A SIMILARITY-GUARANTEED CLUSTERING ALGORITHM AND ITS SEARCH TREE FOR HANDLING AN INCREASED WEIGHT

KREADTISAK LAPPANITCHAYAKUL

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF SCIENCE IN COMPUTER SCIENCE

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2008

KMITL-2008-SC-M-002-372

หัวข้อวิทยานิพนธ์	อัลกอริทึมการจัดกลุ่มที่รับประกันความคล้ายคลึงกันและต้นไม้การค้นหาสำหรับการจัดการค่าน้ำหนักที่เพิ่มขึ้น
นักศึกษา	นายเกียรติศักดิ์ ลาภพาณิชยกุล
รหัสนักศึกษา	47068001
ปริญญา	วิทยาศาสตร์มหาบัณฑิต
สาขาวิชา	วิทยาการคอมพิวเตอร์(หลักสูตรนานาชาติ)
พ.ศ.	2551
อาจารย์ผู้ควบคุมวิทยานิพนธ์	ดร.นวลสวาท หิรัญสกลวงศ์

บทคัดย่อ

ต้นไม้การจัดกลุ่มเป็นโครงสร้างข้อมูลที่ใช้ในการจัดกลุ่มข้อมูล(Clustering Tree) นอกจากนี้ยังทำหน้าที่เป็นต้นไม้การค้นหาสำหรับการค้นหาหมายเลขกลุ่มข้อมูล โดยที่ต้นไม้การจัดกลุ่มถูกสร้างจากต้นไม้แบบทอดข้ามเล็กที่สุด(Minimum Spanning Tree) ในกรณีที่ ถ้าค่าน้ำหนักเส้นเชื่อมของต้นไม้แบบทอดข้ามเล็กสุดมีค่า ไม่เพิ่มขึ้นหรือก็ลดลง ทำให้ต้องเรียกใช้อัลกอริทึมที่ใช้สร้างต้นไม้การจัดกลุ่ม ส่งผลให้เวลาทำงานมีค่าสูงขึ้น อัลกอริทึมที่ได้ออกแบบไว้ในบทความนี้หลีกเลี่ยงการสร้างต้นไม้การจัดกลุ่มใหม่ในทุกๆครั้ง ที่ค่าน้ำหนักเส้นเชื่อมมีค่าลดลง ในงานวิทยานิพนธ์เล่มนี้จะพูดถึงอัลกอริทึมใหม่ เพื่อหลีกเลี่ยงการสร้างต้นไม้การจัดกลุ่มใหม่ทุกๆครั้งเมื่อมีค่าเพิ่มขึ้น การทำงานของอัลกอริทึมนี้ใช้เวลาทำงานเป็น O(n) ต่อการเปลี่ยนแปลงค่าน้ำหนัก 1 ครั้ง และ n คือจำนวนข้อมูล(Nodes) ในกลุ่มข้อมูล

Thesis Title	A Similarity-Guaranteed Clustering Algorithm and Its Search Tree for Handling an Increased Weight
Student	Mr. Kreadtisak Lappanitchayakul
Student ID	47068001
Degree	Master of Science
Program	Computer Science (International Program)
Year	2007
Thesis Advisor	Dr. Nualsawat Hiransakolwong

ABSTRACT

A clustering tree, which is a kind of data structures, is not only used for clustering data but also a search tree for searching cluster number. Normally, it is constructed from a minimum spanning tree. In the case of some weight of the edges is later adjusted: either increased or decreased weight. It is necessary to invoke the Clustering Tree constructing algorithm again. This situation affects the overall running time which can be greater. There was an algorithm for handling the decreased weight. In this thesis, a new algorithm is presented. This new algorithm avoids reconstructing the Clustering Tree at each increased weight. Additionally, its running time is O(n) per weight adjustment where n is the number of nodes in the Clustering Tree.

ACKNOWLEDGMENT

This thesis is completed by advice, encouragement, recommendation and correction from Dr. Nualsawat Hiransakolwong, who is my advisor. She extremely spends her time for me. Therefore, I am very pleased to thank and appreciate for those valuable help.

I would like to express my thanks to thesis committee, Asst. Prof. Dr. Korakot Prachumrak, Asst. Prof. Dr. Jeeraporn Werapun, and Dr. Chakrit Watcharopas who kindly give me the useful comments and suggestions to complete and succeed this thesis.

I would like to thank all lecturers that give me the valuable lectures while studying at King Mongkut's Institute of Technology Ladkabang (KMITL).

I am very thankful to Dr. Nualsawat Hiransakolwong, correcting and recommending the grammar and style writing of this thesis.

I would like to thank my family that encourage me in this study and undertake all responsibilities of mine instead while I am studying.

Thank you all friends for helpful comments, suggestions and other help.

Kreadtisak Lappanitchayakul

TABLE OF CONTENT

PAGE

PAGE

TABLE OF FIGURE

TABLE OF FIGURE (cont.)

TABLE OF TABLE

Chapter 1

Introduction

1.1 Introduction

Clustering, or partitioning into dissimilar groups of similar objects, is a problem with many variants in mathematics and applied sciences. In general major clustering methods can be classified into 5 categories: *partitioning* [10,13,16,18,20,24] , *hierarchical* [8,19,22], *density-based* [1,14,15], *grid-based* [9,17,23], and *model-based* method [4,21] . In addition, some clustering algorithms integrate the ideas of these clustering methods [6, 7]. All of these methods are either heuristic-based or experiential or both. The nature of heuristic approach cannot guarantee quality of the clustering. The typical justification of the clustering quality given by practitioners and experimentalists is case-by-case. Moreover, experimental results also leave the burden of interpretation of the clustering results to the users. Theoreticians, on the other hand, have been busy studying quality measures that are simple to define (e.g. *K*-medians, minimum sum, minimum diameter, etc) and these methods most often use optimization on certain criteria or objective functions. It is well known in clustering literature that there are simple examples in which the right clustering is obvious but optimizing these measures produces undesirable solutions. Clustering problem is an extensive research problem in Data Mining. An example of clustering algorithm is K-Mean Algorithm [2, 11].

Thus far, there have been some quantitative measures for the goodness of clustering results [10, 25]. For instance, the popular k-means algorithm uses a squared-error function to indicate the clustering quality. Even though this function is quantitative, it does not guarantee how good the clustering is. All existing clustering algorithms require a number of parameters as their inputs and these parameters can significantly affect the cluster quality. These parameters are also *case-specific*. That is, some values are good for some certain cases and appropriate values can be found only through trial-and-error. This thesis avoids experiential methods and advocates the idea of *need-specific* as opposed to case-specific because users always know the needs of their applications.

Presently the problem of clustering is usually not associated with the problem of searching points in *d*-dimensional space. Even though there are some clustering algorithms as in [6, 7] that use multidimensional indexing structures such as *R*-trees and *k-d* trees, the two problems are

usually considered independently. In this thesis, clustering method, however, simultaneously produces both a search tree of *d*-dimensional points and clusters (hence, the title). This salient feature is useful in several applications [12]. The Similarity-Guaranteed Clustering Algorithm and Its Search Tree is a Clustering Algorithm of the data in d dimension that can specify the distance value in Inner-Cluster and the distance of the Intra-cluster. Other Clustering Algorithms can not specify the distance value certainly. The Similarity-Guaranteed Clustering Algorithm and Its Search Tree still support searching data. However, the other Clustering Algorithms do not support the searching property.

The Similarity-Guaranteed Clustering Algorithm and Search Tree take the Minimum Spanning Tree (MST), built from Completed Graph, to create the Clustering Tree in Binary Tree for searching cluster. In the case of some weight of the edge in the MST is later adjusted: either increase or decrease weight (weight is the Euclidean Distance of connected node that uses to measure how similarity between these connected nodes). It is necessary to invoke the Clustering Tree constructing algorithm again. This situation affects the overall running time which can be greater. In this thesis, a new algorithm is presented. It avoids reconstructing the Clustering Tree at each weight adjustment and Its running time is O(n) per weight adjustment where n is the number of nodes in the Clustering Tree.

1.2 Objective

A Clustering Tree, which is a kind of data structures, is not only used for clustering data but also a search tree for searching cluster number. Normally, it is constructed from a minimum spanning tree. In the case of some weight of the edges is later adjusted: either increase or decrease weight. It is necessary to invoke the Clustering Tree constructing algorithm again. This situation affects the overall running time which can be greater. In this thesis, a new algorithm is presented. This new algorithm avoids reconstructing the Clustering Tree at each weight adjustment. Additionally, its running time is O(n) per weight adjustment where n is the number of nodes in the Clustering Tree.

1.3 Hypothesis

When a connected line in MST is later adjusted: either increase or decrease weight, this effects the MST and Clustering Tree. This thesis tries to find algorithms to modify the old MST

and Clustering Tree to become the corrected MST that corresponds with the adjusted weight. This new algorithms avoid reconstructing the Clustering Tree at each weight adjustment.

1.4 Frameworks

This thesis builds the Clustering Tree from the MST. Then study the effect of MST and Clustering Tree when a connected line in MST is later adjusted: either increase or decrease weight. Find algorithms to modify the old MST and Clustering Tree to become the corrected MST that corresponds with the adjusted weight. This new algorithms avoid reconstructing MST and the Clustering Tree at each weight adjustment.

1.5 The content in the thesis

Chapter 2 addresses the theory and the related researches distributed into 3 parts. First part addresses about the theory of Minimum Spanning Tree. Second part addresses the effect of MST and Clustering Tree when a connected line in MST is later decrease weight. Third part addresses the algorithm to adjust Clustering Tree.

Chapter 3 addresses the theory distributed into 3 parts. First part addresses how to find Minimum Spanning Tree, MST, from a completed graph. Second part addresses the effect of MST and Clustering Tree when a connected line in MST is later increase weight. Third part addresses the algorithms to adjust MST and Clustering Tree by avoiding reconstruction them again, and then ending with proof theories.

Chapter 4 does some simulations, and then ending with the remarkable conclusions in Chapter 5.

Chapter 2

Related works

The weight of a connection line in MST can be decrease and increase. This chapter is presented a decreased weight of a connection line which distributed into 3 part. First part addresses about the theory of Minimum Spanning Tree. Second part addresses the effect of MST and Clustering Tree when a connected line in MST is later decrease weight. Third part addresses the algorithm to adjust Clustering Tree.

2.1 MST-based clustering algorithm

A. Minimum Spanning Trees (MST)

Given a connected, undirected graph $G = (V,E)$, where V is the set of nodes, E is the set of edges between pairs of nodes, and a weight $w(u, v)$ specifying the weight of the edge (u, v) for each edge $(u, v) \in E$, the Minimum Spanning Tree problem is to find an acyclic subset $T \subseteq$ E that connects all of the vertices and whose total weight $(T) = \sum_{(u,v)\in T} w(u,v)$ is minimized [26,27,28].

B. Clustering Problem

A clustering of a set is a partition of its elements into subsets (clusters) according to some measure of optimality [29]. Although there are many variations of optimization criteria, they can be divided into two board classes depending on whether some *inner-cluster* measure or *intra-cluster* measure is used. The main emphasis, however, is that elements in the same cluster should be more similar to one another than they are to elements in the other clusters. Several empirical and heuristic clustering algorithms exist to solve the clustering problem as in [11].

C. MST Clustering Model

It is obvious that a Minimum Spanning Tree can be used to model a clustering of *d*-dimensional points. A point in the clustering problem can be represented by a corresponding node in the graph. A distance between a pair of points can be represented by a corresponding weighted edge. In [30] clustering problem under the two optimization criteria were considered and both maximum and Minimum Spanning Trees were used in their algorithms. However, only points in two dimensions were considered in this paper. In addition, each of the optimization criteria was used separately. In this thesis algorithm is also based on the Minimum Spanning

Tree but is not limited to two-dimensional points. Moreover, the proposed algorithm produces clusters of *d*-dimensional points with a guaranteed diameter and a naturally appropriate inter-cluster distance.

D. The algorithm

The MST-based clustering algorithm works on a data structure called "Clustering Tree". A Clustering Tree is a binary tree that assists the algorithm in the clustering process. The definition of a Clustering Tree and the recursive algorithm to build the Clustering Tree is addressed as follows:

2.2 The effect of MST and Clustering Tree of MST

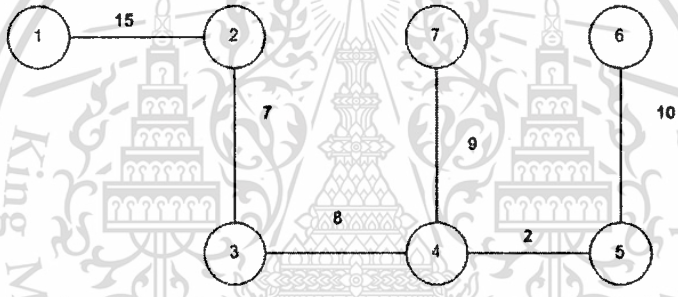

Figure 2.1: Minimum Spanning Tree

Definition 2.1: Given a Minimum Spanning Tree (MST) T , a Clustering Tree is a binary tree in which each node contains, among other things, the weight of a connection line between a pair of points a, b in T and the corresponding point identification numbers of a and b. Each parent node always contains a larger weight than that of its children.

Figure 2.1 and 2.2 illustrate a Minimum Spanning Tree and its corresponding Clustering Tree respectively.

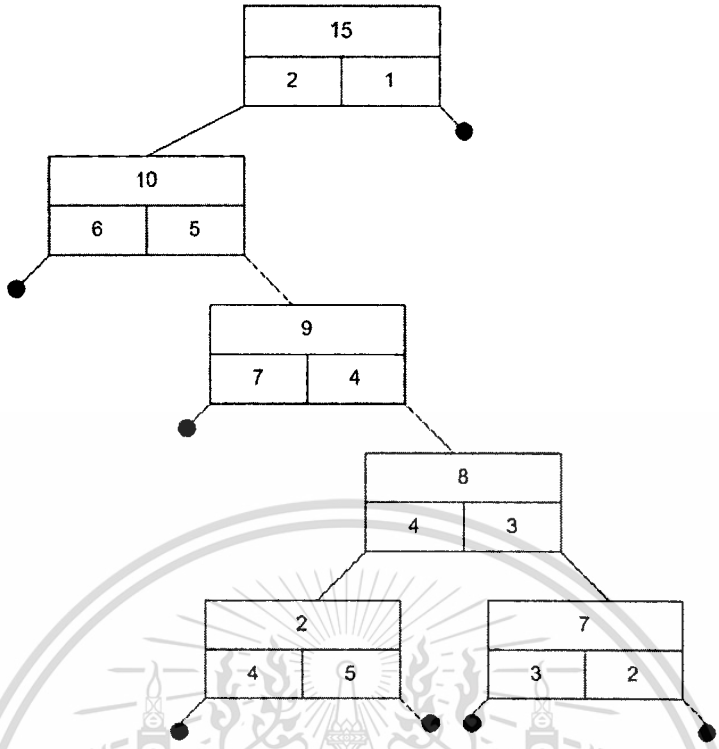

Figure 2.2: Clustering Tree

Clustering Tree, in Figure 2.2, is build from MST in Figure 2.1. Root node of Clustering Tree consists of the distance between node 2 and node 1 which connect together. The algorithm to build Clustering Tree is as follows.

(Hierarchical Structure)

Algorithm: Clustering Tree

Input: a Minimum Spanning Tree T and its root r

Output: a Clustering Tree S.

```
1.  if (T has some edge)
2.      e = get-max-edge(T)
3.      rs = make-bank-node()
4.      fill-info(rs, e)
5.      return (Tree(leftSubtree(r), r.left-node-id), Tree(rightSubtree(r),
        r.right-node-id), rs)
6.  else
7.      rs = make-blank-node()
8.      return rs
```

The Clustering Tree algorithm recursively transforms a Minimum Spanning Tree T to a corresponding Clustering Tree S. It first checks to see if T still contains some edge (line 1). If so, it finds a maximum edge e and creates a blank root node rs of the tree S (line 2-4). It then recursively solves the left and right minimum spanning subtrees (line 5) until T has no edge and S is returned (lines 7-8).

Theorem 2.1: The Clustering Tree algorithm takes $O(|E|^2)$ time.

Proof: A recurrence from the algorithms can be derived as follows. Let T(|E|) be the number of step of the Algorithm.

$$T(|E|) = T(J) + T(|E| - (1 + J)) + |E|$$

Where $1 \leq J \leq |E| - 2$ and $T(1) = 1$. From this recurrence, T (|E|) is maximized when $J = 1$ or $J = |E| - 2$. Without loss of generality, let J=1. The iteration method is used to derive the following equation.

$$T(|E|) = I|E| - I(I - 2) + T(|E| - 2I)$$

There I is and integer. This equation stops iterating when $T(|E| - 2I) = T(1)$ or $I = \frac{|E|-1}{2}$. Substitute $\frac{|E|-1}{2}$ for I in the equation. Then $T(|E|)=O(|E|^2)$.

Once a Clustering Tree is constructed, the Clustering algorithm is used to get clusters of points. The Clustering algorithm can be described as follows.

Algorithm Clustering
Input: a Clustering Tree *S* and a desired diameter *d*.
Output: *k* clusters where *k* is also an output.

```
Clustering(S, d)
1. if (diameter(S) > d)
2.     Clustering(left-subtree(root(S)), d)
3.     Clustering(right-subtree(root(S)), d)
4. else
5.     root(S).cluster-number = cluster-number()
6.     print-cluster-number()
7.     traverse-and-print(S)
8. return
```

The Clustering algorithm firstly checks to see if the diameter of all points in S is greater

than the given d (line1). If so, it recursively solves the sub problems both left and right subtrees (line 2-3) until the diameter becomes less than or equal to d. It then proceeds to record the cluster number in the root node of the current subtree for the current cluster (line5). Note that function *cluster-number()* in line 5 generates a sequence of number and is a global function. The algorithm reports the cluster number and its members in lines 6-7.

Theorem 2.2: The Clustering algorithm produces k clusters, each of which has a diameter $\leq d$.

Proof: In line 1-3 the algorithm recursively separates a larger tree into two smaller subtrees if the diameter of points in the larger tree is greater than d. In line 4-6 the algorithm outputs points in a cluster only when the diameter of points in the cluster is smaller than or equal to d.

In Figure 2.2, if user insert the value of d = 9, the Clustering Tree can be arranged into 3 groups e.g. set of {1}, set of {6} and set of {2,3,4,5,7} and the running time of this algorithm is $O(n^3)$ when n is the amount node of Clustering Tree.

Theorem 2.3: The Clustering algorithm takes $O(n^3)$ time.

Proof: Let n be the number of nodes in the Clustering Tree. In the worst case the Clustering Tree S is recursively divided into two subtrees of sizes n-1 and 1 until each point becomes a singleton cluster (i.e., d is smaller than the least weight of all edges in the Minimum Spanning Tree). Because the cost of checking the diameter of the tree is n^2, a recurrence can be derived as follows. Let $T(n)$ be the number of steps in the algorithm.

$$T(n) = T(n-J) + T(J) + n^2$$

$T(n)$ is maximized when $J = 1$ or $J = n - 1$. Without loss of generally, let $J = 1$. Using the method of the iteration, the following summation can be derived.

$$T(n) = \sum_{i=0}^{n-2} [(n-i)^2] + n$$

$$= \frac{1}{6}[2n^3+3n^2+7n-6] = O(n^3)$$

Thus, the Clustering algorithm takes $O(n^3)$ time.

The Clustering Tree algorithm, on the other hand, produces a search tree with a well-defined structure that enables the search time to be proportional to the height of the tree. If the trec is balanced, the height will be logarithmic. The Clustering Tree algorithm is as follows.

Algorithm Search

Input: a Clustering Tree *S*, its root node *r*, and a query point *p*.

Output: cluster membership number of the point *p*.

```
Search(S, r, p)
1.  if (r.cluster-number = nil)
2.      if (d(r.left-point, p) > d(r.right-point, p))
3.          Search(right-subtree(r), r.right-node, p)
4.      else
5.          if (d(r.left-point, p) < d(r.right-point, p))
6.              Search(left-subtree(r), r.left-node, p)
7.  else
8.      print(r.cluster-number)
9.  return
```

In this algorithm $d(x, y)$ is an Euclidian function that returns the distance between points x, y. Assume that there is no case in which d (*r*.left-point, p) = d (*r*.right-point, p). The algorithm begins by checking the cluster number of *r* (line 1). If it is nil, it recursively searches one of the two subtrees, depending upon the nearer distance to *p* (lines 2-6). The algorithm continues until it finds a cluster number in *r*, which indicates the cluster membership of *p* (line 8). Please note that the algorithm returns a cluster membership number, even though in actuality the point p may not exist in the cluster. In other words, it returns the cluster number to which the point *p* should belong.

Lemma 2.1: Let *S* be a Clustering Tree, *r* the root node of *S*, and *p* a given point. If $d(y, p) < d(z, p)$ where *y* is one point in root node *r* and *z* is the other point in root node *r*, then $d(x_1, p) < d(x_2, p)$ for all points x_1 in the same subtree as *y* and for all points x_2 in the same subtree as *z*.

Proof: *The* proof is processed by using contradiction. Suppose there exists some point x_1 in the same subtree as *y* and some point x_2 in the same subtree as *z* such that $d(x_1, p) > d(x_2, p)$ (i.e., Assume that there is no the equal case). Because $d(x_1, p) > d(x_2, p)$, *p* must be in the same subtree as x_2 for otherwise the edge incident on *p* in the corresponding minimum spanning tree would not be minimized. It follows that there must exist a *minimum* edge

(m, p) in the corresponding Minimum Spanning Tree for some point m in the same subtree as z. If $m = z$, $d(y, p) < d(z, p)$. If $m \neq z$, we know that $d(y,p) < d(m, p) = d(z, p) + \Sigma$, for some constant Σ. Hence, this is a contradiction since $d(m, p)$ is not minimized. Lemma 2.1 is true.

Theorem 2.4: The Search algorithm is correct.

Proof: The proof is an induction on the depth of the Clustering Tree with the application of Lemma 2.1.

2.3 The algorithm to remedy Clustering Tree

Clustering Tree is Binary Tree that is built from Minimum Spanning Tree. In case the value of connected line in MST is decreased then the Clustering Tree is adjusted as follows.

Algorithm Adjust
Input: a Clustering Tree node i.
Output: a new Clustering Tree node i.
Adjust(i)
1.if (i<i.left and i.right $\leq$ i.left)
2. if(d(i,i.left.left) $\leq$ d(i,i.left.right))
3. i=Rotate_Left_Right(i)
4. return Adjust(i)
5. else
6. i=Rotate_Left_Left(i)
7. return Adjust(i)
8.else if(i<i.right and i.left $\leq$ i.right)
9. if (d(i,i.right.left) $\leq$ d(i,i.right.right))
10. i=Rotate_Right_Left(i)
11. return Adjust(i)
12. else
13. i=Rotate_Right_Right(i)
14. return Adjust(i)
15. else
16. return null

Algorithm Rotate_Left_Left
Input: a Clustering Tree node i.
Output: a Clustering Tree node i.left.
Rotate_Left_Left(i)
1. x=i.left
2. x.left=i
3. x.left.left=i.left.left
4. i=x
5. return i.left

Algorithm Rotate_Left_Right
Input: a Clustering Tree node i.
Output: a Clustering Tree node i.right.
Rotate_Left_Right(i)
1. x=i.left
2. x.right=i
3. x.right.left=i.left.right
4. i=x
5. return i.right

Algorithm Rotate_Right_Left
Input: a Clustering Tree node i.
Output: a Clustering Tree node i.left.
Rotate_Right_Left(i)
1. x=i.right
2. x.left=i
3. x.left.right=i.right.left
4. i=x
5. return i.left

Algorithm Rotate_Right_Right
Input: a Clustering Tree node i.
Output: a Clustering Tree node i.right.
Rotate_Right_Right(i)
1. x=i.right
2. x.right=i
3. x.right.right=i.right.right
4. i=x
5. return i.right

The Adjust algorithm works by continuously rotating node of Clustering Tree in Figure 2.4(a)-(c) until the Clustering Tree getting its property. This algorithm consists of four types of rotation: Rotate_Left_Left, Rotate_Left_Right, Rotate_Right_Left, and Rotate_Right_Right. The Adjust algorithm will begin to adjust node i when i is a node of Clustering Tree that corresponds with the decreased weight in MST.

For example in Figure 2.2 when the weight of the connected line between node 7 and node 4 decreases into 1, then the MST is changed as shown in Figure 2.3. Its Clustering Tree is shown in Figture 2.4(a). This Figure 2.4(a) does not have the right property because parent node has smaller weight than their child nodes. Therefore the Adjust algorithm is processed with node (3). The algorithm gets the condition in line 8-9, and then does the Rotate_Right_Left as shown in Figure 2.4(b). The Figure 2.4(b) does not have the right property because parent node has smaller weight than their child nodes. Therefore the Adjust algorithm is processed with node (3). The algorithm gets the condition in line 8-9, and then does the Rotate_Right_Left as shown in Figure 2.4(c). The Figure 2.4(c) has the right property now and then stop the process.

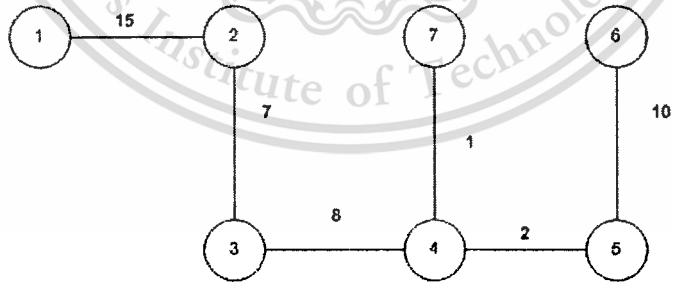

Figure 2.3: The MST when decrease weight between node 7 and node 4

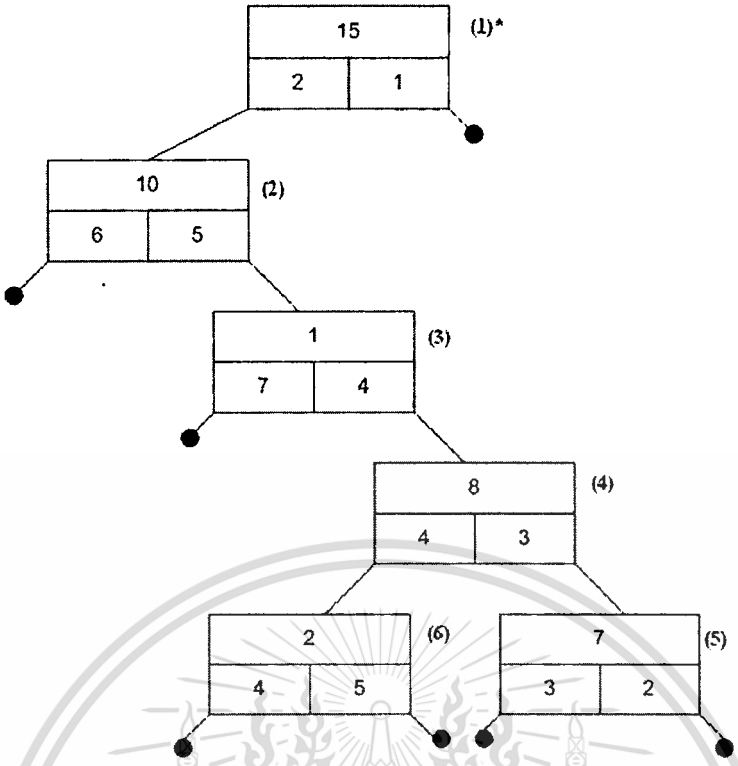

Figure 2.4(a): Clustering Tree from Figure 2.2 changed data on node (3)

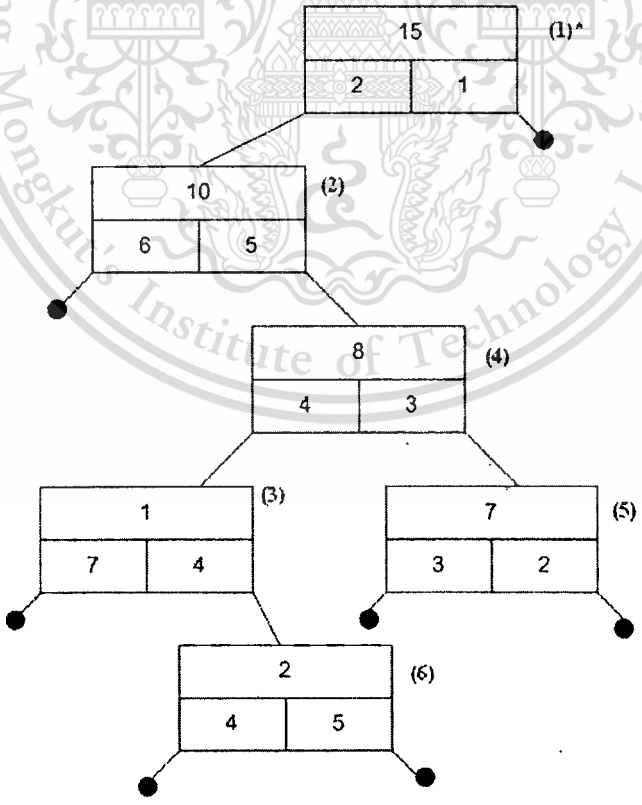

Figure 2.4(b): Clustering Tree from Figure 2.4(a)
with Rotate_Right_Left on node (3)

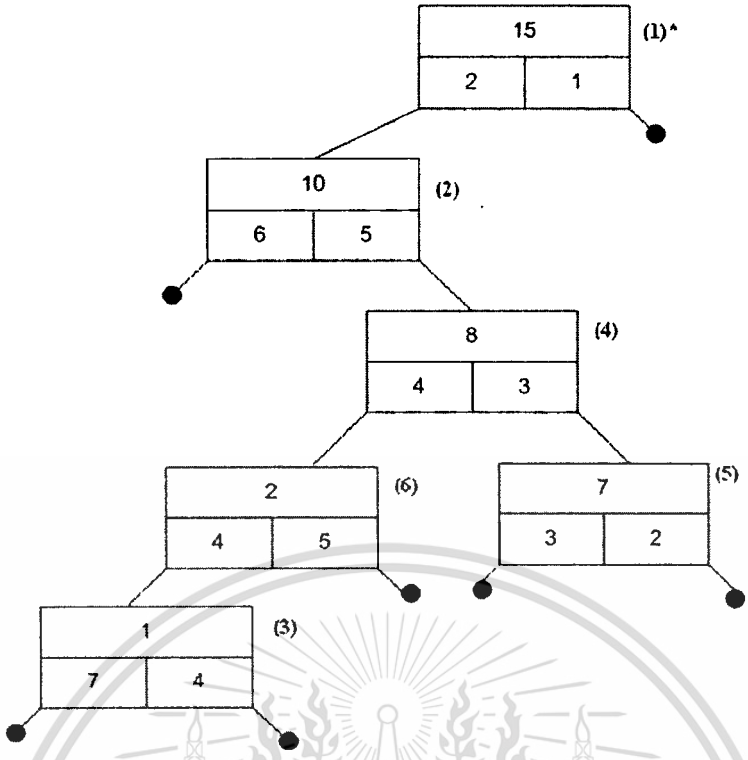

Figure 2.4(c): Clustering Tree from Figure 2.4(b)
with Rotate_Right_Left on node (3)

The rotating node one time makes node down one level. For the worst case, node move down beginning from root to leave level, the height of the tree. Therefore the time complexity of the Adjust algorithm is O(h), where h is the height of the tree.

Theorem 2.5: Time complexity of the Adjust algorithm is O (h) where h is the height of Clustering Tree.

Proof: Decide T (m) is the running time and m is the level of Clustering Tree thus can write Recurrence and solving the equation by Iteration Method as follows.

$$
\begin{aligned}
T(m) &= T(m-1) + O(1) \\
&= T(m-2) + 2O(1) \\
&= T(m-(m-1)) + (m-1)O(1) \\
&= T(1) + mO(1) - O(1) \\
&= O(m) = h = O(h)
\end{aligned}
$$

From Theorem 2.5, the time complexity of the Adjust algorithm is O (n) when n is the number of nodes in Clustering Tree because the Clustering Tree may become a sparse tree as a straight line.

Theorem 2.6: Rotating the node of Clustering can preserve the property of Clustering Tree one level.

Proof: prove by enumerating the cause with the theorem as follows.

Case 1: i<il, ir<il and d(i,ill) $\leq$d(i,ilr)

Casc 2: i<il, ir<il and d(i,ill) >d(i,ilr)
Case 3: i<ir, il<ir and d(i,irl) $\leq$d(i,irr)
Case 4: i<ir, il<ir and d(i,irl) >d(i,irr)

Where i represents node *i* that adjust its weight. *ir* represents *i*.right and *il* represents *i*.left. *ill* represents *i*.left.left and *ilr* represents *i*.left.rigth. *irl* represents *i*.right.left and *irr* represents *i*.right.right.

For the case 1, follows a MST in Figure 2.5 that corresponds with Clustering Tree in Figure 2.6. Node *i* is changed. Node will rotate as shown in Figure 2.7(a). This preserves the property Sub clustering with node i because (from the Lemma 2.1) the weight between node *i* and node *il* is less than the distance (the minus weight between 2 node in Clustering Tree) between node *ilr* and node *il* and this rotating preserve the correct relation in Clustering Tree one level. Without Loss Generality, the proof can be processed in the similar way for case 2-4.

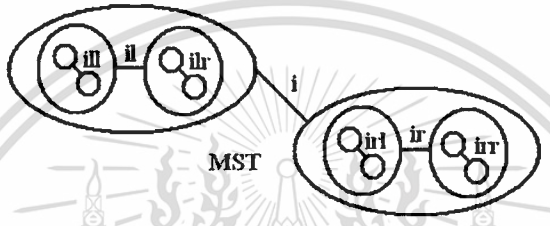

Figure 2.5: a general MST

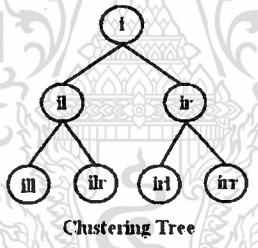

Figure 2.6: a Clustering Tree

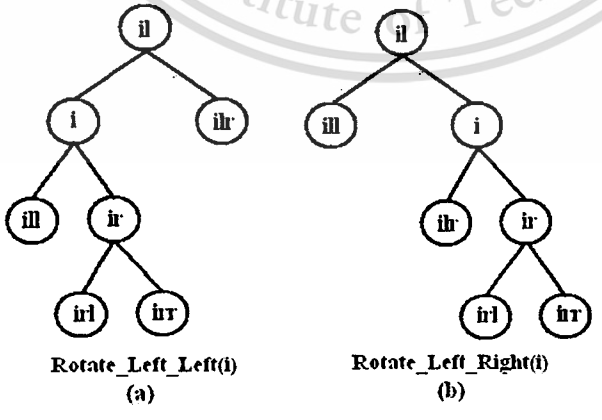

Figure 2.7(a): Rotating_Left_Left and Rotate_Left_Right in a Clustering Tree

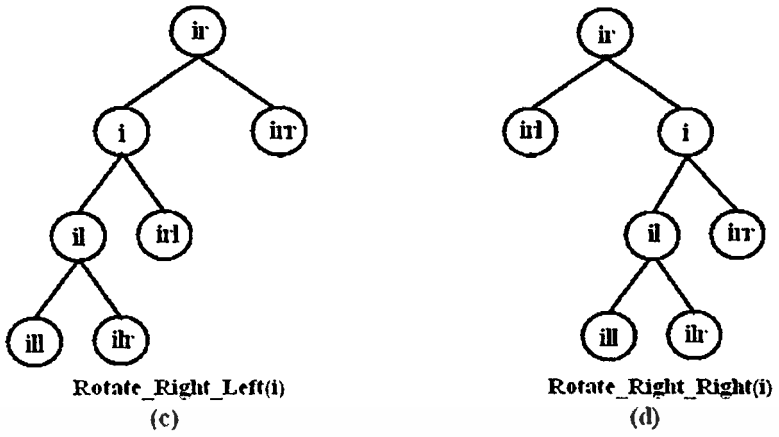

Figure 2.7(b): Rotating_Right_Left and Rotate_Right_Right in a Clustering Tree

Theorem 2.7: adjust algorithm can work correctly

Proof: Let n be the level of the tree. This proof uses induction manner with the Theorem 2.6 as follows. In case of the base when n=1 be corrected by the Theorem 2.6 because the Clustering Tree has 3 nodes. Leaf nodes will rotate go to root node by Theorem 2.6. Inductive Setup n>L, the rotating of Clustering Tree at n=L preserve the property of Clustering Tree by Theorem 2.6 and node will not rotate beyond height value of the tree at n=h. Thus the Adjust algorithm can work correctly.

A Clustering Tree, which is a kind of data structures, is not only used for clustering data but also a search tree for searching cluster numbers. Normally, it is constructed from a Minimum Spanning Tree. In the case of some weight of the edges is later decrease weight k time. It is necessary to invoke the Clustering Tree constructing algorithm again. This situation affects the overall running time O (kn^2) which can be greater. The Adjust algorithm is presented. This new algorithm avoids reconstructing the Clustering Tree at each weight adjustment. Additionally, its running time is O (kn) where n is the number of nodes in the Clustering Tree.

The Adjust algorithm in this chapter can support for the decrease weight. In the Chapter 3, the new algorithms will be proposed that can support for any increase weight of a connection line in the MST.

Chapter 3

The proposed algorithm

3.1 Finding Minimum Spanning Tree

In Chapter 2, whenever the weight of a connection line (in the MST graph) is decreased, other connection lines do not get any effect. Therefore, the MST graph still qualifies its shortest properties. There is an effect only the Clustering Tree. The Clustering Tree can be adjusted by algorithm in Chapter 2. When the weight of a connection line (in the MST graph) is increased, other connection lines may get an affect because the MST graph may not qualify its shortest properties. Therefore both MST and Clustering Tree may get an effect. This chapter will be explained algorithms how to adjust the MST and Clustering Tree to preserve their properties by avoiding reconstruction them.

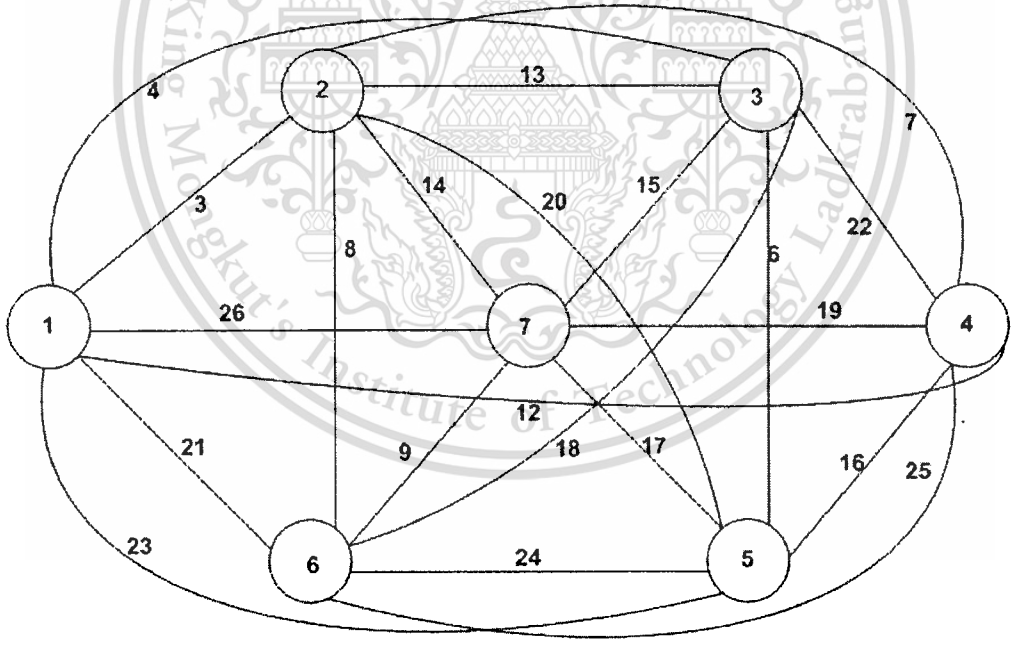

Figure 3.1: A completed graph

A completed graph is a graph that there is a connection line between any two nodes. The database tables of the Completed graph, as shown in Table 3.1, are the information of the connection lines and their weights. An ascending sort of weights from Table 3.1 is shown in Table 3.2. This information is useful for finding the MST of the completed graph.

Table 3.1: The DB tables of a Completed Graph

start point	end point	weight
1	2	3
1	3	4
1	4	12
1	5	23
1	6	21
1	7	26

(1)

start point	end point	weight
2	1	3
2	3	13
2	4	7
2	5	20
2	6	8
2	7	14

(2)

start point	end point	weight
3	1	4
3	2	13
3	4	22
3	5	6
3	6	18
3	7	15

(3)

start point	end point	weight
4	1	12
4	2	7
4	3	22
4	5	16
4	6	25
4	7	19

(4)

start point	end point	weight
5	1	23
5	2	20
5	4	6
5	5	16
5	6	24
5	7	17

(5)

start point	end point	weight
6	1	21
6	2	8
6	3	18
6	4	25
6	5	24
6	7	9

(6)

start point	end point	weight
7	1	26
7	2	14
7	3	15
7	4	19
7	5	17
7	6	9

(7)

Table 3.2: An ascending sort of weights from Table 3.1

start point	end point	weight
1	2	3
1	3	4
1	4	12
1	6	21
1	5	23
1	7	26

(1)

start point	end point	weight
2	1	3
2	4	7
2	6	8
2	3	13
2	7	14
2	5	20

(2)

start point	end point	weight
3	1	4
3	5	6
3	2	13
3	7	15
3	6	18
3	4	22

(3)

start point	end point	weight
4	2	7
4	1	12
4	5	16
4	7	19
4	3	22
4	6	25

(4)

start point	end point	weight
5	3	6
5	4	16
5	7	17
5	2	20
5	1	23
5	6	24

(5)

start point	end point	weight
6	2	8
6	7	9
6	3	18
6	1	21
6	5	24
6	4	25

(6)

start point	end point	weight
7	6	9
7	2	14
7	3	15
7	5	17
7	4	19
7	1	26

(7)

The MST can be created from the completed graph by many algorithms. This thesis uses Prim's Algorithm. The Prim's Algorithm is as follows.

Prim's algorithm

Rather than build a subgraph one edge at a time, Prim's algorithm builds a tree one vertex at a time.

```
Prim's algorithm:
let T be a single vertex x
while (T has fewer than n vertices)
{
    find the smallest edge connecting T to G-T
    add it to T
}
```

Using Prim's algorithm, the MST graph of the completed graph in Figure 3.1 is shown in Figure 3.2 and its Clustering Tree is shown in Figure 3.3.

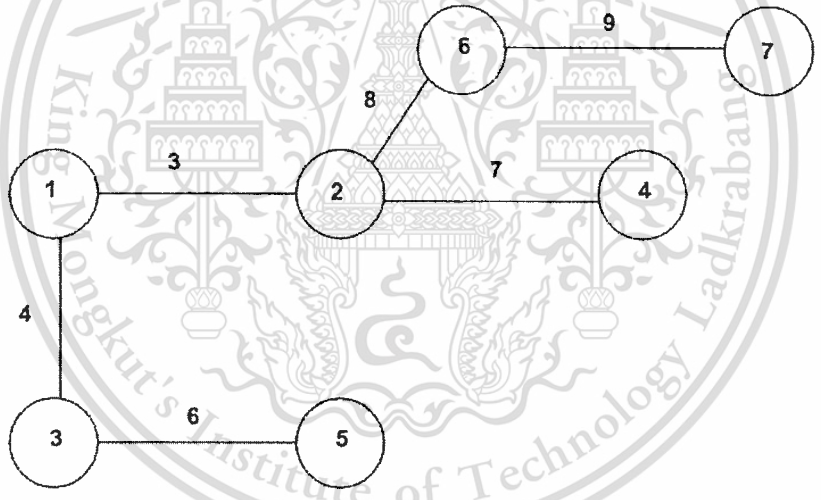

Figure 3.2: The MST of Figure 3.1

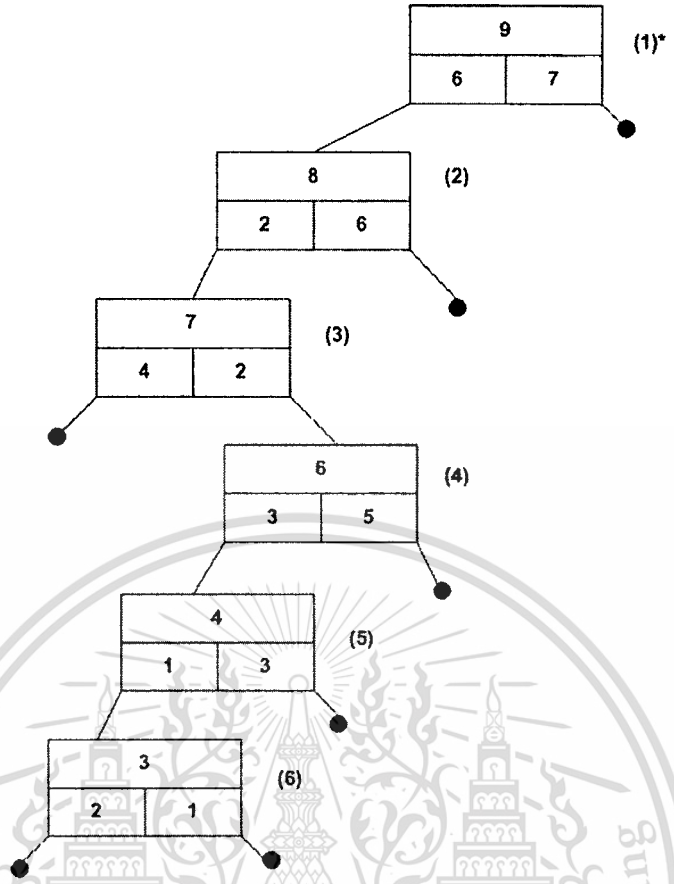

Figure 3.3: A Clustering Tree of the MST from Figure 3.2

3.2 The effect of MST and Clustering Tree

In the Figure 3.4, the weight of a connection line in MST between node 3 and node 5 is decreased from 6 to be 2, the value of new weight. The MST graph has a decreased weight of a connection line that does not affect other connection lines. Therefore, the MST graph still qualifies its shortest properties. There is an effect only the Clustering Tree. The Clustering Tree can be adjusted by algorithm in Chapter 2. In the Figure 3.5, the MST graph has an increased weight of a connection line between node 3 and node 5 from 6 to be 18, the value of new weight. It may affect other connection lines. Therefore, the MST graph may not qualify its shortest properties. There are effects both MST and the Clustering Tree. This chapter will explain algorithms how to handle the adjustment of both MST and the Clustering Tree to preserve their properties by avoiding reconstruction them again.

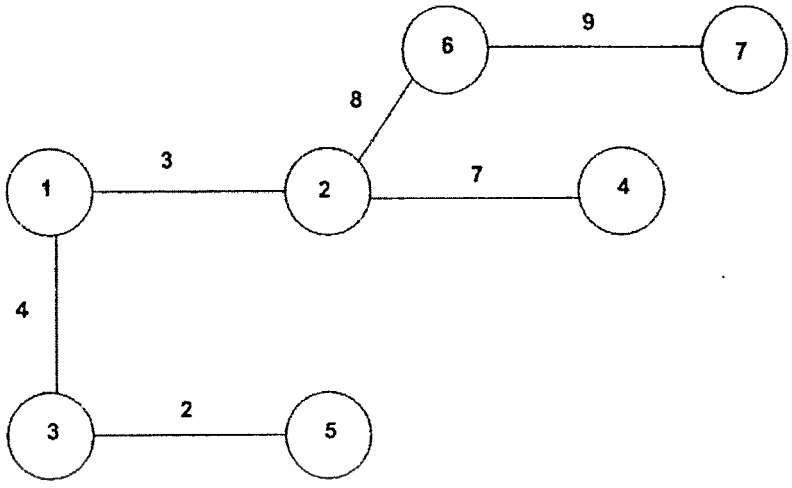

Figure 3.4: A connection line of MST is decreased between nodes 3 and 5

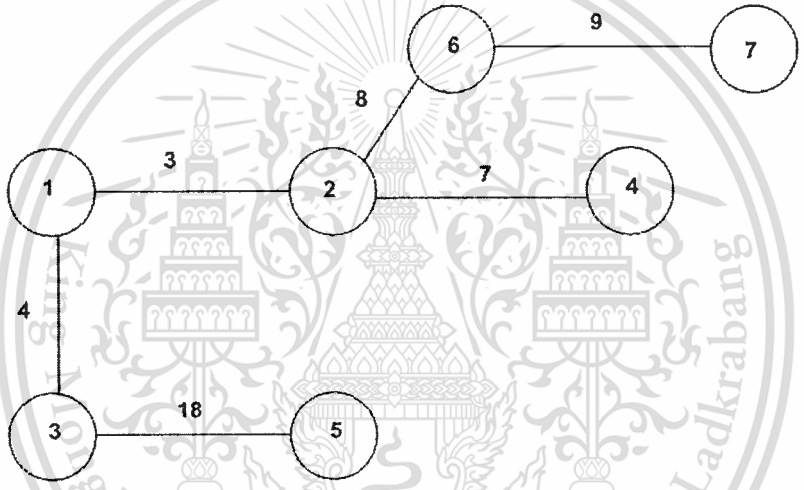

Figure 3.5: A connection line of MST is increased between nodes 3 and 5

From Figure 3.5, the weight of a connection line in MST between node 3 and node 5 is increased from 6 to be 18, the value of new weight. The increased weight of a connection line in the MST may affect other connection lines. The MST may not qualify its properties because it may not preserve the shortest connection. After remove the increased weight of a connection line, MST is divided into two graphs as shown in Figure 3.6. From Figure 3.6, each graph can be found its element set. There are two sets; {5}, and {1, 2, 3, 4, 6, 7}. To find a new shortest weight of a connection line connecting these two graphs uses the DB tables of the set that has fewer members. Here {5} has fewer members. The new weight adjusts the DB table of node 5 from Table 3.2 to become a new DB table as shown in Table 3.3. The information in Table 3.3 will be used to find the new shortest weight of a connection line to connect those two graphs as shown in Figure 3.7.

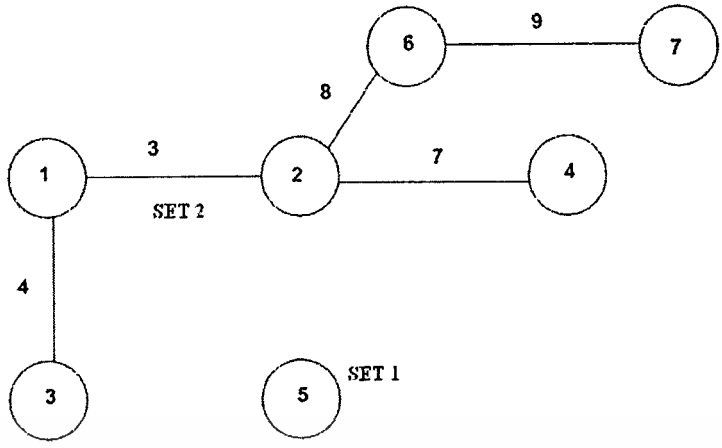

Figure 3.6: The MST from Figure 3.5 is divided between nodes 3 and 5

Therefore the new DB table of node 5, as shown in Table 3.3, will be used for finding the new shortest weight of a connection line. To find the new connection line, choose the first line in the DB table which its start point and its end point are from the different sets. The new connection line is the line between nodes 5 and 4 which its weight is 16. Then the new MST is shown in Figure 3.7. The Clustering Tree of MST from Figure 3.7 is shown in Figure 3.8.

Table 3.3: The new DB table of node 5

start point	end point	weight
5	4	16
5	7	17
5	3	18
5	2	20
5	1	23
5	6	24

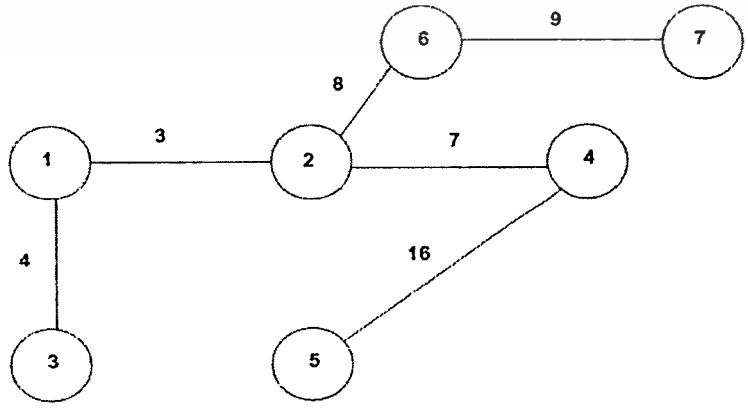

Figure 3.7: The new MST graph

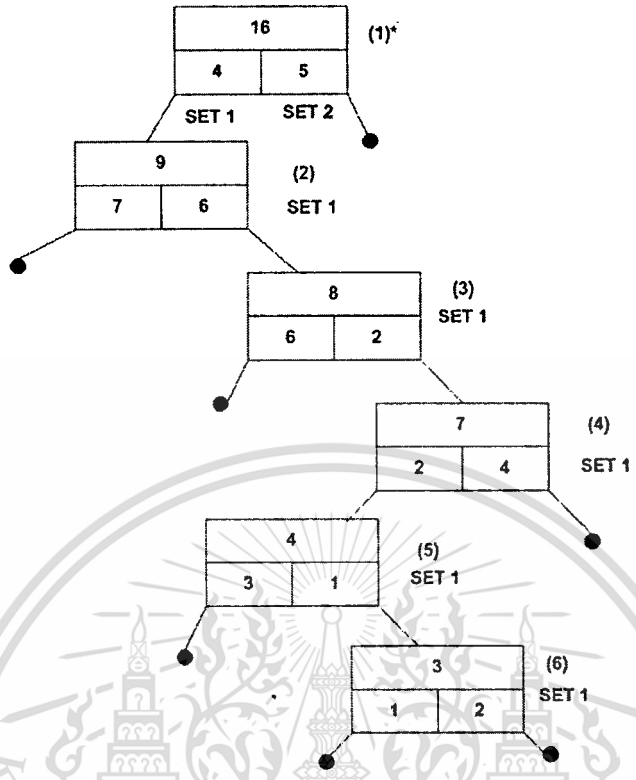

Figure 3.8: A Clustering Tree of the MST from Figure 3.7

From Figure 3.5, the weight of a connection line in MST between node 3 and node 5 is increased from 6 to be 18, the value of new weight. The increased weight of a connection line in the MST affects MST and Clustering Tree. The MST graph from Figure 3.5 is changed to be the MST graph in Figure 3.7 and the Clustering Tree from Figure 3.3 is changed to be the Clustering Tree in Figure 3.8. Next session will explain algorithms how to handle the adjustment of both MST and the Clustering Tree to preserve their properties by avoiding reconstruction them again.

3.3 The algorithms to adjust MST and Clustering Tree

From Figure 3.5, the weight of a connection line in MST between node 3 and node 5 is increased from 6 to be 18, the value of new weight. The increased weight of a connection line in the MST may affect other connection lines. The MST may not qualify its properties because it may not preserve the shortest connection. After remove the increased weight of a connection line, MST is divided into two graphs as shown in Figure 3.6. From F

Chapter 4

Simulations

This chapter presents the simulations to solve problems for an increased weight of its connection line in MST by creating a program from algorithm which is presented in Chapter 3 as follows:

1. Program create n nodes, $n \leq 500$.
2. Program randomly generates weights corresponding with its connection lines to be a completed graph.
3. Keep the ascending DB tables for each node in the text file.
4. Find the MST from those DB tables using Prim's algorithm.
5. Program randomly change a connection line and increase its weight randomly.
6. Adjust the DB table with the new information from step 5.
7. Find the new MST and adjust Clustering Tree as algorithm in Chapter 3.

This chapter presents simulation examples to solve problems for an increased weight of its connection line in MST. The increased weight will affect the MST and its Clustering Tree. These examples, from the completed graph of 10 nodes or 13 nodes, show how to adjust the MST and its Clustering Tree.

The completed graph of 10 nodes is shown in Figure 4.1. The database tables of the Completed graph, as shown in Table 4.1, are the information of the connection lines and their weights. An ascending sort of weights from Table 4.1 is shown in Table 4.2. This information is useful for finding the MST of the completed graph.

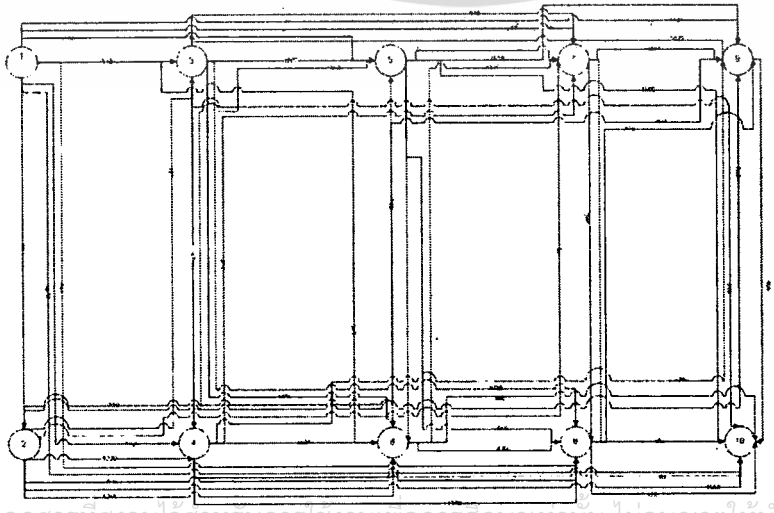

Figure 4.1: A Completed Graph of 10 nodes

Table 4.1: The DB tables of a Completed Graph with 10 nodes

start point	end point	weight
1	2	87
1	3	512
1	4	1620
1	5	1655
1	6	353
1	7	297
1	8	743
1	9	1149
1	10	1978

(1)

start point	end point	weight
2	1	87
2	3	127
2	4	1030
2	5	739
2	6	1245
2	7	52
2	8	846
2	9	722
2	10	1059

(2)

start point	end point	weight
3	1	512
3	2	127
3	4	1147
3	5	1807
3	6	1659
3	7	945
3	8	1932
3	9	1029
3	10	959

(3)

start point	end point	weight
4	1	1602
4	2	1030
4	3	1147
4	5	1893
4	6	1501
4	7	362
4	8	1832
4	9	1798
4	10	51

(4)

start point	end point	weight
5	1	1655
5	2	739
5	3	1807
5	4	1893
5	6	748
5	7	1538
5	8	548
5	9	644
5	10	1500

(5)

start point	end point	weight
6	1	1655
6	2	1245
6	3	1659
6	4	1501
6	5	748
6	7	1540
6	8	430
6	9	650
6	10	369

(6)

Table 4.1(cont.): The DB tables of a Completed Graph with 10 nodes

start point	end point	weight
7	1	297
7	2	52
7	3	945
7	4	362
7	5	1538
7	6	1540
7	8	865
7	9	1461
7	10	463

(7)

start point	end point	weight
8	1	743
8	2	846
8	3	1932
8	4	1832
8	5	548
8	6	430
8	7	865
8	9	879
8	10	463

(8)

start point	end point	weight
9	1	1149
9	2	722
9	3	1029
9	4	1798
9	5	644
9	6	650
9	7	1461
9	8	879
9	10	486

(9)

start point	end point	weight
10	1	1978
10	2	1059
10	3	959
10	4	51
10	5	1500
10	6	369
10	7	463
10	8	465
10	9	486

(10)

Table 4.2: An ascending sort of weights from Table 4.1

start point	end point	weight
1	2	87
1	7	297
1	6	353
1	3	512
1	8	743
1	9	1149
1	4	1620
1	5	1655
1	10	1978

(1)

start point	end point	weight
2	7	52
2	1	87
2	3	127
2	9	722
2	5	739
2	8	846
2	4	1030
2	10	1059
2	6	1245

(2)

start point	end point	weight
3	2	127
3	1	512
3	7	945
3	10	959
3	9	1029
3	4	1147
3	6	1659
3	5	1807
3	8	1932

(3)

start point	end point	weight
4	10	52
4	7	362
4	2	1030
4	3	1147
4	6	1501
4	1	1602
4	9	1798
4	8	1832
4	5	1893

(4)

start point	end point	weight
5	8	548
5	9	644
5	2	739
5	6	748
5	10	1500
5	7	1538
5	1	1655
5	3	1807
5	4	1893

(5)

start point	end point	weight
6	10	369
6	8	430
6	9	650
6	5	748
6	2	1245
6	4	1501
6	7	1540
6	1	1655
6	2	1659

(6)

Table 4.2(cont.): An ascending sort of weights from Table 4.1

start point	end point	weight
7	2	52
7	1	297
7	4	362
7	10	463
7	8	865
7	3	945
7	9	1461
7	5	1538
7	6	1540

(7)

start point	end point	weight
8	6	430
8	10	463
8	5	548
8	1	743
8	2	846
8	7	865
8	9	879
8	4	1832
8	3	1932

(8)

start point	end point	weight
9	10	486
9	5	644
9	6	650
9	2	722
9	8	879
9	3	1029
9	1	1149
9	7	1461
9	4	1798

(9)

start point	end point	weight
10	4	51
10	6	369
10	7	463
10	8	465
10	9	486
10	3	959
10	2	1059
10	5	1500
10	1	1978

(10)

Using Prim's algorithm, the MST graph of the completed graph in Figure 4.1 is shown in Figure 4.2.

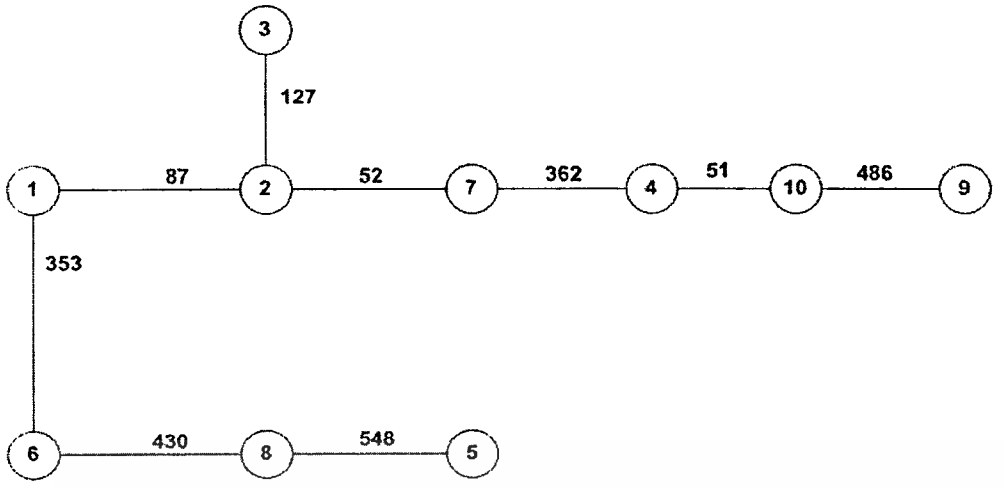

Figure 4.2: The MST on Figure 4.1

Using Clustering Tree algorithm in Chapter 2, the Clustering Tree of the MST graph in Figure 4.2 is shown in Figure 4.3.

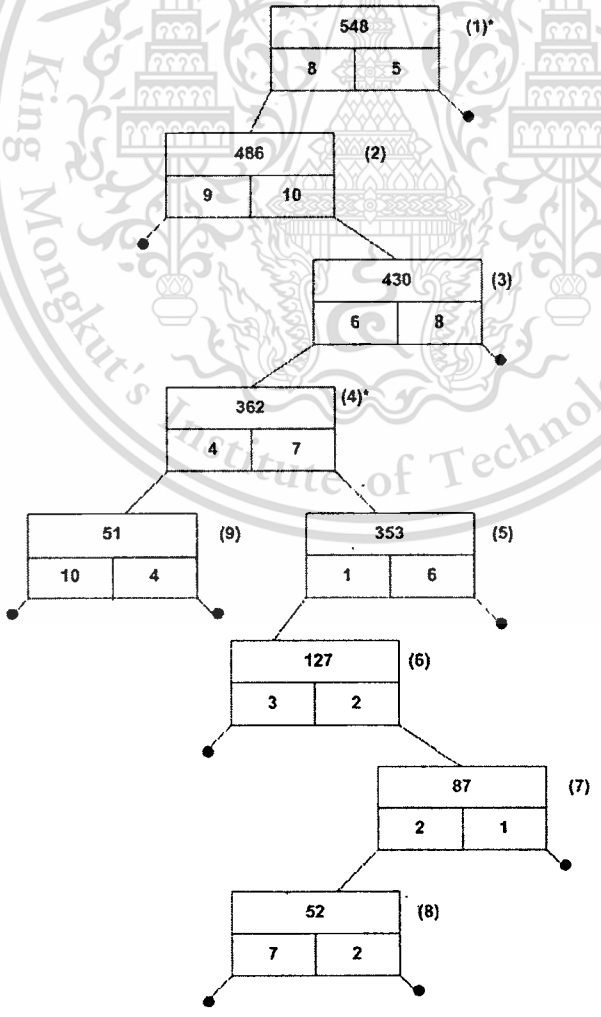

Figure 4.3: A Clustering Tree for the MST on Figure 4.2

The completed graph of 13 nodes is shown in Figure 4.4. The database tables of the Completed graph, as shown in Table 4.3, are the information of the connection lines and their weights. An ascending sort of weights from Table 4.3 is shown in Table 4.4. This information is useful for finding the MST of the completed graph.

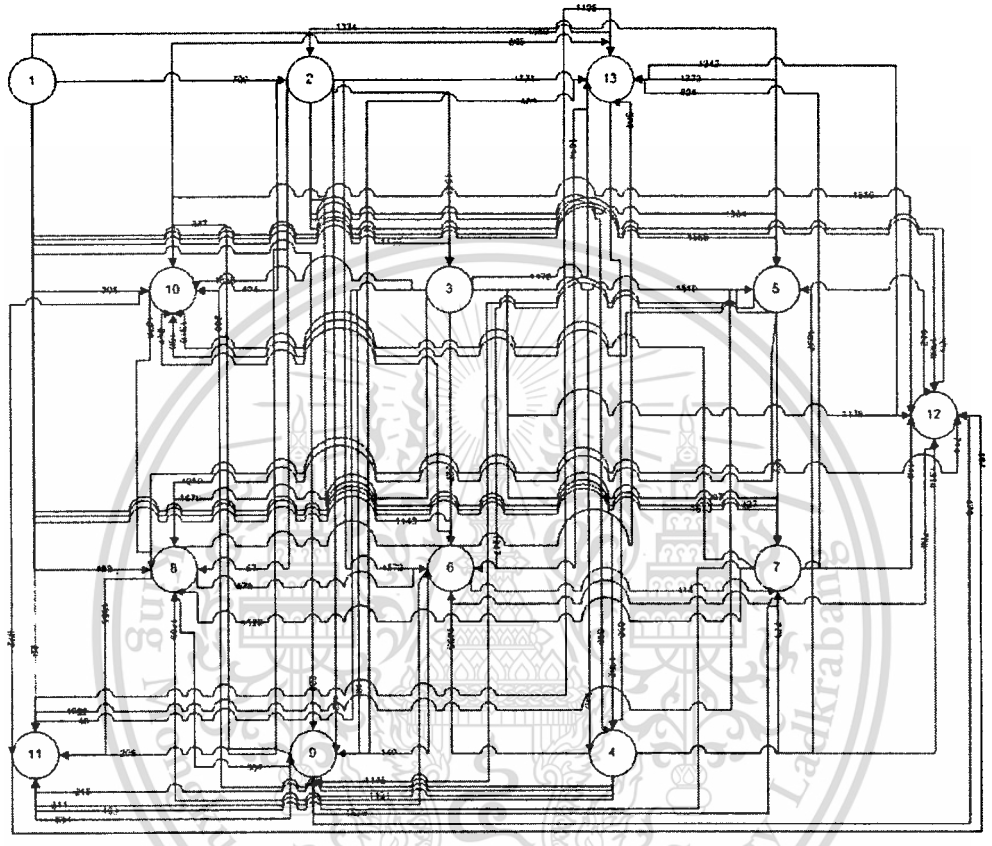

Figure 4.4: A Completed graph of 13 nodes

Table 4.3: The DB tables of a Completed Graph of 13 nodes

start point	end point	weight
1	2	700
1	3	1136
1	4	1382
1	5	1586
1	6	1143
1	7	1673
1	8	462
1	9	833
1	10	205
1	11	20
1	12	1866
1	13	1380

(1)

start point	end point	weight
2	1	700
2	3	1841
2	4	396
2	5	1334
2	6	1572
2	7	327
2	8	57
2	9	462
2	10	404
2	11	608
2	12	351
2	13	1721

(2)

start point	end point	weight
3	1	1136
3	2	1841
3	4	546
3	5	1510
3	6	527
3	7	1297
3	8	1670
3	9	1105
3	10	1866
3	11	40
3	12	1128
3	13	1176

(3)

start point	end point	weight
4	1	1382
4	2	396
4	3	546
4	5	1837
4	6	1495
4	7	779
4	8	1760
4	9	1131
4	10	203
4	11	215
4	12	1314
4	13	693

(4)

Table 4.3(cont.): The DB tables of a Completed Graph of 13 nodes

start point	end point	weight
5	1	1586
5	2	1334
5	3	1510
5	4	1837
5	6	1347
5	7	122
5	8	1212
5	9	1146
5	10	108
5	11	1092
5	12	982
5	13	1373

(5)

start point	end point	weight
6	1	1143
6	2	1572
6	3	527
6	4	1495
6	5	1347
6	7	114
6	8	878
6	9	149
6	10	438
6	11	811
6	12	733
6	13	1814

(6)

start point	end point	weight
7	1	1673
7	2	327
7	3	1297
7	4	779
7	5	122
7	6	114
7	8	1529
7	9	1678
7	10	1310
7	11	463
7	12	146
7	13	824

(7)

start point	end point	weight
8	1	462
8	2	57
8	3	1670
8	4	1760
8	5	1212
8	6	878
8	7	1529
8	9	354
8	10	1078
8	11	1355
8	12	714
8	13	383

(8)

Table 4.3(cont.): The DB tables of a Completed Graph of 13 nodes

start point	end point	weight
9	1	833
9	2	462
9	3	1105
9	4	1131
9	5	1146
9	6	149
9	7	1678
9	8	354
9	10	337
9	11	634
9	12	473
9	13	459

(9)

start point	end point	weight
10	1	205
10	2	404
10	3	1866
10	4	203
10	5	108
10	6	438
10	7	1310
10	8	1078
10	9	337
10	11	972
10	12	1539
10	13	605

(10)

start point	end point	weight
11	1	20
11	2	608
11	3	40
11	4	215
11	5	1092
11	6	811
11	7	463
11	8	1355
11	9	634
11	10	972
11	12	187
11	13	1195

(11)

start point	end point	weight
12	1	1866
12	2	351
12	3	1128
12	4	1314
12	5	982
12	6	733
12	7	146
12	8	714
12	9	473
12	10	1539
12	11	187
12	13	1247

(12)

Table 4.3(cont.): The DB tables of a Completed Graph of 13 nodes

start point	end point	weight
13	1	1380
13	2	1721
13	3	1176
13	4	693
13	5	1373
13	6	1814
13	7	824
13	8	383
13	9	459
13	10	605
13	11	1195
13	12	1247

(13)

Table 4.4: An ascending sort of weights from Table 4.3

start point	end point	weight
1	11	20
1	10	205
1	8	462
1	2	700
1	9	833
1	3	1136
1	6	1143
1	13	1380
1	4	1382
1	5	1586
1	7	1673
1	12	1866

(1)

start point	end point	weight
2	8	57
2	7	327
2	12	351
2	4	396
2	10	404
2	9	462
2	11	608
2	1	700
2	5	1334
2	6	1572
2	13	1721
2	3	1841

(2)

start point	end point	weight
3	11	40
3	6	527
3	4	546
3	9	1105
3	12	1128
3	1	1136
3	13	1176
3	7	1297
3	5	1510
3	8	1670
3	2	1841
3	10	1866

(3)

start point	end point	weight
4	10	203
4	11	215
4	2	396
4	3	546
4	13	693
4	7	779
4	9	1131
4	12	1314
4	1	1382
4	6	1495
4	8	1760
4	5	1837

(4)

Table 4.4(cont.): An ascending sort of weights from Table 4.3

start point	end point	weight
5	10	108
5	7	122
5	12	982
5	11	1092
5	9	1146
5	8	1212
5	13	1334
5	6	1347
5	13	1373
5	3	1510
5	1	1586
5	4	1837

(5)

start point	end point	weight
6	7	114
6	9	149
6	10	438
6	3	527
6	12	733
6	1	811
6	8	878
6	1	1143
6	5	1347
6	4	1495
6	2	1572
6	13	1814

(6)

start point	end point	weight
7	6	114
7	5	122
7	12	146
7	2	327
7	11	463
7	4	779
7	13	824
7	3	1297
7	10	1310
7	8	1529
7	1	1673
7	9	1678

(7)

start point	end point	weight
8	2	57
8	9	354
8	13	383
8	1	462
8	12	714
8	6	878
8	10	1078
8	5	1212
8	11	1355
8	7	1529
8	3	1670
8	4	1760

(8)

Table 4.4(cont.): An ascending sort of weights from Table 4.3

start point	end point	weight
9	6	149
9	10	337
9	8	354
9	13	459
9	2	462
9	12	473
9	11	634
9	1	833
9	3	1105
9	4	1131
9	5	1146
9	7	1678

(9)

start point	end point	weight
10	5	108
10	4	203
10	1	205
10	9	337
10	2	404
10	6	438
10	13	605
10	11	972
10	8	1078
10	7	1310
10	12	1539
10	3	1866

(10)

start point	end point	weight
11	1	20
11	3	40
11	12	187
11	4	215
11	7	463
11	2	608
11	9	634
11	6	811
11	10	972
11	5	1042
11	13	1195
11	8	1355

(11)

start point	end point	weight
12	7	146
12	11	187
12	2	351
12	9	473
12	8	714
12	6	733
12	5	982
12	3	1128
12	13	1247
12	4	1314
12	10	1539
12	1	1866

(12)

Table 4.4(cont.): An ascending sort of weights from Table 4.3

start point	end point	weight
13	8	383
13	9	459
13	10	605
13	4	693
13	7	824
13	3	1176
13	11	1195
13	12	1247
13	5	1373
13	1	1380
13	2	1721
13	6	1814

(13)

Using Prim's algorithm, the MST graph of the completed graph in Figure 4.4 is shown in Figure 4.5.

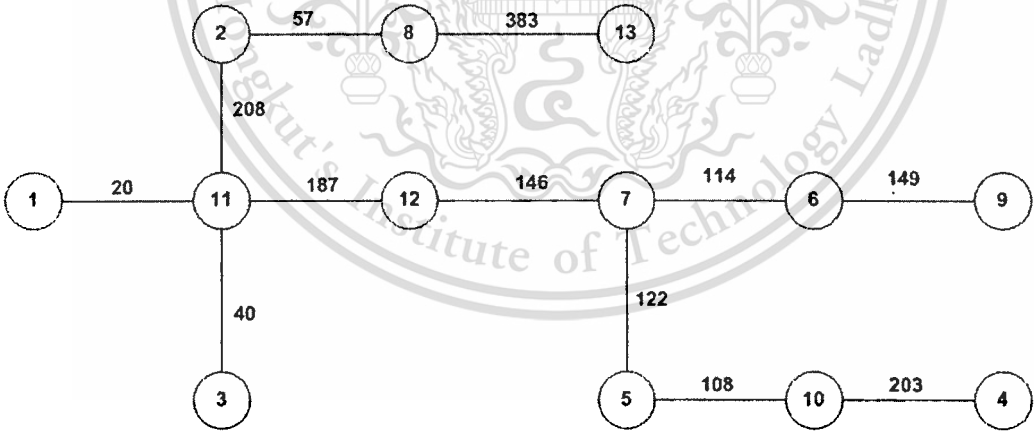

Figure 4.5: MST graph of Figure 4.4

Using Clustering Tree algorithm in chapter 2, the Clustering Tree of the MST graph in Figure 4.5 is shown in Figure 4.6.

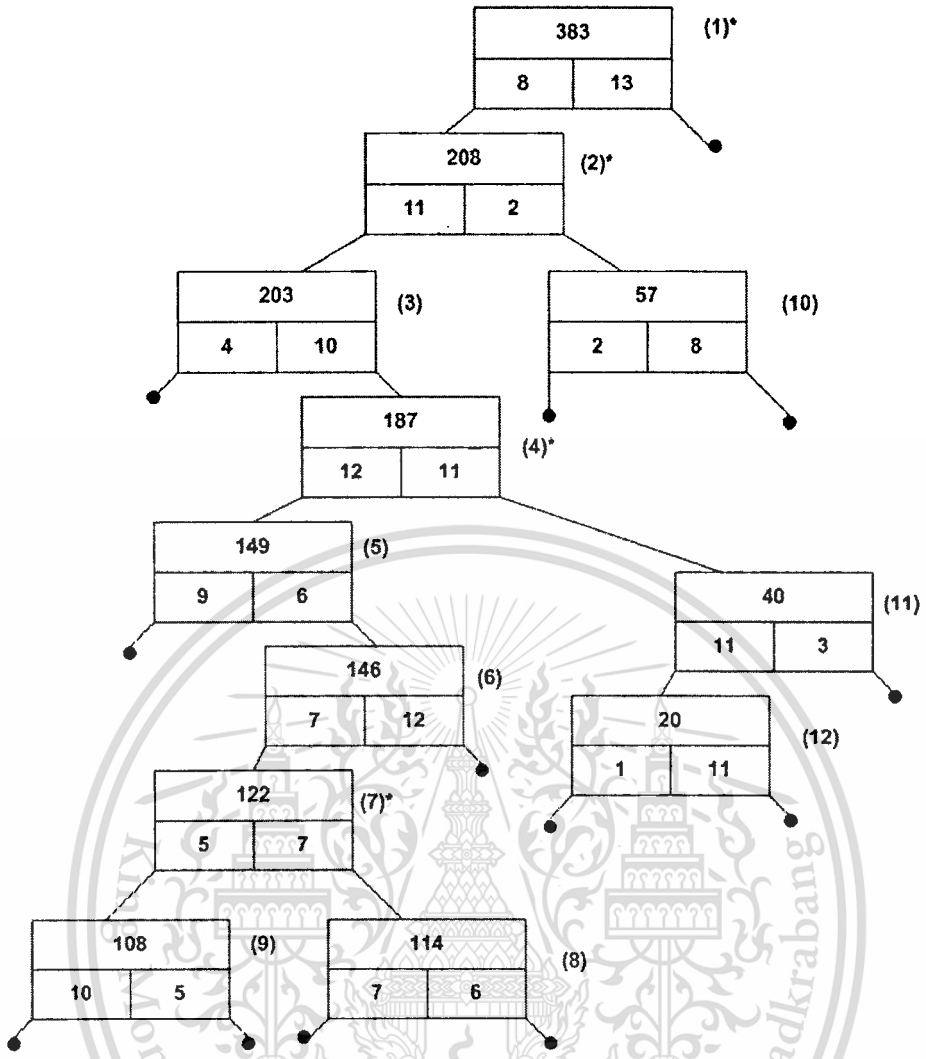

Figure 4.6: A Clustering Tree of Figure 4.5

Example 1: A connection line of MST is increased its weight between node 2 and node 3.

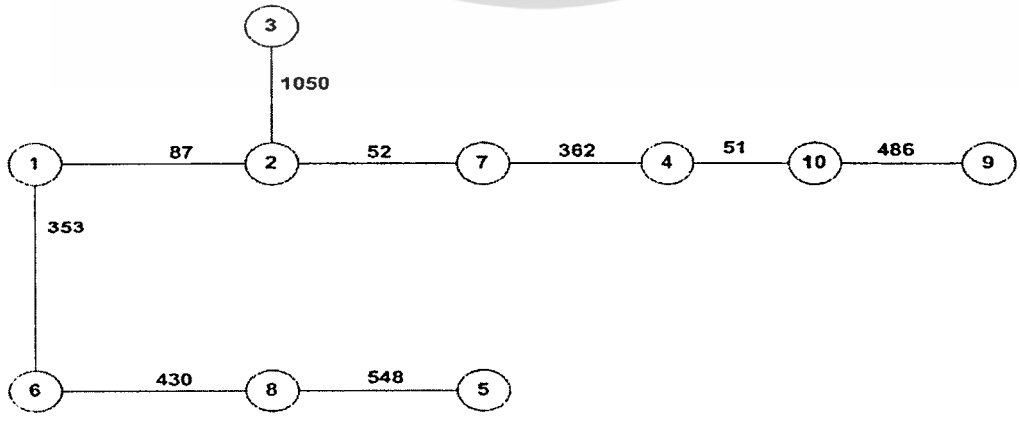

Figure 4.7: A connection line of MST is increased between node 2 and node 3

From Figure 4.7, the weight of a connection line in MST between node 2 and node 3 is increased from 127 to be 1050, the value of new weight. The increased weight of a connection line in the MST may affect other connection lines. The MST may not qualify its properties because it may not preserve the least weight connection. After remove the increased weight of a connection line, MST is divided into two graphs as shown in Figure 4.8. From Figure 4.8, each graph can be found its element set. There are two sets; {3}, and {1, 2, 4, 5, 6, 7, 8, 9, 10}. To find a new least weight of a connection line connecting these two graphs uses the DB tables of the set that has fewer members. Here {3} has fewer members. The new weight adjusts the DB table of node 3 from Table 4.2 to become a new DB table as shown in Table 4.5. The information in Table 4.5 will be used to find the new least weight of a connection line to connect those two graphs in Figure 4.9.

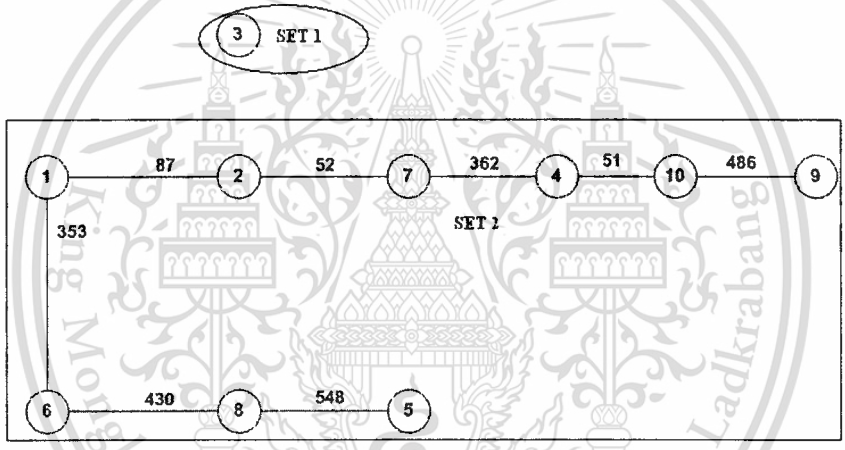

Figure 4.8: The MST from Figure 4.7 is divided

Table 4.5: The new DB table of node 3

3	1	512
3	7	945
3	10	959
3	9	1029
3	2	1050
3	4	1147
3	6	1659
3	5	1807
3	8	1932

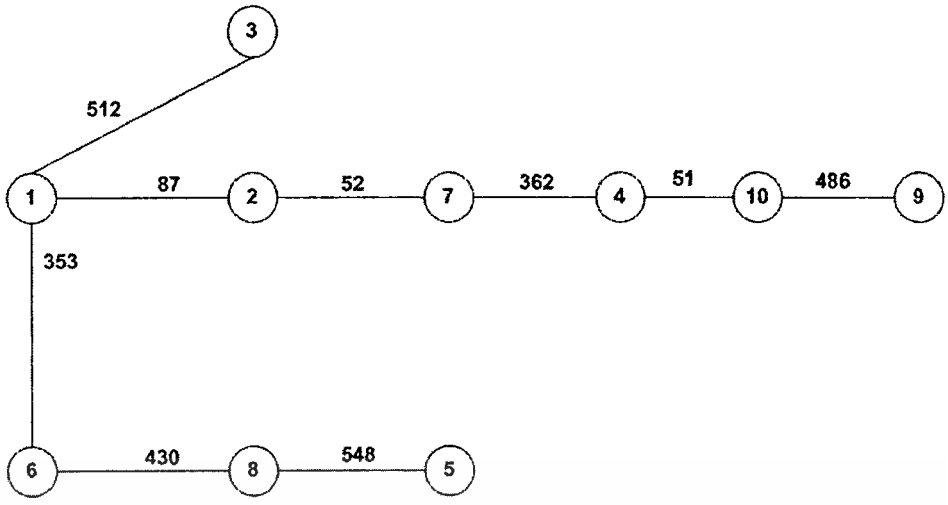

Figure 4.9: New MST graph

Using Clustering Tree algorithm, the Clustering Tree of the MST graph in Figure 4.9 is improved from the Clustering Tree in Figure 4.3 which has a changed some data on node (6) as shown in Figure 4.10(a). The Clustering Tree does not qualify its properties because the value 512 on node (6) is greater than the value 353 of its parent node, node (5). Then move node (6) to the right position as shown on Figure 4.10(b) and Figure 4.10(c).

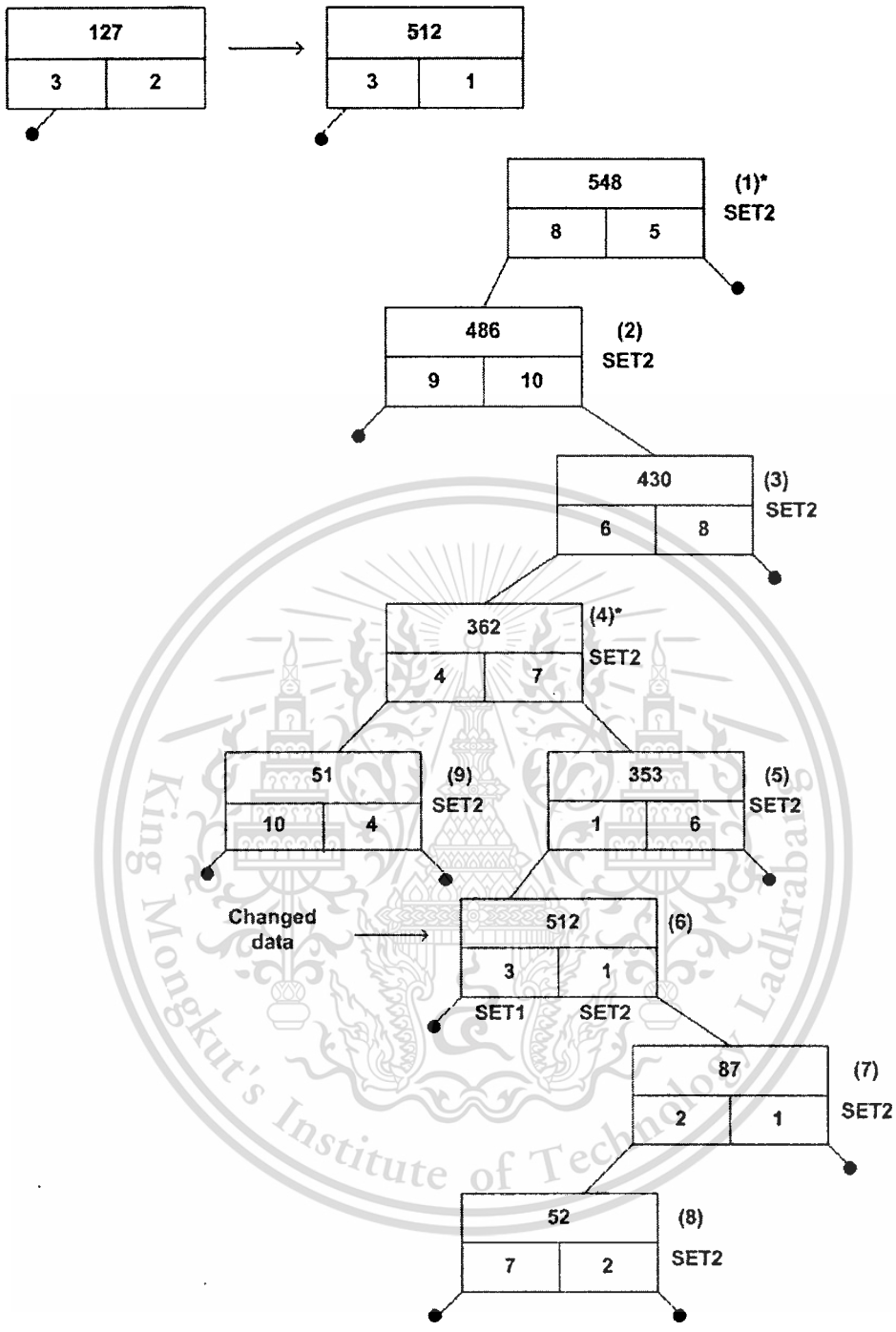

Figure 4.10(a): Clustering Tree from Figure 4.3 changed information on node (6)

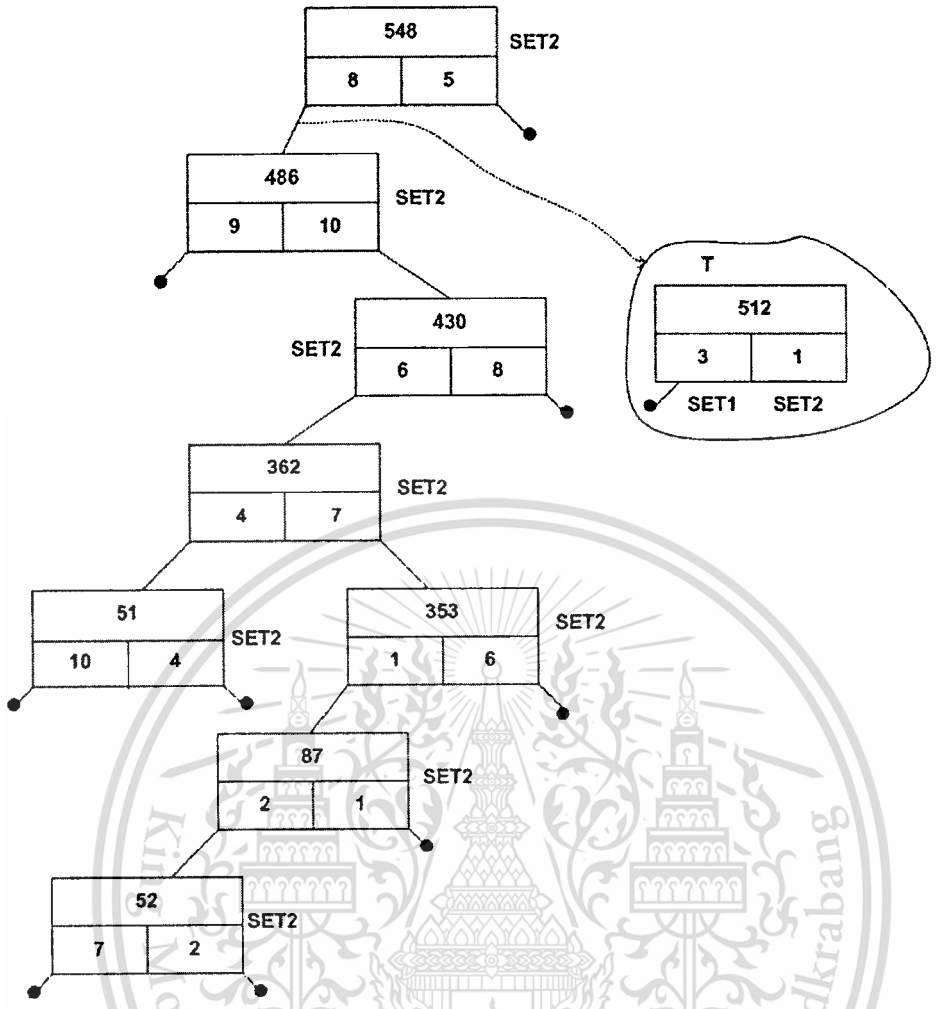

Figure 4.10(b): Clustering Tree from Figure 4.10(a) with moved node (6) to the right position

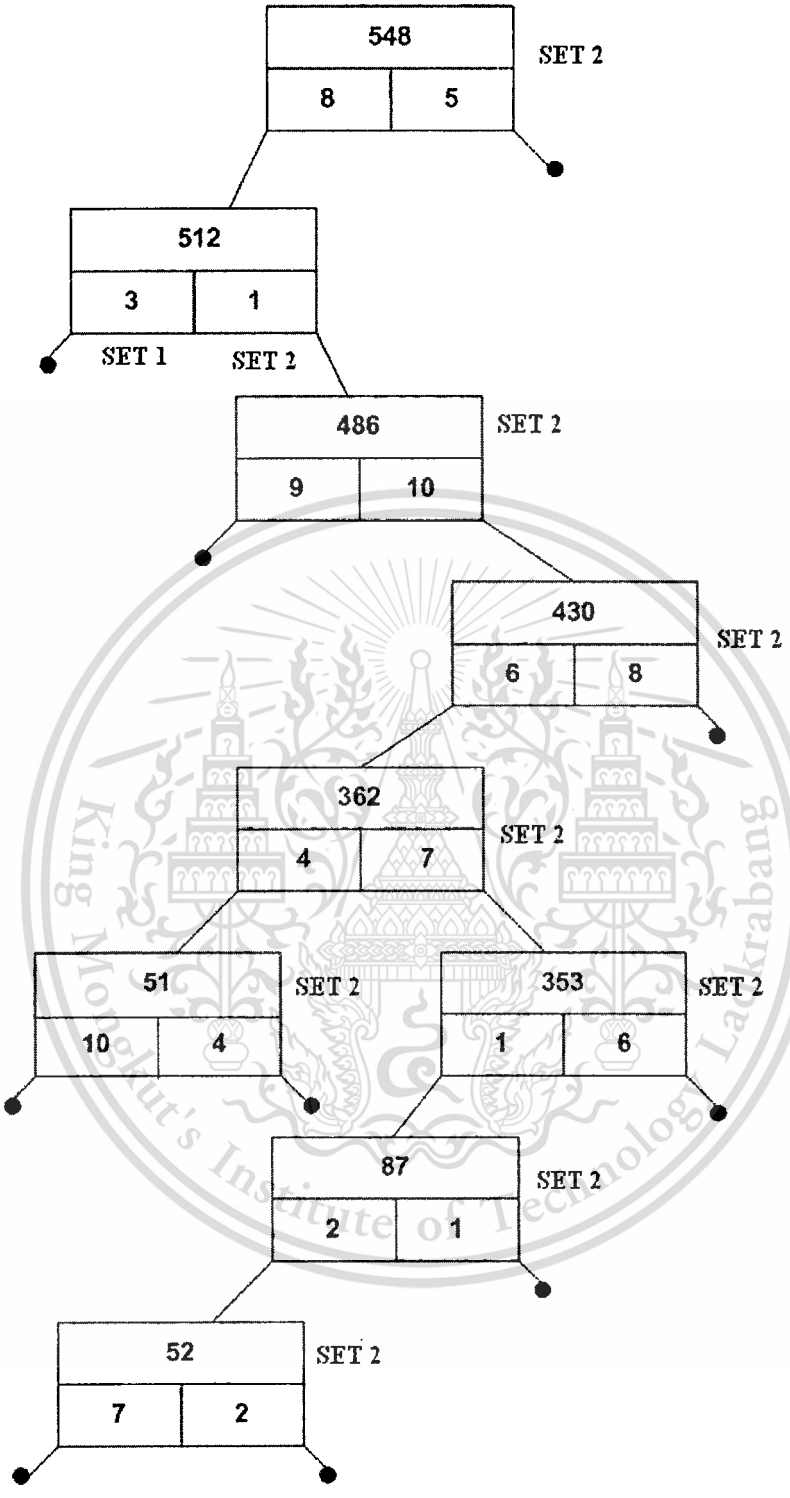

Figure 4.10(c): Clustering Tree for the MST on Figure 4.9

Example 2: A connection line of MST is increased its weight between node 5 and node 8.

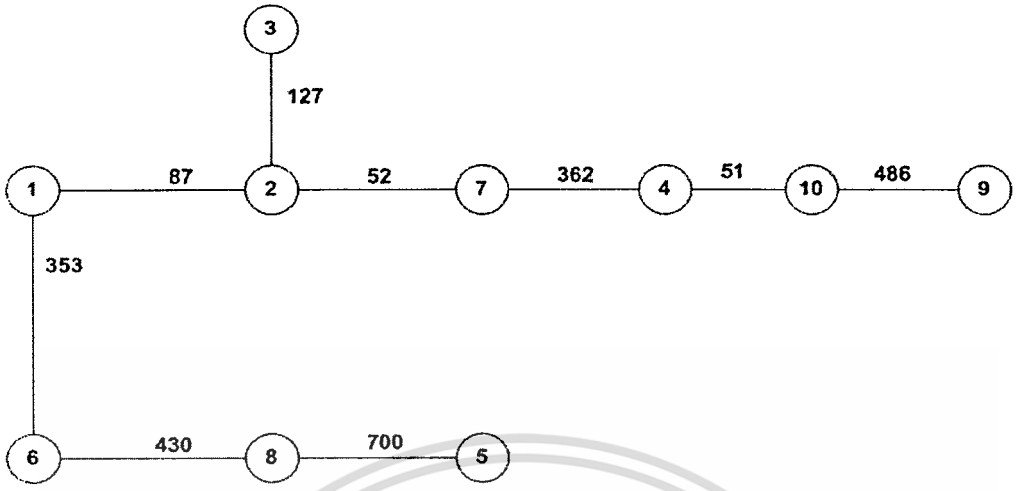

Figure 4.11: A connection line of MST is increased between node 5 and node 8

From Figure 4.11, the weight of a connection line in MST between node 5 and node 8 is increased from 548 to be 700, the value of new weight. The increased weight of a connection line in the MST may affect other connection lines. The MST may not qualify its properties because it may not preserve the least weight connection. After remove the increased weight of a connection line, MST is divided into two graphs as shown in Figure 4.12. From Figure 4.12, each graph can be found its element set. There are two sets; {5}, and {1, 2, 3, 4, 6, 7, 8, 9, 10}. To find a new least weight of a connection line connecting these two graphs uses the DB tables of the set that has fewer members. Here {5} has fewer members. The new weight adjusts the DB table of node 5 from Table 4.2 to become a new DB table as shown in Table 4.6. The information in Table 4.6 will be used to find the new least weight of a connection line to connect those two graphs in Figure 4.13.

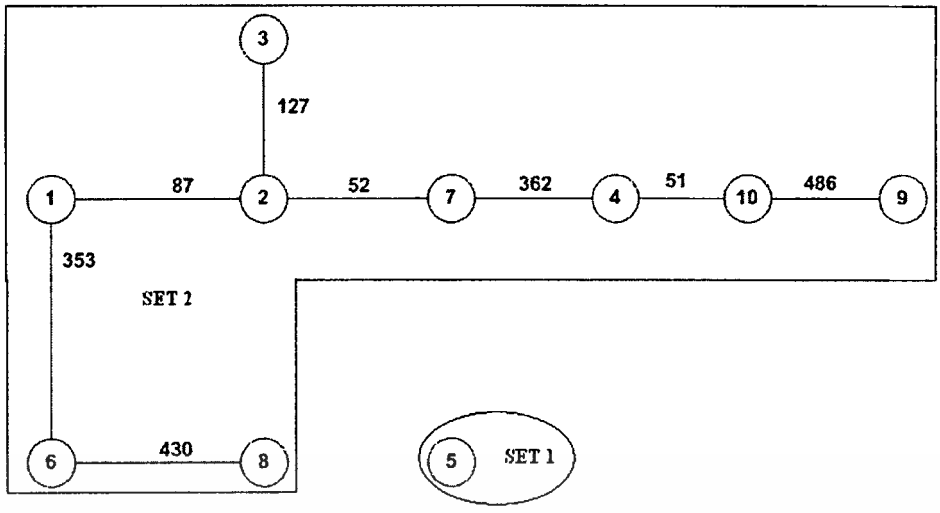

Figure 4.12: The MST from Figure 4.11 is divided

Table 4.6: The new DB table of node 5

5	9	644
5	8	700
5	2	739
5	6	748
5	10	1500
5	7	1538
5	1	1655
5	3	1807
5	4	1893

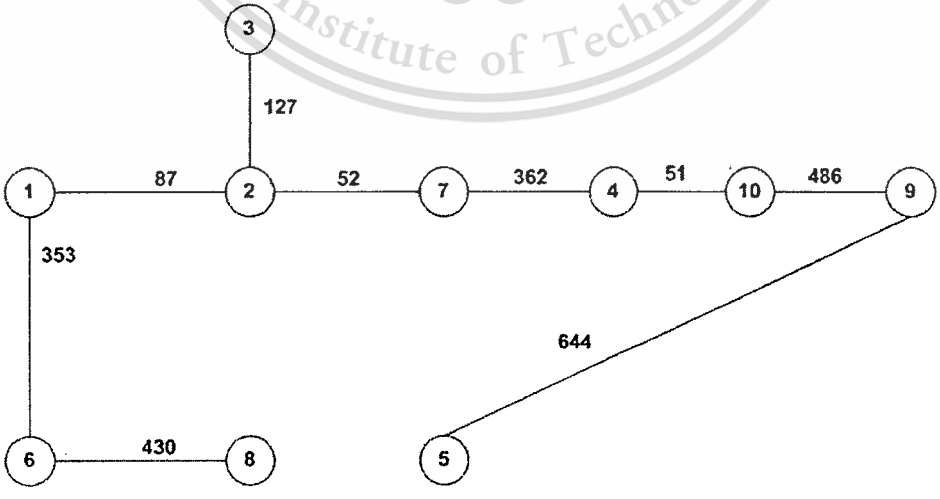

Figure 4.13: New MST graph

Using Clustering Tree algorithm, the Clustering Tree of the MST graph in Figure 4.13 is improved from the Clustering Tree in Figure 4.3 which has a changing some data on node (1) as shown in Figure 4.14. The Clustering Tree qualifies its properties on the right position. Therefore the Clustering Tree does not have to adjust any more.

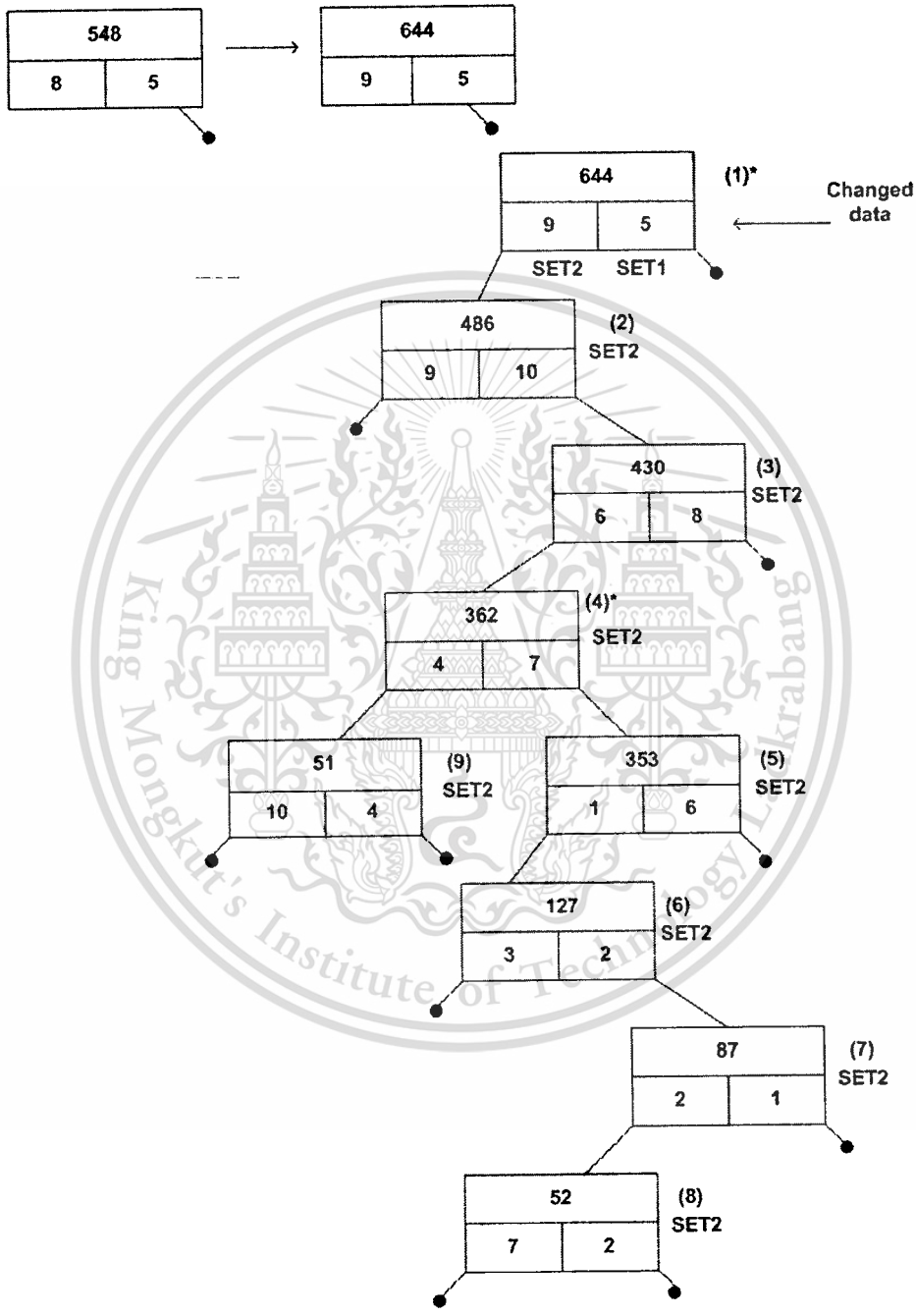

Figure 4.14: Clustering Tree for the MST on Figure 4.13

From Figure 4.14, after change data on node (1), this node is on the right position. Therefore, this Clustering Tree is not adjusted. This Clustering Tree is final Clustering Tree for MST from Figure 4.13.

Example 3: A connection line of MST is increased its weight between node 9 and node 10.

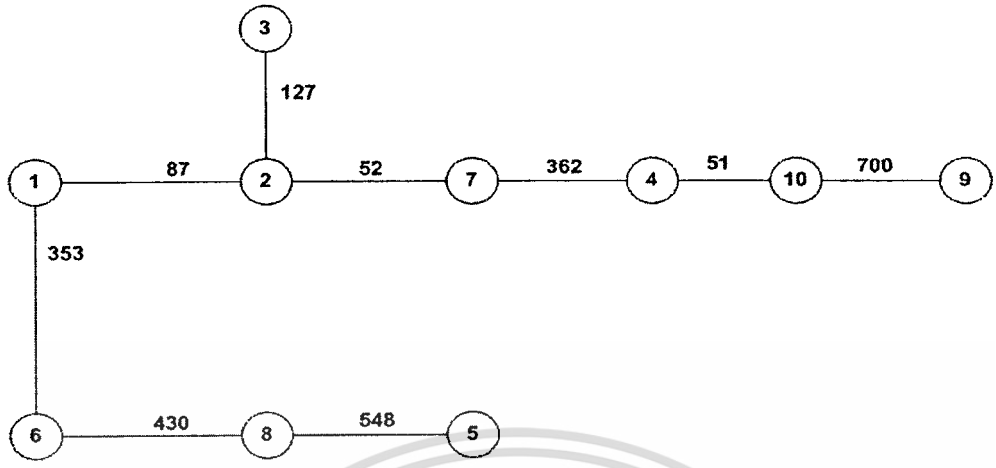

Figure 4.15: A connection line of MST is increased between node 9 and node 10

From Figure 4.15, the weight of a connection line in MST between node 9 and node 10 is increased from 486 to be 700, the value of new weight. The increased weight of a connection linc in the MST may affect other connection lines. The MST may not qualify its properties because it may not preserve the least weight connection. After remove the increased weight of a connection line, MST is divided into two graphs as shown in Figure 4.16. From Figure 4.16, each graph can be found its element set. There are two sets; {9}, and {1, 2, 3, 4, 5, 6, 7, 8, 10}. To find a new least weight of a connection line connecting these two graphs uses the DB tables of the set that has fewer members. Here {9} has fewer members. The new weight adjusts the DB table of node 9 from Table 4.2 to become a new DB table as shown in Table 4.7. The information in Table 4.7 will be used to find the new least weight of a connection line to connect those two graphs in Figure 4.17.

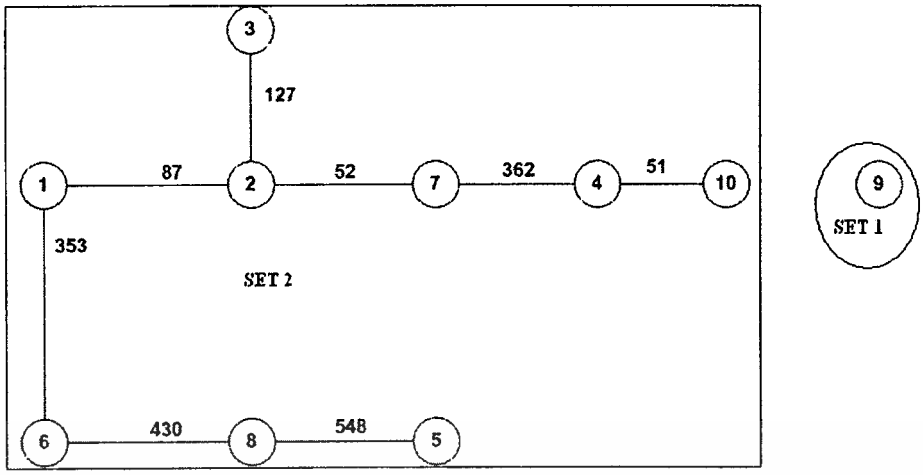

Figure 4.16: The MST from Figure 4.15 is divided

Table 4.7: The new DB table of node 9

9	5	644
9	6	650
9	10	700
9	2	722
9	8	879
9	3	1029
9	1	1149
9	7	1461
9	4	1798

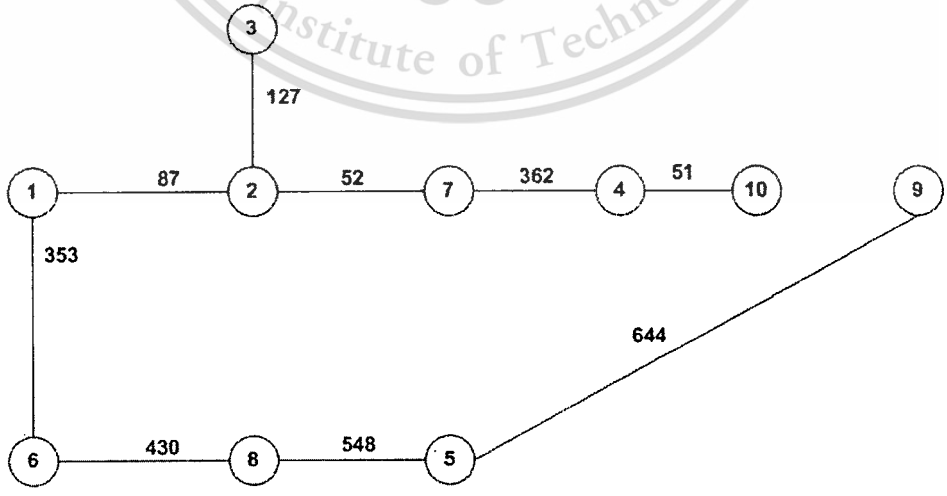

Figure 4.17: New MST graph

Using Clustering Tree algorithm, the Clustering Tree of the MST graph in Figure 4.17 is improved from the Clustering Tree in Figure 4.3 which has a changed some data on node (2) as shown in Figure 4.18(a). The Clustering Tree does not qualify its properties because the value 644 on node (2) is greater than the value 548 of its parent node, node (1). Then move node (2) to the right position as shown on Figure 4.18(b) and Figure 4.18(c).

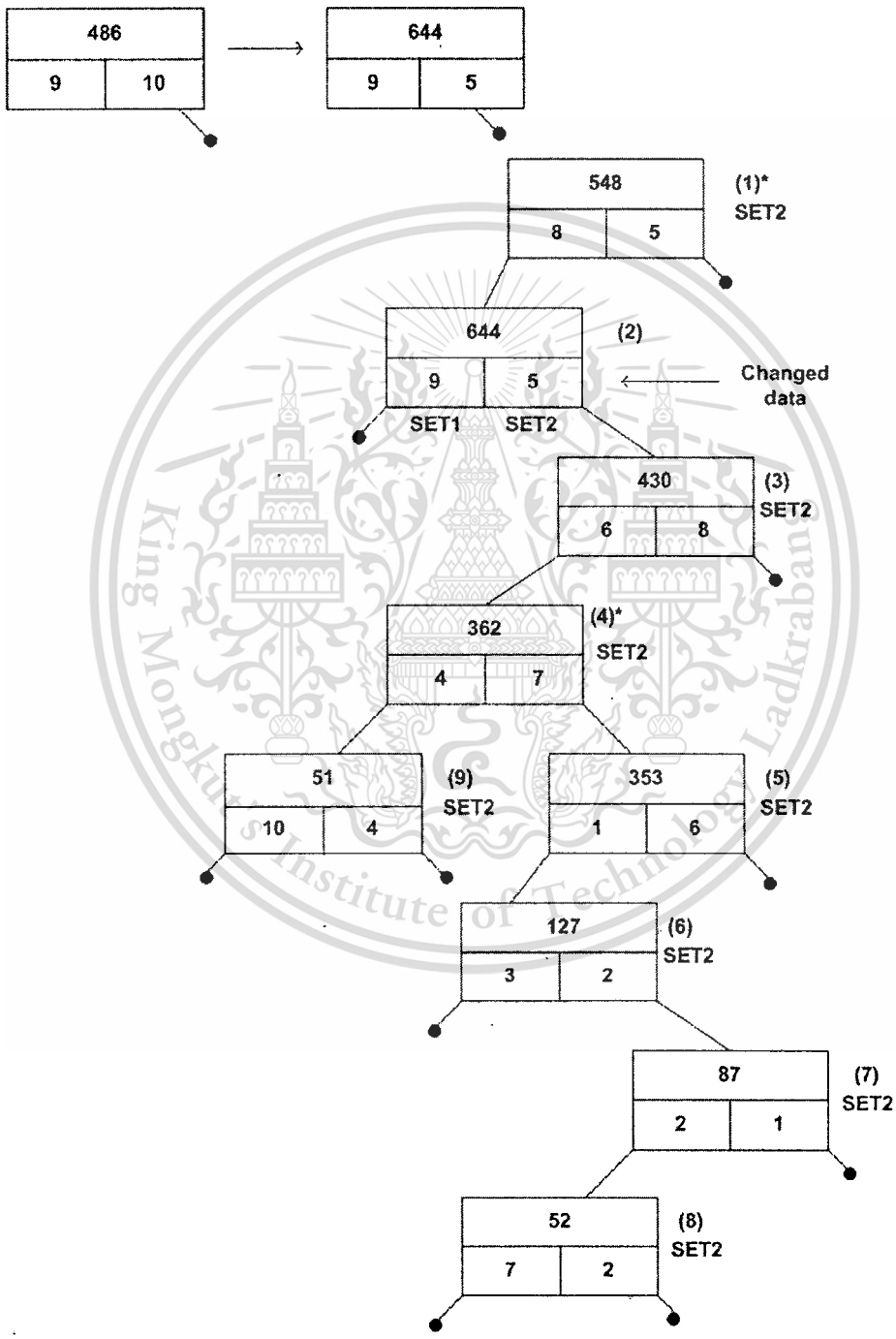

Figure 4.18(a): Clustering Tree from Figure 4.3 with changed data on node (2)

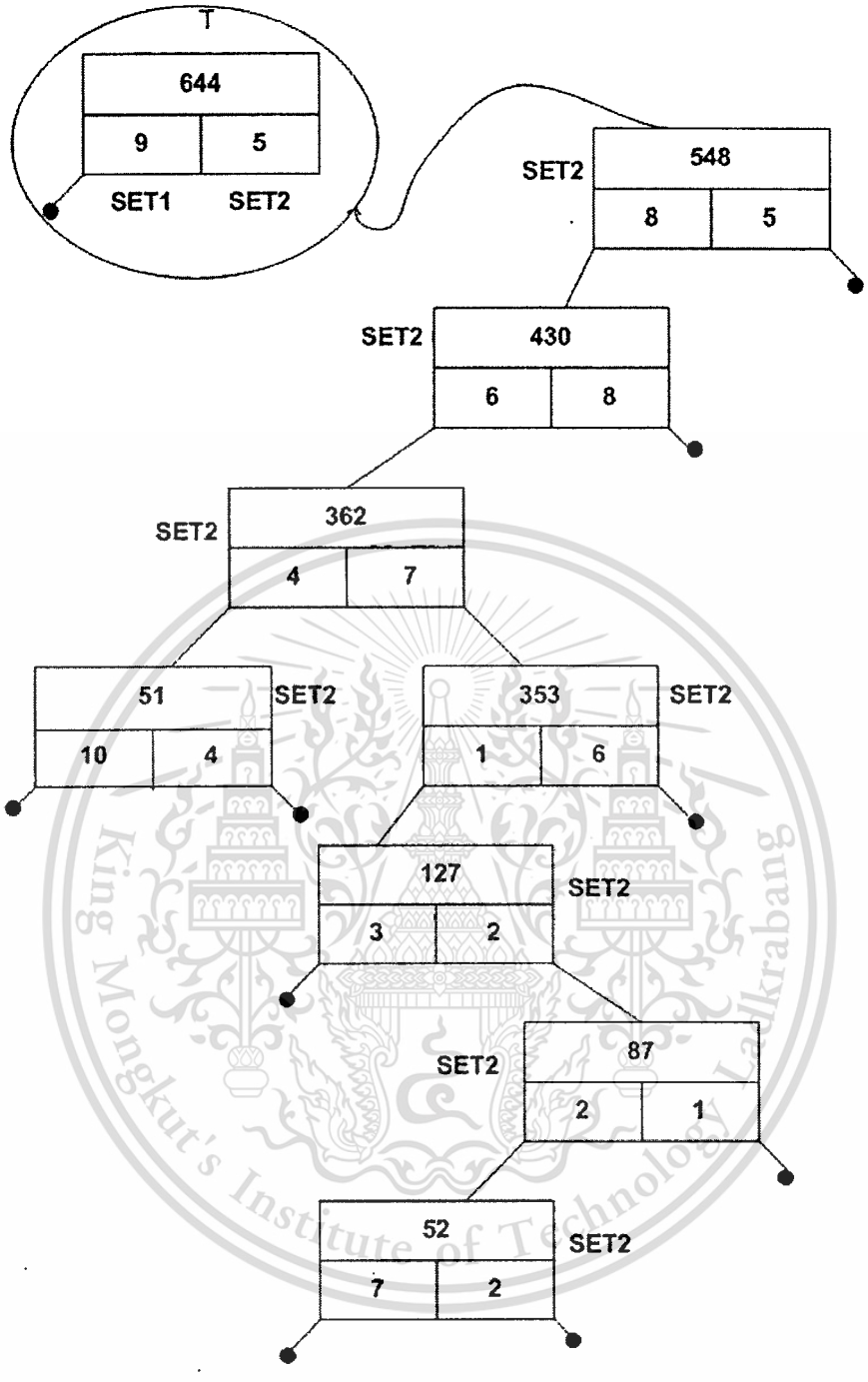

Figure 4.18(b): Clustering Tree from Figure 4.18(a) with moved node (2) to the right position

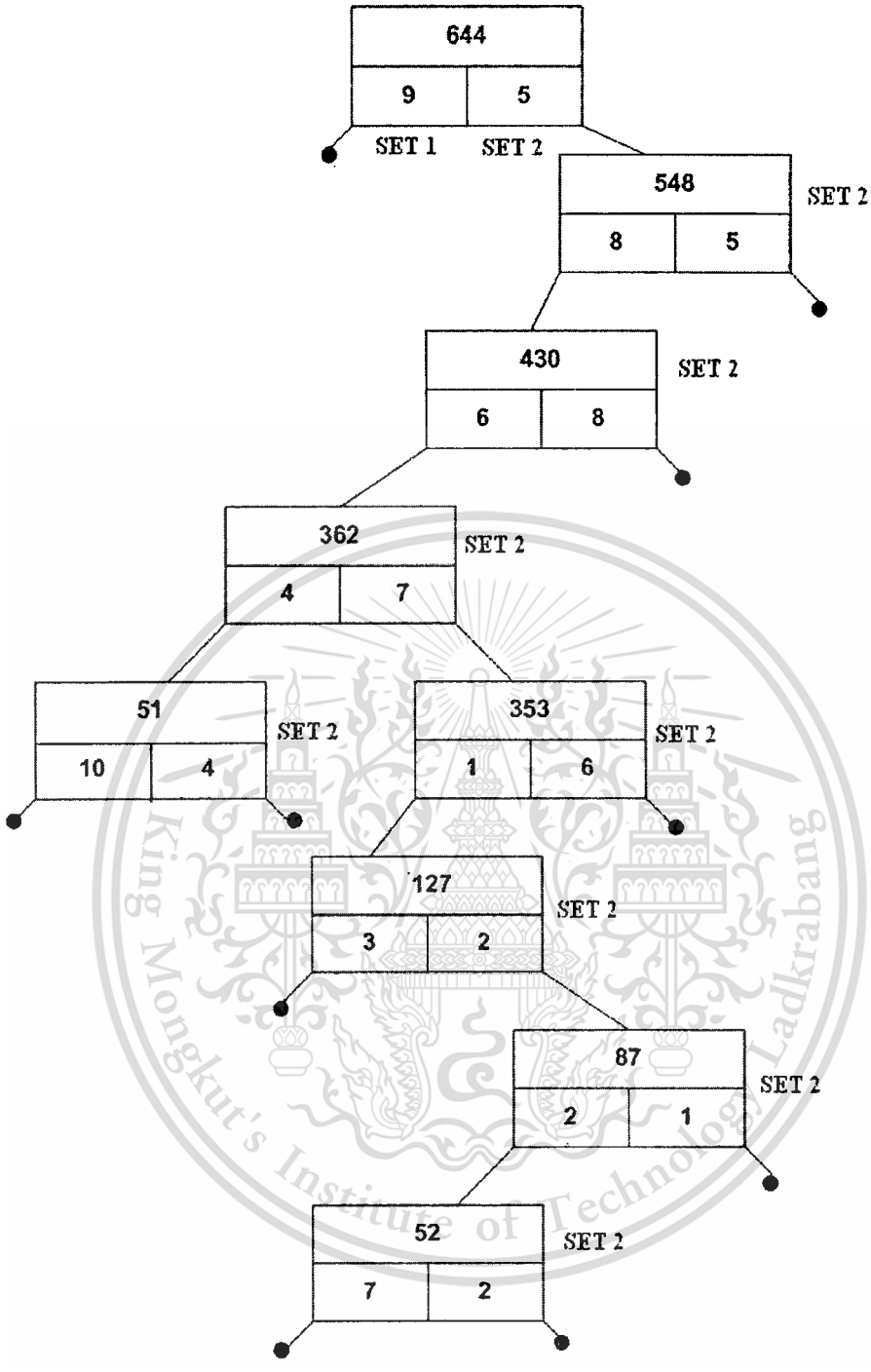

Figure 4.18(c): Clustering Tree for the MST on Figure 4.17

Example 4: A connection line of MST is increased its weight between node 5 and node 10.

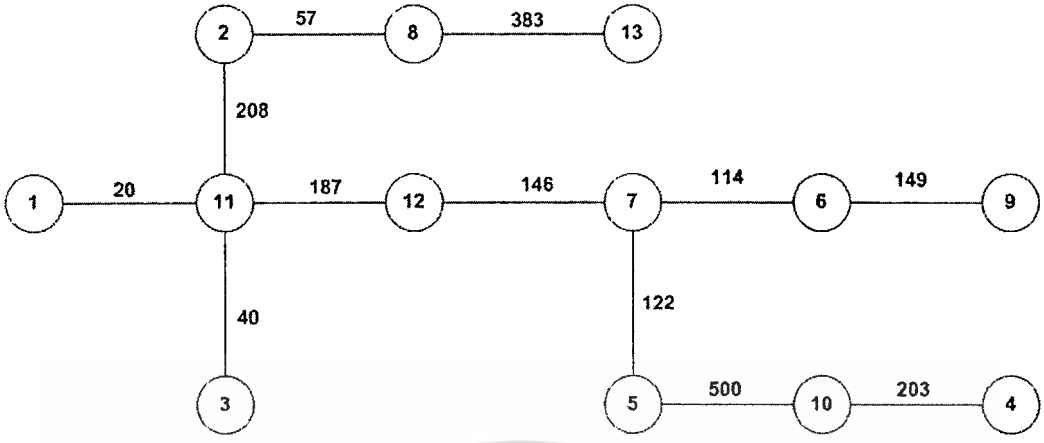

Figure 4.19: A connection line of MST is increased between nodes 5 and 10

From Figure 4.19, the weight of a connection line in MST between node 5 and node 10 is increased from 108 to be 500, the value of new weight. The increased weight of a connection line in the MST may affect other connection lines. The MST may not qualify its properties because it may not preserve the least weight connection. After remove the increased weight of a connection line, MST is divided into two graphs as shown in Figure 4.20. From Figure 4.20, each graph can be found its element set. There are two sets; {10, 4}, and {1, 2, 3, 5, 6, 7, 8, 9, 11, 12, 13}. To find a new least weight of a connection line connecting these two graphs uses the DB tables of the set that has fewer members. Here {10, 4} has fewer members. The new weight adjusts the DB table of node 10 and DB tables of node 4 from Table 4.4 to become a new DB table as shown in Table 4.8. The information in Table 4.8 will be used to find the new least weight of a connection line to connect those two graphs in Figure 4.21. From the DB tables, select the first line, in each table, that its start point and end point are from the different sets. Then select the minimum weight value from those selected lines.

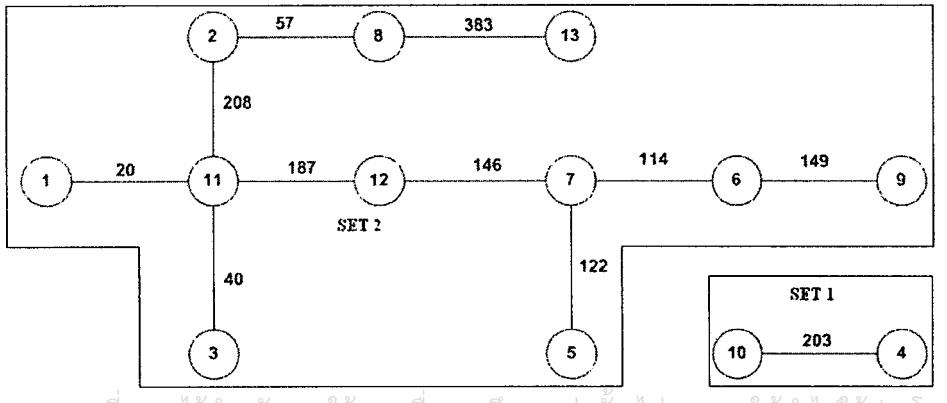

Figure 4.20: The MST graph from Figure 4.19 is divided

Table 4.8: The new DB tables of nodes 4 and 10

4	10	203
4	11	215
4	2	396
4	3	546
4	13	693
4	7	779
4	9	1131
4	1	1314
4	6	1382
4	8	1495
4	8	1760
4	5	1837

10	4	203
10	1	205
10	9	337
10	2	404
10	6	438
10	5	500
10	13	605
10	11	972
10	8	1078
10	7	1310
10	12	1539
10	3	1866

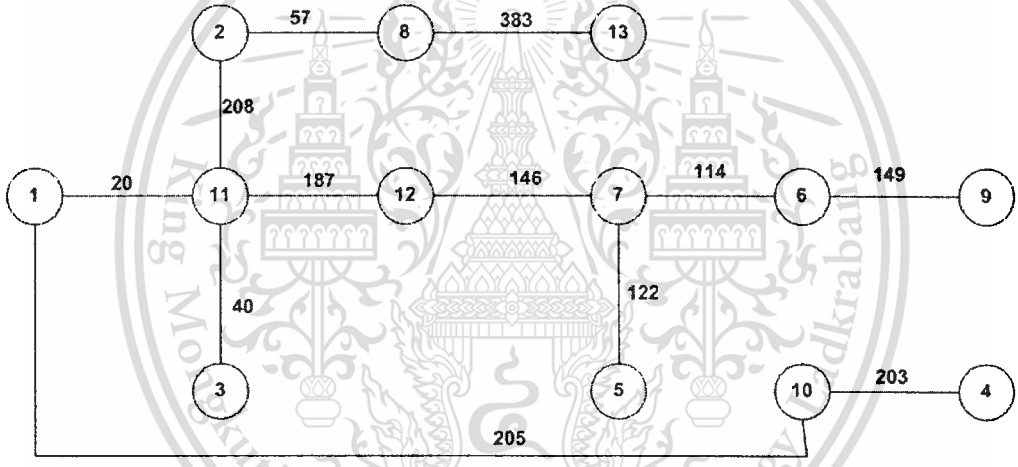

Figure 4.21: New MST graph

Using Clustering Tree algorithm, the Clustering Tree of the MST graph in Figure 4.21 is improved from the Clustering Tree in Figure 4.6 which has a changed some data on node (9) as shown in Figure 4.22(a). Node (9) is the leaf node because this node has no child.

The data have been changed as follows: The node 5 changes to node 1, because {5, 1} is in the same set of one graph.

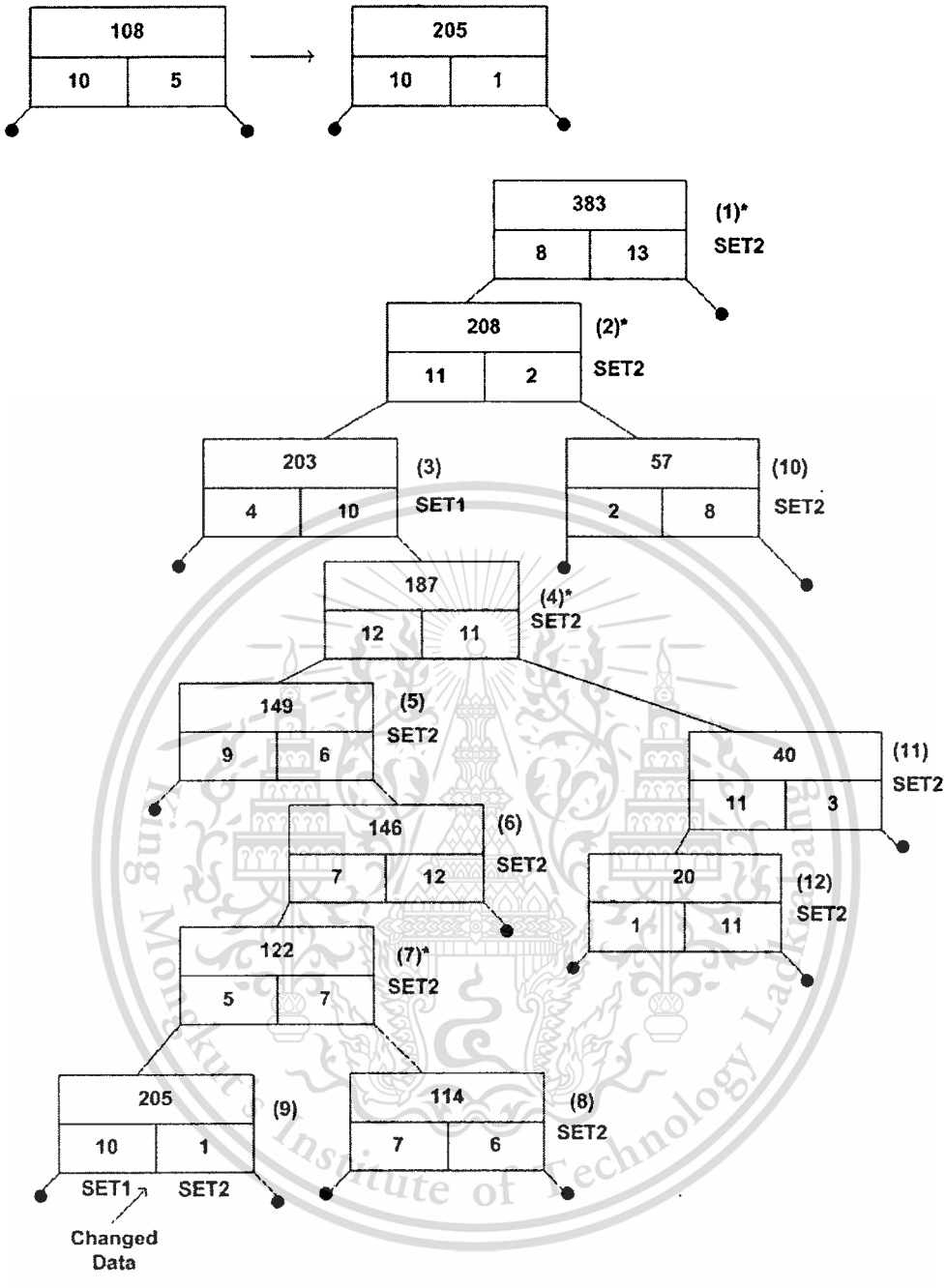

Figure 4.22(a): Clustering Tree from Figure 4.6 with changed data on node (9)

From Figure 4.22(a), after change data on node (9), this node is not on the right position because it does not preserve its properties. The Clustering Tree does not qualify its properties because the value 205 on node (9) is greater than the value 122 of its parent node, node (7). Then move node (9) to the right position as shown on Figure 4.22(b) and Figure 4.22(c). Therefore, this Clustering Tree is adjusted. The children nodes of the left/right of the previous node (9) are from the same set and preserve properties. This Clustering Tree is final Clustering Tree for MST from Figure 4.21.

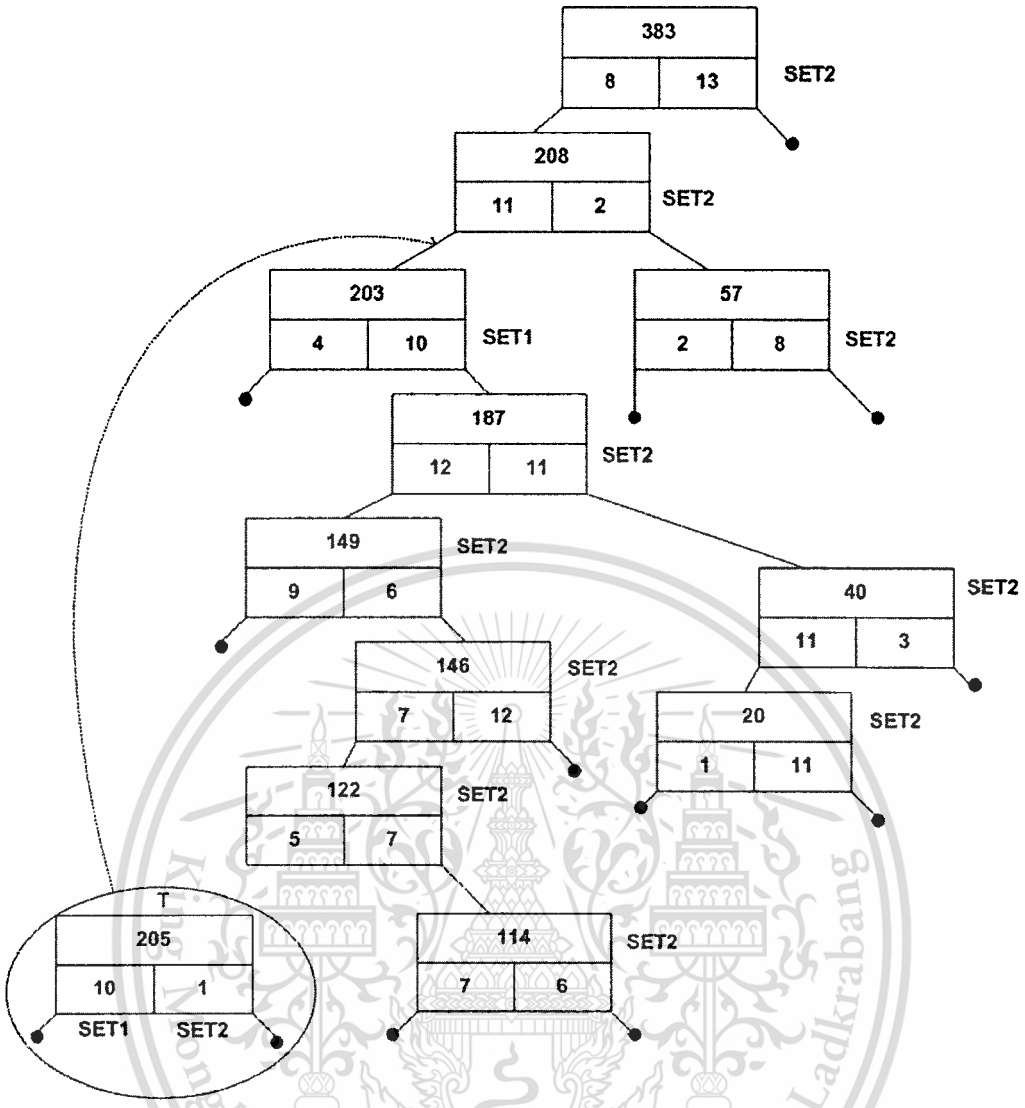

Figure 4.22(b): Clustering Tree from Figure 4.22(a) with moved node (9) to the right position

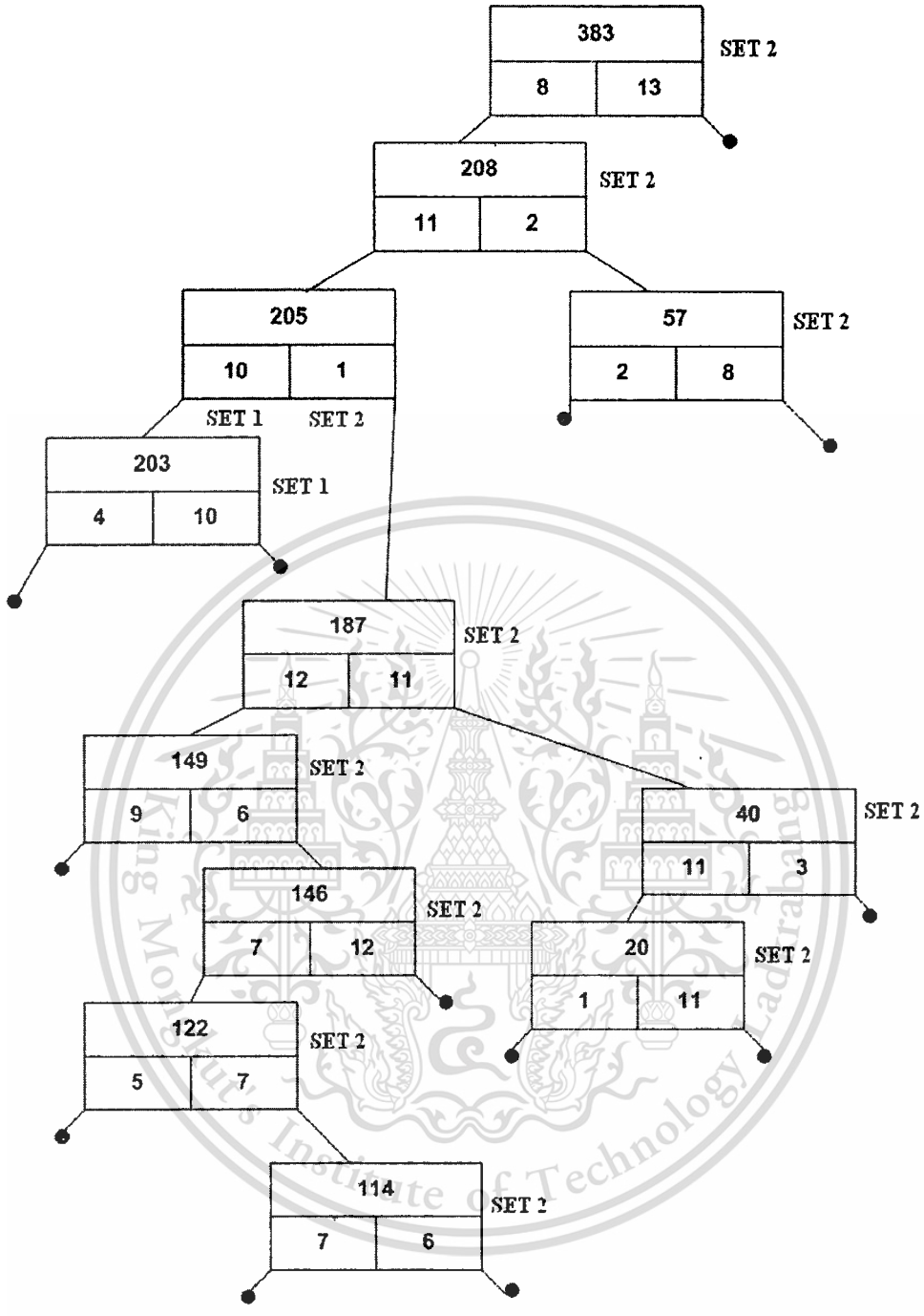

Figure 4.22(c): A Clustering Tree for the MST on Figure 4.21

Example 5: A connection line of MST is increased its weight between node 11 and node 12.

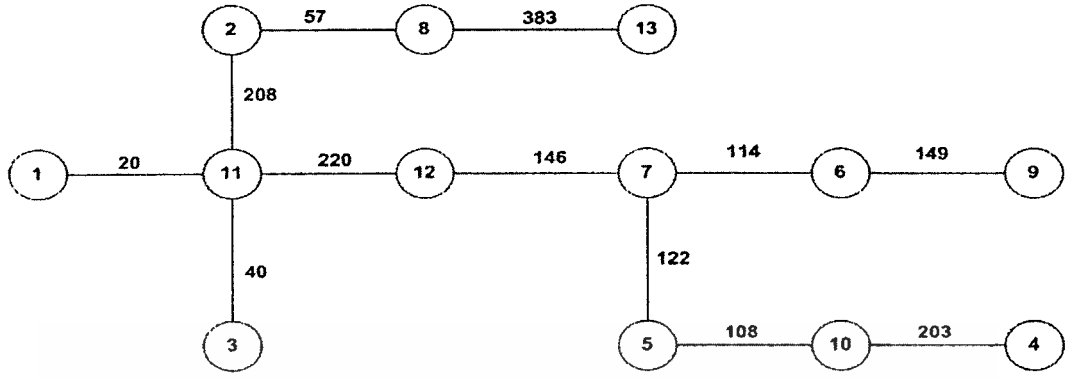

Figure 4.23: A connection line of MST is increased between nodes 11 and 12

From Figure 4.23, the weight of a connection line in MST between node 11 and node 12 is increased from 187 to be 220, the value of new weight. The increased weight of a connection line in the MST may affect other connection lines. The MST may not qualify its properties because it may not preserve the least weight connection. After remove the increased weight of a connection line, MST is divided into two graphs as shown in Figure 4.24. From Figure 4.24, each graph can be found its element set. There are two sets; {1, 11, 3, 2, 8, 13}, and {4, 5, 6, 7, 9, 10, 12}. To find a new least weight of a connection line connecting these two graphs uses the DB tables of the set that has fewer members. Here {1, 11, 3, 2, 8, 13} has fewer members. The new weight adjusts the DB table of node 11 and DB table of nodes 1, 3, 2, 8, and 13 from Table 4.4 to become a new DB table as shown in Table 4.9. The information in Table 4.9 will be used to find the new least weight of a connection line to connect those two graphs in Figure 4.25. From the DB tables, select the first line, in each table, that its start point and end point are from the different sets. Then select the minimum weight value from those selected lines.

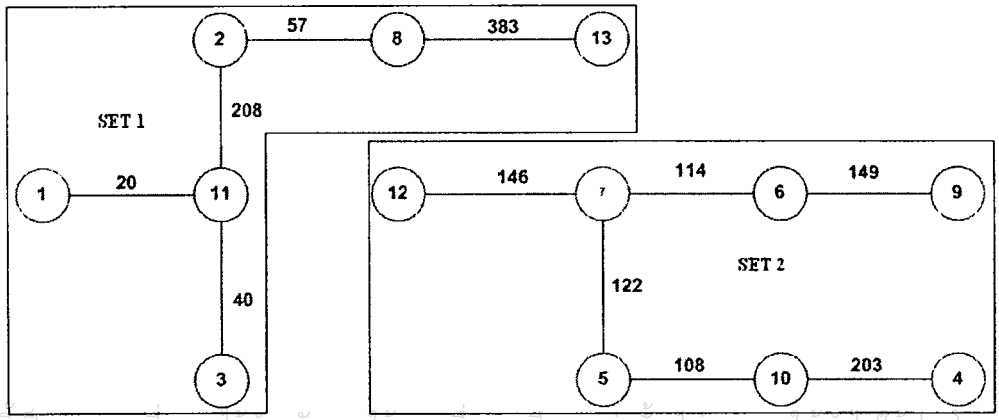

Figure 4.24: A MST graph from Figure 4.55 is divided

Table 4.9: The new DB tables of nodes 8, 11, and 13

1	11	20
1	10	205
1	8	462
1	9	833
1	2	700
1	3	1136
1	6	1143
1	13	1380
1	4	1382
1	5	1586
1	7	1673
1	12	1866

2	8	57
2	11	208
2	7	327
2	12	351
2	4	396
2	10	404
2	9	462
2	1	700
2	5	1334
2	6	1572
2	13	1721
2	3	1841

3	11	40
3	6	527
3	4	546
3	9	1105
3	12	1128
3	1	1136
3	13	1176
3	7	1297
3	5	1510
3	8	1670
3	2	1841
3	10	1866

8	2	57
8	9	354
8	13	383
8	1	462
8	12	714
8	6	878
8	10	1078
8	5	1212
8	11	1355
8	7	1529
8	3	1670
8	4	1760

11	1	20
11	3	40
11	2	208
11	4	215
11	12	220
11	7	463
11	9	634
11	6	811
11	10	972
11	5	1092
11	13	1195
11	8	1355

13	8	383
13	9	459
13	10	605
13	4	693
13	7	824
13	3	1176
13	11	1195
13	12	1247
13	5	1373
13	1	1380
13	2	1721
13	6	1814

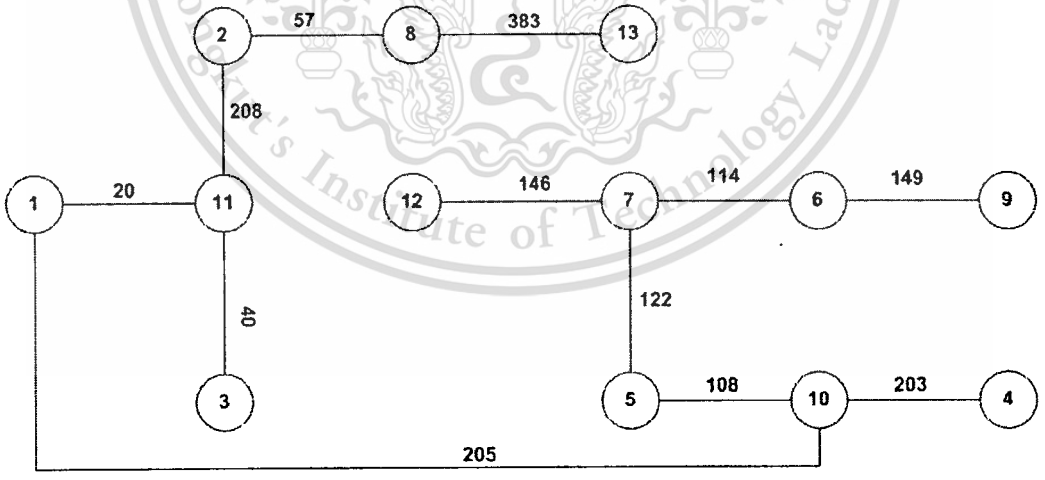

Figure 4.25: New MST graph

Using Clustering Tree algorithm, the Clustering Tree of the MST graph in Figure 4.25 is improved from the Clustering Tree in Figure 4.6 which is changed some data on node (4) as shown in Figure 4.26(a). Node (4) is the branch node that has two children.

The data have been changed as follows: The node 12 changes to node 10 and node 11 changes to node 1, because {12, 10} is in the same set of one graph and {11, 1} is in the same set of the other graph.

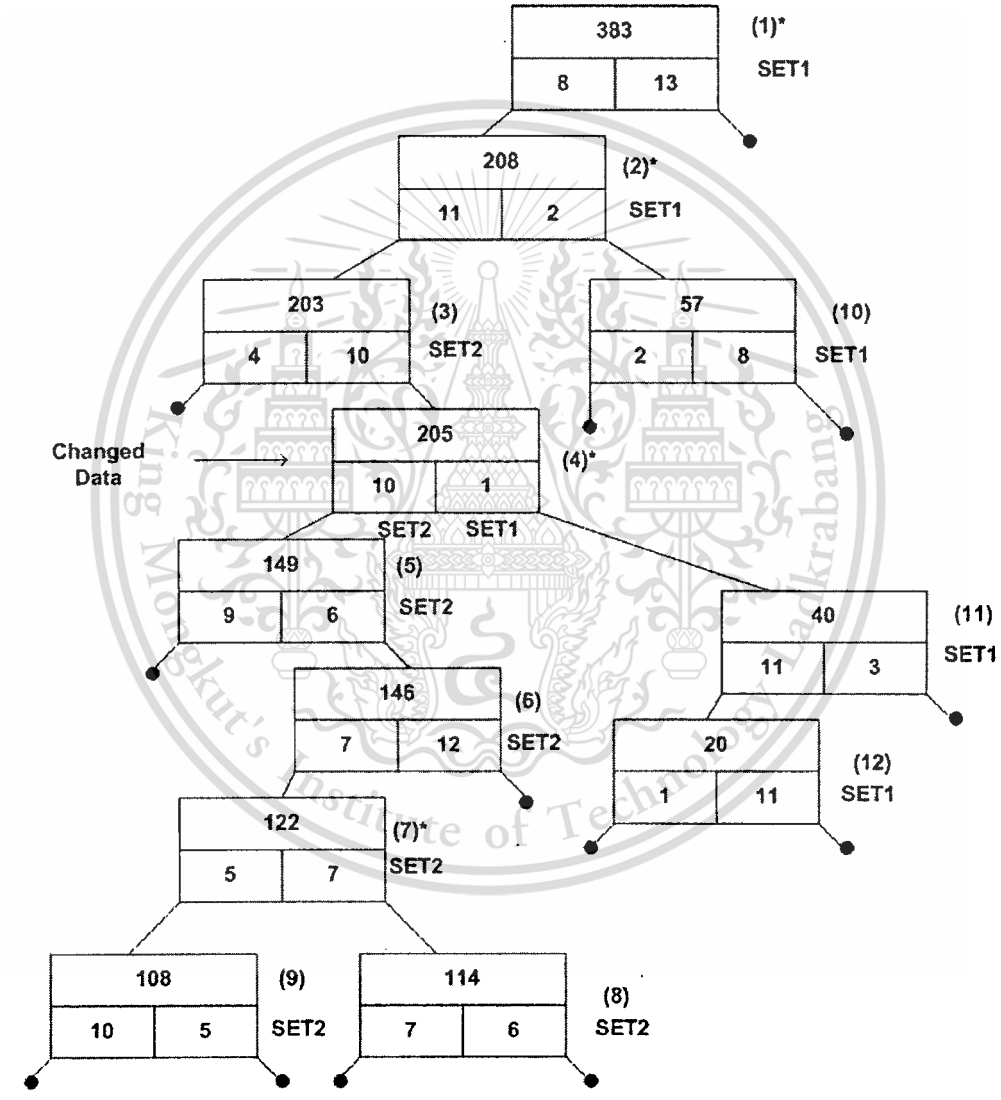

Figure 4.26(a): Clustering Tree from Figure 4.6 with changed data on node (4)

From Figure 4.26(a), after change data on node (4), this node is not on the right position because it does not preserve its properties. The Clustering Tree does not qualify its properties because the value 205 on node (4) is greater than the value 203 of its parent node, node (3). Then move node (4) to the right position as shown on Figure 4.26(b) and Figure 4.26(c).

Therefore, this Clustering Tree is adjusted. The children nodes of the left/right of the previous node (4) are from the same set and preserve properties. This Clustering Tree is final Clustering Tree for MST from Figure 4.25.

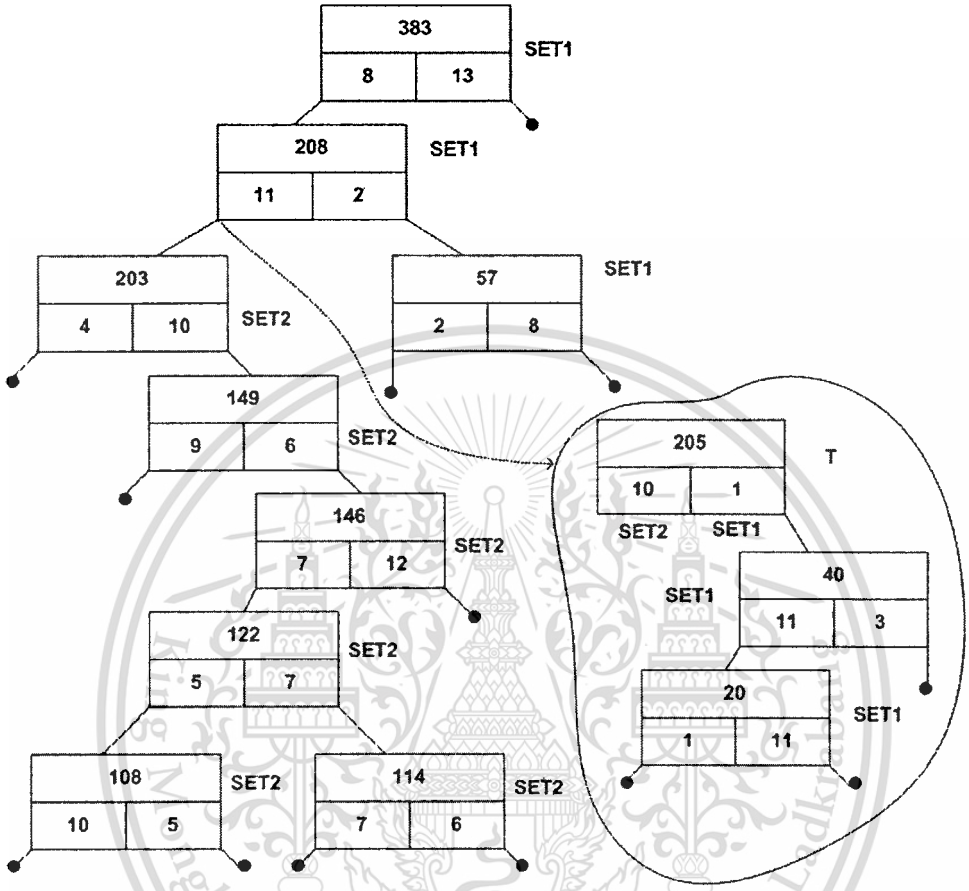

Figure 4.26(b): Clustering Tree from Figure 4.26(a) with moved T group to the right position

T group is a group of nodes in Clustering Tree that comes from the same set 1.

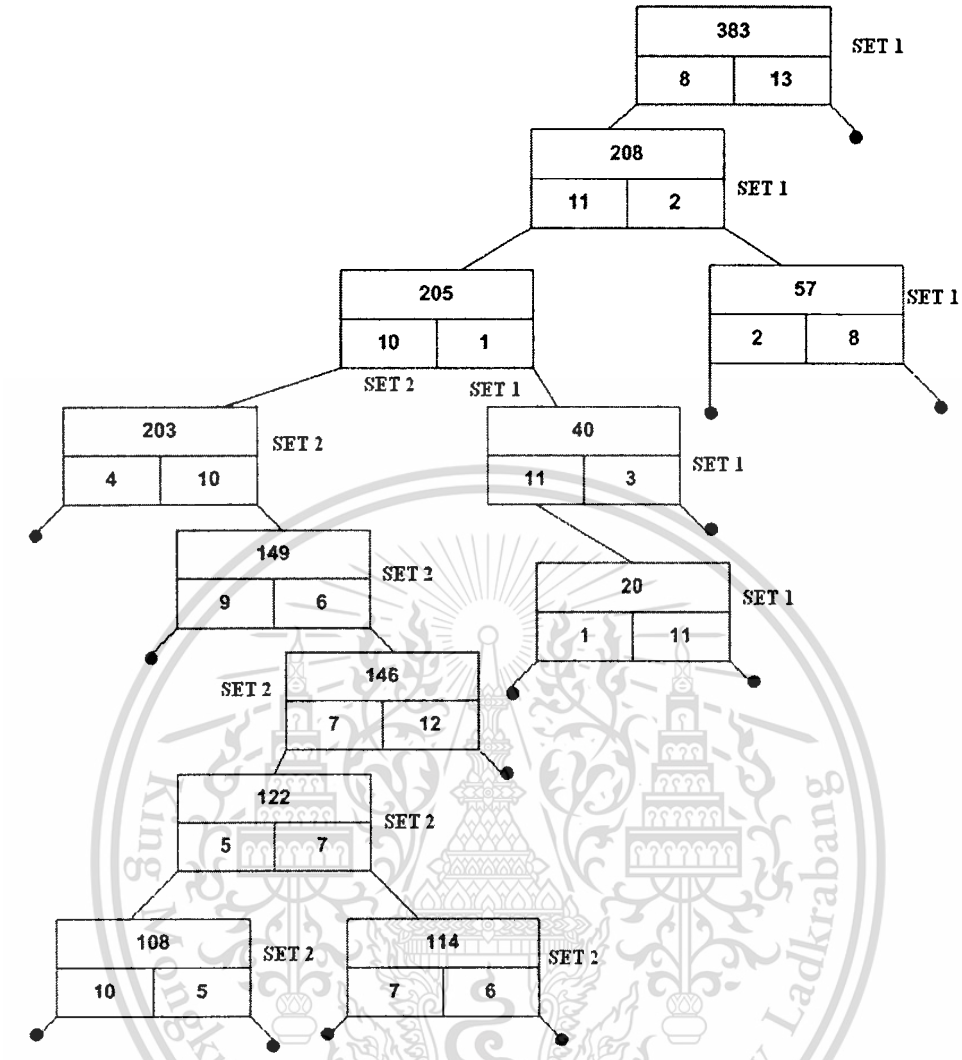

Figure 4.26(c): Clustering Tree for the MST on Figure 4.25

Whenever the weight of a connection line in the MST graph is increased, other connection lines may get an affect because the MST graph may not qualify its shortest properties. Therefore both MST and Clustering Tree may get an effect. This Chapter presents the simulations of the algorithms and the theories for adjusting Clustering Tree and MST by avoiding reconstruction them which is tested 100 times. This chapter showed only 5 examples of them. All tests show that the theories and the algorithms in Chapter 3 are correct.

Chapter 5

Conclusions

Clustering, or partitioning into dissimilar groups of similar objects, is a problem with many variants in mathematics and applied sciences. Presently the problem of clustering is usually not associated with the problem of searching points in d-dimensional space. In this thesis, clustering method, however, simultaneously produces both a search tree of d-dimensional points and clusters (hence, the title). This salient feature is useful in several applications, such as network systems, searching systems, etc. The Similarity-Guaranteed Clustering Algorithm and Its Search Tree is a Clustering Algorithm of the data in d dimension that can specify the distance value in Inner-Cluster and the distance of the Intra-cluster. Other Clustering Algorithms can not specify the distance value certainly. The Similarity-Guaranteed Clustering Algorithm and Its Search Tree still support searching data. However, the other Clustering Algorithms do not support the searching property.

The Similarity-Guaranteed Clustering Algorithm and Search Tree take the Minimum Spanning Tree (MST), built from Completed Graph, to create the Clustering Tree in Binary Tree for searching cluster. In the case of some weight of the edge in the MST is later adjusted: either increase or decrease weight. (Weight is the Euclidean Distance of connected node that uses to measure how similarity between these connected nodes. Weights may represent for the interesting features.) It is necessary to invoke the Clustering Tree constructing algorithm again. This situation affects the overall running time which can be greater. There is previous research for solving only the decreased weight. In this thesis, a new algorithm is presented to extend for the increased weight. It avoids reconstructing the Clustering Tree at each weight adjustment and Its running time is O(n) per weight adjustment where n is the number of nodes in the Clustering Tree. This research makes the Similarity-Guaranteed Clustering Algorithm and Search Tree complete with any adjusted weight: either increase or decrease weight. Theories and algorithms are presented in Chapter 3. Simulations run for testing are presented some examples in Chapter 4. All tested showed algorithms and theories in Chapter 3 are correct.

Reference

[1] A.Hinneburg and D.A. Keim. "An efficient approach to clustering in large multimedia databases with noise," *in Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining* New York, pp. 58-65 , Aug. 1198.

[2] A.K Jain, M.N . Murty and P.J. Flymn. Data Clustering: A Review. *ACM Computing Surveys*, vol.31 , No.3 ,1999.

[3] T.H. Conmen, C.E. Leiserson and R.L. Rivest; Introduction to Algorithms New York : *McGraw-Hill Book Company* , 1992.

[4] D.Fisher. "Improving inference through conceptual clustering," *in Proc. 1987 AAAI Conf.*, Seattle, WA, pp.461-465, July 1987.

[5] D. Jungnickel. Graphs, Networks and Algorithms. New York: *Springer-Verlag*, 2002.

[6] E.M. Knorr and T.Ng. Raymond "A unified notion of outliers: Properties and computation," *in Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining* , Newport Beach , CA, pp.219-222 , Aug 1997

[7] E. Knorr and T.Ng. Raymond "Algorithms for mining distance-based outliers in large datasets," *in Proc.1998 Int. Conf. Very Large Data Bases*, New York, pp.392-403, Aug. 1998

[8] G. Karypis, E.-H. Han , and V. Kumar. "CHAMELON: A hierarchical clustering algorithm using dynamic modeling" *COMPUTER*, vol. 32, pp.68-75, 1999.

[9] G. Sheikholeslami, S. Chatterjee, and A. Zhang. "WaveCluster: A multi-resolution clustering approach for very large spatial databases," *in Proc. 1998 Int. Conf. Very Large Data Bases*, New York, pp. 428-439, Aug. 1998.

[10] J.MacQueen. "Some methods for classification and analysis of multivariate observations," *in Proc. 5th Berkeley Symp. Math. Statist, Prob.*, vol. 1, pp 281-297 , 1967.

[11] J.Han and M. Kamber. Data Mining: Concepts and Techniques. San Francisco: Morgan Kaufimann Publishers, 2001.

[12] S. Kantabutra and C. Bunkhumpornpat; "Two Birds With One Stone: A Similarity-Guaranteed Clustering Algorithm and Its Search Tree", *in Proc. of the IEEE TENCON 2004*. Chiang Mai, Thailand, 2004.

[13] L. Kaufman and P.J.Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis. New York: John Wiley & Sons, 1990.

[14] M. Ankerst, M. Breuning , H.-P. Kriegel, and J. Sander. "OPTICS: Ordering points to identify the cluster structure," *in Proc. 1990 ACM-SIGMOD Int. Conf. Management of Data*, Philadelphia, PA, pp.49-60, June 1999.

[15] M.Ester, H.-P. Kriegel, J. Sander, and X. Xu. "A density-based algorithm for discovering clusters in large spatial databases," *in Proc. 1996 Int. Conf. Knowledge Discovery and Data Mining*, Portland, OR, pp.226-231, Aug. 1996.

[16] P.S. Bradley, U. Fayyad, and C.Reina. "Scaling clustering algorithms to large databases," *in Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining,* New York, pp.9-15, Aug 1998.

Reference(cont.)

[17] R. Agrawal, J. Gehrke, D. Gunopulos, and P.Raghavan . "Automatic subspace clustering of high dimensional data for data mining applications," *in Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data.* Seattle , WA, pp.94-105, June 1998.

[18] T.Ng. Raymond and J.Han. "Efficient and effective clustering method for spatial data mining," *in Proc .1994 Int. Conf. Very Large Data Bases,* Santiago, Chile, pp. 144-155 Sept. 1994.

[19] S. Guha, R. Rastogi, and K. Shim. "CURE: An efficient clustering algorithm for large databases," *in Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, Seattle, WA, pp.73-84, June 1998.

[20] S. L. Lauritzen. "The EM algorithm for graphical association models with missing data" *Computational Statistics and Data Analysis*, Vol. 19, pp. 191-201, 1995.

[21] T. Kohonen. "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, pp. 59-69, 1982

[22] T. Zhang, R. Rammakrishnan , and M. Livny . "BIRCH: An efficient data clustering method for very large databases," *in Proc. 1996 ACM_SIGMOD Int. Conf. Management of Data, Montreal*, Canada, pp.103-114 June 1196.

[23] W. Wang, J. Yang and R. Muntz. "STING: A statistical information grid approach to spatial data mining," *in Proc. 1997 Int. Conf. Very Large Data Bases*, Athens, Greece, pp.186-195, Aug. 1997.

[24] Z. Huang "Extensions to the k-means algorithms for clustering large data sets with categorical values," *Data Mining and Knowledge Discovery*, vol.2, pp.283-304. 1998

[25] R. Kannan, S Vempala, and A. Vetta. "On clustering: Good, Bad , Spectral," *in Proc. Of the 41st Foundation of Computer Science*, Redondo Beach, 2000.

[26] J.B. Kruskal. "On the shortest spanning subtree of a graph and the traveling salesman problem," *in Proc, Of the American Mathematical Society*, vol.7, pp.48-50, 1956.

[27] R.C. Prim. "Shortest connection networks and some generalizations," *Bell System Technical Jurnal*, vol.36, pp. 1389-140, 1957.

[28] R. E.Tarjan. Data Structures and Network Algorithms. *Society for Industrial and Applied Mathematics*, 1983.

[29] J. A. Hartigan, Clustering Algorithms, John-Wiley, New York,1975.

[30] T. Asano, B. Bhattacharya, M.Keil, and F. Yao. "Clustering algorithms based on minimum and maximum spanning trees," *in Proc. Of the 4th Annual Symposium on Computational Geometry*, Urbana Champaign , IL,pp.252-257 , Jan 1998.

[31] S. Kantabutra and C. Bunkhumpornpat. " Improved Performance Similarity-Guaranteed Clustering Algorithm and Its Search Tree for Handling a Decreased Weight", *9th National Computer Science and Engineering Conference(NCSEC 2005)*,The Thai Chamber of Commerce University, Thailand, October 2005

INDEX

INDEX (cont.)

INDEX (cont.)

INDEX (cont.)

INDEX (cont.)

INDEX (cont.)

INDEX (cont.)

INDEX (cont.)

INDEX (cont.)

Character **Page**

Z

APPENDIX

Class addmst

```
import java.io.*;
import java.lang.*;

class addmst{
        static int x;

        int addmst(String []link,String []listnode,String [][]mstnode,String z,int xi)  throws
        IOException
        {
                int j=1;
                String namefile;
                checknode del=new checknode();
                listnode list=new listnode();
                deletesubloop delete=new deletesubloop();
                for ( int i=0;i<listnode.length;i++)
                {
                        if (link[2].equals(listnode[i]))
                        {
                                j=0;
                        }
                }
                if (j==1)
                {
                        if ( x<mstnode.length)
                        {
                                                mstnode[x][0]=link[0];
                                                mstnode[x][1]=link[1];
                                                mstnode[x][2]=link[2];
                                                del.deletenode(listnode,mstnode,x,z);
                                                xi=list.addlist(link[2],listnode,z,xi);
```

```
                                        x++;
                                        j=0;
                }
                else if(x>=mstnode.length)
                {
                        x=0;
                }
        }
        else if (j==0)
        {
                if ( x<mstnode.length)
                {
                        delete.readfile(link[1],link,mstnode.length,z);
                        delete.readfile(link[2],link,mstnode.length,z);
                }
        }
        return xi;
    }
}
```

Class checknode

```
import java.io.*;
import java.lang.*;
import java.util.StringTokenizer;

class checknode{
        static int x;
        static int y;
        static String []mstlink=new String[3];
        static double min;
        static int t;
        void checklist (String name,String []listnode,String z,String [][] mstnode,int xi)
        throws IOException
```

```
{
        int i=1;
        int l=0;
        int x=xi;
        checknode read=new checknode();
        FileCopy copy=new FileCopy();
        FileCopy1 copy1=new FileCopy1();
        random ran=new random();
        String fromcopy;
        String tocopy,filename;
        for (int j=0;j<listnode.length;j++)
        {
                if ( name.equals(listnode[j]))
                {
                        i=0;
                }
        }
                if (i==1)
                {
                        listnode[x]=name;
                        fromcopy=listnode[x]+".txt";
                        tocopy=z+"/"+listnode[x]+".txt";
                        copy.filecopy(fromcopy,tocopy,z);
                        x++;
                }

        for (int k=0;k<listnode.length;k++)
        {
                if (!(listnode[k].equals("")))
                {
                        l++;
                }
        }
```

```
        for(int m=0;m<l;m++)
        {
                filename=z+"/"+listnode[m]+".txt";
                xi=read.readfile(filename,mstnode,l,min,listnode,z,x);
        }
        if (l<=mstnode.length)
        {
                read.checklist("1",listnode,z,mstnode,xi);
        }
        if (l==(mstnode.length+1))
        {
                filenode file=new filenode();
                file.filenode3("mst",mstnode,z);
                ran.randommst(mstnode,z);
        }
}

int readfile(String filename,String[][] mstnode,int l,double i,String []listnode
,String z,int xi) throws IOException
{
        int h=0;
        String b,c;
        FileReader fin =new FileReader(filename);
        BufferedReader bin = new BufferedReader (fin);
        String link[]=new String[3];
        b =  bin.readLine();
        StringTokenizer st=new StringTokenizer(b);
        while(st.hasMoreTokens())
        {
                c=st.nextToken();
                if (h<link.length)
                {
                        link[h++]=c;
```

```
                }
                else
                {
                        h=0;
                }
        }
        fin.close();
        bin.close();
        checknode check=new checknode();
        xi=check.checkmst(link,l,i,mstnode,listnode,z,xi);
        return xi;
}

int checkmst(String [] link,int l,double i, String[][] mstnode,String [] listnode
,String z,int xi) throws IOException
{
        double j;
        min=i;
        checknode del=new checknode();
        listnode list=new listnode();
        addmst mst=new addmst();
        j=Double.parseDouble(link[0]);
        if(l==1)
        {
                xi=mst.addmst(link,listnode,mstnode,z,xi);
                min=0;
                t=0;
        }
        else
        {
                t++;
                if (t==1)
                {
```

```
                min=j;
                mstlink=link;
        }
        else
        {
                if(j<=min)
                {
                        min=j;
                        mstlink=link;
                        if(l==t)
                        {
                                xi=mst.addmst(mstlink,listnode,
                                mstnode,z,xi);
                                mstlink[0]="";
                                mstlink[1]="";
                                mstlink[2]="";
                                min=0;
                                t=0;
                        }
                }
                else
                {
                        if (l==t)
                        {

                                xi=mst.addmst(mstlink,listnode,
                                mstnode,z,xi);
                                mstlink[0]="";
                                mstlink[1]="";
                                mstlink[2]="";
                                min=0;
                                t=0;
```

```
                                    }
                                }
                            }
                        }
                        return xi;
                }

                void deletenode(String [] listnode,String [][]mstnode,int i,String z)
                throws IOException
                {
                        checknode check=new checknode();
                        read Read =new read();
                        FileCopy copy =new FileCopy();
                        String fromcopy,tocopy,filename;
                        String node[]=new String[2];
                        node[0]=mstnode[i][1];
                        node[1]=mstnode[i][2];
                        for(int j=0;j<node.length;j++)
                        {
                            fromcopy=node[j]+".txt";
                            tocopy=z+"/"+node[j]+".txt";
                            copy.filecopy(fromcopy,tocopy,z);
                            Read.readfile(tocopy,mstnode,mstnode.length,i,z);
                        }
                        }
}
```

Classs createnode

```
import java.util.Random;
import java.text.DecimalFormat;
import java.io.*;
import java.util.StringTokenizer;
import java.lang.*;
```

```
class createnode{
        static int n;
        static int n1;
        static int n2;

        int creanode() throws IOException
        {
                Random rn=new Random();
                int i= rn.nextInt(500)+4;
                quicksort quick=new quicksort();
                System.out.println("i is "+i);
                int i1=i-1;
                String [][][] nodefile=new String[i][i1][3];
                createnode crenode=new createnode();
                for(int i0=0;i0<nodefile.length;i0++)
                {
                        crenode.canode(nodefile,i0,i1,0);
                }
                crenode.senddata(nodefile);
                for (int i2=0;i2<nodefile.length;i2++)
                {
                        quick.quickSort(nodefile[i2]);
                }
                return i;
        }

        void canode(String [][][]nodefile,int n,int number,int loop)
        {
                for(int i=0;i<number;i++)
                {
                        nodefile[n][i][1]=String.valueOf(n+1);
                }
                for(int j=loop;j<=number;j++)
                {
```

```
                if ((j+1)!=(n+1))
                {
                        nodefile[n][loop][2]=String.valueOf(j+1);
                        loop++;
                }
        }
}

void senddata(String[][][] nodefile)
throws IOException
{
        String keepsub[]=new String[2];
        Random ran=new Random();
        DecimalFormat df = new DecimalFormat();
        df.applyPattern("0.00");
        for ( int i=0;i<nodefile.length;i++)
        {
            for(int j=0;j<nodefile.length-1;j++)
            {
                if (nodefile[i][j][0]==null)
                {
                        double k=Double.parseDouble
                        (df.format(ran.nextDouble()*2000));
                        nodefile[i][j][0]=String.valueOf(k);
                        keepsub[0]=nodefile[i][j][1];
                        keepsub[1]=nodefile[i][j][2];
                        for(int k0=0;k0<nodefile.length;k0++)
                        {
                                for (int k1=0;k1<nodefile.length-1;k1++)
                                {
if(nodefile[k0][k1][1].equals(keepsub[1]) && nodefile[k0][k1][2].equals(keepsub[0]))
{
            nodefile[k0][k1][0]=nodefile[i][j][0];
```

```
                keepsub[0]=keepsub[1]="";
                keepsub[0]=keepsub[1]="";
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
```

Class deletedata

```
import java.io.*;

class deletedata{
 public static void delete(String fileName)
 {
          fileName=fileName+".txt";
          deletedata del=new deletedata();
   try {
    File target = new File(fileName);
    if (!target.exists()) {
     System.err.println("File " + fileName
       + " not present to begin with!");
     return;
    }
    if (target.delete())
     System.err.println("** Deleted " + fileName + " **");
   } catch (SecurityException e) {
    System.err.println("Unable to delete " + fileName + "("
      + e.getMessage() + ")");
   }
   catch(Exception e) {
```

```
   System.out.println("Git");
        }
  }
}
```

Class deletesubloop

```
import java.io.*;
import java.lang.*;
import java.util.StringTokenizer;

class deletesubloop{
        void readfile (String namefile,String [] subloop,int num,String z ) throws IOException
        {
                int j=0;
                int k=0;
                deletesubloop Read = new deletesubloop ();
                String subfile[][]=new String [num][3];
                String b,c,c1;
                String link[]=new String[3];
                namefile=z+"/"+namefile+".txt";
                FileReader fin = new FileReader(namefile);
                BufferedReader bin = new BufferedReader (fin);
                for (int i1=0;i1<subfile.length;i1++)
                {
                        for (int i2=0;i2<3;i2++)
                        {
                                subfile[i1][i2]="";
                        }
                }
                while ((b =  bin.readLine()) != null) {
                        StringTokenizer st=new StringTokenizer(b);
                        while(st.hasMoreTokens())
                        {
```

```
                    c1=st.nextToken();
                    if (j<link.length)
                    {
                    link[j++]=c1;
                    }
                    lse
                    {
                            j=0;
                    }
            }
        if(k<subfile.length)
        {
                subfile[k][0]=link[0];
                subfile[k][1]=link[1];
                subfile[k++][2]=link[2];
                link[0]="";
                link[1]="";
                link[2]="";
                j=0;
        }
        else
        {
                k=0;
        }

    }
  fin.close();
  bin.close();
  Read.deletedata(subloop,subfile,z);
}

  void deletedata(String []subloop,String [][] subfile,String z)
  throws IOException
```

```
{
        for (int l=0;l<subfile.length;l++)
        {
                if((subfile[l][0].equals(subloop[0]) &
                        subfile[l][1].equals(subloop[1]) &
                        subfile[l][2].equals(subloop[2])))
                  {
                        subfile[l][0]="";
                        subfile[l][1]="";
                        subfile[l][2]="";
                  }
                 else if ((subfile[l][0].equals(subloop[0]) &
                        subfile[l][1].equals(subloop[2]) &
                        subfile[l][2].equals(subloop[1])))
                  {
                        subfile[l][0]="";
                        subfile[l][1]="";
                        subfile[l][2]="";
                  }
        }
        read Read=new read();
        Read.swapdata(subfile,z);
}

void swapdata(String [][]subfile,String z) throws IOException
{
        String []link=new String[3];
        link[0]=link[1]=link[2]="";
        for ( int i=0;i<subfile.length;i++)
        {
                if (subfile[i][0].equals("") & subfile[i][1].equals("") &
                        subfile[i][2].equals(""))
                {
```

```
                for(int j=i+1;j<subfile.length;j++)
                {
                        if (!(subfile[j][0].equals("") & subfile[j][1].equals("") &
                                subfile[j][2].equals("")))
                                {
                                        subfile[i][0]=subfile[j][0];
                                        subfile[i][1]=subfile[j][1];
                                        subfile[i][2]=subfile[j][2];
                                        subfile[j][0]="";
                                        subfile[j][1]="";
                                        subfile[j][2]="";
                                        break;
                                }
                }
        }
    }
    filenode file=new filenode();
    file.filenode2(subfile,z);
}

void swapdata(String [][]subfile,String z,String name) throws IOException
{
    String []link=new String[3];
    link[0]=link[1]=link[2]="";
    for ( int i=0;i<subfile.length;i++)
    {
        if (subfile[i][0].equals("") & subfile[i][1].equals("") &
                subfile[i][2].equals(""))
        {
                for(int j=i+1;j<subfile.length;j++)
                {
                        if (!(subfile[j][0].equals("") & subfile[j][1].equals("") &
                                subfile[j][2].equals("")))
```

```
                                        {
                                                subfile[i][0]=subfile[j][0];
                                                subfile[i][1]=subfile[j][1];
                                                subfile[i][2]=subfile[j][2];
                                                subfile[j][0]="";
                                                subfile[j][1]="";
                                                subfile[j][2]="";
                                                break;
                                        }
                                }
                        }
                }
                filenode file=new filenode();
                file.filenode4(name,subfile,z);
        }
}
```

Class FileCopy

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;

public class FileCopy {
        static int x;

  public static void filecopy(String fromcopy,String tocopy,String z) {
        String m1;
        try {
    copy(fromcopy,tocopy,z);
    } catch (IOException e) {
```

```
    System.err.println(e.getMessage());
  }
}

public static void copy(String fromFileName, String toFileName,String directory)
    throws IOException {
  File fromFile = new File(fromFileName);
  File toFile = new File(toFileName);
  if (!fromFile.exists())
    throw new IOException("FileCopy: " + "no such source file: "
        + fromFileName);
  if (!fromFile.isFile())
    throw new IOException("FileCopy: " + "can't copy directory: "
        + fromFileName);
  if (!fromFile.canRead())
    throw new IOException("FileCopy: " + "source file is unreadable: "
        + fromFileName);

  if (toFile.isDirectory())
    toFile = new File(toFile, fromFile.getName());

  if (toFile.exists()) {
    if (!toFile.canWrite())
      throw new IOException("FileCopy: "
          + "destination file is unwriteable: " + toFileName);
    String response="n";
    if (!response.equals("Y") && !response.equals("y"));
      throw new IOException();
  } else {
    String parent = toFile.getParent();
    if (parent == null)
      parent = System.getProperty("user.dir");
    File dir = new File(parent);
```

```
if (!dir.exists())
{
                    File file = new File(directory);
            file.mkdir();
}
if (dir.isFile())
  throw new IOException("FileCopy: "
     + "destination is not a directory: " + parent);
if (!dir.canWrite())
  throw new IOException("FileCopy: "
     + "destination directory is unwriteable: " + parent);
}
FileInputStream from = null;
FileOutputStream to = null;
try {
 from = new FileInputStream(fromFile);
 to = new FileOutputStream(toFile);
 byte[] buffer = new byte[4096];
 int bytesRead;

 while ((bytesRead = from.read(buffer)) != -1)
  to.write(buffer, 0, bytesRead);
} finally {
 if (from != null)
  try {
   from.close();
  } catch (IOException e) {
   ;
  }
 if (to != null)
  try {
   to.close();
  } catch (IOException e) {
```

```
        ;
      }
    }
  }
}
```

Class FileCopy1

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;


public class FileCopy1 {

        public static void filecopy(String []listnode,String z)
        {
                String fromcopy,tocopy;
                for(int i=0;i<listnode.length;i++)
                {
                        fromcopy=listnode[i]+".txt";
                        tocopy=z+"/"+fromcopy;
                        filerecieve(fromcopy,tocopy,z);
                        fromcopy="";
                        tocopy="";
                }
        }

        public static void filecopy(String fromcopy,String tocopy,String z)
        {
        filerecieve(fromcopy,tocopy,z);
        }
```

```
        public static void filerecieve(String fromcopy,String tocopy,String z)
        {
                try {
 copy(fromcopy,tocopy,z);
 } catch (IOException e) {
 System.err.println(e.getMessage());
}
        }

 public static void copy(String fromFileName, String toFileName,String directory)
  throws IOException {
File fromFile = new File(fromFileName);
File toFile = new File(toFileName);

if (!fromFile.exists())
  throw new IOException("FileCopy: " + "no such source file: "
     + fromFileName);
if (!fromFile.isFile())
  throw new IOException("FileCopy: " + "can't copy directory: "
     + fromFileName);
if (!fromFile.canRead())
  throw new IOException("FileCopy: " + "source file is unreadable: "
     + fromFileName);

if (toFile.isDirectory())
  toFile = new File(toFile, fromFile.getName());

if (toFile.exists()) {
  if (!toFile.canWrite())
    throw new IOException("FileCopy: "
       + "destination file is unwriteable: " + toFileName);
  System.out.print("");
  System.out.flush();
```

```
BufferedReader in = new BufferedReader(new InputStreamReader(
    System.in));
String response="y";
if (!response.equals("Y") && !response.equals("y"))
  throw new IOException("FileCopy: "
    + "existing file was not overwritten.");
} else {
String parent = toFile.getParent();
if (parent == null)
  parent = System.getProperty("user.dir");
File dir = new File(parent);
if (!dir.exists())
{
                    File file = new File(directory);
                    file.mkdir();
}
if (dir.isFile())
  throw new IOException("FileCopy: "
    + "destination is not a directory: " + parent);
if (!dir.canWrite())
  throw new IOException("FileCopy: "
    + "destination directory is unwriteable: " + parent);
}
FileInputStream from = null;
FileOutputStream to = null;
try {
  from = new FileInputStream(fromFile);
  to = new FileOutputStream(toFile);
  byte[] buffer = new byte[4096];
  int bytesRead;

  while ((bytesRead = from.read(buffer)) != -1)
    to.write(buffer, 0, bytesRead); // write
```

```
    } finally {
     if (from != null)
       try {
         from.close();
       } catch (IOException e) {
         ;
       }
     if (to != null)
       try {
         to.close();
       } catch (IOException e) {
         ;
       }
    }
  }
}
```

Class filenode

```
import java.io.*;
import java.lang.*;

class filenode {
        String [][]a;

        public void filenode1 (String [][]a)  throws IOException{
        int length;
        String namefile,c1;
        c1=" ";
        length=a.length;
        namefile=a[0][1];
        namefile=namefile+".txt";
        FileWriter fileWriter = new FileWriter(namefile);
        for (int i=0;i<length;i++)
        {
```

```
                for (int j=0;j<3;j++)
                {

                        fileWriter.write(a[i][j]);
                        fileWriter.write(" ");
                }
                fileWriter.write('\n');
        }
fileWriter.close();
}

public void filenode2 (String [][]a,String z)  throws IOException{
int length;
String namefile,c1;
c1=" ";
length=a.length;
namefile=a[0][1];
namefile=z+"/"+namefile+".txt";
FileWriter fileWriter = new FileWriter(namefile);
for (int i=0;i<length;i++)
        {
                for (int j=0;j<3;j++)
                {
                        fileWriter.write(a[i][j]);
                        fileWriter.write(" ");
                }
                fileWriter.write('\n');
        }
fileWriter.close();
}

public void filenode3 (String namefile,String [][]mstnode,String z)  throws
IOException{
```

```
int length;
String c1;
c1=" ";
length=mstnode.length;
namefile=z+"/"+namefile+".txt";
FileWriter fileWriter = new FileWriter(namefile);
for (int i=0;i<length;i++)
        {
                for (int j=0;j<3;j++)
                {

                        fileWriter.write(mstnode[i][j]);
                        fileWriter.write(" ");
                }
                fileWriter.write('\n');
        }
fileWriter.close();
}

public void filenode4 (String namefile,String [][]mstnode,String z)  throws
IOException{
int length;
String c1;
c1=" ";
length=mstnode.length;
FileWriter fileWriter = new FileWriter(namefile);
for (int i=0;i<length;i++)
        {
                for (int j=0;j<3;j++)
                {

                        fileWriter.write(mstnode[i][j]);
                        fileWriter.write(" ");
```

```
                }
                fileWriter.write('\n');
        }
fileWriter.close();
}

public void filenode4 (String namefile,String []a,String z)  throws IOException{
int length;
String c1;
c1=" ";
length=a.length;
namefile=z+"/"+namefile+".txt";
FileWriter fileWriter = new FileWriter(namefile);
for (int i=0;i<length;i++)
        {
                fileWriter.write(a[i]);
                fileWriter.write(" ");
        }
fileWriter.close();
}

public void filenode4 (String namefile,String num,String z)  throws IOException{
        int length;
        namefile=z+"/"+namefile+".txt";
        FileWriter fileWriter = new FileWriter(namefile);
        fileWriter.write(num);
        fileWriter.close();
 }

public void filenode5 (String namefile,String [][]subfile,String z)  throws IOException{
int length;
String c1;
c1=" ";
length=subfile.length;
```

```
        namefile=z+"/"+namefile+"ch.txt";
        FileWriter fileWriter = new FileWriter(namefile);
        for (int i=0;i<length;i++)
                {
                        for (int j=0;j<3;j++)
                        {

                                fileWriter.write(subfile[i][j]);
                                fileWriter.write(" ");
                        }
                        fileWriter.write('\n');
                }
        fileWriter.close();
        }
 }
```

Class listnode

```
import java.io.*;
import java.lang.*;

class listnode
{
        static int x;
        int addlist(String name,String []listnode,String z,int xi)
                throws IOException
        {
                int i=1;
                checknode read=new checknode();
                FileCopy copy=new FileCopy();
                String fromcopy;
                String tocopy,filename;
                for (int j=0;j<listnode.length;j++)
                {
```

```
                    if( name.equals(listnode[j]))
                    {
                        i=0;
                    }
                }
                if (i==1)
                {
                    listnode[xi ]=name;
                    xi++;
                    tocopy=z+"/"+fromcopy;
                    copy.filecopy(fromcopy,tocopy,z);*/
                }
                return xi;
        }
}
```

Class nodeSearch

```
import java.io.*;
import java.util.StringTokenizer;
import java.lang.*;
import java.util.*;
import java.sql.*;

class nodeSearch
{
        static int k;
        static int n;
        static int n1;
        static int t;

        void keeplnode(String node,String [][]mstnode,String [] slsearch,String z)
        throws IOException
        {
            String [][] knode=new String[mstnode.length-1][2];
```

```
String [][] prmsts=new String[mstnode.length][3];
String []senode=new String[mstnode.length+1];
nodeSearch nodes=new nodeSearch();
prmst nsearch=new prmst();
for(int i1=0;i1<prmsts.length;i1++)
{
        for(int i2=0;i2<3;i2++)
        {
                prmsts[i1][i2]=mstnode[i1][i2];
        }
}
if (n<slsearch.length)
{
        slsearch[n++]=node;
        nodes.keeplnode1(prmsts,slsearch,knode);
}
else
{
        n=0;
}
nodes.ksenode(senode,slsearch);
for(int i=0;i<senode.length;i++)
{
        if (!senode[i].equals(""))
        {
                nodes.besearch(senode[i],z,senode,mstnode.length);
        }
        System.out.println("senode["+i+"]="+senode[i]);
}
filenode file=new filenode();
file.filenode4("senode",senode,z);
double startTime;
Calendar c2 = Calendar.getInstance();
```

```
        Timestamp time = new Timestamp(c2.getTimeInMillis());
        String x10=time.toString();
        System.out.println("Start Time : " + x10);
        String[] result = x10.split(":");
        System.out.println(result[result.length-1]);
        startTime=Double.parseDouble(result[result.length-1]);
        nsearch.searchmst(mstnode,z,senode,true,startTime);
        n=0;
        n1=0;
        t=0;
}

void keeplnode1(String [][]prmsts,String[] slsearch,String [][] knode)
{
        int l=0;
        int i=0;
        nodeSearch nodes=new nodeSearch();
        for( int i2=0;i2<slsearch.length;i2++)
        {
                if(!(slsearch[i2].equals("")))
                {
                        l++;
                }
        }
        System.out.println("the slsearch node data have ="+l);
        for(int i0=t;i0<l;i0++)
        {
                for(int i1=0;i1<prmsts.length;i1++)
                {
                        if(slsearch[i0].equals(prmsts[i1][1]) ||
                        slsearch[i0].equals(prmsts[i1][2]))
                        {
                                i=1;
```

```
                    }
                    if (i==1)
                    {
                    nodes.keeplnode2(prmsts,i1,knode,slsearch);
                    i=0;
                    }
                }
            }
            if ( i==0)
            {
                if (t<=l)
                {
                    t++;
                    keeplnode1(prmsts,slsearch,knode);
                }
                else
                {
                    t=0;
                }
            }
}

void keeplnode2(String [][]prmsts,int i,String [][]knode,String []slsearch)
{
            int j=1;
            int l=0;
            System.out.println("kepplnode2");
            nodeSearch nodes=new nodeSearch();
            for(int i0=0;i0<knode.length;i0++)
            {
                if ((prmsts[i][1].equals(knode[i0][0]) &&
                        prmsts[i][2].equals(knode[i0][1])) ||
                        (prmsts[i][1].equals(knode[i0][1]) &&
```

```
                prmsts[i][2].equals(knode[i0][0])))
                {
                        j=0;
                }
        }
        if (j==1)
        {
                if (n1<knode.length)
                {
                        knode[n1][0]=prmsts[i][1];
                        knode[n1++][1]=prmsts[i][2];
                        prmsts[i][0]="";
                        prmsts[i][1]="";
                        prmsts[i][2]="";
                }
                else
                {
                        n1=0;
                }
        }
        for(int i1=0;i1<knode.length;i1++)
        {
                if (!(knode[i1][0]==null))
                {
                        l++;
                }
        }
        for(int i2=0;i2<l;i2++)
        {
                nodes.kslsnode(knode[i2][0],slsearch);
                nodes.kslsnode(knode[i2][1],slsearch);
        }
```

```
        for(int i3=0;i3<slsearch.length;i3++)
        {
                System.out.println("Slearch["+i3+"]="+slsearch[i3]);
        }
}

void kslsnode(String node,String []slsearch)
{
        int j=1;
        for(int i=0;i<slsearch.length;i++ )
        {
                if (node.equals(slsearch[i]))
                {
                        j=0;
                }
        }
        if( j==1)
        {
                if (n<slsearch.length)
                {
                        slsearch[n++]=node;
                }
                else
                {
                        n=0;
                }
        }
}

void swapdata(String [][]prmsts)
{
        for ( int i=0;i<prmsts.length;i++)
        {
                if (prmsts[i][0].equals("") & prmsts[i][1].equals("") &
```

```
                prmsts[i][2].equals(""))
            {
                for(int j=i+1;j<prmsts.length;j++)
                {
                    if (!(prmsts[j][0].equals("") &
                    prmsts[j][1].equals("") &
                    prmsts[j][2].equals("")))
                    {
                        prmsts[i][0]=prmsts[j][0];
                        prmsts[i][1]=prmsts[j][1];
                        prmsts[i][2]=prmsts[j][2];
                        prmsts[j][0]="";
                        prmsts[j][1]="";
                        prmsts[j][2]="";
                        break;
                    }
                }
            }
        }
    }

    void ksenode(String[] senode,String[] slsearch)
    {
        int l=0;
        for (int i=0;i<slsearch.length;i++)
        {
            if (!(slsearch[i].equals("")))
            {
                l++;
            }
        }
        if (l<= ( slsearch.length/2))
        {
```

```
            for(int i3=0;i3<slsearch.length;i3++)
            {
                senode[i3]=slsearch[i3];
            }
        }
        else
        {
            for (int i0=0;i0<senode.length;i0++)
            {
                senode[i0]=String.valueOf(i0+1);
            }
            for(int i1=0;i1<senode.length;i1++)
            {
                for(int i2=0;i2<slsearch.length;i2++)
                {
                    if (senode[i1].equals(slsearch[i2]))
                    {
                        senode[i1]="";
                    }
                }
            }
            for ( int i3=0;i3<senode.length;i3++)
            {
                if (senode[i3].equals(""))
                {
                    for(int i4=i3+1;i4<senode.length;i4++)
                    {
                        if (!(senode[i4].equals("")))
                        {
                            senode[i3]=senode[i4];
                            senode[i4]="";
                            break;
                        }
```

```
                                }
                            }
                        }
                    }
                }
void besearch(String namefile,String z,String []senode,int num) throws IOException
{
        nodeSearch nodes=new nodeSearch();
        String filename;
        filename=z+"/"+namefile+"ch.txt";
        String subfile[][]=new String [num][3];
        int j=0;
String b,c,c1;
String link[]=new String[3];
FileReader fin = new FileReader(filename);
BufferedReader bin = new BufferedReader (fin);
for (int i1=0;i1<subfile.length;i1++)
{
        for (int i2=0;i2<3;i2++)
        {
                subfile[i1][i2]="";
        }
}
while ((b =  bin.readLine()) != null) {
StringTokenizer st=new StringTokenizer(b);
while(st.hasMoreTokens())
        {
                c1=st.nextToken();
                if (j<link.length)
                {
                link[j++]=c1;
                }
```

```
                else
                {
                        j=0;
                }
        }
if (k<subfile.length)
{
                subfile[k][0]=link[0];
                subfile[k][1]=link[1];
                subfile[k++][2]=link[2];
                link[0]="";
                link[1]="";
                link[2]="";
                j=0;
}
else
{
        k=0;
}
}
        fin.close();
        bin.close();
        k=0;
        nodes.deletebe(subfile,senode,z);
}

void deletebe(String [][]subfile,String[] senode,String z) throws IOException
{
        nodeSearch nodes=new nodeSearch();
        for(int i=0;i<senode.length;i++)
        {
                for(int j=0;j<subfile.length;j++)
                {
```

```
                    if(senode[i].equals(subfile[j][2]))
                    {
                        subfile[j][0]="";
                        subfile[j][1]="";
                        subfile[j][2]="";
                    }
                }
            }
            nodes.swapsub(subfile,z);
}

void swapsub(String [][]subfile,String z) throws IOException
{
            filenode file=new filenode();
            String []link=new String[3];
            link[0]=link[1]=link[2]="";
            for ( int i=0;i<subfile.length;i++)
            {
                if (subfile[i][0].equals("") & subfile[i][1].equals("") &
                        subfile[i][2].equals(""))
                {
                    for(int j=i+1;j<subfile.length;j++)
                    {
                        if (!(subfile[j][0].equals("") & subfile[j][1].equals("")
                        &
                            subfile[j][2].equals("")))
                        {
                            subfile[i][0]=subfile[j][0];
                            subfile[i][1]=subfile[j][1];
                            subfile[i][2]=subfile[j][2];
                            subfile[j][0]="";
                            subfile[j][1]="";
                            subfile[j][2]="";
```

```
                                        break;
                                    }
                                }
                            }
                        }
                        file.filenode5(subfile[0][1],subfile,z);
                }
        }
```

Class prmst

```
import java.io.*;
import java.lang.*;
import java.util.StringTokenizer;
import java.util.*;
import java.sql.*;

class prmst{
        static double min;
        static int t;
        static String []mstlink=new String[3];
        static int k;
        static double stime;

        void swapdata(String [][]mstnode,String z)
        {
                String []link=new String[3];
                for ( int i=0;i<mstnode.length;i++)
                {
                        if (mstnode[i][0].equals("") & mstnode[i][1].equals("") &
                                mstnode[i][2].equals(""))
                        {
                                for(int j=i+1;j<mstnode.length;j++)
                                {
```

```
                    if (!(mstnode[j][0].equals("") &
                    mstnode[j][1].equals("") &
                            mstnode[j][2].equals("")))
                    {
                            mstnode[i][0]=mstnode[j][0];
                            mstnode[i][1]=mstnode[j][1];
                            mstnode[i][2]=mstnode[j][2];
                            mstnode[j][0]="";
                            mstnode[j][1]="";
                            mstnode[j][2]="";
                            break;
                    }
                }
            }
        }
}

void searchmst(String [][]mstnodech,String z,String []senode,boolean d,double Stime)
throws IOException
{
        int l=0;
        double endTime,escapetime;
        endTime=0;
        stime=Stime;
        filenode file=new filenode();
        prmst search=new prmst();
        if (d)
        {
                for (int i=0;i<senode.length;i++)
        {
                if (!(senode[i].equals("")))
                {
                        l++;
```

```
                }
        }
        for (int i1=0;i1<senode.length;i1++)
        {
                if (!(senode[i1].equals("")))
                {
                        search.readdata(mstnodech,senode[i1],z,l,senode);
                }
        }
        }
        else
        {
                Calendar c2 = Calendar.getInstance();
                Timestamp time = new Timestamp(c2.getTimeInMillis());
                String x10=time.toString();
                System.out.println("End Time : " + x10);
                String[] result = x10.split(":");
                System.out.println(result[result.length-1]);
                endTime=Double.parseDouble(result[result.length-1]);
                escapetime=endTime - stime;
                System.out.println("Using Times is " + escapetime);
                file.filenode4("time",String.valueOf(escapetime), z);
        }
}

void readdata(String [][]mstnodech,String name,String z,int l,String []senode)
throws IOException
{
        int h=0;
        String b,c,filename;
        filename=z+"/"+name+"ch.txt";
        FileReader fin =new FileReader(filename);
        BufferedReader bin = new BufferedReader (fin);
```

```
            String link[]=new String[3];
            b =  bin.readLine();
            StringTokenizer st=new StringTokenizer(b);
            while(st.hasMoreTokens())
            {
                    c=st.nextToken();
                    System.out.print(c);
                    if (h<link.length)
                    {
                    link[h++]=c;
                    }
                    else
                    {
                        h=0;
                    }
            }
            fin.close();
            bin.close();
            prmst check=new prmst();
            check.checkdata(link,l,min,mstnodech,z,senode);
}

void checkdata(String []link,int l,double i,String [][]mstnodech,String z,String
[]senode)
 throws IOException
{
            double j;
            min=i;
            String name;
            filenode file=new filenode();
            j=Double.parseDouble(link[0]);
            prmst check=new prmst();
            if (l==1)
```

```
{
        mstlink[0]=link[0];
        mstlink[1]=link[1];
        mstlink[2]=link[2];
        check.addmst(mstlink,mstnodech,z,senode);
        mstlink[0]="";
        mstlink[1]="";
        mstlink[2]="";
}
else
{
        t++;
        if (t==1)
        {
            min=j;
            mstlink[0]=link[0];
            mstlink[1]=link[1];
            mstlink[2]=link[2];
        }
        else
        {
            if(j<=min)
            {
                min=j;
                mstlink[0]=link[0];
                mstlink[1]=link[1];
                mstlink[2]=link[2];
                if(l==t)
                {
                    check.addmst(mstlink,mstnodech,z,senode);
                    min=0;
                    t=0;
                }
```

```
                    }
                    else
                    {
                        if (l==t)
                        {
                            check.addmst(mstlink,mstnodech,z,senode);
                            min=0;
                            t=0;
                        }
                    }
                }
            }
        }

    void addmst(String []mstlink,String [][]mstnodech,String z,String []senode)
     throws IOException
    {
            int n=1;
            String name;
            long endtime;
            prmst delete=new prmst();
            filenode file=new filenode();
            mstnodech[mstnodech.length-1][0]=mstlink[0];
            mstnodech[mstnodech.length-1][1]=mstlink[1];
            mstnodech[mstnodech.length-1][2]=mstlink[2];
            delete.searchmst(mstnodech,z,senode,false,stime);
            name=z+"/"+"newmst.txt";
            file.filenode4(name,mstnodech,z);
     }
    }
```

Class quicksort


```
import java.io.*;
```

```
import java.lang.*;
public class quicksort {

 public static void quickSort(String [][] strArray) throws IOException{
        filenode f =new filenode();
        recQuickSort(strArray, 0, strArray.length - 1);
  f.filenode1(strArray);
 }

 public static void quickSort(String[][] strArray,String floder,String name) throws IOException
 {
        filenode f =new filenode();
        recQuickSort(strArray, 0, strArray.length - 1);
  f.filenode3(name,strArray,floder);
 }

 public static void recQuickSort(String [][] strArray, int left, int right) {
  int size = right - left + 1;
  if (size <= 3)
   manualSort(strArray, left, right);
  else {
   double median = medianOf3(strArray, left, right);
   int partition = partitionIt(strArray, left, right, median);
   recQuickSort(strArray, left, partition - 1);
   recQuickSort(strArray, partition + 1, right);
  }
 }

 public static double medianOf3(String [][] strArray, int left, int right) {
  int center = (left + right) / 2;
  double w;

  if (Double.parseDouble(strArray[left][0]) > Double.parseDouble(strArray[center][0]))
   swap(strArray, left, center);
```

```
 if (Double.parseDouble(strArray[left][0]) > Double.parseDouble(strArray[right][0]))
  swap(strArray, left, right);

 if (Double.parseDouble(strArray[center][0]) > Double.parseDouble(strArray[right][0]))
  swap(strArray, center, right);

 swap(strArray, center, right - 1);
 w=Double.parseDouble(strArray[right - 1][0]);
 return w;
}

public static void swap(String [][] strArray, int dex1, int dex2) {
 String [][] temp =new String [1][3];
  temp[0]= strArray[dex1];
 strArray[dex1] = strArray[dex2];
 strArray[dex2] = temp[0];
}

public static int partitionIt(String[][] strArray, int left, int right, double pivot) {
 int leftPtr = left;
 int rightPtr = right - 1;
 while (true) {
  while (Double.parseDouble(strArray[++leftPtr][0]) < pivot)
   ;
  while (Double.parseDouble(strArray[--rightPtr][0]) > pivot)
   ;
  if (leftPtr >= rightPtr)
   break;
  else
   swap(strArray, leftPtr, rightPtr);
 }
 swap(strArray, leftPtr, right - 1);
 return leftPtr;
}
```

```
public static void manualSort(String [][] strArray, int left, int right) {
  int size = right - left + 1;
  if (size <= 1)
    return;
  if (size == 2) {
    if (Double.parseDouble(strArray[left][0]) > Double.parseDouble(strArray[right][0]))
      swap(strArray, left, right);
    return;
  } else {
    if (Double.parseDouble(strArray[left][0]) > Double.parseDouble(strArray[right - 1][0]))
      swap(strArray, left, right - 1);
    if (Double.parseDouble(strArray[left][0]) > Double.parseDouble(strArray[right][0]))
      swap(strArray, left, right);
    if (Double.parseDouble(strArray[right - 1][0]) > Double.parseDouble(strArray[right][0]))
      swap(strArray, right - 1, right);
  }
 }
}
```

Class random

```
import java.util.Random;
import java.text.DecimalFormat;
import java.io.*;
import java.util.StringTokenizer;
import java.lang.*;

class random{
        static int k;

        void randommst(String [][]mstnode,String z) throws IOException
        {
                prmst create=new prmst();
                nodeSearch nodes=new nodeSearch();
```

```
String []senode=new String[mstnode.length+1];
String [] subloop=new String[3];
String [] slsearch=new String[mstnode.length+1];
System.out.println("length of slsearch ="+slsearch.length);
for (int i0=0;i0<slsearch.length;i0++)
{
        slsearch[i0]="";
}
FileCopy copy=new FileCopy();
DecimalFormat df = new DecimalFormat();
df.applyPattern("0.00");
filenode file=new filenode();
int i=mstnode.length;
Random rn=new Random();
Random generator = new Random();
random ran=new random();
int j=rn.nextInt(i);
System.out.println("j is"+j);
                double
k=Double.parseDouble(df.format(generator.nextDouble()*1001));
ran.checknumber(mstnode,j,k);
file.filenode3("mstch",mstnode,z);
subloop[0]=mstnode[j][0];
subloop[1]=mstnode[j][1];
subloop[2]=mstnode[j][2];
System.out.println("subloop[0] is"+subloop[0]);
System.out.println("subloop[0] is"+subloop[1]);
System.out.println("subloop[0] is"+subloop[2]);
ran.chdata(mstnode[j],z,mstnode.length);
ran.upmstnode(mstnode[j],mstnode,z);
nodes.keeplnode(subloop[1],mstnode,slsearch,z);
ran.chdata1(subloop,z,mstnode.length);
}
```

```
void checknumber(String [][]mstnode,int j1,double k1)
{
        random ran=new random();
        DecimalFormat df = new DecimalFormat();
        df.applyPattern("0.00");
        Random generator = new Random();
        if (Double.parseDouble(mstnode[j1][0])< k1)
        {
                mstnode[j1][0]=Double.toString(k1);
        }
        else
        {
                double
                k=Double.parseDouble(df.format(generator.nextDouble()*3001));
                ran.checknumber(mstnode,j1,k);
        }
}

void chdata(String []loop,String z,int l)
throws IOException
{
        String name,fcopy,tcopy;
        FileCopy copy=new FileCopy();
        FileCopy1 copy1=new FileCopy1();
        String [] list=new String[l+1];
        random ran=new random();
        ran.copydata(loop,loop[1],z,l);
        ran.copydata(loop,loop[2],z,l);
        for (int i=0;i<list.length;i++)
        {
                list[i]=String.valueOf(i+1);
                name=list[i];
                fcopy=name+".txt";
```

```
                tcopy=z+"/"+name+".txt";
                copy1.filecopy(fcopy,tcopy,z);
                fcopy=z+"/"+name+".txt";
                tcopy=z+"/"+name+"ch.txt";
                copy.filecopy(fcopy,tcopy,z);
        }
}

void chdata1(String []loop,String z,int l)
throws IOException
{
        String name,fcopy,tcopy;
        FileCopy1 copy1=new FileCopy1();
        String [] list=new String[l+1];
        random ran=new random();
        for (int i=0;i<list.length;i++)
        {
                list[i]=String.valueOf(i+1);
                name=list[i];
                fcopy=z+"/"+name+".txt";
                tcopy=z+"/"+name+"ch.txt";
                copy1.filecopy(fcopy,tcopy,z);
        }
        ran.copydata(loop,loop[1],z,l);
        ran.copydata(loop,loop[2],z,l);
}

void copydata(String []loop,String name,String z,int l)
 throws IOException
{
        int m=0;
        int j=0;
        String b,c,namefile;
        random ran=new random();
```

```
        namefile=name+".txt";
        String []link=new String [3];
        String [][]subfile=new String[1][3];
        FileReader fin = new FileReader(namefile);
        BufferedReader bin = new BufferedReader (fin);
        while ((b =  bin.readLine()) != null)
        {
                StringTokenizer st=new StringTokenizer(b);
                while(st.hasMoreTokens())
                {
                        c=st.nextToken();
                        link[j++]=c;
                }
                subfile[k][0]=link[0];
                subfile[k][1]=link[1];
                subfile[k++][2]=link[2];
                link[0]="";
                link[1]="";
                link[2]="";
                j=0;
        }
        k=0;
        fin.close();
        bin.close();
        ran.updata(loop,subfile,z,name);
}

void updata(String []loop,String[][] subfile,String z,String name) throws IOException
{
        quicksort sort=new quicksort();
        for (int i=0;i<subfile.length;i++)
        {
                if (subfile[i][1].equals(loop[1]) & subfile[i][2].equals(loop[2]))
```

```
                {
                        subfile[i][0]=loop[0];
                }
                else if(subfile[i][1].equals(loop[2]) & subfile[i][2].equals(loop[1]))
                {
                        subfile[i][0]=loop[0];
                }
        }
        name=name+"ch";
        sort.quickSort(subfile,z,name);
}
void upmstnode(String[]loop,String [][]mstnode,String z) throws IOException
{
        for(int i=0;i<mstnode.length;i++)
        {
                if (loop[0].equals(mstnode[i][0]) & loop[1].equals(mstnode[i][1])
                        & loop[2].equals(mstnode[i][2]) )
                        {
                                mstnode[i][0]="";
                                mstnode[i][1]="";
                                mstnode[i][2]="";
                        }
        }
        prmst mst=new prmst();
        mst.swapdata(mstnode,z);
}

}
```

Class read

```
import java.io.*;
import java.lang.*;
import java.util.StringTokenizer;
```

```
class read{

        void readfile (String namefile,String [][]mstnode,int num,int position,
                                        String z ) throws IOException {
   int j=0;
   int k=0;
   int lofsub=0;
   read Read = new read();
        String subfile[][]=new String [num][3];
   String b,c,c1;
   String link[]=new String[3];
   FileReader fin = new FileReader(namefile);
   BufferedReader bin = new BufferedReader (fin);
   for (int i1=0;i1<subfile.length;i1++)
   {
        for (int i2=0;i2<3;i2++)
        {
                subfile[i1][i2]="";
        }
   }
   while ((b =  bin.readLine()) != null) {
    StringTokenizer st=new StringTokenizer(b);
    while(st.hasMoreTokens())
                {
                        c1=st.nextToken();
                        if (j<link.length)
                        {
                        link[j++]=c1;
                        }
                        else
                        {
                                j=0;
```

```
                    }
                }
    if (k<subfile.length)
    {
                subfile[k][0]=link[0];
                subfile[k][1]=link[1];
                subfile[k++][2]=link[2];
                link[0]="";
                link[1]="";
                link[2]="";
                j=0;
    }
    else
    {
          k=0;
    }
}
        fin.close();
        bin.close();
        k=0;
        for( int i0=0;i0<subfile.length;i0++)
        {
                if(!(subfile[i0][0].equals("")))
                {
                        System.out.println("subfile["+i0+"]["+0+"]="+subfile[i0][0]+"--
                        "+subfile[i0][1]);
                        lofsub++;
                }
        }
        System.out.println("lofsub is  "+lofsub);
    Read.deletedata(mstnode,subfile,position,z,lofsub);
}
```

```
void deletedata(String [][]mstnode,String [][] subfile,int position,String z,int z1)
throws IOException
{
        System.out.println("DeleteData git"+ subfile[0][1]);
       for (int l=0;l<z1;l++)
       {
              if((subfile[l][0].equals(mstnode[position][0]) &
                     subfile[l][1].equals(mstnode[position][1]) &
                     subfile[l][2].equals(mstnode[position][2])) ||
                (subfile[l][0].equals(mstnode[position][0]) &
                 subfile[l][2].equals(mstnode[position][1]) &
                 subfile[l][1].equals(mstnode[position][2])))
                 {
                     subfile[l][0]="";
                     subfile[l][1]="";
                     subfile[l][2]="";
                 }
       }
       read Read=new read();
       Read.swapdata(subfile,z);
}
void swapdata(String [][]subfile,String z) throws IOException
{
        String []link=new String[3];
        link[0]=link[1]=link[2]="";
        System.out.println("Git subfile"+subfile[1][1]);
       for ( int i=0;i<subfile.length;i++)
       {
            if (subfile[i][0].equals("") & subfile[i][1].equals("") &
                     subfile[i][2].equals(""))
            {
                     for(int j=i+1;j<subfile.length;j++)
```

```
                {
                        if (!(subfile[j][0].equals("") & subfile[j][1].equals("") &
                                subfile[j][2].equals("")))
                                {
                                        link[0]=subfile[j][0];
                                        link[1]=subfile[j][1];
                                        link[2]=subfile[j][2];
                                        subfile[j][0]="";
                                        subfile[j][1]="";
                                        subfile[j][2]="";
                                        subfile[i][0]=link[0];
                                        subfile[i][1]=link[1];
                                        subfile[i][2]=link[2];
                                        break;
                                }
                                else if ((subfile[j][0].equals("") &
                                subfile[j][1].equals("") &
                                subfile[j][2].equals("")))
                                {
                                        break;
                                }
                }
        }
    }
        filenode file=new filenode();
        file.filenode2(subfile,z);
 }}
```

Class thesis

```
import java.io.*;
import java.lang.*;
class thesis{
```

```
static int i;

public static void main(String args[])throws IOException{
        checknode check=new checknode();
        BufferedReader in = new BufferedReader(new InputStreamReader(
    System.in));
        int i0;
        boolean respon=true;
        deletedata del=new deletedata();
        String listnode[];
        String mstnode [][];
        while (respon)
        {
            createnode create=new createnode();
            i0=create.creanode();
            System.out.println("i0 is "+i0);
            listnode=new String[i0];
            mstnode=new String[i0-1][3];
            for(int x1=0;x1<listnode.length;x1++)
        {
            listnode[x1]="";
        }
        check.checklist("1",listnode,String.valueOf(i),mstnode,0);
            for( int i1=0;i1<listnode.length;i1++)
            {
                del.delete(listnode[i1]);
            }
            i++;
            System.out.println("Please keyboard y is continue  and N is stop");
            String response = in.readLine();
            if (response.equals("y") || response.equals("Y"))
            {
                respon=true;
```

```
                }
                else
                {
                        respon=false;
                }
        }
        in.close();
    }
}
```

BIOGRAPHY

PERSONAL INFORMATION

NAME	KREADTISAK LAPPANITCHAYAKUL
NATIONALITY	THAI
DATE OF BIRTH	November 16, 1981
ADDRESS	65 Saranit road, Takhli Nakhonsawan
E-MAIL	PARADISENU@HOTMAIL.COM

STUDIES

2003 Bachelor's Degree of Computer Science, Narasuan University