蛟龙五班

图

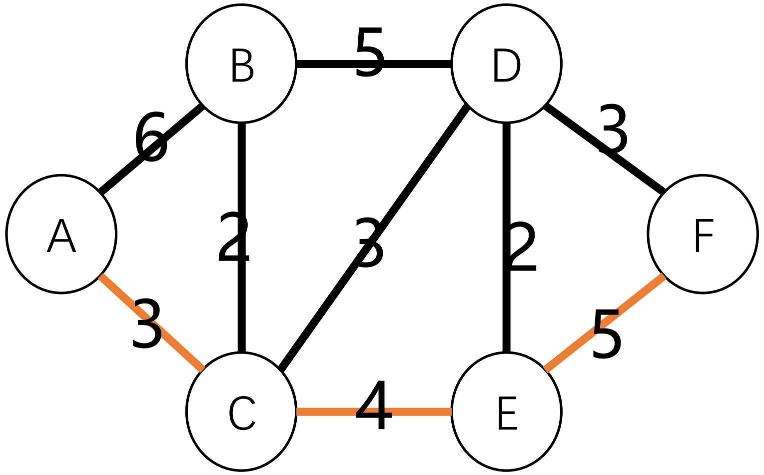
Mas

图中的路径

- 路径，即一系列点和边：
- $v_1, e_1, v_2, e_2, \cdots, v_{n-1}, e_{n-1}, v_n$
- 其中 e_i 是一条连接 v_i 和 v_{i+1} 的路径，即以 v_i 为起点，v_{i+1}为终点
- 称为v_1到v_n的路径
- 由于每条边有边长，因此路径的总长度为边长之和

图中的路径

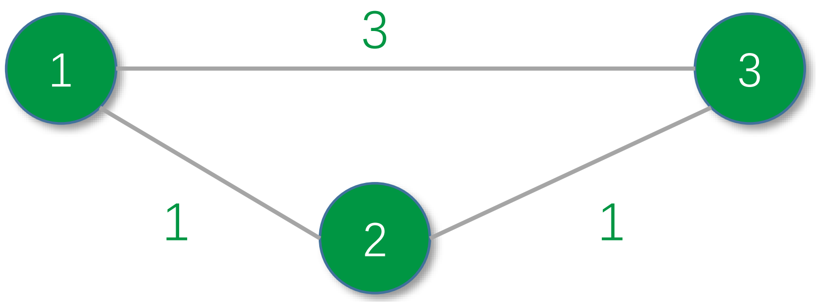
- 路径，即一系列点和边：
- $v_1, e_1, v_2, e_2, \cdots, v_{n-1}, e_{n-1}, v_n$
- 其中 e_i 是一条连接 v_i 和 v_{i+1} 的路径，即以 v_i 为起点，v_{i+1}为终点
- 称为v_1到v_n的路径



- 由于每条边有边长，因此路径的总长度为边长之和

最短路

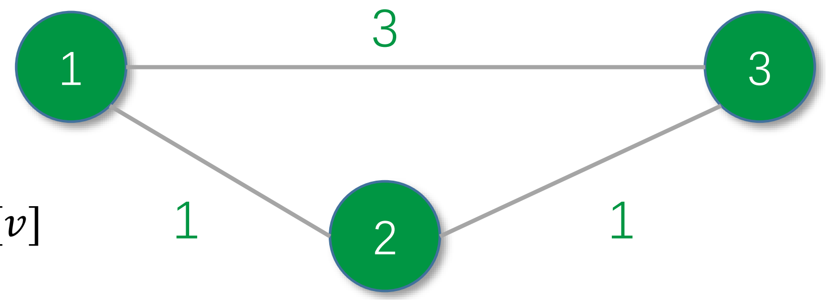
- 两点之间长度最短的路径，被称为**最短路径**
- 最短路径不会经过重复的边和点
- **单源最短路**问题：要求的是某个点到其他点的最短路径
- **多源最短路**问题：要求的是任意两点之间的最短路径



Floyd-Warshall(多源最短路)

$Floyd$(弗洛伊德)算法，是最简单的最短路径算法，可以计算图中任意两点间的最短路径。

$Floyd$的时间复杂度是$O\ (N^3)$，适用于出现负边权的情况。



$dis[u][v]$记录从u点到v点的最短路径

枚举一个点k，如果k能够将u、v松弛，那么更新 $dis[u][v]\ =\ dis[u][k]\ +\ dis[k][v]$

K需要在最外层, 只有当 $dis[i][k]$ 和 $dis[k][j]$ 已经被确定时,才能被正确松弛

若 k 在内层, 会出现 $dis[i][k]$ 和 $dis[k][j]$ 未被确定, 导致未能正确松弛

5 5
1 2 4
2 3 5
1 5 3
4 3 1
4 5 2
1 3

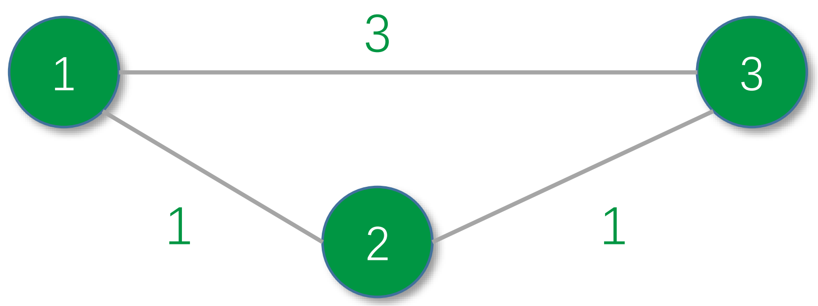
Floyed-Warshall(多源最短路)

Floyed(弗洛伊德)算法，是最简单的最短路径算法，可以计算图中任意两点间的最短路径。Floyed的时间复杂度是O (N3)，适用于出现负边权的情况。

三层循环，第一层循环中间点k，第二第三层循环起点终点i、j，算法的思想很容易理解：如果点i到点k的距离加上点k到点j的距离小于原先点i到点j的距离，那么就用这个更短的路径长度来更新原先点i到点j的距离。

在上图中，因为dis[1][2]+dis[2][3]<dis[1][2]，所以就用dis[1][3]+dis[3][2]来更新原先1到2的距离。

我们在初始化时，把不相连的点之间的距离设为一个很大的数，不妨可以看作这两点相隔很远很远，如果两者之间有最短路径的话，就会更新成最短路径的长度。Floyed算法的时间复杂度是O(N^3)。



#1258、最短路径问题

【题目描述】

给出一个有向图 $G=(V,E)$，和一个源点 $v_0 \in V$，请写一个程序输出 v_0 和图 G 中其它顶点的最短路径。只要所有的有向环权值和都是正的，我们就允许图的边有负值。顶点的标号从 1 到 n（n 为图 G 的顶点数）。

【输入】

第 1 行：一个正数 n（$2 \le n \le 80$），表示图 G 的顶点总数。

第 2 行：一个整数，表示源点 $v0$（$v_0 \in V$，v_0 可以是图 G 中任意一个顶点）。

第 3 至第 $n+2$ 行，用一个邻接矩阵 W 给出了这个图。

【输出】

共包含 $n-1$ 行，按照顶点编号从小到大的顺序，每行输出源点 v_0 到一个顶点的最短距离。每行的具体格式参照样例。

【输入样例】

```
5
1
0 2 - - 10
- 0 3 - 7
- - 0 4 -
- - - 0 5
- - 6 - 0
```

【输出样例】

```
(1 -> 2) = 2
(1 -> 3) = 5
(1 -> 4) = 9
(1 -> 5) = 9
```

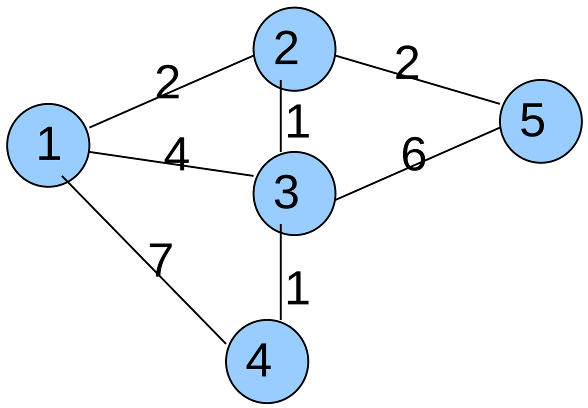
```cpp
#include <bits/stdc++.h>
using namespace std;
int dis[1001][1001], n, s;
char c;
int main()
{
    cin >> n >> s;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            if (!scanf(" %d", &dis[i][j]))
                dis[i][j] = 0x3f3f3f3f;

    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                dis[i][j] = min(dis[i][j], dis[k][j] + dis[i][k]);
    for (int i = 1; i <= n; i++)
        if (s != i)
            printf("(%d -> %d) = %d\n", s, i, dis[s][i]);
    return 0;
}
```

Dijkstra(单源最短路)

主要思想是，将结点分成两个集合：已确定最短路长度的，未确定的。
一开始第一个集合里只有 S。
然后重复这些操作：

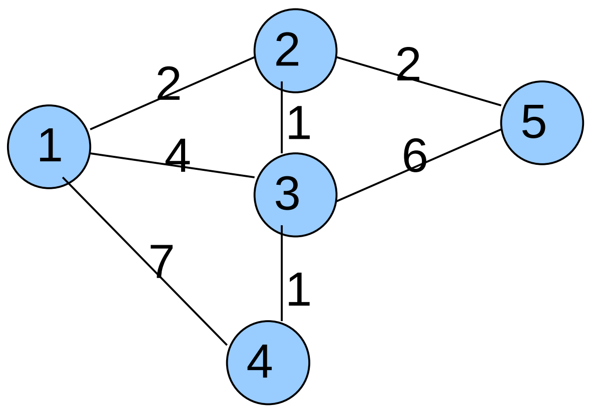
- 对那些刚刚被加入第一个集合的结点的所有出边执行松弛操作。
- 从第二个集合中，选取一个最短路长度最小的结点，移到第一个集合中。

直到第二个集合为空，算法结束。



Dijkstra(单源最短路)

从起点到一个点的最短路径一定会经过至少一个“中转点”（例如下图1到5的最短路径，中转点是2。特殊地，我们认为起点1也是一个“中转点”）。显而易见，如果我们想求出起点到一个点的最短路径，那我们必然要先求出中转点的最短路径（例如我们必须先求出点2的最短路径后，才能求出从起点到5的最短路径）。



我们把点分为两类，一类是已确定最短路径的点，称为“白点”，另一类是未确定最短路径的点，称为“蓝点”。如果我们要求出一个点的最短路径，就是把这个点由蓝点变为白点。从起点到蓝点的最短路径上的中转点在这个时刻只能是白点。

Dijkstra的算法思想，就是一开始将起点到起点的距离标记为0，而后进行n次循环，每次找出一个到起点距离dis[u]最短的点u，将它从蓝点变为白点。随后枚举所有的蓝点vi，如果以此白点为中转到达蓝点vi的路径dis[u]+w[u][vi]更短的话，这将它作为vi的“更短路径”dis[vi]（此时还不确定是不是vi的最短路径）。

就这样，我们每找到一个白点，就尝试着用它修改其他所有的蓝点。中转点先于终点变成白点，故每一个终点一定能够被它的最后一个中转点所修改，而求得最短路径。

Dijkstra(单源最短路)

dis[i]记录从起点s到i点的最小距离
dis[s] = 0,其他元素置为极大值(不连通的情况)

是否能够优化?

```
bool vis[2001];
int n, m, u, v, c, dis[2001], g[2001][2001];
void dijkstra(int s)
{
    dis[s] = 0;
    vis[s] = true;
    for (int i = 2; i <= n; i++)
        dis[i] = g[s][i];
    for (int i = 1; i <= n; i++)
    {
        int minDis = 0x3f3f3f3f;
        for (int j = 2; j <= n; j++)
            if (!vis[j] && minDis > dis[j]) // 最出S中的最小距离u
            {
                minDis = dis[j];
                u = j;
            }
        vis[u] = true;
        for (int v = 1; v <= n; v++) //使用 u 来对进行松弛
            if (!vis[v] && dis[u] + g[u][v] < dis[v])
                dis[v] = dis[u] + g[u][v];
    }
}
```

#1881 城市路

【题目描述】

Mas 被邀请参加一个会议，是在城市 n ，而 Mas 当前所处的城市为 1 ,附近还有很多城市 $2 \sim n-1$ ，有些城市之间没有直接相连的路，有些城市之间有直接相连的路，这些路都是双向的，当然也可能有多条。

现在给出直接相邻城市的路长度， Mas 想知道从城市 1 到城市 n ，最短多少距离。

【输入】

输入 n , m ，表示 n 个城市和 m 条路;

接下来 m 行，每行 a b c ，表示城市 a 与城市 b 有长度为 c 的路。

【输出】

输出 1 到 n 的最短路。如果 1 到达不了 n ，就输出 -1 。

【输入样例】

```
5 5
1 2 20
2 3 30
3 4 20
4 5 20
1 5 100
```

【输出样例】

```
90
```

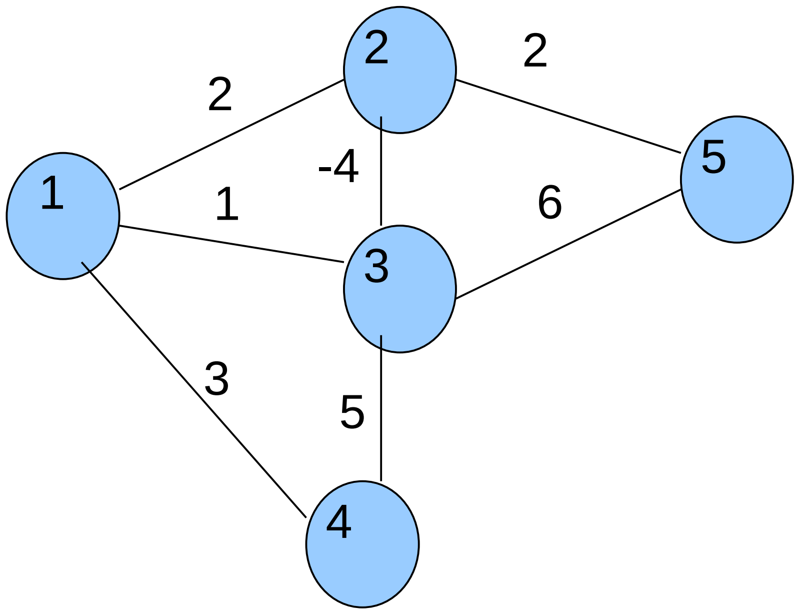
【数据规模和约定】

对于全部数据： $1 \leq n \leq 2000, 1 \leq m \leq 10000, 0 \leq c \leq 10000$

Dijkstra(单源最短路)

Dijkstra无法处理边权为负的情况，例如右图这个例子。

2到3的边权值为-4，显然从起点1到3的最短路径是-2（1→2→3），但是dijskstra在第二轮循环开始时会找到当前dis[i]最小的点3，并标记它为白点。这时的dis[3]=1，然而1却不是从起点到点3的最短路径。因为3已被标记为白点，最短路径值dis[3]不会再被修改了，所以我们在边权存在负数的情况下得到了错误的答案!



Bellman-Ford(单源最短路)

简称Ford（福特）算法，同样是用来计算从一个点到其他所有点的最短路径的算法，是一种单源最短路径算法。能够处理存在负边权的情况，但无法处理存在负权回路的情况。

这个算法主要是构造一个最短路径长度数组的序列: dist[1][u], dist[2][u], · · · , dist[n − 1][u]。

其中 dist[k][u] 表示从源 s 到 u 至多经过 k 条边的最短路径的长度。

我们可以得到: dist[1][u] = w[s][u] // 只走一条边从 s 到 u

dist[k][u] = min(dist[k − 1][u] , dist[k − 1][v] + w[v][u])，其中 (v, u) 是图中的一条边 由于每次计算 dist 数组复杂度都是 O(m) 的，总的复杂度就是 O(nm)。

Bellman-Ford(单源最短路)

```cpp
#include <bits/stdc++.h>
using namespace std;
int n, m, ans, dis[1001], u, v, w;
struct node
{
    int u, v, w;
} g[100001];

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= m; i++)
    {
        cin >> u >> v >> w;
        g[i].u = u, g[i].v = v, g[i].w = w;
    }
    memset(dis, 0x3f, sizeof(dis));
    dis[1] = 0;
    for (int i = 1; i <= n - 1; i++)
        for (int j = 1; j <= m; j++)
            dis[g[j].v] = min(dis[g[j].v], dis[g[j].u] + g[j].w);
    return 0;
}
```

SPFA(单源最短路)

假设 我们现在已经得到了 Bellman-Ford 算法某个阶段的 dist 数组，然后我们发现了一条 s 到 u 的距离比 dist[u] 更加短的路径，更新了 dist[u]。

那么接下来直接受到影响的就是与 u 直接关联的顶点 v。

对于所有的 dist[u] + w[u][v] < dist[v]，s 到 v 的最短路就可以利用 s 到 u 的最短路 加上 u 到v 的边来更新。

这样的话与 v 直接关联的顶点又会受到影 响......不断这样持续下去直到最后没有顶点能被影响。

那么一我们利用队列存储这些需要更新的结点，每次从队 列中取出一个结点，计算是否有结点需要更新，如果有，并且这个节点不在队列中，那么就将它加入队列。

这样的算法被称为 SPFA(Shortest Path Faster Algorithm)——一种带队列优化的 Bellman-Ford 算法。

#1884 最短路(SPFA)

【题目描述】

给定 M 条边， N 个点的带权无向图。求 1 到 N 的最短路。

【输入】

第一行：$N, M(N \leq 100000，M \leq 500000)$；

接下来 M 行 3 个正整数：a_i, b_i, c_i 表示 a_i, b_i 之间有一条长度为 c_i 的路， $c_i \leq 1000$ 。

【输出】

一个整数，表示 1 到 N 的最短距离。

【输入样例】

```
4 4
1 2 1
2 3 1
3 4 1
2 4 1
```

【输出样例】

```
2
```

【提示】

注意图中可能有重边和自环,数据保证 1 到 N 有路径相连。

实验舱

LAB

青少年编程

走近科学 走进名校

谢谢观看