```
#include<iostream>
using namespace std;
typedef int VertexType;
#define MAX_VERTEX_NUM 20
#define INFINITY 65535
//typedef enum { DG, DN, UDG, UDN } GrapgKind;
typedef  int AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM];
//邻接矩阵类型
typedef struct {
    VertexType vexs[MAX_VERTEX_NUM];      //顶点表
    AdjMatrix arcs;     //邻接矩阵
    int vexnum, arcnum;  //图的顶点数和边/弧数
    int Graphkind;
}  MGraph;
typedef struct {
    VertexType adjvex;
    int lowcost;
}closedge[MAX_VERTEX_NUM];
int LocateVertex(MGraph &G, VertexType v)
{
    int i;
    for (i = 0; i < G.vexnum; i++)
        if (v == G.vexs[i])
            return i;
    return -1;
}
void CreateGraph(MGraph &G1)
{
    int i, j, k;
    VertexType v1, v2;
    int w;
    int n, m;
    cin >> n >> m;
    G1.vexnum = n;
    G1.arcnum = m;
    for (i = 0; i < G1.vexnum; i++)
        G1.vexs[i] = i + 1;
    for (i = 0; i < G1.vexnum; i++)
        for (j = 0; j < G1.vexnum; j++)
            G1.arcs[i][j] = INFINITY;
    for (k = 0; k < G1.arcnum; k++)
    {
        cin >> v1 >> v2 >> w;
        i = LocateVertex(G1, v1);
```

```
        j = LocateVertex(G1, v2);
        G1.arcs[i][j] = w;
        G1.arcs[j][i] = w;
    }
}
int min(closedge &cls, MGraph &G)
{
    int min = INFINITY, k = 0;
    for (int i = 0; i < G.vexnum; i++)
        if (cls[i].lowcost != 0 && cls[i].lowcost < min)
        {
            min = cls[i].lowcost;
            k = i;
        }
    return k;
}
int  MST_PRIM(MGraph &G, VertexType u)
{
    closedge cls;
    int k, j, sum = 0, arc_num = 0;
    k = LocateVertex(G, u);
    for (j = 0; j < G.vexnum; j++)
    {
        if (j != k)
            cls[j] = { u,G.arcs[k][j] };
    }
    cls[k].lowcost = 0;
    int i;
    for (i = 1; i < G.vexnum; i++)
    {
        k = min(cls, G);
        sum += cls[k].lowcost;
        arc_num++;
        cls[k].lowcost = 0;
        for (j = 0; j < G.vexnum; j++)
            if (G.arcs[k][j] < cls[j].lowcost)
                cls[j] = { G.vexs[k],G.arcs[k][j] };
    }
    if (arc_num == G.vexnum - 1)
        return sum;
    else
        return  -1;
}
```

```
int main()
{
    MGraph G1;
    CreateGraph(G1);
    cout << MST_PRIM(G1, 1);
    return 0;
}

#include<iostream>
using namespace std;
#define MAX_VERTEX_NUM 20
int AdjMatrix[5][5] = {
            0, 1, 1, 0, 1,
            1, 0, 1, 0, 1,
            1, 1, 0, 1, 1,
            0, 0, 1, 0, 1,
            1, 1, 1, 1, 0
};   //邻接矩阵
char s[MAX_VERTEX_NUM];
void  DFS(int x, int k, char s[])
{

    s[k] = x + 1 + '0';
    if (k >= 8)
    {
        cout << s << endl;
        return;
    }
    int y = 0;
    for (y = 0; y < 5; y++)
        if (AdjMatrix[x][y] == 1)
        {
            AdjMatrix[x][y] = 0;
            AdjMatrix[y][x] = 0;
            DFS(y, k + 1, s);
            AdjMatrix[x][y] = 1;
            AdjMatrix[y][x] = 1;
        }
}
int main()
{
    int x = 0;
    DFS(x, 0, s);
    return 0;
```

```
}


#include<iostream>
#include<iomanip>
#include<queue>
#include<vector>
#include<string.h>
using namespace std;
#define MAX_VERTEX_NUM 10000
int vis[MAX_VERTEX_NUM];
int vexnum, arcnum;
float  BFSTraverse(int k, vector<int>AdjMatrix[])
{
    int v, depth = 1;//广度优先的层数
    float count = 1;//顶点个数
    queue<int>Queue;
    int w;
    vis[k] = 1;
    Queue.push(k);
    Queue.push(-1);
    while (!Queue.empty())
    {
        v = Queue.front();
        Queue.pop();
        if (v == -1)
        {
            depth++;
            continue;
        }
        for (w = 0; w < AdjMatrix[v].size(); w++)
            if (!vis[AdjMatrix[v][w]] && depth < 6)
            {
                count++;
                vis[AdjMatrix[v][w]] = 1;
                Queue.push(AdjMatrix[v][w]);
            }
            else if (!vis[AdjMatrix[v][w]] && depth == 6)
            {
                count++;
                vis[AdjMatrix[v][w]] = 1;
            }
        if (Queue.front() == -1)
            Queue.push(-1);
```

```
    }
    return count;
}
int main()
{
    vector<int>AdjMatrix[MAX_VERTEX_NUM];
    float percent;
    int v1, v2, k;
    cin >> vexnum >> arcnum;
    for (k = 1; k <= arcnum; k++)
    {
        cin >> v1 >> v2;
        AdjMatrix[v1].push_back(v2);
        AdjMatrix[v2].push_back(v1);
    }
    for (k = 1; k <= vexnum; k++)
    {
        memset(vis, 0, sizeof(vis));
        cout << k << ": ";
        percent = (BFSTraverse(k, AdjMatrix) / vexnum)*100.0;
        cout << setiosflags(ios::fixed) << setprecision(2) << percent <<
"%" << endl;
    }
    return 0;
}
```