给定一张 n 个点的无向带权图，节点的编号从 1 至 n，求从 S 到 T 的最短路径长度。

```
/*邻接矩阵的Dijkstra算法*/
#include<iostream>
using namespace std;
typedef int VertexType;
#define MAX_VERTEX_NUM 1700
#define INFINITY 2147483647
int num[MAX_VERTEX_NUM] = { 0 };
//typedef enum { DG, DN, UDG, UDN } GrapgKind;
typedef  long long 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  int ShortPathTable[MAX_VERTEX_NUM];  //最短路径长度
int LocateVertex(MGraph &G, VertexType v)
{
    int i;
    for (i = 1; i <= G.vexnum; i++)
        if (v == G.vexs[i])
            return i;
    return -1;
}
void CreateGraph(MGraph &G, int n, int m)
{
    int i, j, k;
    VertexType v1, v2;
    int w;
    G.vexnum = n;
    G.arcnum = m;
    for (i = 1; i <= G.vexnum; i++)
        G.vexs[i] = i;
    for (i = 1; i <= G.vexnum; i++)
        for (j = 1; j <= G.vexnum; j++)
            G.arcs[i][j] = INFINITY;
    for (k = 0; k < G.arcnum; k++)
    {
        cin >> v1 >> v2 >> w;
        i = LocateVertex(G, v1);
        j = LocateVertex(G, v2);
        G.arcs[i][j] = w;
        G.arcs[j][i] = w;
    }
}
void ShortPath_DIJ(MGraph&G, int v0, ShortPathTable &D)
{
    //用Dijkstra算法求有向网v0到其余各顶点的最短路径P[v]及其带权长度
D[v];
    //P[v][w]=true，则w是当前最短路径上的顶点
    //final[v]=true当且仅当v∈S,即已经求得v0到v的最短路径
    int final[MAX_VERTEX_NUM] = { 0 };
    int v;
    for (v = 1; v <= G.vexnum; v++)
        D[v] = G.arcs[v0][v];//初始化最短距离
    D[v0] = 0;
    final[v0] = 1;//初始化，v0顶点属于S;
    int min, w, i;
    for (i = 2; i <= G.vexnum; i++)
    {
        min = INFINITY;
        for (w = 1; w <= G.vexnum; w++)
            if (!final[w] && D[w] < min)
            {
                v = w;
                min = D[w];
            }
        final[v] = 1;
        //更新当前最短路径P[][]及最短距离D[];
        for (w = 1; w <= G.vexnum; w++)
            if (!final[w] && min + G.arcs[v][w] < D[w])
                D[w] = min + G.arcs[v][w];
    }
}
int main()
{
    MGraph G;
    ShortPathTable D;
    int n, m;
    int v0, v1;
    cin >> n >> m >> v0 >> v1;
    CreateGraph(G, n, m);
    ShortPath_DIJ(G, v0, D);
    cout << D[v1] << endl;
    return 0;
}

/*邻接链表的Dijkstra算法*/
#include<iostream>
using namespace std;
typedef int VertexType;
#define MAX_VERTEX_NUM 20000
#define INFINITY  2147483647
typedef struct ArcNode
{
    int adjvex;//弧指向的顶点的位置
    ArcNode *nextarc;//指向下一个与该顶点邻接的顶点
    long long info;//弧的相关信息
}ArcNode;//边表结点
typedef struct VNode
{
    VertexType data;//用于存储顶点
    ArcNode *firstarc;//指向第一个与该顶点邻接的顶点
}VNode, AdjList[MAX_VERTEX_NUM];//表头节点，顺序表存储
typedef struct
{
    AdjList vertices;//邻接表
    int vexnum, arcnum;//边数，顶点数
    int kind;//图的种类
}ALGraph;
typedef  long long int ShortPathTable[MAX_VERTEX_NUM];  //最短路径长度
int LocateVertex(ALGraph &G, VertexType& v)
{
    int i;
    for (i = 1; i <= G.vexnum; i++)
        if (G.vertices[i].data == v)
            return i;
    return -1;
}
void CreateGraph_AdjList(ALGraph &G, int n, int m)
{
    int i, j, k, w;
    VertexType v1, v2;
    ArcNode *p;
    G.vexnum = n;
    G.arcnum = m;
    for (i = 1; i <= G.vexnum; i++)
    {
        G.vertices[i].data = i;
        G.vertices[i].firstarc = NULL;
    }
    for (k = 1; k <= G.arcnum; k++)//无向图
    {
        cin >> v1 >> v2 >> w;
        i = LocateVertex(G, v1);
        j = LocateVertex(G, v2);
        /*j为入i为出创建邻接链表*/
        p = new ArcNode;
        p->adjvex = j;
        p->info = w;
        p->nextarc = G.vertices[i].firstarc;
        G.vertices[i].firstarc = p;
        /*i为入j为出创建邻接链表*/
        p = new ArcNode;
        p->adjvex = i;
        p->info = w;
        p->nextarc = G.vertices[j].firstarc;
        G.vertices[j].firstarc = p;
    }
}
void ShortPath_DIJ(ALGraph&G, int v0, ShortPathTable &D)
{
    //用Dijkstra算法求有向网v0到其余各顶点带权长度D[v];
    //final[v]=true当且仅当v∈S,即已经求得v0到v的最短路径
    int final[MAX_VERTEX_NUM] = { 0 };
    long long min;
    int v, i, w, j;
    for (int i = 1; i <= G.vexnum; i++)
        D[i] = INFINITY;
    ArcNode *p;
    p = G.vertices[v0].firstarc;
    while (p)
    {
        w = p->adjvex;
        D[w] = p->info;
        p = p->nextarc;
    }
    D[v0] = 0;
    final[v0] = 1;//初始化，v0顶点属于S;
    for (i = 1; i < G.vexnum; i++)
    {
        min = INFINITY;
        for (j = 1; j <= G.vexnum; j++)
        {
            if (!final[j] && D[j] < min)
            {
                v = j;
                min = D[j];
            }
        }
        final[v] = 1;
        //更新当前最短距离D[];
        p = G.vertices[v].firstarc;
        while (p)
        {
            w = p->adjvex;
            if (!final[w] && min + p->info < D[w])
                D[w] = min + p->info;
            p = p->nextarc;
        }
    }
}
int main()
{
    ALGraph G;
    ShortPathTable D;
    int n, m;
    int v0, v1;
    cin >> n >> m >> v0 >> v1;
    CreateGraph_AdjList(G, n, m);
    ShortPath_DIJ(G, v0, D);
    cout << D[v1] << endl;
    return 0;
}

/*静态邻接表的Dijkstra算法*/
#include<iostream>
using namespace std;
typedef int VertexType;
#define MAX_VERTEX_NUM 26000
#define INFINITY  2147483647
typedef  long long ShortPathTable[MAX_VERTEX_NUM];   //最短路径长度
int first[MAX_VERTEX_NUM];
int u[MAX_VERTEX_NUM];//起始顶点表;
int v[MAX_VERTEX_NUM];//终止顶点表;
int w[MAX_VERTEX_NUM];//权值表;
int _next[MAX_VERTEX_NUM * 2];//邻接点表
void ShortPath_DIJ(int n, int m, int v0, ShortPathTable &D)
{
    //用Dijkstra算法求有向网v0到其余各顶点带权长度D[v];
    //final[v]=true当且仅当v∈S,即已经求得v0到v的最短路径
    int final[MAX_VERTEX_NUM] = { 0 };
    long long min;
    int i, j, l;
    for (int i = 1; i <= n; i++)
    {
        D[i] = INFINITY;
        first[i] = -1;
    }
    j = 1;
    for (i = 1; i <= m; i++)
    {
        /*构建无向网*/
        /*相当于输入2*m条边，即u->v;v->u 并同时
        将它们存储在静态链表为基础的邻接表中*/
        cin >> u[i] >> v[i] >> w[i];
        _next[j] = first[u[i]];
        first[u[i]] = j;
        _next[j + 1] = first[v[i]];
        first[v[i]] = ++j;
        if (u[i] == v0)
            D[v[i]] = w[i];
        if (v[i] == v0)
            D[u[i]] = w[i];
        j++;
    }
    final[v0] = 1;
    D[v0] = 0;
    int k;
    for (k = 1; k <= n; k++)
    {
        min = INFINITY;
        for (j = 1; j <= n; j++)
        {
            /*找到最离源点最近的一个点l*/
            if (final[j] == 0 && min > D[j])
            {
                l = j;
                min = D[j];
            }
        }
        final[l] = 1;
        int p;
        for (i = first[l]; i != -1; i = _next[i])
        {
            /*在静态邻接表中遍历更新D[]*/
            p = (i + 1) / 2;
            if (i % 2 == 0 && !final[u[p]])
            {
                if (D[l] + w[p] < D[u[p]])
                    D[u[p]] = D[l] + w[p];
            }
            else if (!final[v[p]] && i % 2 == 1)
            {
                if (D[l] + w[p] < D[v[p]])
                    D[v[p]] = D[l] + w[p];
            }
        }
    }
}
int main()
{
    ShortPathTable D;
    int n, m;
    int v0, v1;
    cin >> n >> m >> v0 >> v1;
    ShortPath_DIJ(n, m, v0, D);
    cout << D[v1] << endl;
    return 0;
}


/*完整Dijkstra算法*/
#include<iostream>
using namespace std;
typedef int VertexType;
#define MAX_VERTEX_NUM 20
#define INFINITY 65535
int num[MAX_VERTEX_NUM] = { 0 };
//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 int PathMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM]; //最短路径数组
typedef  int ShortPathTable[MAX_VERTEX_NUM];  //最短路径长度
int LocateVertex(MGraph &G, VertexType v)
{
    int i;
    for (i = 1; i <= G.vexnum; i++)
        if (v == G.vexs[i])
            return i;
    return -1;
}
void CreateGraph(MGraph &G)
{
    int i, j, k;
    VertexType v1, v2;
    int w;
    int n, m;
    cin >> n >> m;
    G.vexnum = n;
    G.arcnum = m;
    for (i = 1; i <= G.vexnum; i++)
        G.vexs[i] = i;
    for (i = 1; i <= G.vexnum; i++)
        for (j = 1; j <= G.vexnum; j++)
            G.arcs[i][j] = INFINITY;
    for (k = 0; k < G.arcnum; k++)
    {
        cin >> v1 >> v2 >> w;
        i = LocateVertex(G, v1);
        j = LocateVertex(G, v2);
        G.arcs[i][j] = w;
    }
}
void ShortPath_DIJ(MGraph&G, int v0, PathMatrix &P, ShortPathTable &D)
{
    //用Dijkstra算法求有向网v0到其余各顶点的最短路径P[v]及其带权长度D[v];
    //P[v][w]=true，则w是当前最短路径上的顶点
    //final[v]=true当且仅当v∈S,即已经求得v0到v的最短路径
    int final[MAX_VERTEX_NUM] = { 0 };
    int v;
    for (v = 1; v <= G.vexnum; v++)
    {
        D[v] = G.arcs[v0][v];//初始化最短距离
        for (int w = 1; w <= G.vexnum; w++)
        {
            P[v][w] = 0;//路径为空
            if (D[v] < INFINITY)
            {
                P[v][v0] = 1;
                P[v][v] = 1;
            }
        }
    }
    D[v0] = 0;
    final[v0] = 1;//初始化，v0顶点属于S;
    int min, w, i;
    for (i = 2; i <= G.vexnum; i++)
    {
        min = INFINITY;
        for (w = 1; w <= G.vexnum; w++)
            if (!final[w] && D[w] < min)
            {
                v = w;
                min = D[w];
            }
        final[v] = 1;
        //更新当前最短路径P[][]及最短距离D[];
        for (w = 1; w <= G.vexnum; w++)
        {
            if (!final[w] && min + G.arcs[v][w] < D[w])
            {
                D[w] = min + G.arcs[v][w];
                for (int i = 1; i <= G.vexnum; i++)
                    if (P[v][i] == 1)
                        P[w][i] = P[v][i];
                P[w][w] = 1;
            }
        }
    }
}
```