Written Assignment 1

薛劭杰 1930026143

Question1.

(a) The adjacency matrix representation of this graph:

$A = [a_{ij}], a_{ij} = 0\ if\ (v_i, v_j) \in E.$

$a_{ab} = 0, a_{ac} = 0, a_{ad} = 0, a_{ae} = 0, a_{af} = 1, a_{ah} = 1, a_{ai} = 0, a_{ak} = 0, a_{as} = 1$

$a_{ba} = 0, a_{bc} = 0, a_{bd} = 0, a_{be} = 1, a_{bf} = 0, a_{bh} = 0, a_{bi} = 0, a_{bk} = 1, a_{bs} = 1$

$a_{ca} = 0, a_{cb} = 0, a_{cd} = 1, a_{ce} = 0, a_{cf} = 0, a_{ch} = 0, a_{ci} = 1, a_{ck} = 0, a_{cs} = 1$

$a_{da} = 0, a_{db} = 0, a_{dc} = 1, a_{de} = 1, a_{df} = 1, a_{dh} = 0, a_{di} = 0, a_{dk} = 0, a_{ds} = 0$

$a_{ea} = 0, a_{eb} = 1, a_{ec} = 0, a_{ed} = 1, a_{ef} = 0, a_{eh} = 1, a_{ei} = 0, a_{ek} = 0, a_{es} = 0$

$a_{fa} = 1, a_{fb} = 0, a_{fc} = 0, a_{fd} = 1, a_{fe} = 0, a_{fh} = 0, a_{fi} = 0, a_{fk} = 1, a_{fs} = 0$

$a_{ha} = 1, a_{hb} = 0, a_{hc} = 0, a_{hd} = 0, a_{he} = 1, a_{hf} = 0, a_{hi} = 1, a_{hk} = 0, a_{hs} = 0$

$a_{ia} = 0, a_{ib} = 0, a_{ic} = 1, a_{id} = 0, a_{ie} = 0, a_{if} = 0, a_{ih} = 1, a_{ik} = 1, a_{is} = 0$

$a_{ka} = 0, a_{kb} = 1, a_{kc} = 0, a_{kd} = 0, a_{ke} = 0, a_{kf} = 1, a_{kh} = 0, a_{ki} = 1, a_{ks} = 0$

$a_{sa} = 1, a_{sb} = 1, a_{sc} = 1, a_{sd} = 0, a_{se} = 0, a_{sf} = 0, a_{sh} = 0, a_{si} = 0, a_{sk} = 0$

And $(v_i, v_i) \notin E$, so the adjacency-matrix A is:

$$\begin{matrix}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{matrix}\ ]$$

(b) The adjacency list representation of this graph:

$Adj[a] = \{f, s, h\}$ $Adj[b] = \{k, s, e\}$ $Adj[c] = \{d, s, i\}$

$Adj[d] = \{f, c, e\}$ $Adj[e] = \{d, b, h\}$ $Adj[f] = \{d, a, k\}$

$Adj[h] = \{e, a, i\}$ $Adj[i] = \{k, c, h\}$ $Adj[k] = \{f, b, i\}$

$Adj[s] = \{a, c, b\}$

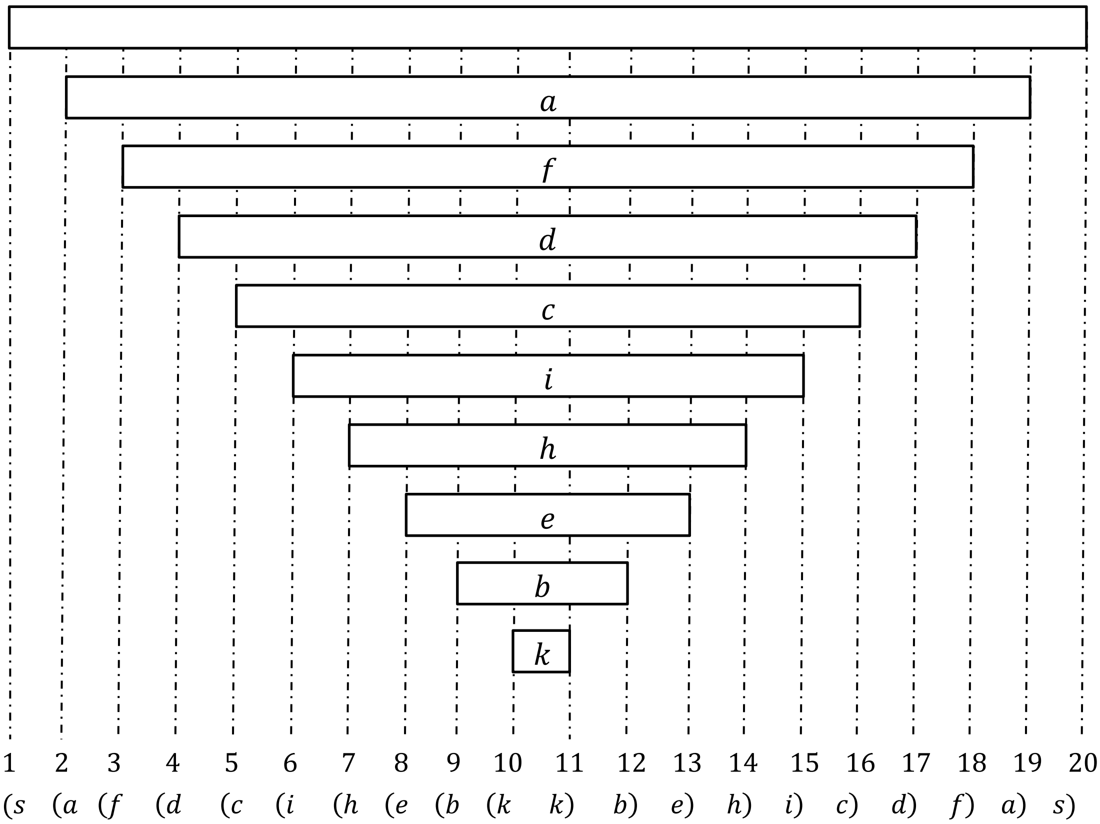
(c) $s \to a \to f \to d \to c \to i \to h \to e \to b \to k$

(d)

Vertex	Discover Time	Finish Time
a	2	19
b	9	12

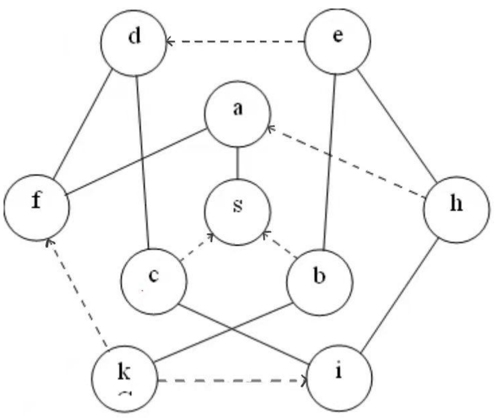
c	5	16
d	4	17
e	8	13
f	3	18
h	7	14
i	6	15
k	10	11
s	1	20

Time Stamp Structure:



(e) The back edges:

$\{b, s\}, \{c, s\}, \{e, d\}, \{k, i\}, \{k, f\}, \{h, a\}$



(f) Firstly, we should find out all the articulation points:

$k\ (10, 10 \to 3)$	$3 < 9$
$b\ (9, 9 \to 3 \to 1)$	$1 < 8$
$e\ (8, 8 \to 1)$	$1 < 7$
$h\ (7, 7 \to 2 \to 1)$	$1 < 6$
$i\ (6, 6 \to 1)$	$1 < 5$
$c(5, 5 \to 1)$	$1 < 4$
$d(4, 4 \to 1)$	$1 < 3$
$f(3, 3 \to 1)$	$1 < 2$
$a(2, 2 \to 1)$	

There are not articulation points in this graph. And the biconnected component of a graph is a maximal biconnected subgraph of the graph, it is not contained in any larger biconnected subgraph, so there is no biconnected component.

Question2:

(a) We should use the Depth-First Search algorithm.

1. Find all the Island in the grid
2. For each certain island, select one of the grids as the source vertex.
3. The zero (water) is the boundary for the DFS algorithm.
4. Use DFS with four directions to get the area of island by recursion.
5. Compare the area of all islands and select the largest value.

(b)

```c
#include <stdio.h>
#include <stdlib.h>
#include <cmath>

int MaxIsland();
int FindIsland(int row, int col);

int main(){

    int maxIsland = MaxIsland();
    printf("%d\n", maxIsland);

    return 0;

}

// Set the Island as a globle variable.
int grid[8][13] = {{0,0,1,0,0,0,0,1,0,0,0,0,0}, {0,0,0,0,0,0,0,1,1,1,0,0,0},
                   {0,1,1,0,1,0,0,0,0,0,0,0,0}, {0,1,0,0,1,1,0,0,1,0,1,0,0},
                   {0,1,0,0,1,1,0,0,1,1,1,0,0}, {0,0,0,0,0,0,0,0,0,0,1,0,0},
                   {0,0,0,0,0,0,0,1,1,1,0,0,0}, {0,0,0,0,0,0,0,1,1,0,0,0,0}};
```

```c
int FindIsland(int row, int col) {

    // The basic case is that reach the boundary, begin from zero and stop at length-1
    if ((row < 0 || row >= 8) || (col < 0 || col >= 13)) {
        return 0;
    }
    if (grid[row][col]==0){
        return 0;
    }

    grid[row][col] = 0;

    // 1 is the first point you visit
    // Then recursively visited the vertex by the four directions.
    return (1 + FindIsland(row + 1,col) + FindIsland(row, col - 1)) + FindIsland(row, col + 1) + FindIsland(row - 1, col) ;
}
```

```
int MaxIsland() {
    int area = 0;
    // Traversal all the grids in the map
    for (int row = 0; row < 8; row++) {
        for (int col = 0;col < 13; col++) {
            int temp = FindIsland(row,col);
            // If the temp is larger than the previous island area
            area = area>temp ? area:temp;
        }
    }
    return area;
}
```

```
问题    输出    终端    调试控制台

6
PS E:\Desktop_xsj\Desktop\A6_1930026143> 
```

Pseudo-code:

$MaxIsland(grid, row\_len, col\_len)$

$\quad area = 0$ $// Initialize$

$\quad for\ each\ row$

$\quad\quad for\ each\ element\ in\ a\ row:$

$\quad\quad\quad new\_area = JudgeIsland(i, j)$

$\quad\quad\quad area = \max(area, new\_area)$

$\quad\quad end$

$\quad end$

$\quad return\ area$

$JudgeIsland(grid, i, j, row\_len, col\_len)$

$\quad if\ i < 0\ or\ j < 0\ do$

$\quad\quad return\ 0$

$\quad if\ i > row\_len\ \ or\ j > col\_len:$

$\quad\quad return\ 0$

$\quad if\ grid[i][j]\ not\ equal\ to\ 1\ do$

$\quad\quad return\ 0$

// Recurse in all four directions

$\quad return\ 1 + JudgeIsland(grid, i-1, j, row\_len, col\_len)$

$\quad\quad + JudgeIsland(grid, i+1, j, row\_len, col\_len)$

$\quad\quad + JudgeIsland(grid, i, j-1, row\_len, col\_len)$

$\quad\quad + JudgeIsland(grid, i, j+1, row\_len, col\_len)$

(c) The code in question (b) is required to go through all elements of the grid. Whether it is an island or water, all of them will get into the recursion and break at the zero one and each them is visited one times. Therefore, the running time of the algorithm is $O(row\_len * col\_len)$, row_len is the length of row of the grid and col_len is the length of column of the grid.

Question3.

```java
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;


public class WaysOfOperation {
    public static void main(String[] args) {
        WaysOfOperation test = new WaysOfOperation();
        System.out.println(test.TransToArrayList("8+6+7*5"));
    }
    List<String> lst = new ArrayList<>();

    public List<Integer> TransToArrayList(String str) {
        List<String> list= Stream.iterate(0, n -> ++n).limit(str.length())
                .map(n -> "" + str.charAt(n))
                .collect(Collectors.toList());
        for(String s:list){
            System.out.println(s);
        }
        for(int i = 0; i< list.size(); i++) {
            lst.add(list.get(i));
        }
        System.out.println(lst);
        return CalculateWays(0, lst.size()-1);
    }

    public List<Integer> CalculateWays(int left, int right){
        List<Integer> num = new ArrayList<>();
        // transform the char to the integer type
        if(left == right){
            num.add(Integer.parseInt(lst.get(left)));
            return num;
        }
        for(int i = left+1; i<right; i += 2){
            List<Integer> new_left = CalculateWays(left, i-1), new_right = CalculateWays(i+1, right);
            for(int x: new_left)
                for(int y: new_right){
                    if("-".equals(lst.get(i))) num.add(x-y);
                    else if("+".equals(lst.get(i)))
                        num.add(x+y);
                    else if("*".equals(lst.get(i)))
                        num.add(x*y);
                    else
                        System.out.println("Please enter the right operation!");
                }
        }
        return num;
    }
}
```

```
F:\Java\JDK_1.8\bin\java.exe ...
[8, +, 6, +, 7, *, 5]
[49, 73, 49, 105, 105]

Process finished with exit code 0
```

By the induction:

$f(2) = f(1) + f(1)$

$f(3) = f(1) + f(2) + f(2) + f(1)$

$f(n) = f(1) + f(2) + \dots + f(n) + f(n) + \dots + f(2) + f(1)$

So $T(n) = T(1)T(n-1) + T(2)T(n-2) + \cdots + T(n-1)T(1)$

The number is the Catalan number the $T(n) = \frac{4^n}{n^{\frac{2}{3}}\sqrt{\pi}}$

So the running time of this algorithm is $\theta(\frac{4^n}{n^{\frac{2}{3}}\sqrt{\pi}})$

Question4.

(a) Step0:

V	s	a	b	c	d	e	f
$Key[v]$	0	∞	∞	∞	∞	∞	∞
$Pred[v]$	NIL						
$Color[v]$	W	W	W	W	W	W	W

Step1:

V	s	a	b	c	d	e	f
$Key[v]$	0	5	6	7	∞	∞	∞
$Pred[v]$	NIL	s	s	s			
$Color[v]$	B	W	W	W	W	W	W

Step2:

V	s	a	b	c	d	e	f
$Key[v]$	0	5	6	1	∞	∞	∞
$Pred[v]$	NIL	s	s	a			
$Color[v]$	B	B	W	W	W	W	W

Step3:

V	s	a	b	c	d	e	f
$Key[v]$	0	5	6	1	2	9	4
$Pred[v]$	NIL	s	s	a	c	c	c
$Color[v]$	B	B	W	B	W	W	W

Step4:

V	s	a	b	c	d	e	f
$Key[v]$	0	5	6	1	2	9	3
$Pred[v]$	NIL	s	s	a	c	c	d
$Color[v]$	B	B	W	B	B	W	W

Step5:

V	s	a	b	c	d	e	f
$Key[v]$	0	5	6	1	2	9	3
$Pred[v]$	NIL	s	s	a	c	c	d

$Color[v]$	B	B	W	B	B	W	B

Step6:

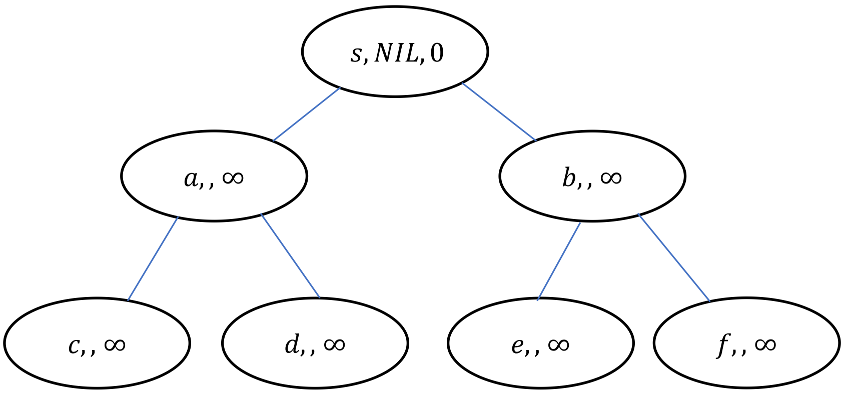
V	s	a	b	c	d	e	f
$Key[v]$	0	5	6	1	2	9	3
$Pred[v]$	NIL	s	s	a	c	c	d
$Color[v]$	B	B	B	B	B	W	B

Step7:

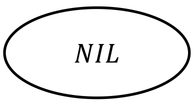
V	s	a	b	c	d	e	f
$Key[v]$	0	5	6	1	2	9	3
$Pred[v]$	NIL	s	s	a	c	c	d
$Color[v]$	B	B	B	B	B	B	B

Q:

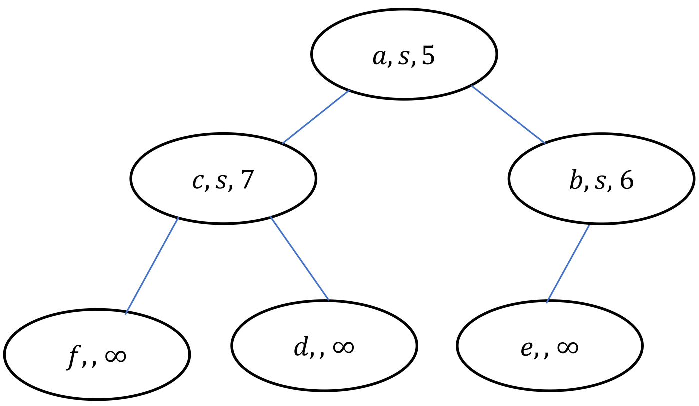
Step0:



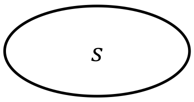
MST:



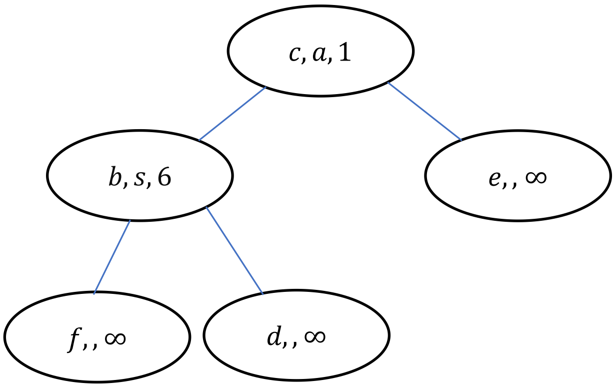
Step1:



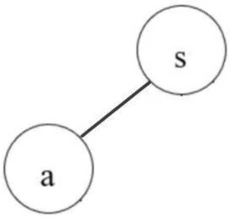
MST



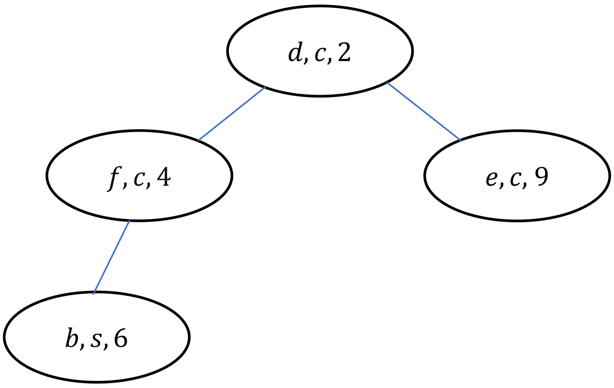
Step2:



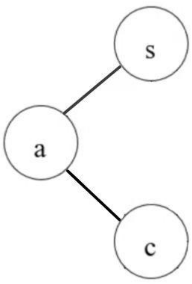
MST:



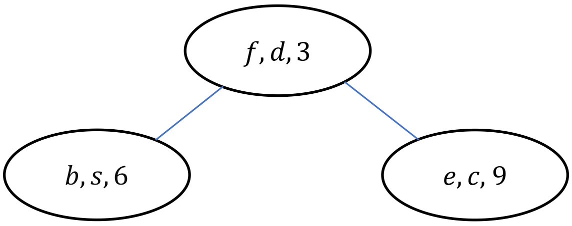
Step3:



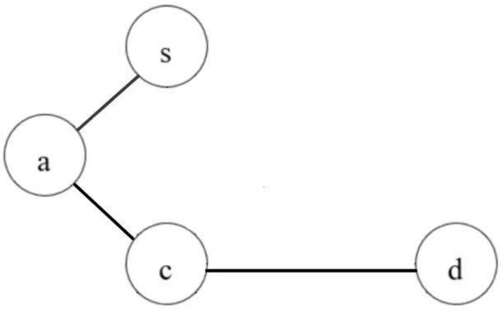
MST:



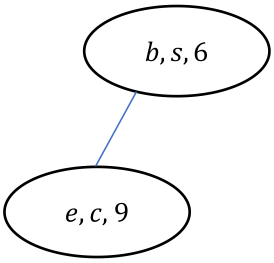
Step4:



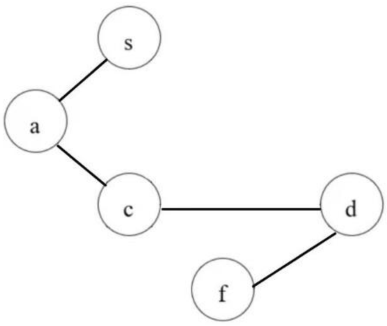
MST:



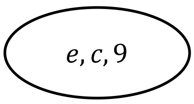
Step5:



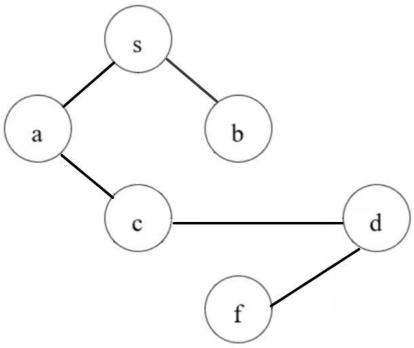
MST:



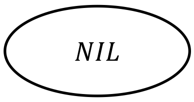
Step6:



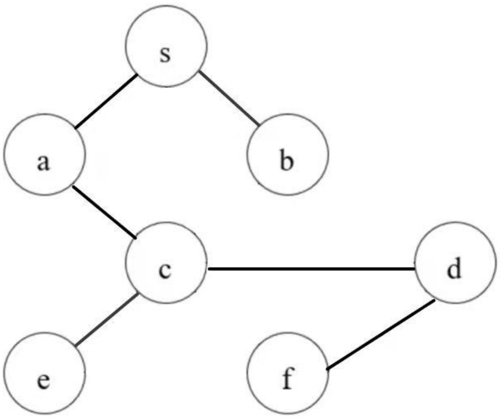
MST:



Step7:



MST:



(b) Initially, sort the value for all edges by ascending order.

$\{ac\} = 1$

$\{cd\} = 2$

$\{df\} = 3$

$\{cf\} = 4$

$\{sa\} = 5$

$\{sb\} = 6$

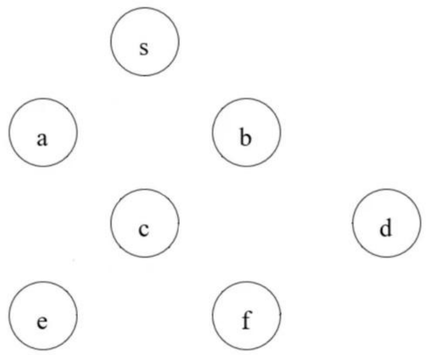
$\{sc\} = 7$

$\{ce\} = 9$

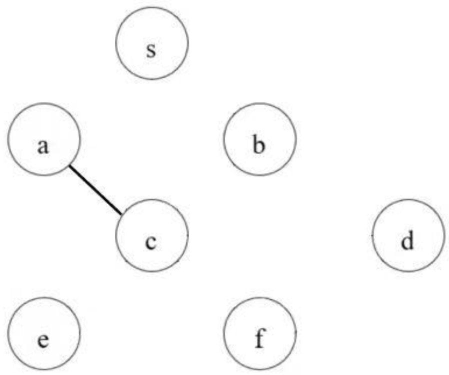
$\{bc\} = 10$

$\{ef\} = 11$

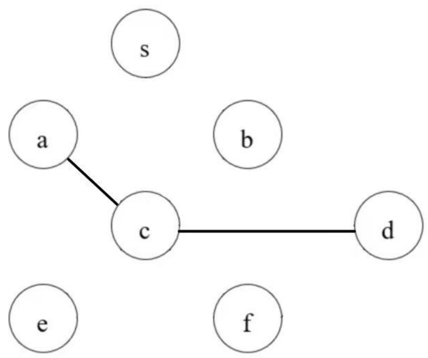
Step0:



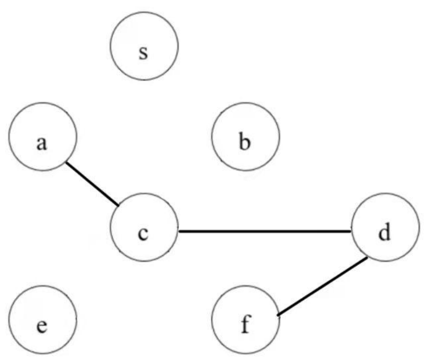
Step1:



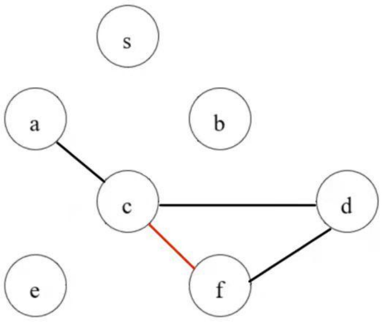
Step2:

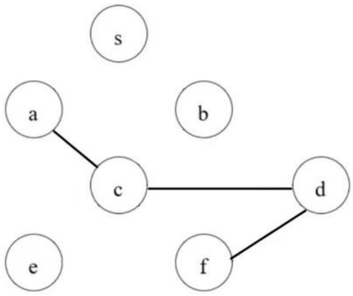


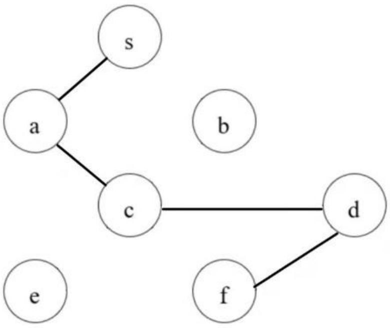
Step3:



Step4:

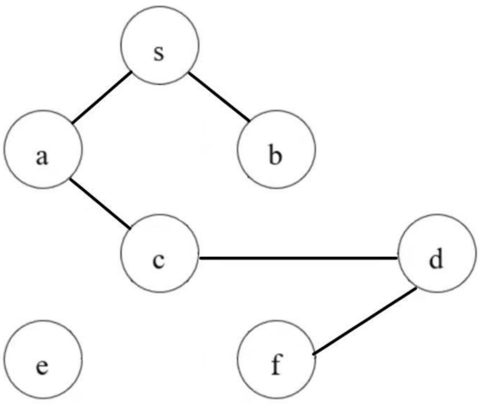


→


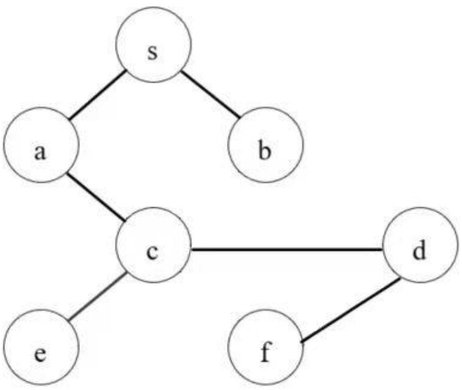


→

Step5:



Step6:



(c) Prim Algorithm (b) and Kruskal's Algorithm (c) get the same MST. Every edge in a graph has a unique weight, and the graph have a unique MST.

Proof: Assume it have two different MST and we set them T1 and T2. Let the e_i be the edge in the T1 but T2 is not contain, the $e_i^{'}$ be the edge in the T2 but T1 is not contain. Then add $e_i^{'}$ to T1 and it must generate a circle in the graph. Then we can remove one edge in the circle to generate a new spanning tree. However, the weights of all the edges are distinct. If we remove the non-$e_i^{'}$ edge e_j. The weight of e_j must be different from $e_i^{'}$. We know that T1 is a MST of the graph, so the sum weight of new spanning tree must be greater than T1 wherever the $e_i^{'}$ to be, which means that T2 is impossible to be another MST of the graph. It is contradictory and the proposition we prove is correct.

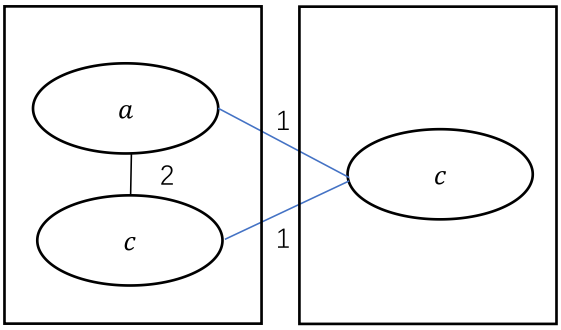
Q5.

(a) Proof: If for every cut of the graph, there is a unique light edge crossing the cut, then the graph has a unique minimum spanning tree.

We can prove it by contradiction, we can set that there are two different MST, so there must be an edge (cut) that equal to each other.

We set there is a cut (e_i, e_j), divide the graph into two trees T_1 and T_2. Remove (e_i, e_j) and let the (e_a, e_b) be the light edge crossing the cut to correct the two part-trees again. We know that (e_a, e_b) is not equal to (e_i, e_j), then the (e_a, e_b) is light edge, the sum weight of the graph with (e_a, e_b) is smaller than that with (e_i, e_j). However, the tree with (e_i, e_j) is the MST, so it is contradictory.

Counter Example:



(b) The following algorithm can find the minimum spanning tree for a graph. Since the remove the edge in a circle is the biggest one. Detail: Referring to Kruskal's Algorithm, followed by adding the edges, when it comes to form a circle, it will remove the edge of the circle which is the latest one, and the latest one is also the largest one. Now without sorting, followed by adding the edges randomly, when it forms a circle, we move the largest one so that can make the weight of left part must be the smaller. So, the following algorithm is just different from Kruskal's Algorithm in order of connections, the result is same, can find the minimum spanning tree for a graph.

Q6.

Ideas: Use MST to solve this problem.

Analysis:

1. Each cost of pipes C_{ij} may be different, just like weight of edges.

2. Each house can pass as a vertex in a graph
3. We should pay attention to the cost of well (different part from MST)
4. Find the minimum cost of pipe
5. Use queue structure to find the lightest edge.

Steps:

1) Connect the all the house and find the minimum cost to connect the all houses.
2) For each house, we construct a well, Wi. Then we find the minimum Wi.
3) Sum the all cost of pipes and the cost of well.

pseudo-code:

$MinCost(V, w, p)$

$// \; w \;$ **is well and p is pipe(weight)**

$\quad foreach\; u \in V\; do$

$\qquad //\;$ **inistalize**

$\qquad key[u] = positive\; infinite$

$\qquad color[u] = W;$

$\quad end$

$\quad Key[r] = 0$

$\quad pred[r] = NIL$

$\quad Q = new\; PriQueue(V)$

$\quad While\; Q\; is\; nonempty\; do$

$\qquad //$ **until all house in the minimum spannning tree**

$\qquad U = Q.extraxtMin();$

$\qquad foreach\; v \in adj[u]\; do$

$\qquad\quad if\; (color[v] = W)\&\&(p[u,v] < key[v])\; then$

$\qquad\qquad key[v] = p[u,v]; \quad //$ **new lightest edge**

$\qquad\qquad Q.decreaseKey(v, key[v]);$

$\qquad\qquad pred[v] = u;$

$\qquad\quad end$

$\qquad end$

$\qquad color[u] = B;$

$\quad end$

$\quad //$ **Find all the house's well cost**

$\quad well\_cost = min\; (the\; elements\; of\; the\; well\; cost)$

$\quad return\; well\_cost + Sum(all\; the\; (pipe\; cost)weight\; in\; p)$

end