图的表示

比如说现在有N个点，M条边，边有边权表示两点间的距离，如果两点间没有边就表示这两点之间不直接连通。如何用程序语言表示该图?

二维数组实现邻接矩阵

开一个大小为 $N \times N$ 的二维数组g，数据类型为int（或者long long，也有可能是double，不用纠结这个）。

二维数组中的值就表示某两点间的距离。

比如说节点2，节点3之间的直接距离为6，用C语言表示就是g[2] [3] = 5。

假设两点间不连通呢？对应之前所讲的用任意不存在的值来表示不连通，比如无穷大，负数，0（当然首先得确保没有边权为0的边）

```
#define N 1005 //点的最大值
#define INF 0x3f3f3f3f
int g[N][N]; //用来表示图

int n; //点的个数
int m; //边的个数

void init() {
    //初始化，一开始都是不连通的
    //假设用无穷大表示不连通
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (i == j) g[i][j] = 0;
            else g[i][j] = INF;
        }
    }
}

void build() {
    //给出m条边，每条边包含的信息有两个顶点和对应的边权
    scanf("%d%d", &n, &m); //输入点的数量和边的数量
    init(); //先初始化
    for (int i = 1; i <= m; i++) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w); //u到v有一条边权为w的边
        a[u][v] = min(a[u][v], w);
        a[v][u] = min(a[v][u], w); //好习惯，防止重边

        //假设是无向边那么两个方向都要设置一下，如果是有向的话根据题目条件加一条就行
    }
}

void work(int x) {
    //图的遍历
    //假设要遍历所有与x直接相连的点
    for (int i = 1; i <= n; i++) {
        if (i == x) continue; //跳过自身
```

```
        if (g[x][i] == INF) continue; //不连通
        DoSomeWork(); //i这个点与x直接相连
    }
}

int main() {
    build(); //建图

    /*

    设计相应的算法解决问题

    */

    return 0;
}
```

此种方法通常最笨比。一般来说点的数量不超过1000时才能使用该方法，当点的数量级到达100000这个级别后该方法就用不了了（二维数组开不了这么大的空间）

vector实现邻接矩阵

不懂的先去看 https://www.bilibili.com/video/BV1PY411K7Vt?spm_id_from=333.999.0.0 这个视频中的vector和pair。

用 vector g[N] 来表示没有边权的图，用vector<pair<int, int>> g[N]来表示有边权的图。

假设a和b之间有一条边权为w的边，那么就

```
//无边权，无向
g[a].push_back(b);
g[b].push_back(a);

//有边权，无向
g[a].push_back(pair<int, int>(b, w));
g[b].push_back(pair<int, int>(a, w));

//有向不再赘述
```

可以看到比普通二维数组的好处是

1. 相对来说能省点空间
2. 不需要定义额外的非法状态来表示点之间的不连通关系（因为只有连通的才能记录进去）

```
#define N 1005 //点的最大值
typedef pair<int, int> pii;
int n; //点的数量
int m; //边的数量
vector<pii> g[MAXN]; //表示图的数据结构

void init() {
    //初始化
    for (int i = 0; i <= n; i++) {
        g[i].clear();
    }
}

```

```cpp
void build() {
    //图的建立
    scanf("%d%d", &n, &m); //输入点的数量和边的数量
    init(); //先初始化
    for (int i = 1; i <= m; i++) {
        int u, v, w; //u到v有一条边权为w的边
        scanf("%d%d%d", &u, &v, &w);
        g[u].push_back(pii(v, w));
        g[v].push_back(pii(u, w));

        //假设是无向边那么两个方向都要设置一下，如果是有向的话根据题目条件加一条就行
    }
}

void work(int x) {
    //图的遍历，假设要遍历x的所有邻点
    for (pii it : g[x]) {
        int y = it.first; //邻点
        int w = it.second; //连接x和y这条边的边权
        DoSomework();
    }
}

int main() {
    build();
    /*

    设计相应的算法解决问题

    */
}
```

和普通int二维数组大同小异，除非题目明确指出 **边的数量不超过xxx** ，否则依然存不下数据量在100000及以上范围下的点。

链式前向星实现邻接表（推荐）

依据链表来建图。

不讲算法原理，因为不重要，感兴趣的同学可以自行去洛谷，AcWing，牛客等平台找找相应资料或者百度上搜搜博客。

这种方法可以提高空间的利用率，就算是一个有很多点的稀疏图，数据结构也存的下。

```cpp
#define N 100005 //点的最大值
#define M 200005 //边的最大值。如果是无向图边的最大值要开乘以2

int head[N]; //表头，最大值依据点的最大值
int tot; //边的总数，同时用于给边编号。我这种写法编号从1开始，假设是无向图，那么正向边为奇数，反向边为偶数，奇数和奇数+1就对应一条边的两个方向。
struct EDGE {
    int to; //该边的另一个点，对应于from
    int next; //指向下一条边
    int val; //边权
}edge[M]; //依据边的最大值

int n, m; //点的数量和边的数量
```

```cpp

void add(int from, int to, int val) {
    //在图中加一条边
    edge[++tot].to = to;
    edge[tot].val = val;
    edge[tot].next = head[from];
    head[from] = tot;
}

void init() {
    //图的初始化
    tot = 0;
    for (int i = 1; i <= n; i++) head[i] = 0;
}

void build() {
    //图的建立
    scanf("%d%d", &n, &m);
    init();
    for (int i = 1; i <= m; i++) {
        int u, v, w; //u到v有一条边权为w的边
        add(u, v, w);
        add(v, u, w);

        //假设是无向边那么两个方向都要设置一下，如果是有向的话根据题目条件加一条就行
    }
}

void work(int now) {
    //遍历点now的所有邻点
    for (int i = head[now]; i; i = edge[i].next) {
        int to = edge[i].to; //邻点
        int val = edge[i].val; //边权
        DoSomework();
    }
}

int main() {
    build();
    /*

    设计相应的算法解决问题

    */
    return 0;
}

```

可以说，百分之九十九的图都能用该算法实现模型的建立。不排除特殊情况的存在，比如说这个https://vjudge.net/contest/483291#problem/C