Erdos Number

Paul Erdos(保罗·厄多斯,1913-1996) is a legendary mathematician of the 20th century. Famous for his eccentric living style, Erdos was one of the most prolific publishers of papers in mathematical history, having written around 1500 mathematical articles, mostly with co-authors. Erdos spent most of his lifetime traveling between scientific conferences and visiting homes of colleagues all over the world, with most of his belongings in a suitcase. He would typically show up at a colleague's doorstep and announce "my brain is open", staying long enough to collaborate on a few papers before moving on a few days later.

In many cases, he would ask the current collaborator whom he should visit next. Because of his prolific output, friends created the Erdos number as a humorous tribute, which has been a part of the folklore of mathematicians throughout the world for many years. Erdos himself is assigned the Erdos number of 0.

Those who have co-written a paper with Erdos have an Erdos number of 1. In turn, authors who have collaborated with those whose Erdos number is 1(but not with Erdos himself) have an Erdos number of 2. In general, if the lowest Erdos number of the cowriters of an author is S, then his Erdos number is S + 1. A person with no such coauthorship chain connecting to Erdos has an infinite Erdos number.

For example, in the picture above, the lady collaborates with Erdos on one paper, and then with the man on the right on another, so this man is given an Erdos number of 2(assuming he has never collaborated with Erdos himself).

The interesting thing with Erdos number is that about 90% of the world's active mathematicians have an Erdos number smaller than 8, and many mathematicians have an Erdos number surprisingly smaller than they expect. For example, Andrew Wiles, who have proved the Fermat's Last Theorem and whose research area is very different from that of Erdos, has an Erdos number of only 3. All Fields prize winners have an Erdos number at most 5 and all Wolf prize winners' Erdos numbers do not exceed 6. In addition, a very large number of non-mathematicians also have finite Erdos numbers. For example, Albert Einstein has an Erdos number of 2 and Bill Gates has an Erdos number of 4.

Your task is, given a list of co-writing activities of authors, to answer a series of queries according to the list, which have the form: "What is the Erdos number of author A?"

Standard Input

The first line of the input file is an integer N, number of co-authoring activities. The next N lines each contain two names, separated with spaces, meaning these two authors have co-written a paper. On the next line is an integer M, number of queries. The next M lines each contain the name of an author.

Constraints: $N \leq 50000$, $M \leq 50000$

Notes

- Names contain letters('A'-'Z', 'a'-'z') and '.' only and don't contain white spaces.
- Each name has at most 20 characters.
- Names are case-sensitive, e.g. Erdos is not the same person as erdos.
- Authors with the same name should be considered the same person.
- Paul Erdos himself is represented with exactly the name "Erdos".
- The same co-authoring activity is listed at most once.
- No author will be co-authoring with himself.

- **Standard Output**

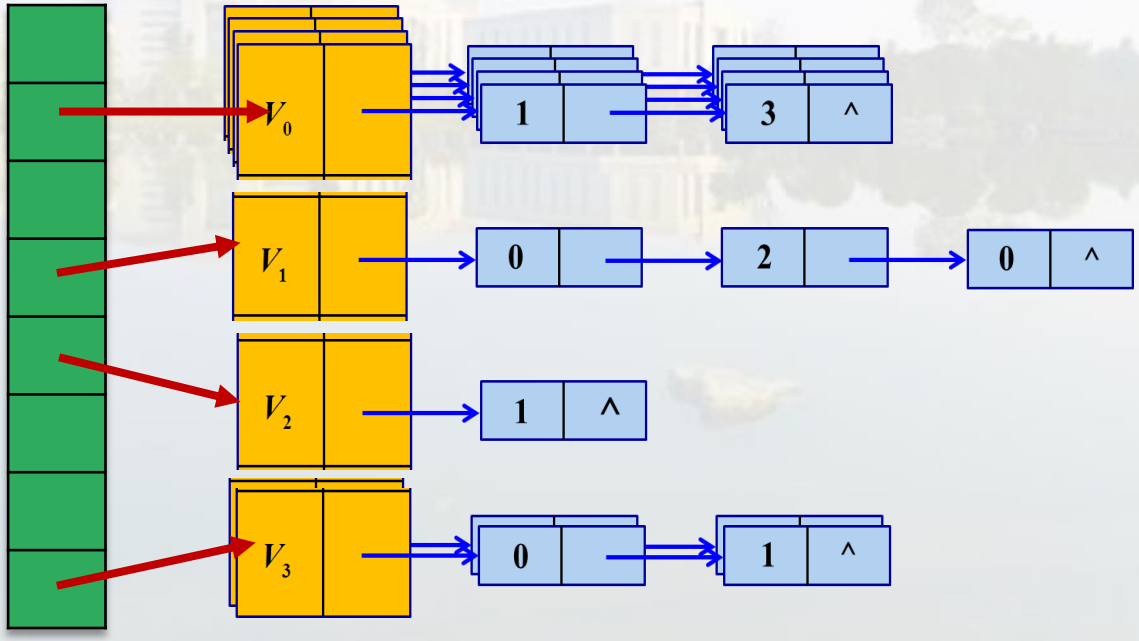
For each query, output the corresponding author's Erdos number on a line. If the author doesn't have a finite Erdos number, output "infinity" instead(quotes for clarity only).

- **Samples**

Input	Output
7 Erdos Andrew.Odlyzko Andrew.Odlyzko Chris.M.Skinner Chris.M.Skinner Andrew.Wiles Erdos Pavol.Hell Pavol.Hell Xiao.Tie.Deng Xiao.Tie.Deng Chris.Papadimitriou Chris.Papadimitriou Bill.Gates 3 Andrew.Wiles Bill.Gates Albert.Einstein	3 4 infinity

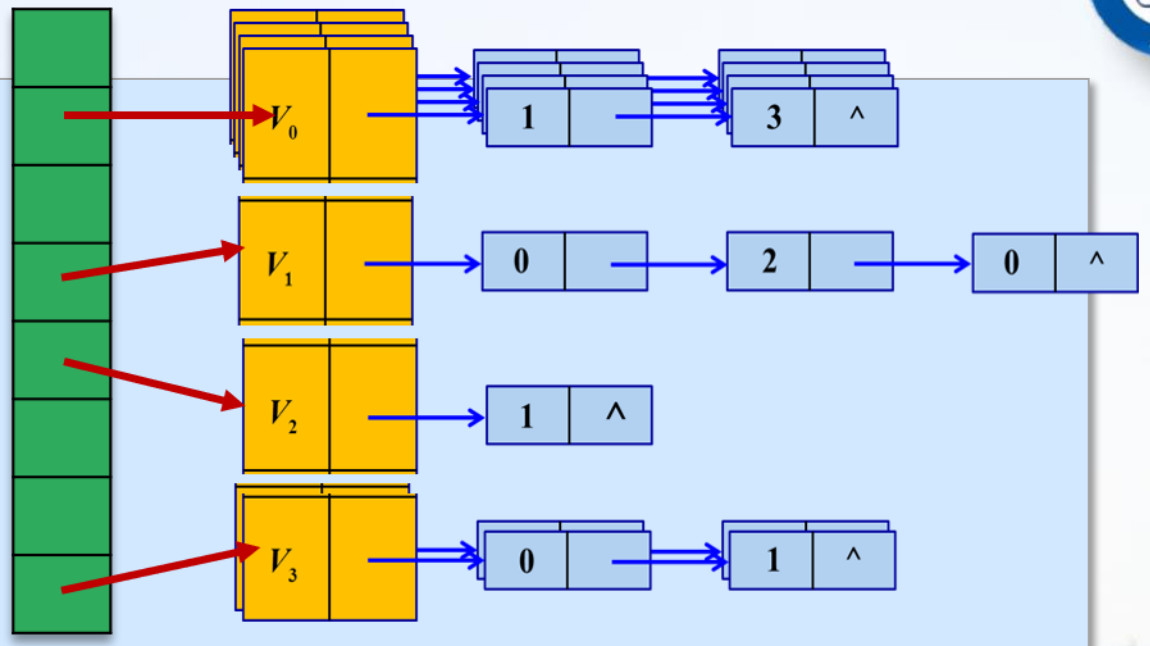
题目分析

- 先建立合作图的邻接表, 两个节点有边当且仅当这两个作者合作, 然后进行遍历操作.
- 以Erdos为起始点直接BFS扩展一棵宽度优先树.
- 每个节点在树中到Erdos(树根)的距离就是它的Erdos数.
- 陷阱: 输入的合作关系里边可能没有Erdos这个人, 但是查询中可能会有Erdos, 这时应当输出0而不是infinity.
- 为了迅速的找到Erdos, 可构造经典的字符串Hash函数实现.

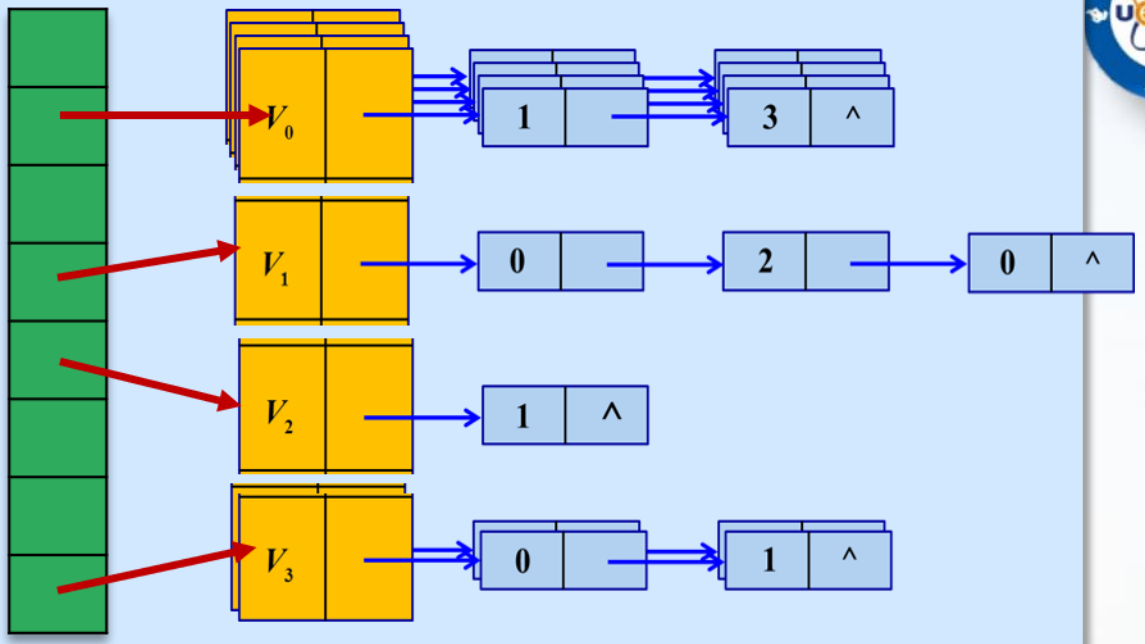


参考程序

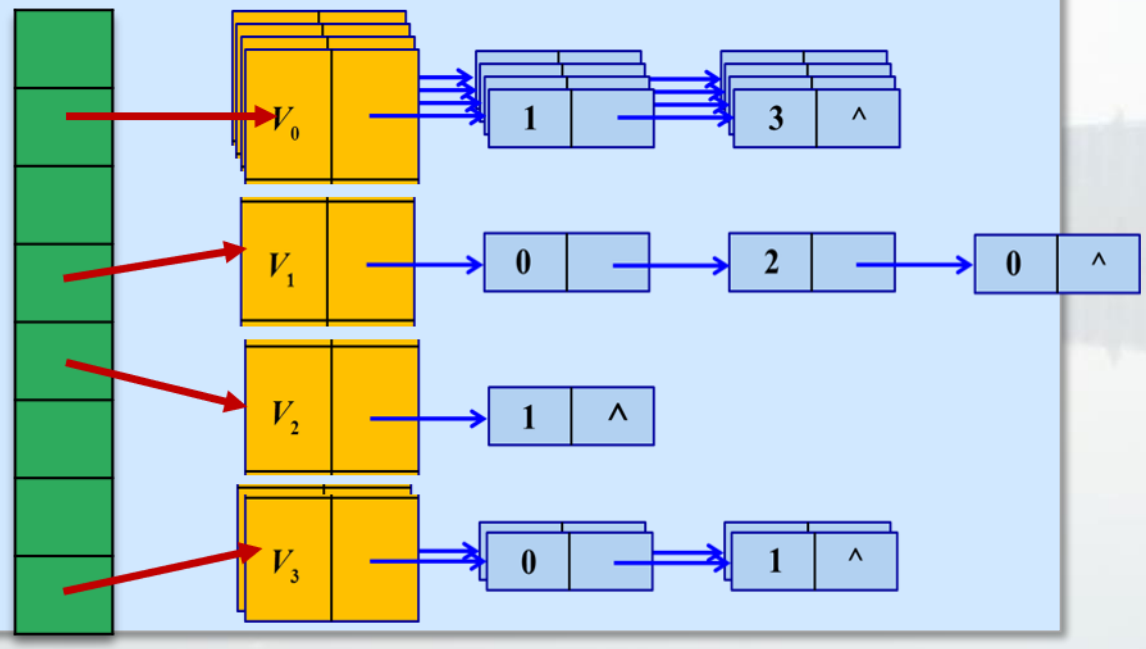
```
#include  <list>
#include  <queue>
#include  <iostream>
const unsigned  MAX_LEN  = 20;
const unsigned  HASH_SCALE = 400000;
struct Node
{
    char  name[MAX_LEN + 1];
    int   dist;
    Node *next; //记录冲突结点
    std::list<Node*>  neib;//记录合作者结点
    Node(const char _name[], Node *_next):dist(-1), next(_next)
    {
        strcpy(name, _name);
    }
};
```



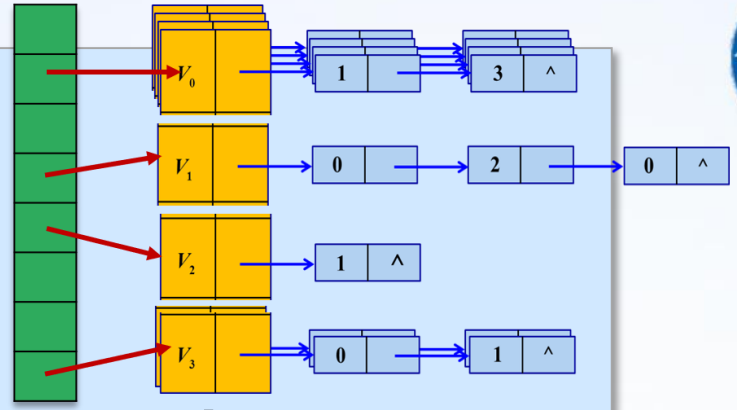
```
Node*    hashTable[HASH_SCALE];
unsigned ELFhash(const char *key)
{
    unsigned h = 0;
    while (*key != '\0')
    {
        h = (h << 4) + *key++;
        unsigned g = h & 0xf0000000;
        if (g)
            h ^= g >> 24;
        h &= ~g;
    }
    return h % HASH_SCALE;
}
Node* enhash(const char name[])
{
    unsigned h = ELFhash(name);
    Node    *p = hashTable[h];
    while (p != NULL && strcmp(p->name, name) != 0)
        p = p->next; //冲突处理，检查是否有相同名字
    return p ?  p : hashTable[h] = new Node(name, hashTable[h]);
}
```



```
void get_coauthor_activities()
{  //读取数据，添加合作者
    int N;
    scanf("%d", &N);

    while (N--)
    {
        char name1[MAX_LEN + 1], name2[MAX_LEN + 1];
        Node *p, *q;

        scanf("%s %s", name1, name2);
        p = enhash(name1);
        q = enhash(name2);

        p->neib.push_back(q);
        q->neib.push_back(p);
    }
}
```



```
void bfs()
{
   std::queue<Node*> Q;
   Q.push( enhash("Erdos") );
   Q.front()->dist = 0;
   while ( !Q.empty() )
   {
      typedef std::list<Node*>::iterator  IP;
      int d = Q.front()->dist + 1;
      std::list<Node*>&  L = Q.front()->neib;
      Q.pop();
      for (IP p = L.begin(); p != L.end(); p++)
      {
         if ( (*p)->dist == -1 )
         {
            Q.push( *p );
            (*p)->dist = d;
         }
      }
   }
}
```

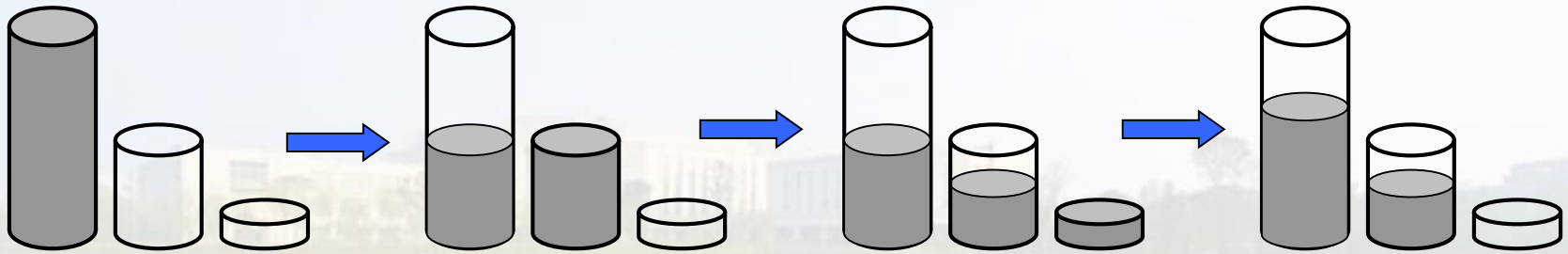



```
void do_query()
{
   int M;
   scanf("%d", &M);
   while (M--)
   {
      char name[MAX_LEN + 1];
      int  d;
      scanf("%s", name);
      d = enhash(name)->dist;
      if (d != -1)    printf("%d\n", d);
      else            puts("infinity");
   }
}
int main()
{
   get_coauthor_activities();
   bfs();
   do_query();
   return 0;
}
```

思考题：倒水问题

有装满水的6升的杯子、空的3升杯子和1升杯子，3个杯子中都没有刻度。在不使用其他道具的情况下，是否可以量出4升的水呢?

方法如下图所示：

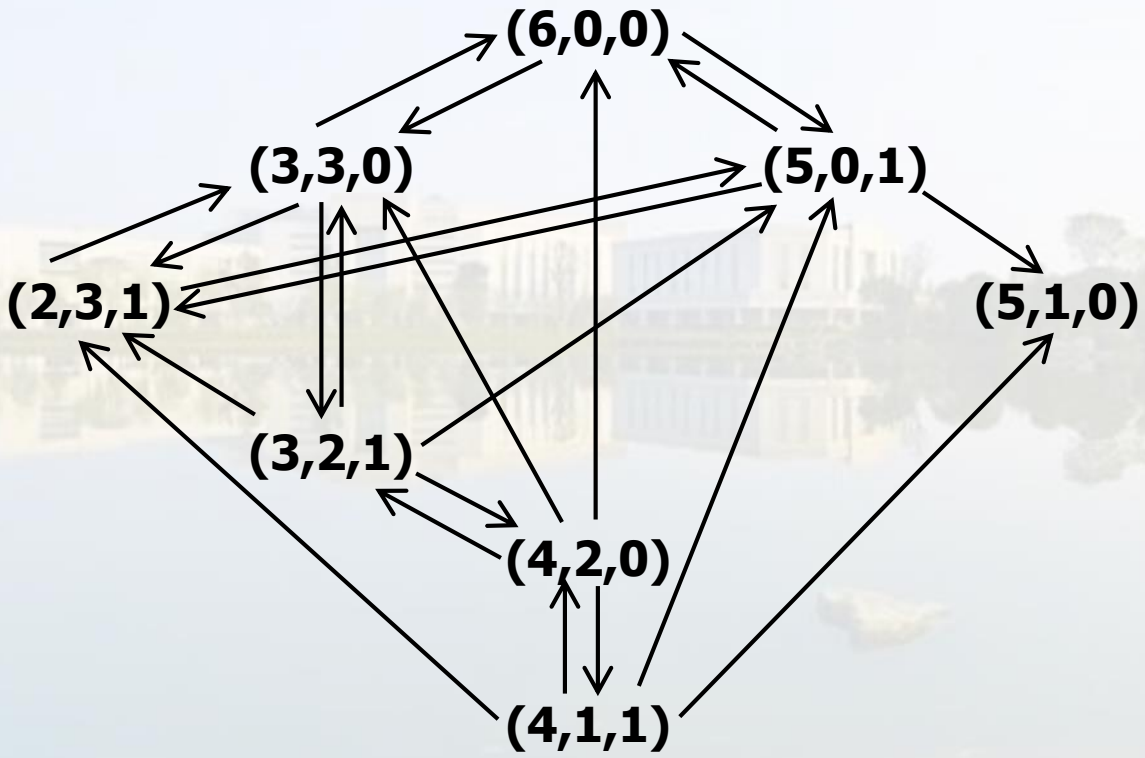


即，倒水问题的一种方法为：

(6,0,0) → (3,3,0) → (3,2,1) → (4,2,0)

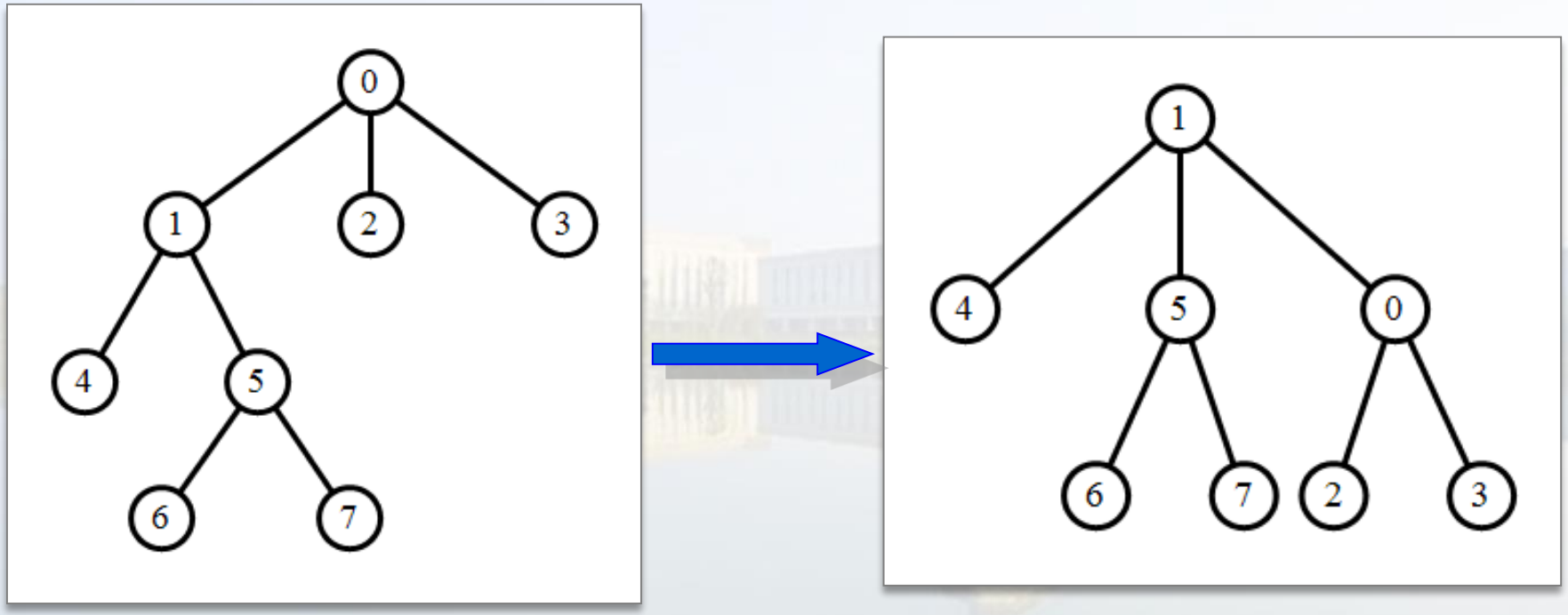
注意: 由于没有刻度, 用杯子x给杯子y倒水时必须一直把杯子y倒满或者把杯子x倒空, 而不能中途停止.

你的任务是解决一般性的问题: 设大、中、小3个杯子的容量分别为a,b,c, 最初只有大杯子装满水, 其他两个杯子为空. 最少需要多少步才能让某一个杯子中的水有x升呢? 你需要打印出每步操作后各个杯子中的水量(0<c<b<a<1000).



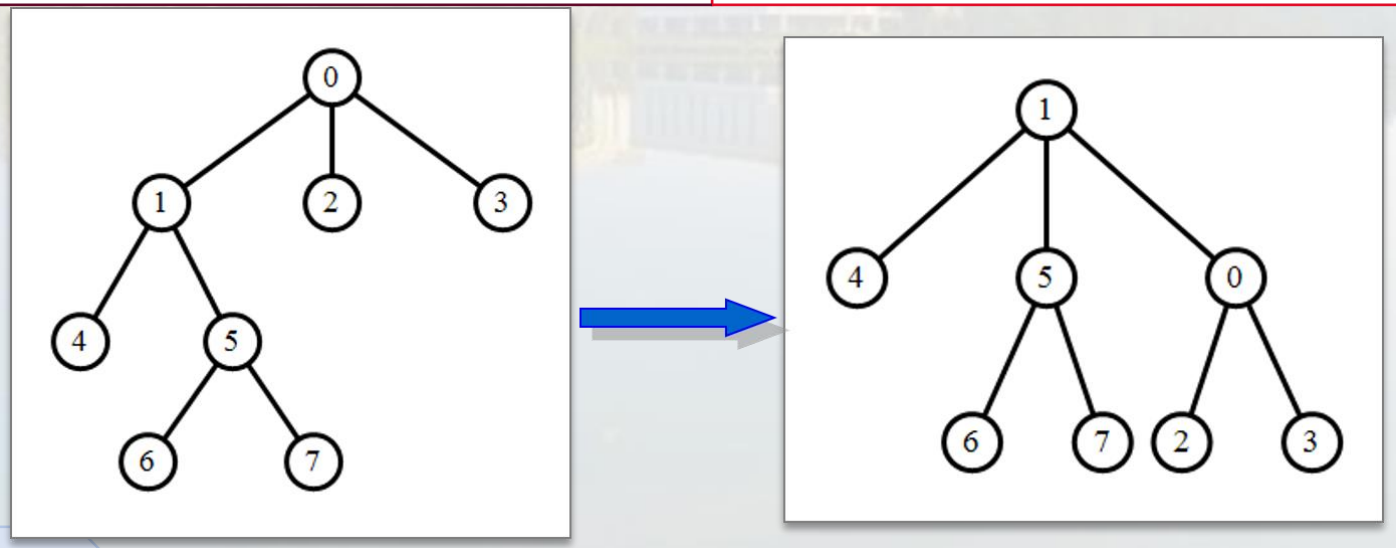
四、无根树转有根树

输入一个n个结点的无根树的各条边，并指定一个根结点，要求把该树转化为有根树，输出各个结点的父结点编号。n<=10^6。



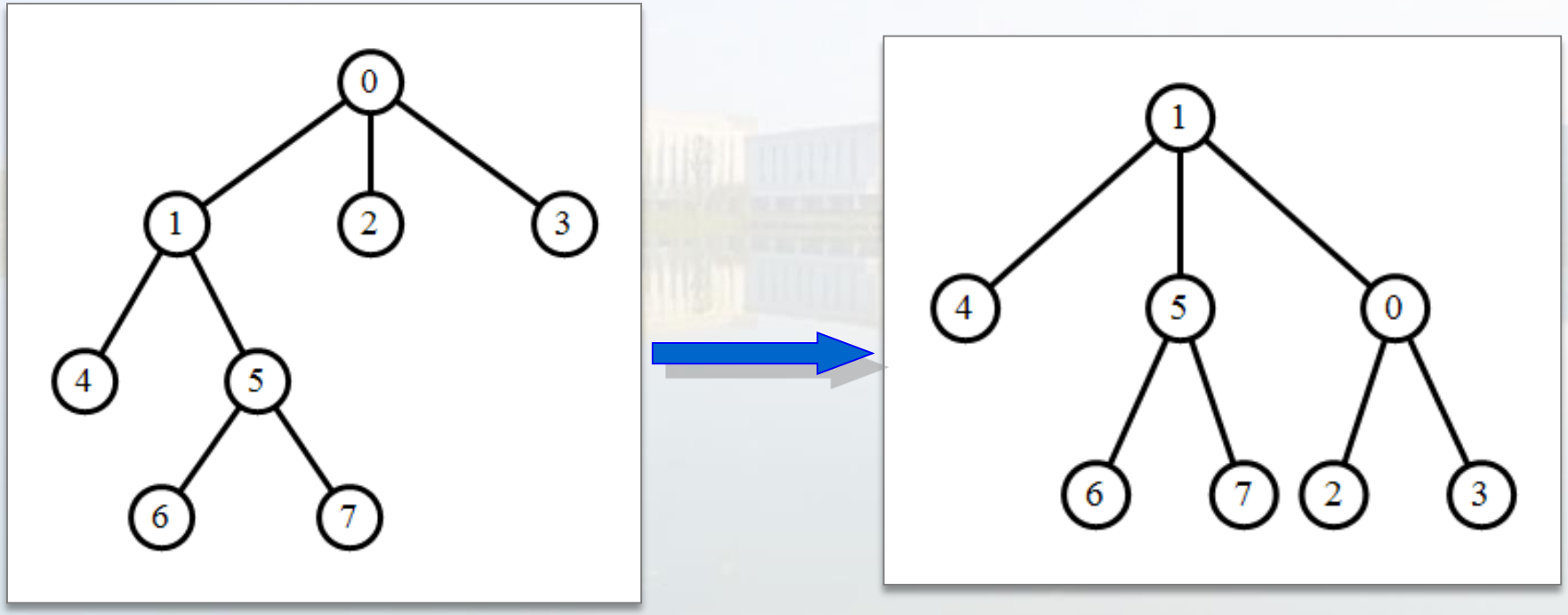
- **Samples**

Input	Output
8 1 0 1 0 2 0 3 1 4 1 5 5 6 5 7	1 -1 0 0 1 1 5 5



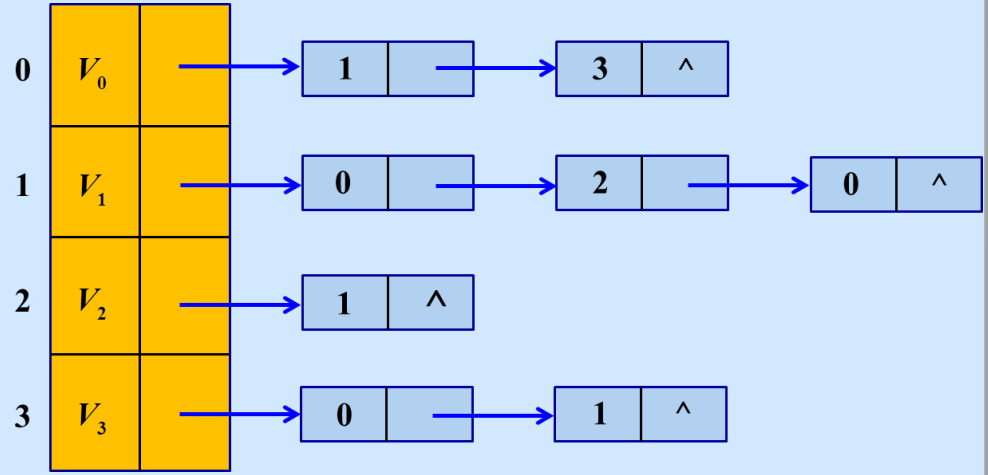
题目分析

- 先建立图(无向树)的邻接表;
- 然后从指定点深度优先搜索即可;
- 由于原图是树, 所以结果具有唯一性.

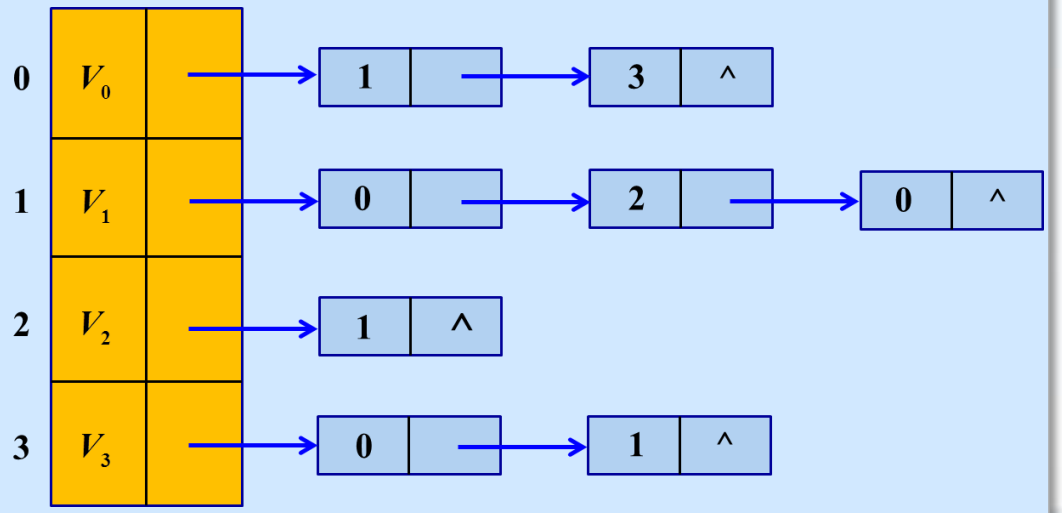


参考程序

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define N 1000002
struct edge
{
    int node;
    struct edge* next;
} mynode[N], myedge[2 * N];
int n, root, father[N];
void dfs(int u);
int main()
{
    while(scanf("%d%d", &n, &root) == 2)
    {
        int i, k = 0, s, t;
        memset(father, -1, sizeof(father));
        memset(mynode, 0, sizeof(mynode));
        for(i = 0; i < n - 1; i++) //建立邻接表
        {
            scanf("%d%d", &s, &t);
            myedge[k].node = t;
            myedge[k].next = mynode[s].next;
            mynode[s].next = &myedge[k];
            k++;
```



```
            myedge[k].node = s;
            myedge[k].next = mynode[t].next;
            mynode[t].next = &myedge[k];
            k++;
        }
        dfs(root);
        for(i = 0; i < n; i++)
            printf("%d ", father[i]);
        printf("\n");
    }
    return 0;
}
void dfs(int u)
{
    struct edge *p = mynode[u].next;
    while(p != NULL)
    {
        int v = p->node;
        if(v != father[u])
        {
            father[v] = u;
            dfs(v);
        }
        p = p->next;
    }
}
```



五、最小生成树

设$G=<V,E>$是连通的赋权图,T是G的一棵生成树,T的每个树枝所赋权值之和称为T的权,记为$W(T)$.G中具有最小权的生成树称为G的最小生成树.

一个无向图的生成树不是唯一的,除非该图就是树;同样的,一个赋权图的最小生成树也不一定是惟一的.

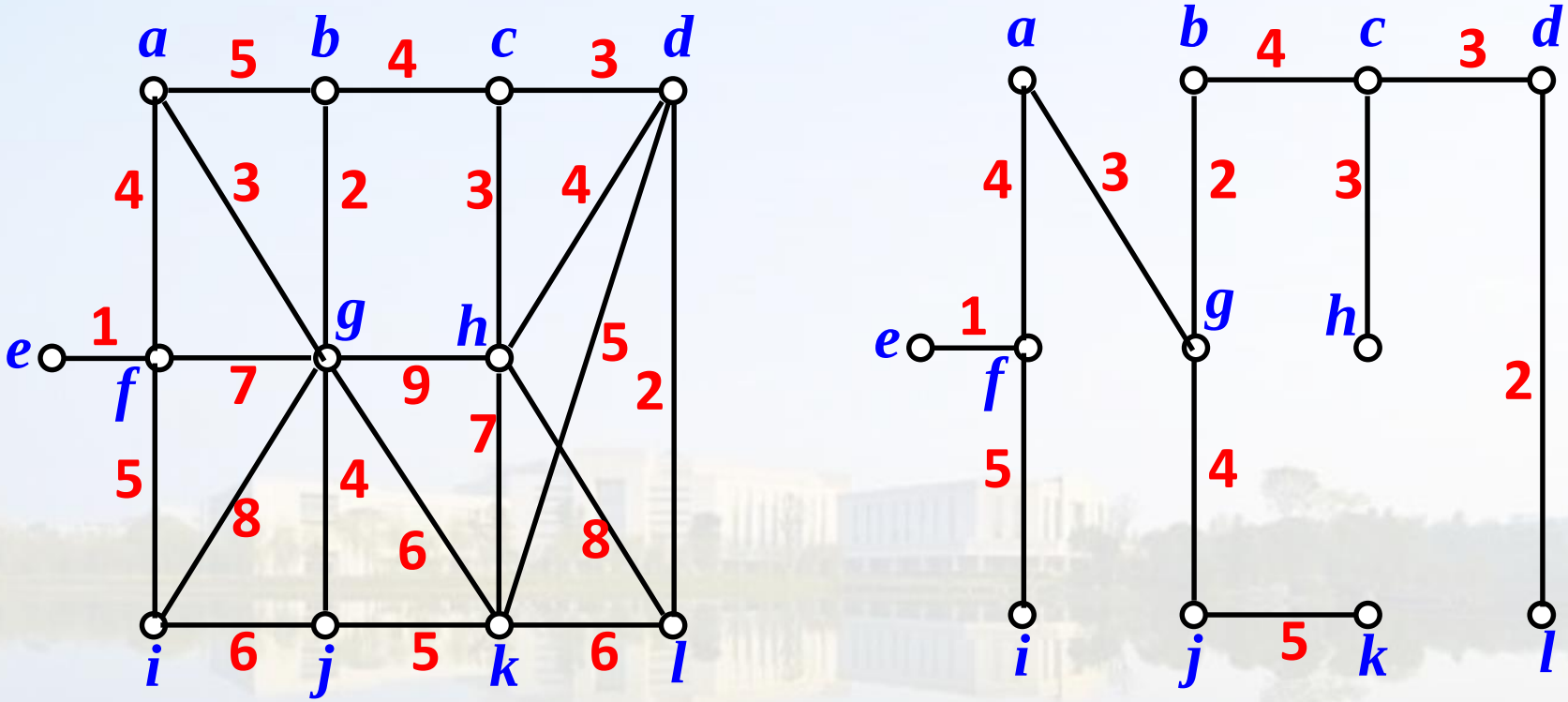
寻找最小生成树的常用算法是Kruskal算法和Prim算法.

1、Kruskal算法

Kruskal算法是Joseph Kruskal在1956年发表. 用来寻找赋权图的最小生成树. 其算法流程如下:

1. 在G中选取最小权边e_1,置$i=1$;
2. 当$i=n-1$时,结束,否则转3;
3. 设已选取的边为$e_1,e_2,\ldots,e_i$,在G中选取不同于$e_1,e_2,\ldots,e_i$的边e_{i+1},使$\{e_1,e_2,\ldots,e_i,e_{i+1}\}$中无回路且$e_{i+1}$是满足此条件的最小权边.
4. 置$i=i+1$,转2.

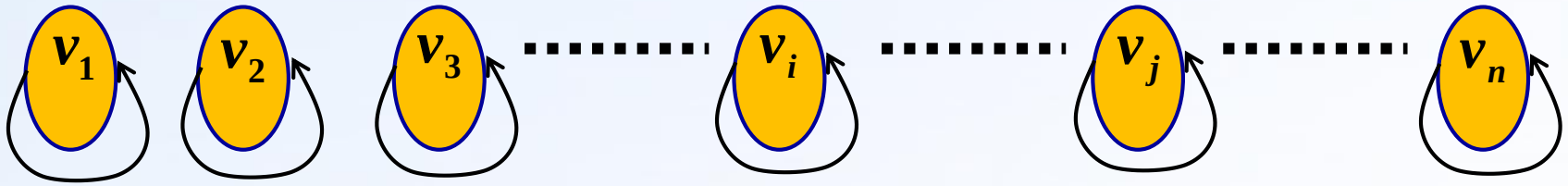
注意Kruskal算法更适合找稀疏图的最小生成树.



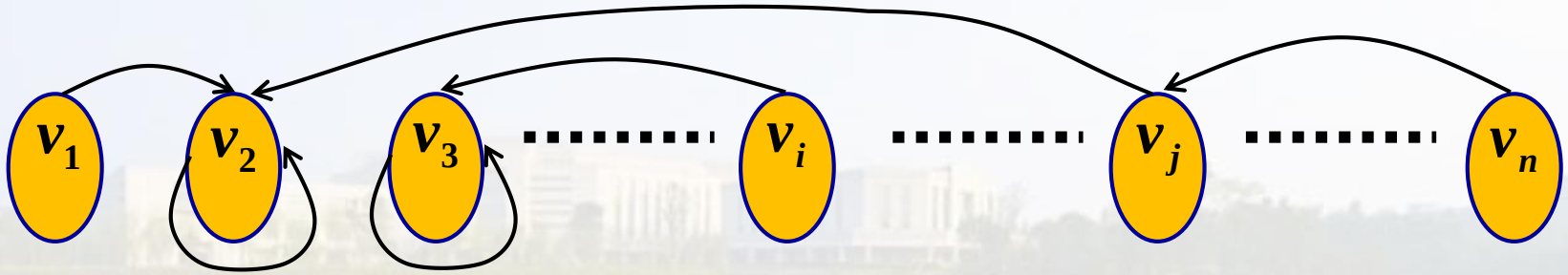
Kruskal算法　w(T) = 36

Kruskal算法实现要点

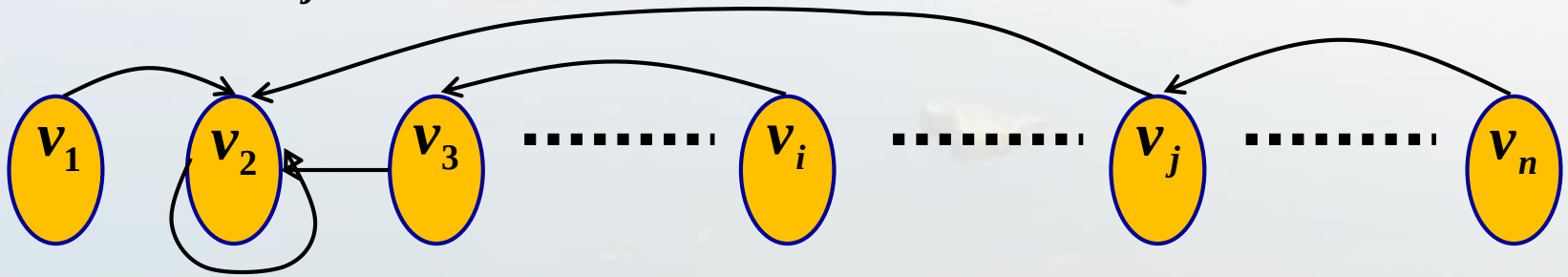
- 并查集的初始状态



- 添加部分边后的并查集

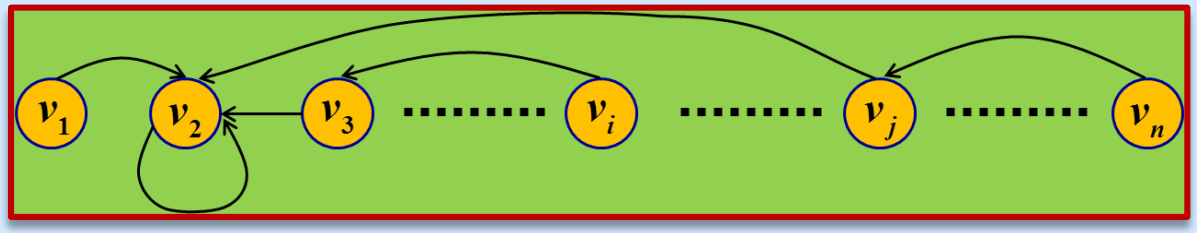


- 添加边(v_i, v_j)后的并查集

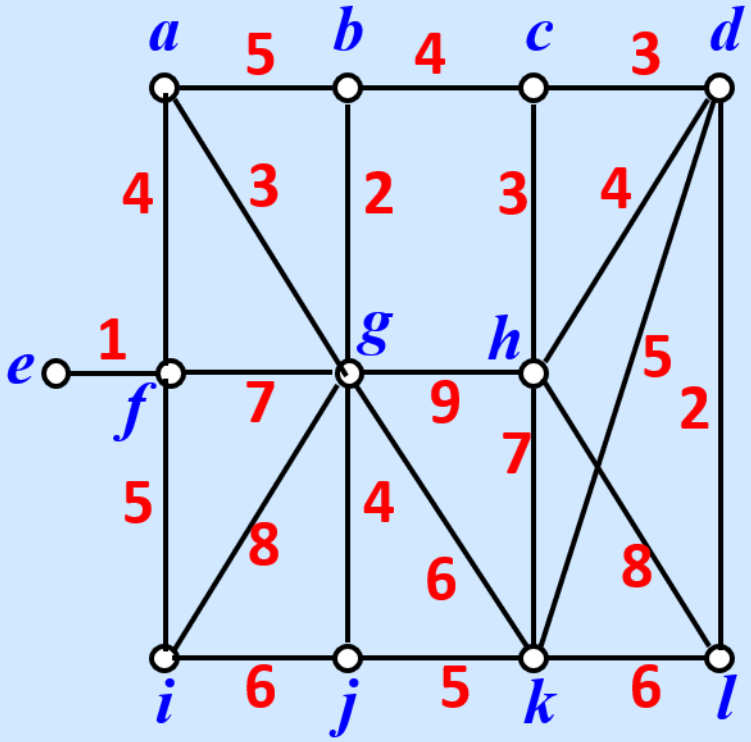


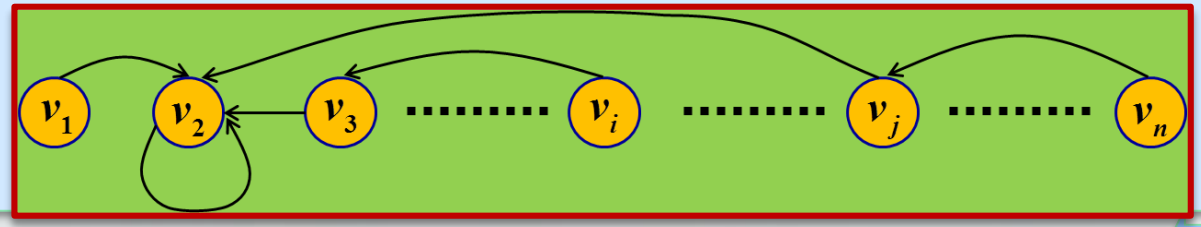
参考程序

```
#include <bits/stdc++.h>
using namespace std;
#define N 100
struct edge {
    int u, v, value;
}myedge[N*N];
int n, m, p[N];
bool cmp(const edge& p1, const edge& p2) {
    return p1.value < p2.value;
}
int find(int x) {
    return (p[x] == x) ? x : (p[x] = find(p[x]));
}
int main() {
    while(scanf("%d%d", &n, &m) == 2) {
        char a[2], b[2];
        int i, x, y, sum = 0, value, counter = 0;
        for(i = 0; i < m; i++) {
            scanf("%s%s%d", a, b, &myedge[i].value);
```



```
            myedge[i].u = a[0] - 'a';
            myedge[i].v = b[0] - 'a';
        }
        for(i = 0; i < n; i++)  p[i] = i;
        sort(myedge, myedge + m, cmp);
        for(i = 0; i < m; i++) {
            value = myedge[i].value;
            x = find(myedge[i].u);
            y = find(myedge[i].v);
            if(x != y) {
                sum += value;
                p[y] = x;
                counter++;
                if(counter >= n-1)  break;
            }
        }
        printf("%d\n", sum);
    }
    return 0;
}
```





2、Prim算法

Prim算法是美国计算机科学家Robert C. Prim1957年独立发现.其算法描述如下.

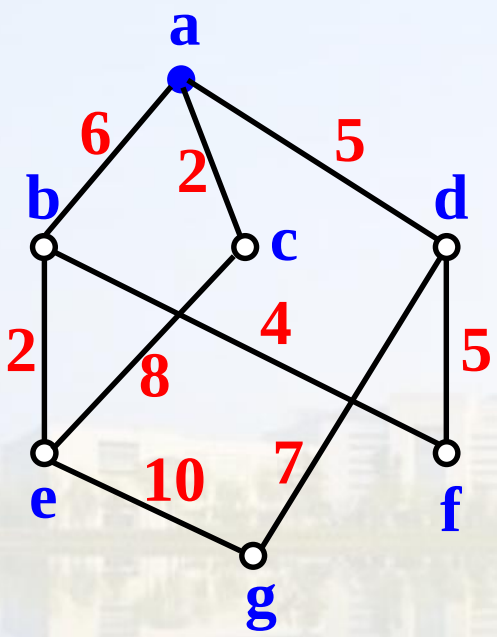
1. 在G中任意选取一个结点v_1,置$V_T=\{v_1\},E_T=\phi,k=1$;

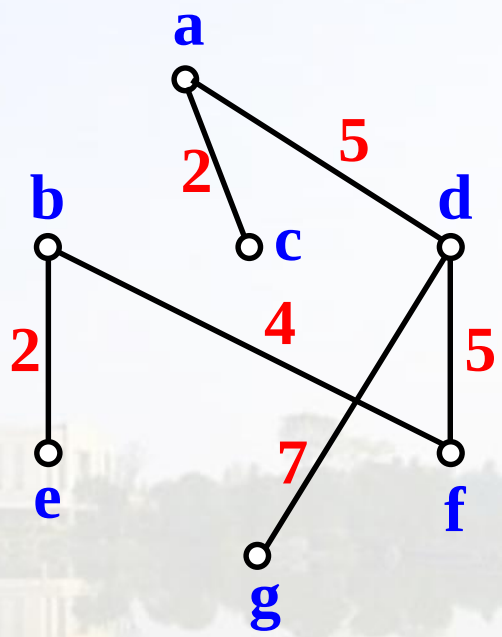
2. 在$V-V_T$中选取与某个$v_i\in V_T$邻接的结点v_j,使得(v_i,v_j)的权最小,置$V_T=V_T\cup\{v_j\},E_T=E_T\cup\{(v_i,v_j)\}$,$k=k+1$;

3. 重复步骤2,直到$k=|V|$.

相对于Kruskal算法, Prim算法也适用于稠密图.

Prim算法

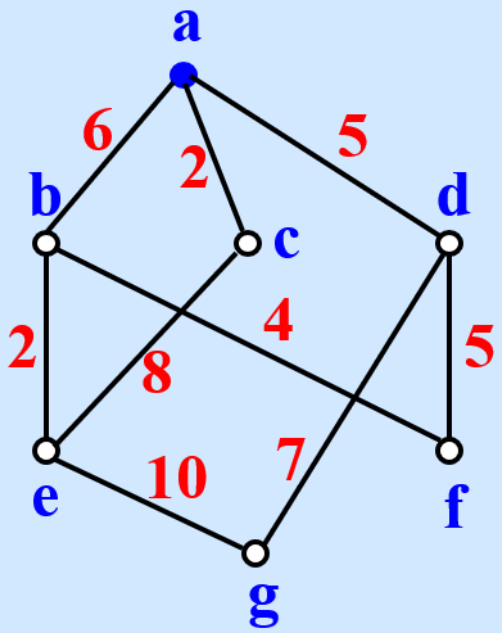




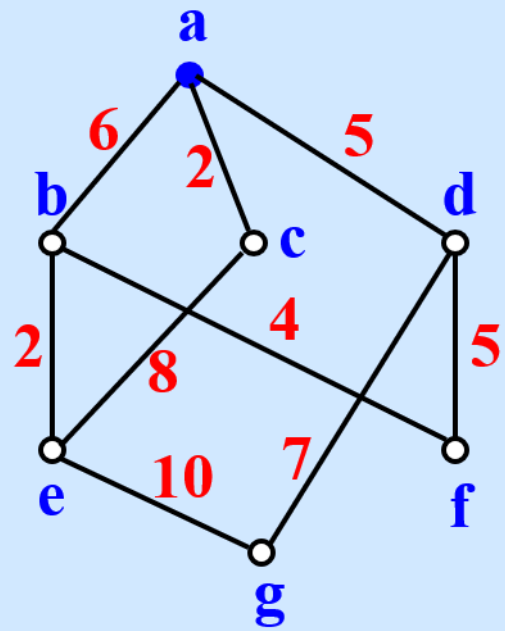
$$w(T) = 25$$

参考程序

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 100;
bool vis[maxn];
int dis[maxn], g[maxn][maxn];
int n, m, ans;
int main()
{
    while(scanf("%d%d", &n, &m) == 2)
    {
        char a[2], b[2];
        memset(g, 0x7f, sizeof(g));
        memset(vis, false, sizeof(vis));
        for(int i = 0; i < m; i++)
        {
            scanf("%s%s%d", a, b, &ans);
            g[a[0]-'a'][b[0]-'a'] =  g[b[0]-'a'][a[0]-'a'] = ans;
        }
        for(int i = 0; i < n; i++)
            dis[i] = g[0][i];
```



```
        ans = 0;
        vis[0] = true;
        dis[n] = 0x7fffffff;
        for(int i = 0; i < n-1; i++)
        {
            int k = n;
            for(int j = 0; j < n; j++)
            {
                if(vis[j]) continue;
                if(dis[j] < dis[k]) k = j;
            }
            ans += dis[k];
            vis[k] = true;
            for(int j = 0; j < n; j++)
                if(g[k][j] < dis[j])
                    dis[j] = g[k][j];
        }
        printf("%d\n", ans);
    }
}
```



修路

N个村庄，从1到N编号，现在请你兴建一些路使得任何两个村庄彼此连通．我们称两个村庄A和B是连通的，当且仅当在A和B之间存在一条路，或者存在一个村庄C，使得A和C之间有一条路，并且C和B是连通的．

已知在一些村庄之间已经有了一些路，你的工作是再兴建一些路，使得所有的村庄都是连通的，并且兴建的路的长度是最小的．

- **Standard Input**

第一行是一个整数N(3<=N<=100)，代表村庄的数目．后面的N行，第i行包含N个整数，这个N个整数中的第j个整数是第i个村庄和第j个村庄之间的距离，距离值在[1,1000]之间．

然后是一个整数Q(0<=Q<=N*(N+1)/2)．后面给出Q行，每行包含两个整数a和b(1<=a<b<=N)，表示在村庄a和b之间已经兴建了路．

- **Standard Output**

输出一行仅有一个整数, 表示为使所有的村庄连通要新建公路的长度的最小值.

- **Samples**

Input	Output
3 0 990 692 990 0 179 692 179 0 1 1 2	179

题目分析

- 可以考虑最小生成树.
- 已有的路把权记为0, 那么在找最小生成树时, 只要不构成回路, 这些边一定会被选上.
- 本题用Kruskal算法.

参考程序

```
#include <bits/stdc++.h>
using namespace std;
const int N = 100;
struct edge {
    int u, v, value;
}myedge[N*N];
int n, m, q, p[N], g[N][N], counter, sum;
bool cmp(const edge& p1, const edge& p2) {
    return p1.value < p2.value;
}
int find(int x) {
    return (p[x] == x) ? x : (p[x] = find(p[x]));
}
int main() {
    int x, y;
    scanf("%d", &n);
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            scanf("%d", &g[i][j]);
    scanf("%d", &q);
    for(int i = 0; i < q; i++) {
        scanf("%d%d", &x, &y);
        g[x-1][y-1] = g[y-1][x-1] = -1;
    }
```

```
    m = 0;
    for(int i = 0; i < n; i++) {
        for(int j = i+1; j < n; j++) {
            if(g[i][j] != 0) {
                myedge[m].u = i; myedge[m].v = j;
                myedge[m].value = (g[i][j]>0) ? (g[i][j]) : 0;
                m++;
            }
        }
    }
    for(int i = 0; i < n; i++) p[i] = i;
    sort(myedge, myedge + m, cmp);
    for(int i = 0; i < m; i++) {
        x = find(myedge[i].u);  y = find(myedge[i].v);
        if(x != y) {
            sum += myedge[i].value;
            p[y] = x;
            counter++;
            if(counter >= n-1)  break;
        }
    }
    printf("%d\n", sum);
    return 0;
}
```

六、最短路径算法

- 最短路径算法是图论中的一个经典问题, 目的是找出图中两点间的最短路径.
- 主要算法有: Dijkstra、Bellman-Ford与SPFA、Floyd.
- 前两个是单源最短路径, 旨在找某个点到其他所有点的最短路. Dijkstra适合不含负权的图, SPFA可以解决有负权边的图 (但不能含负权回路).
- Floyd算法可以找出任意两点间的最短路.

1、Dijkstra算法

Dijkstra算法适用于求边权为正, 从单个源点出发的最短路. 它是由Dijkstra于1959年提出的. 实际它能求出始点到其它所有顶点的最短路径.

Dijkstra算法是一种标号法：给赋权图的每一个顶点记一个数, 称为顶点的标号(临时标号,称T标号, 或者固定标号, 称为P标号). T标号表示从始点到该标点的最短路长的上界; P标号则是从始点到该顶点的最短路长.

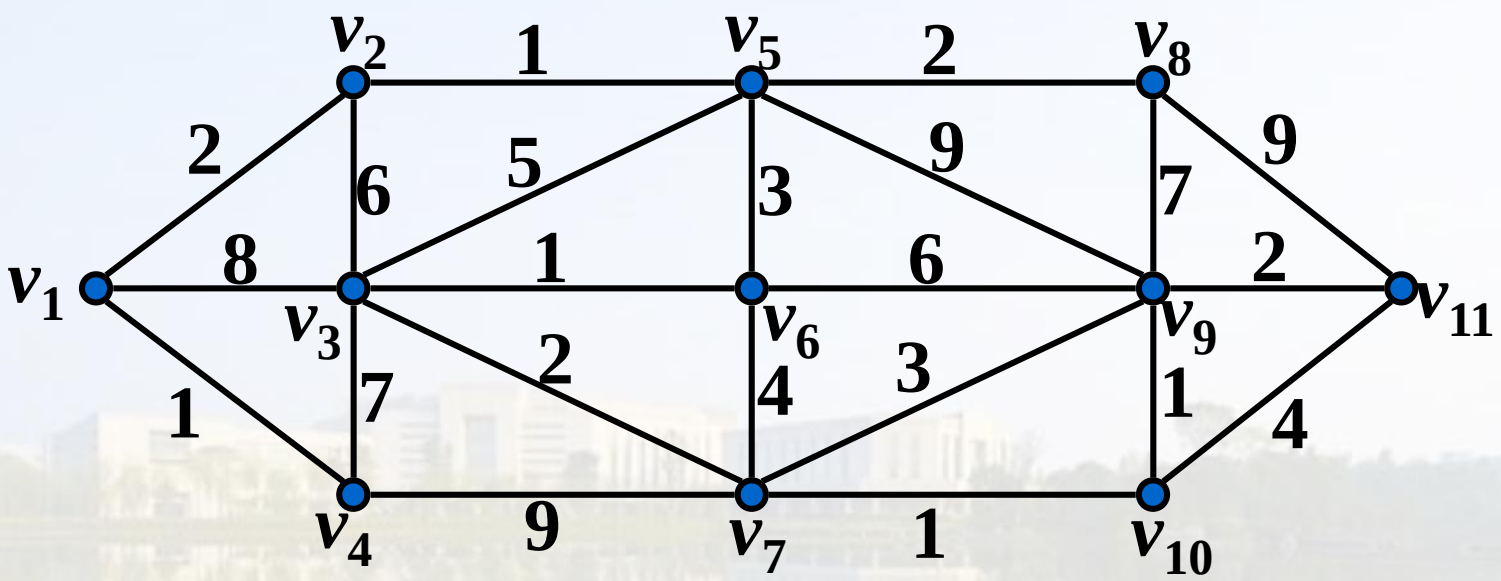
Dijkstra算法步骤如下(假设v_1是始点):

(1)给顶点v_1标P标号$d(v_1)=0$,给顶点$v_j(j=2,3,\cdots,n)$标T标号$d(v_j)=l_{1j}$;

(2)在所有T标号中取最小值,譬如,$d(v_{j_0})=l_{1j_0}$最小,则把v_{j_0}的T标号改为P标号,并重新计算具有T标号的其它各顶点的T标号:选顶点v_j的T标号$d(v_j)$与$d(v_{j_0})+l_{j_0j}$中较小者作为v_j的新的T标号.

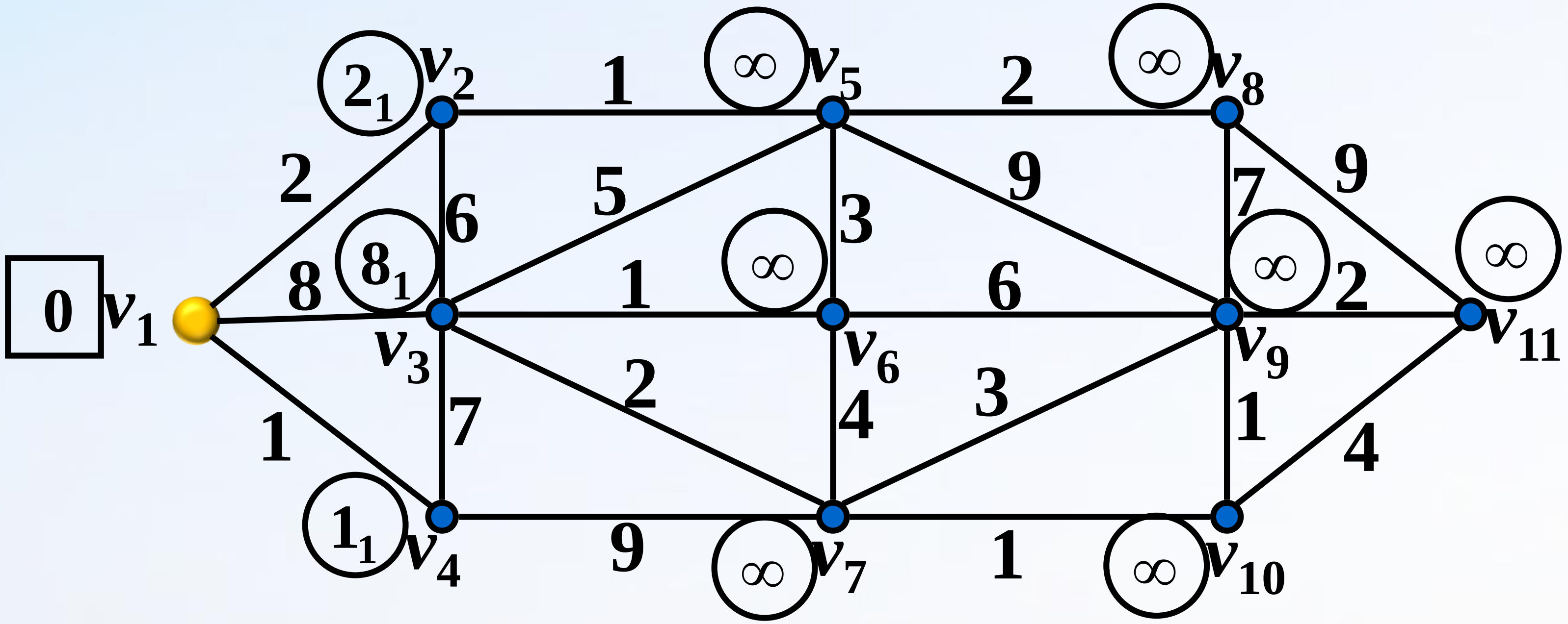
(3)重复上述步骤,直到目标顶点的标号改为P标号.

例 求出下图G中v_1到v_{11}的最短路长。

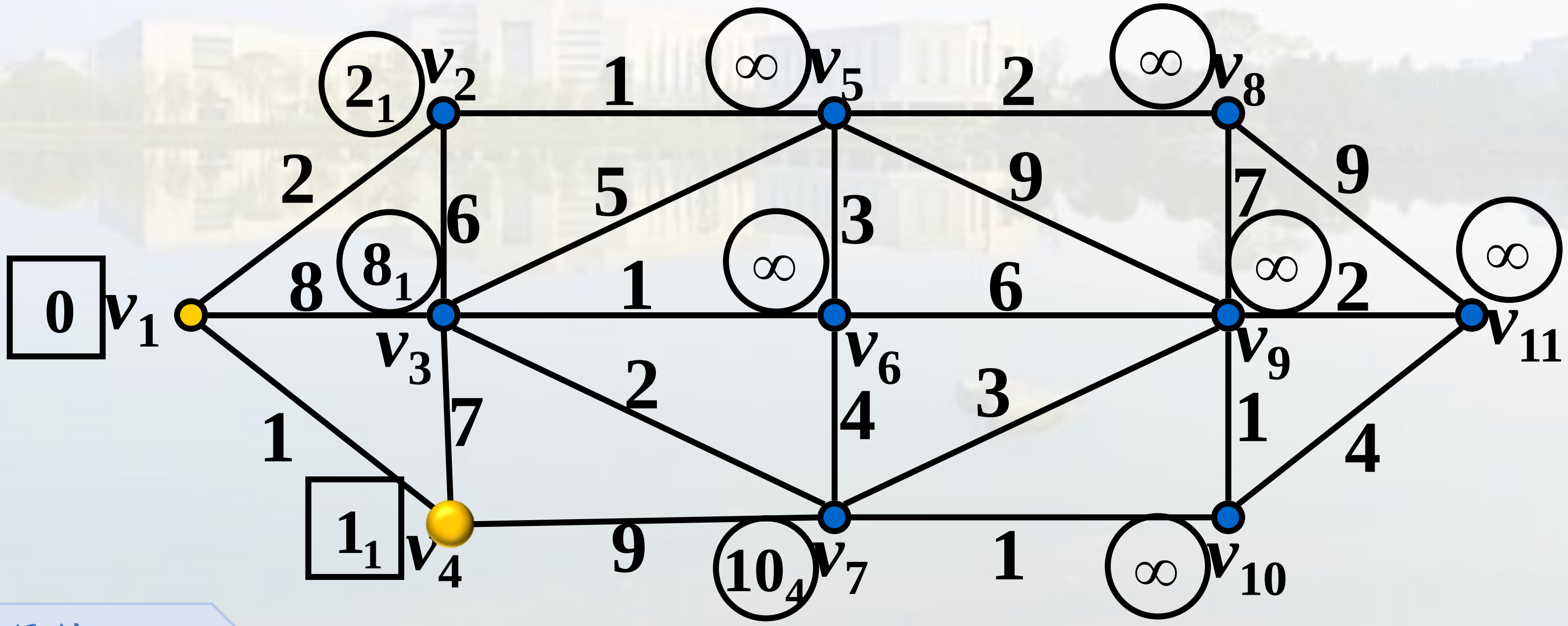


$\boxed{3_2}$---固定编号，下标表示前趋顶点

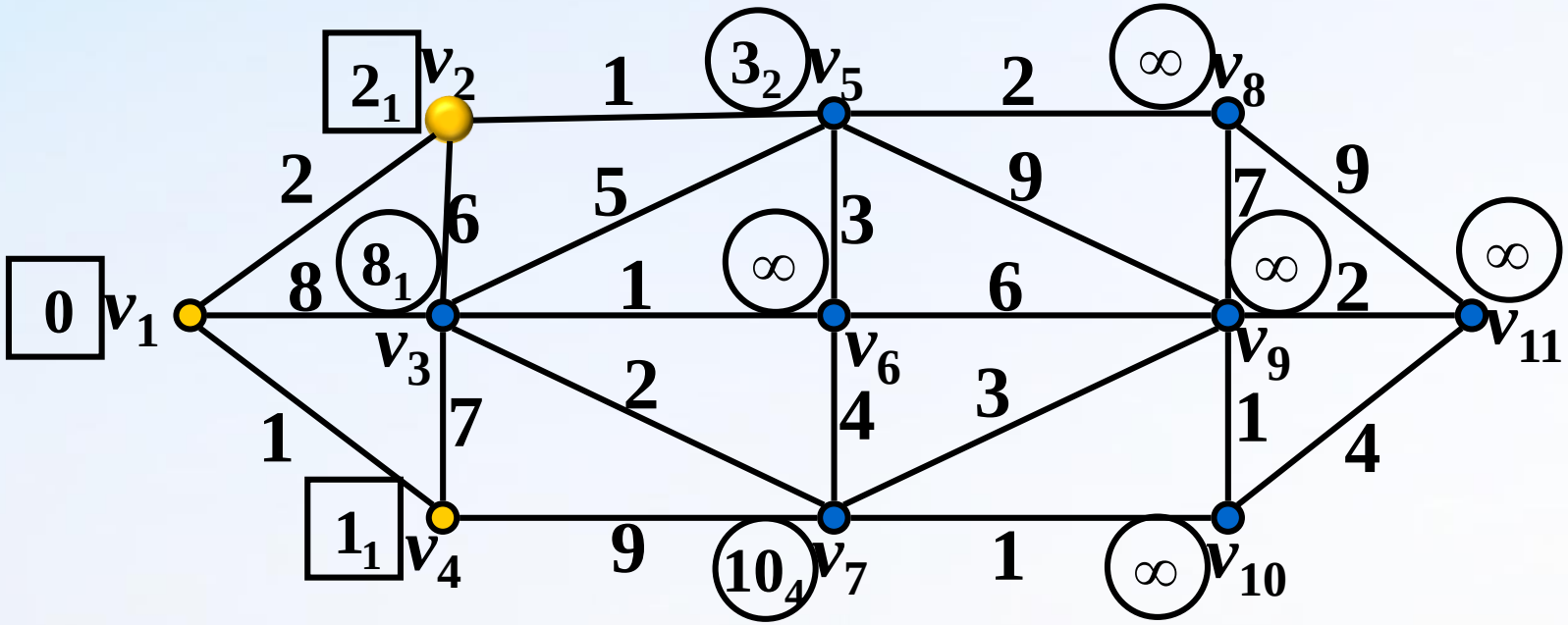
(8_1)---临时编号，下标表示前趋顶点

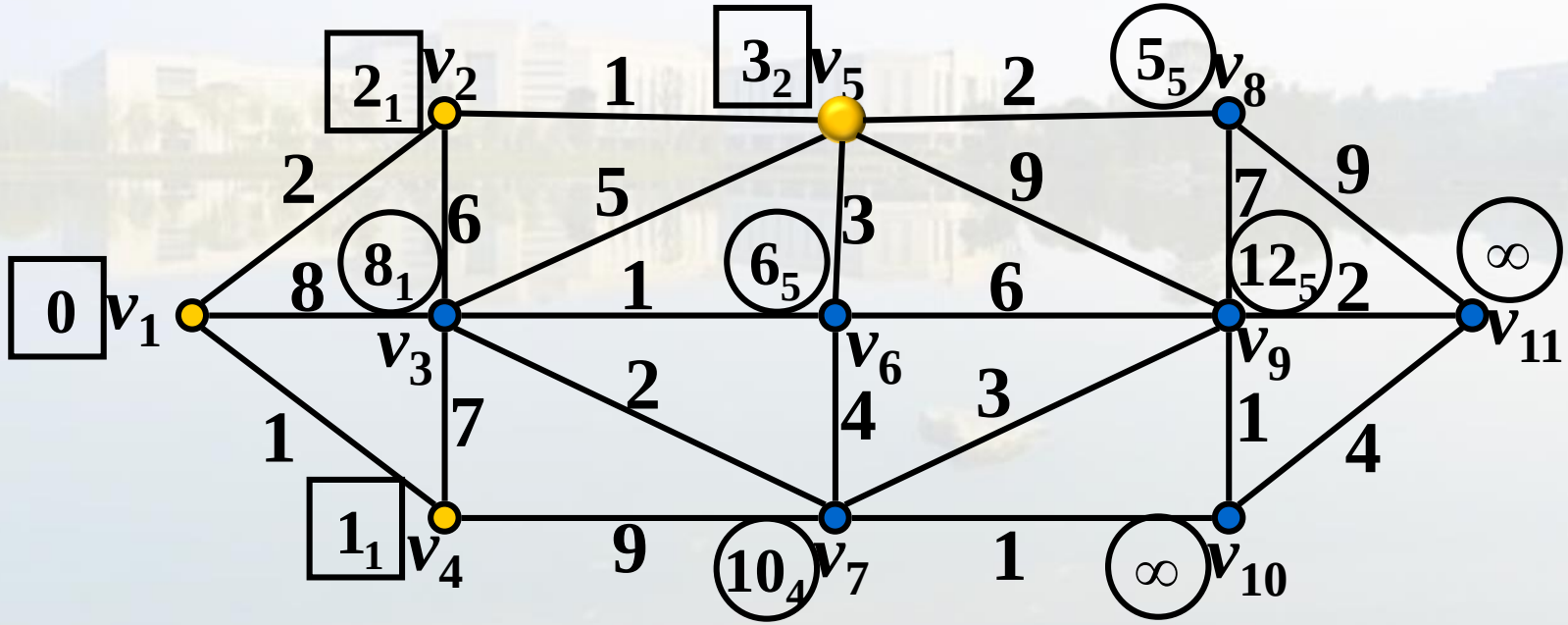


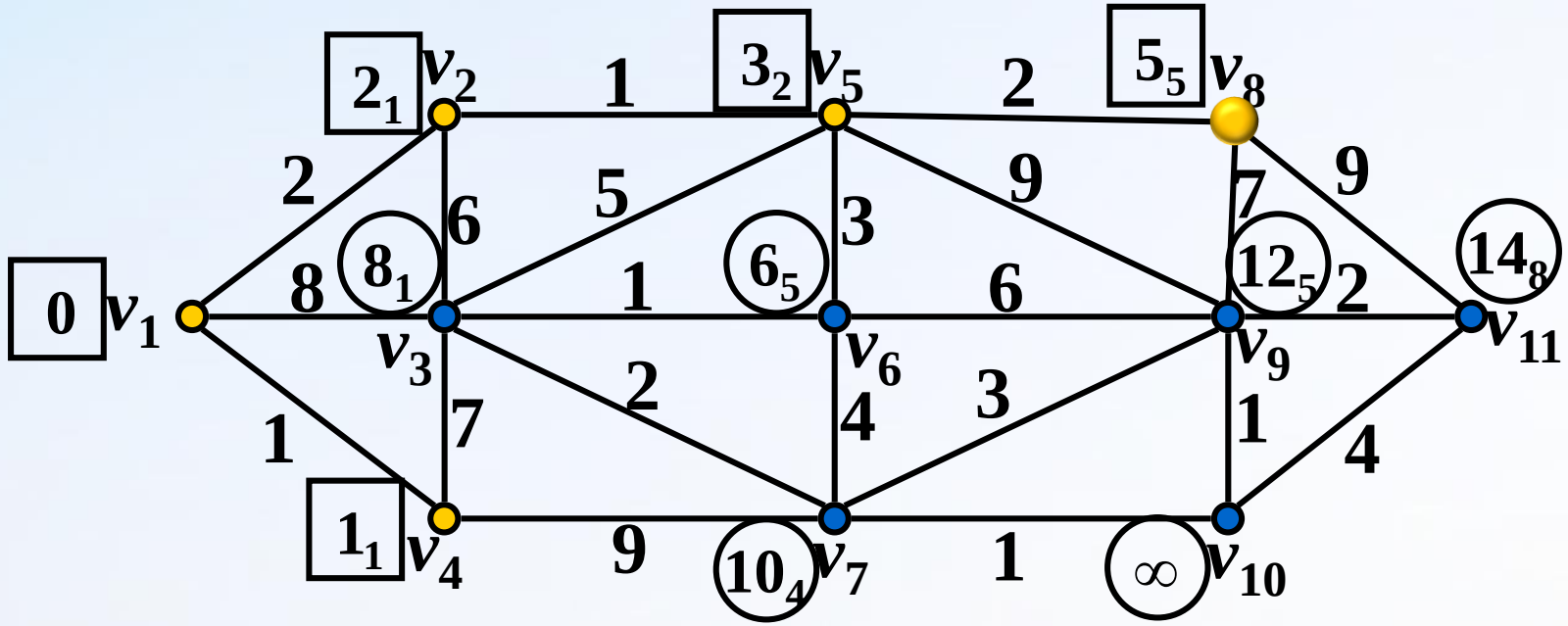
0
v1
2₁ v2
8₁ v3
1₁ v4
∞ v5
∞ v6
∞ v7
∞ v8
∞ v9
∞ v10
∞ v11
2 8 1 1 6 7 5 2 9 1 3 4 2 9 6 3 7 1 9 2 4 1

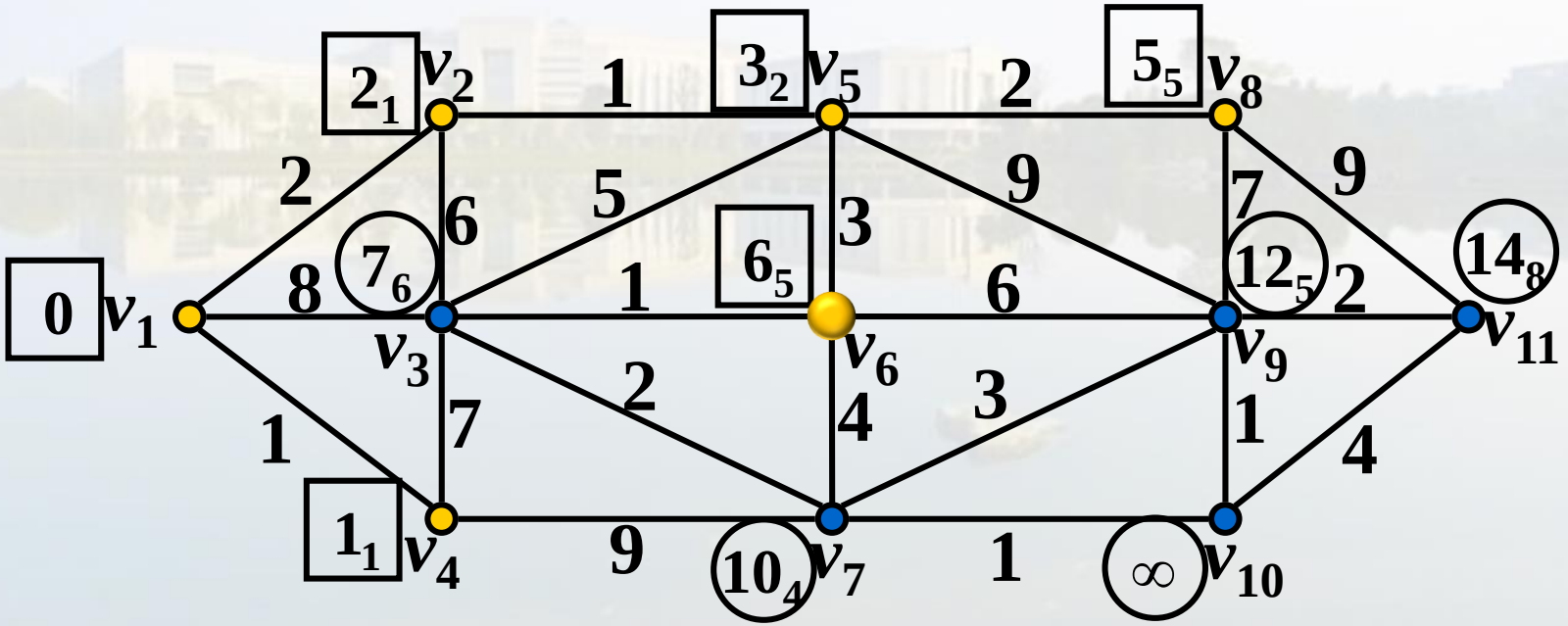


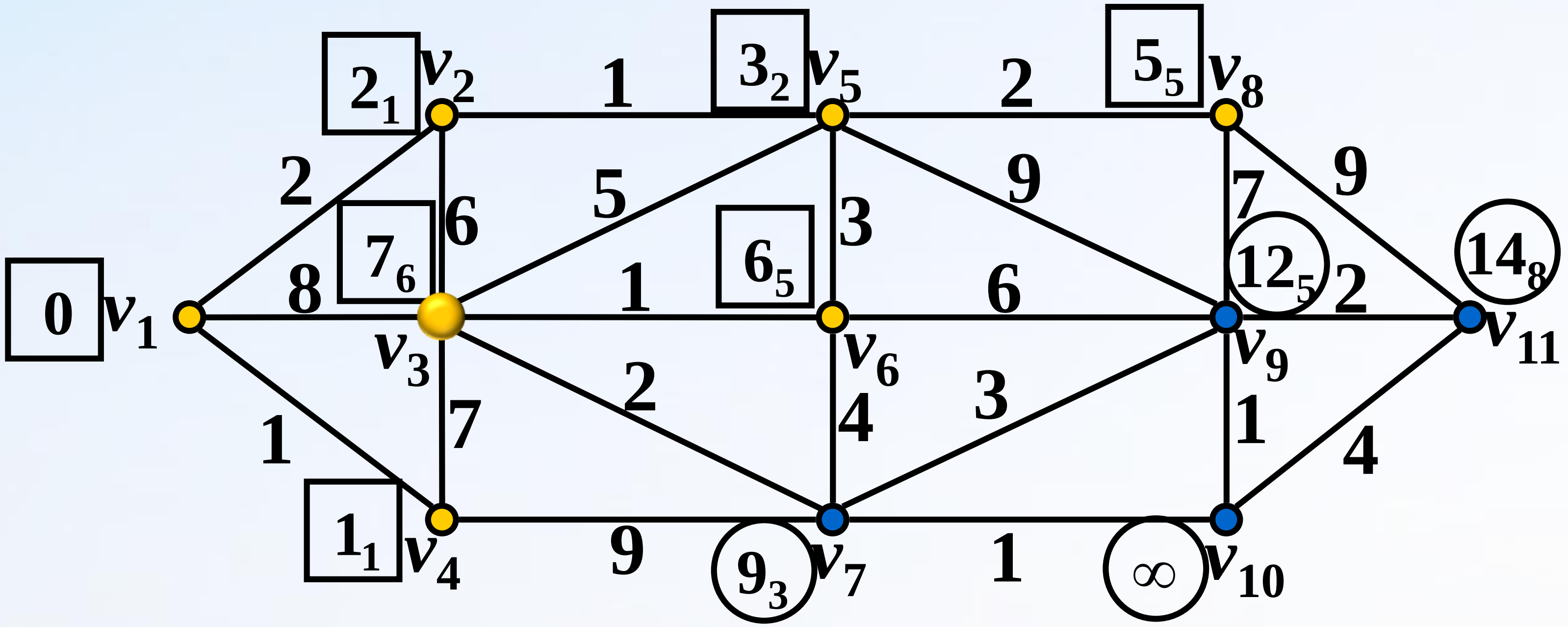
0
v1
2₁ v2
8₁ v3
1₁ v4
∞ v5
∞ v6
10₄ v7
∞ v8
∞ v9
∞ v10
∞ v11
2 8 1 1 6 7 5 2 9 1 3 4 2 9 6 3 7 1 9 2 4 1

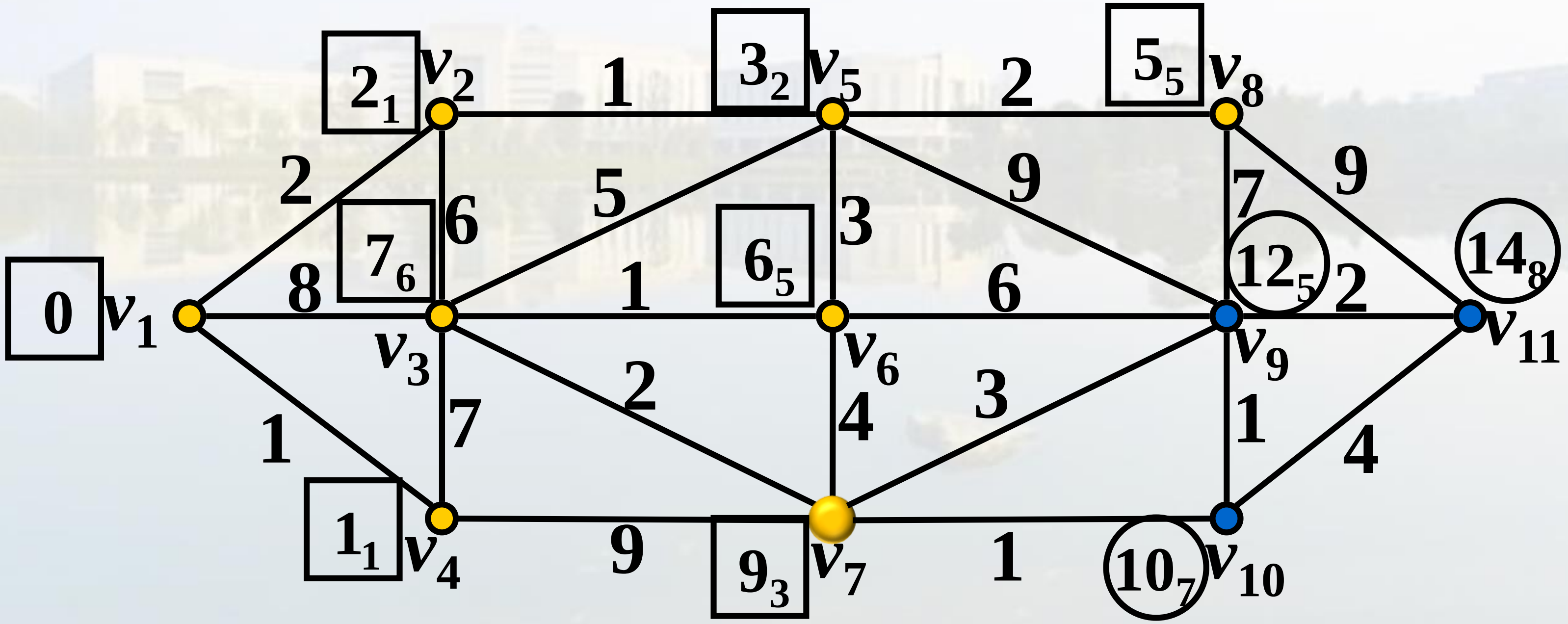

2_1 v_2
3_2 v_5
∞ v_8
8_1
∞
∞
∞
0 v_1
v_3
v_6
v_9
v_{11}
1_1 v_4
10_4 v_7
∞ v_{10}

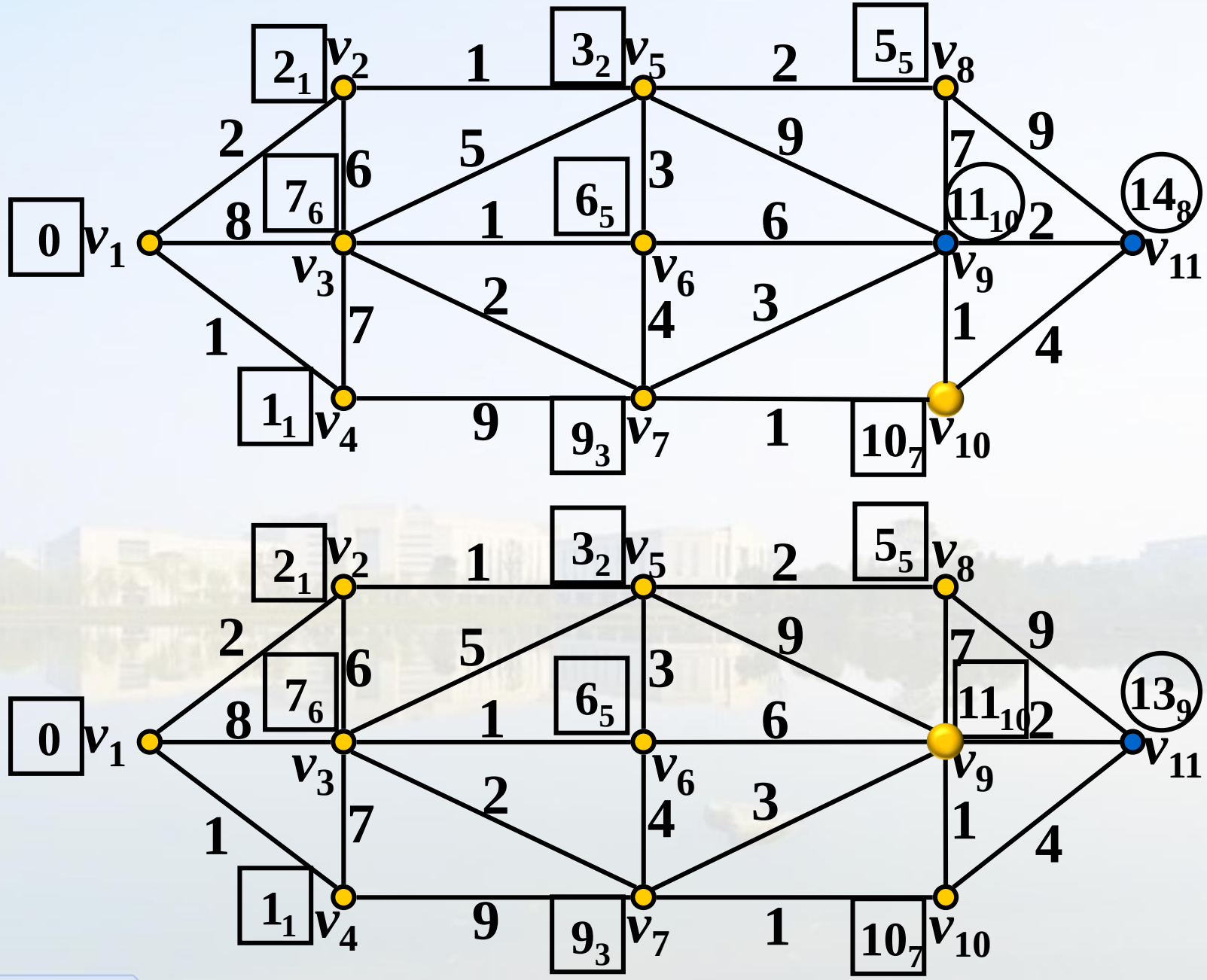

2_1 v_2
3_2 v_5
5_5 v_8
8_1
6_5
12_5
∞
0 v_1
v_3
v_6
v_9
v_{11}
1_1 v_4
10_4 v_7
∞ v_{10}

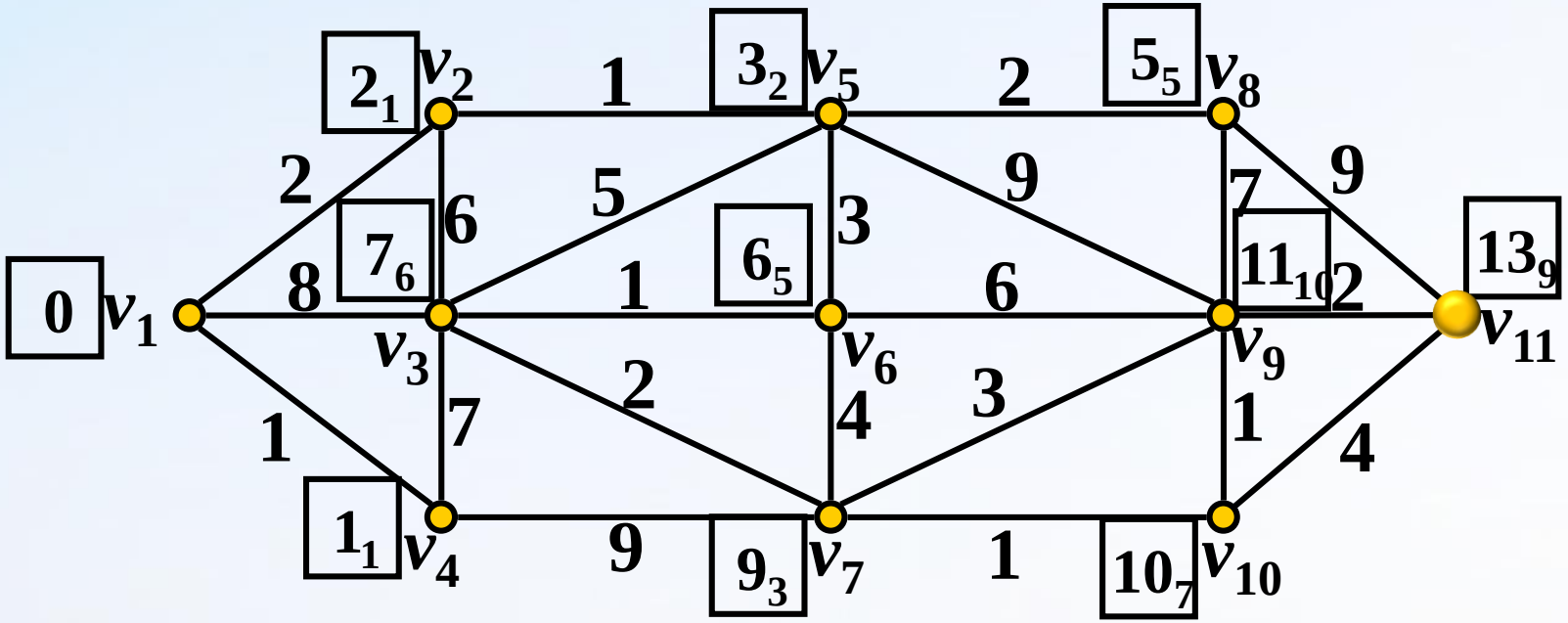

0 v1
2 1 v2
3 2 v5
5 5 v8
8 1
6 5
12 5
14 8
v3
v6
v9
v11
1 1 v4
10 4 v7
∞ v10

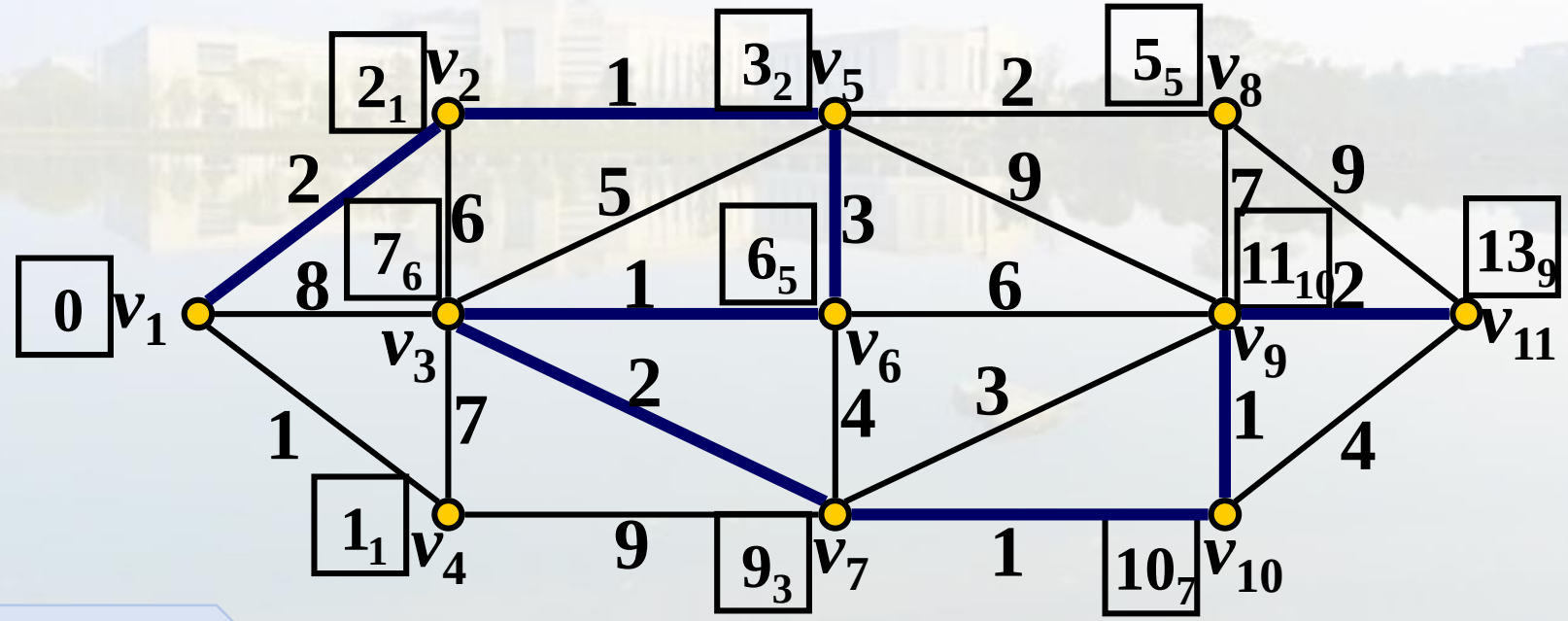

0 v1
2 1 v2
3 2 v5
5 5 v8
7 6
6 5
12 5
14 8
v3
v6
v9
v11
1 1 v4
10 4 v7
∞ v10


0
v1
2
8
1
2₁
v2
7₆
6
v3
7
1₁
v4
1
5
1
2
9
3₂
v5
6₅
3
v6
4
9₃
v7
2
9
6
3
1
5₅
v8
7
12₅
v9
1
∞
v10
9
2
4
14₈
v11


0
v1
2
8
1
2₁
v2
7₆
6
v3
7
1₁
v4
1
5
1
2
9
3₂
v5
6₅
3
v6
4
9₃
v7
2
9
6
3
1
5₅
v8
7
12₅
v9
1
10₇
v10
9
2
4
14₈
v11


0 v1
2 1 v2
3 2 v5
5 5 v8
14 8 v11
7 6
6 5
11 10
v3
v6
v9
1 1 v4
9 3 v7
10 7 v10
13 9


2_1 v_2
1
3_2 v_5
2
5_5 v_8
2
9
7
9
7_6
6
5
6_5
3
11_{10}
2
13_9
0 v_1
8
1
6
v_3
v_6
v_9
v_{11}
2
3
7
4
1
1
4
1_1 v_4
9
9_3 v_7
1
10_7 v_{10}


2_1 v_2
1
3_2 v_5
2
5_5 v_8
2
9
7
9
7_6
6
5
6_5
3
11_{10}
2
13_9
0 v_1
8
1
6
v_3
v_6
v_9
v_{11}
2
3
7
4
1
1
4
1_1 v_4
9
9_3 v_7
1
10_7 v_{10}