图论入门与最短路

2020年1月19日

黄哲威 hzwer

北京大学16级计算机科学

自我介绍

北京大学16级计算机科学方向

计算概论 A，数据结构与算法 A ，算法设计与分析讨论班助教

NOI银牌，CTSC金牌，ACM区域赛金牌

旷视科技 算法工程师 计算机视觉

hzwer.com

课程安排 1

STL 简单入门

图的概念与两种存储方式

图的遍历

单源，多源最短路算法，掌握拆点，分层的处理技巧

前置技能 - STL 模板

- vector 是一个动态开空间的数组
- pair 和 greater<>
- priority_queue，二叉堆，可以在 O(logn) 的时间内插入删除，O(1) 的时间查询最小值
- map 可以当一个哈希表，使用形如一个下标范围扩大的数组
- 参考 https://wenku.baidu.com/view/93f33b3b192e45361066f5eb.html

前置技能 - STL 模板

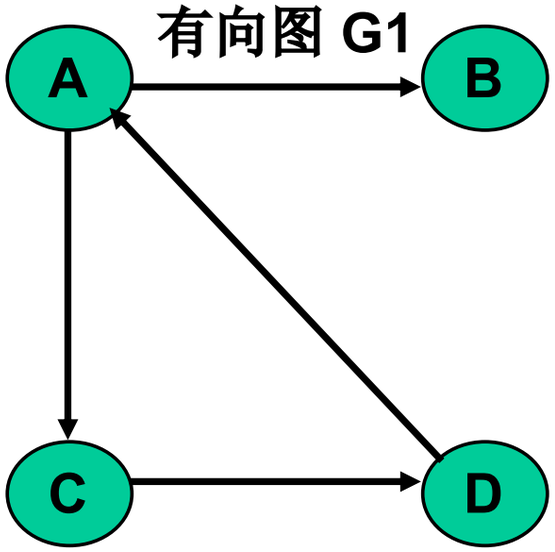
```
#include<vector>
#include<iostream>
#include<algorithm>
#define pa pair<int, int>
using namespace std;
vector<pa> v;
int main()
{
    for(int i = 1; i <= 10; i++)
        v.push_back(make_pair(rand(), i));
    sort(v.begin(), v.end());
    for(int i = 0; i < v.size(); i++)
        cout << v[i].first << ' ' << v[i].second << endl;
    return 0;
}
```

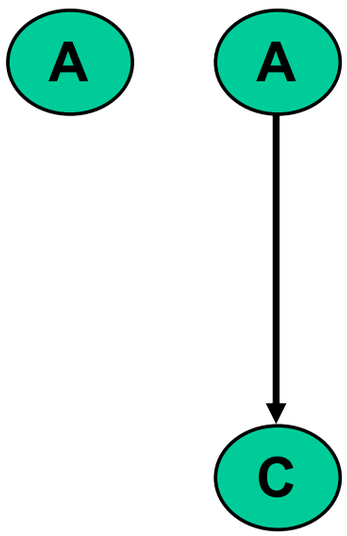
前置技能 - STL 模板

```
#include<queue>
#include<iostream>
using namespace std;
priority_queue<int, vector<int> >q; // 默认大根堆
// priority_queue<int, vector<int>, greater<int> >q; // 小根堆
int main()
{
    for(int i = 1; i <= 10; i++)
        q.push(rand());
    for(int i = 1; i <= 10; i++)
    {
        cout << q.top() << endl;
        q.pop();
    }
    return 0;
}
```

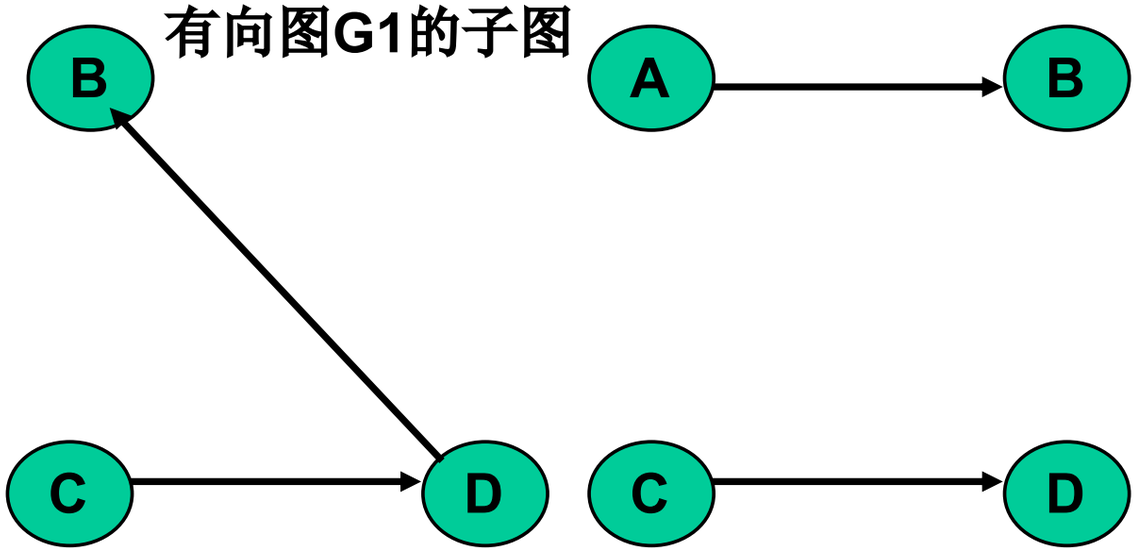
基本概念

有向图 G1

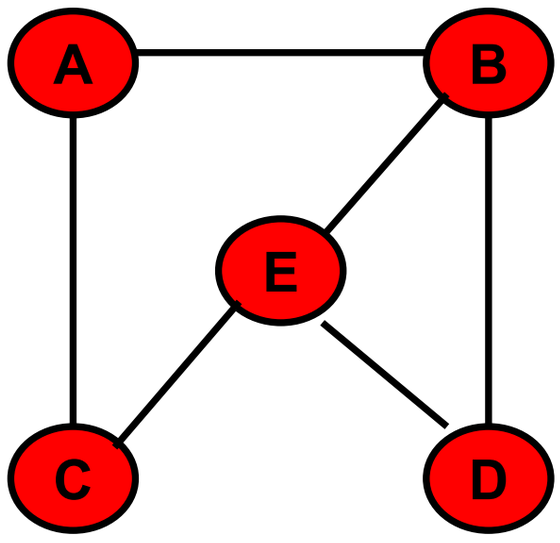




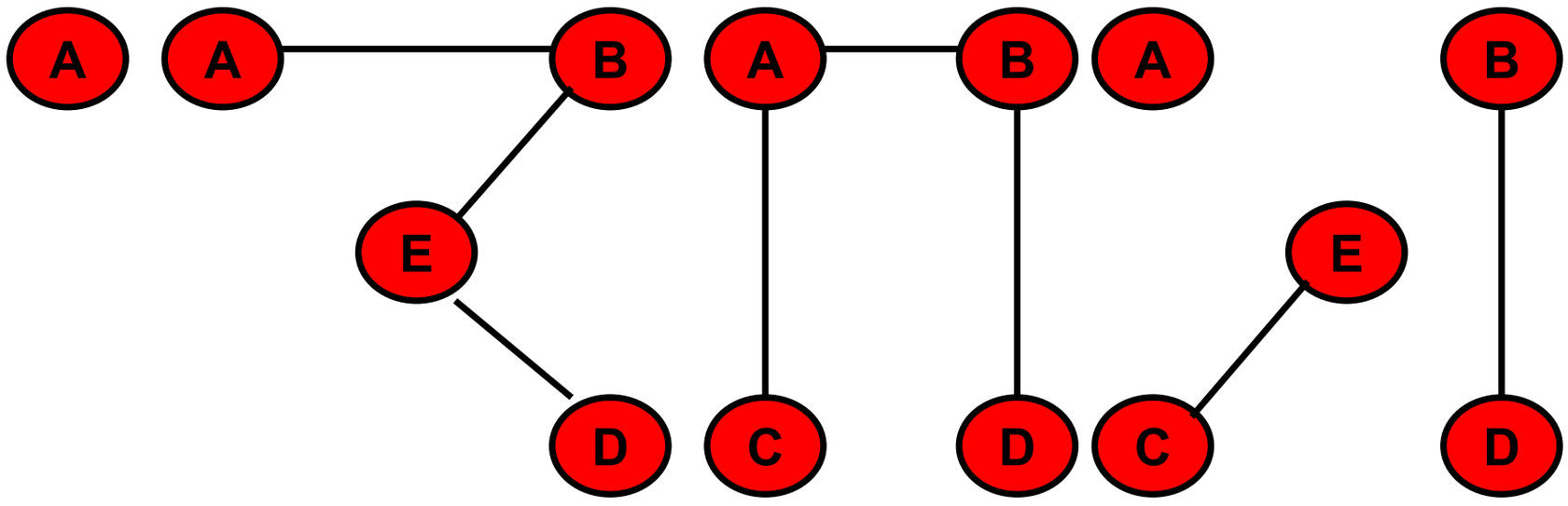
有向图G1的子图



无向图 G2



无向图G2的子图



基本概念

- 顶点的度:在无向图中，某个顶点的度是与它相关联的边的数目，或者说是它邻居的个数。在有向图中，一个顶点的出度是以它为起始的边的数目， 入度是以它为终止的边的数目。
- 简单路径:顶点不重复的路径。
- 自环:从某个顶点出发连向它自身的边。
- 环:从某个顶点出发再回到自身的路径，又称回路。
- 重边:从一个顶点到另一个顶点有两条边直接相连

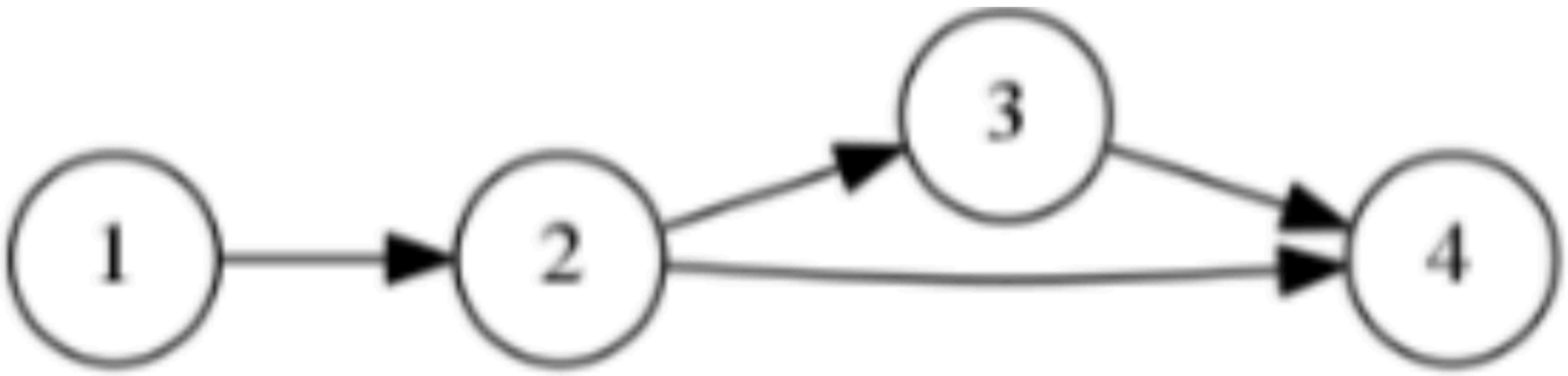
基本概念

- 在无向图中，若从顶点 u 到 v 存在路径，那么称顶点 u 和 v 是连通的。

- 如果无向图中任意一对顶点都是连通的，那么称此图为连通图。

- 如果一个无向图不是连通的，则称它的一个极大连通子图为连通分量。这里的极大是指顶点个数极大。

图的邻接矩阵存储



图可以直接用二维数组来存储。具体来说，如果一张图有 n 个结点，那么就使用 n*n 的二维数组来存储。

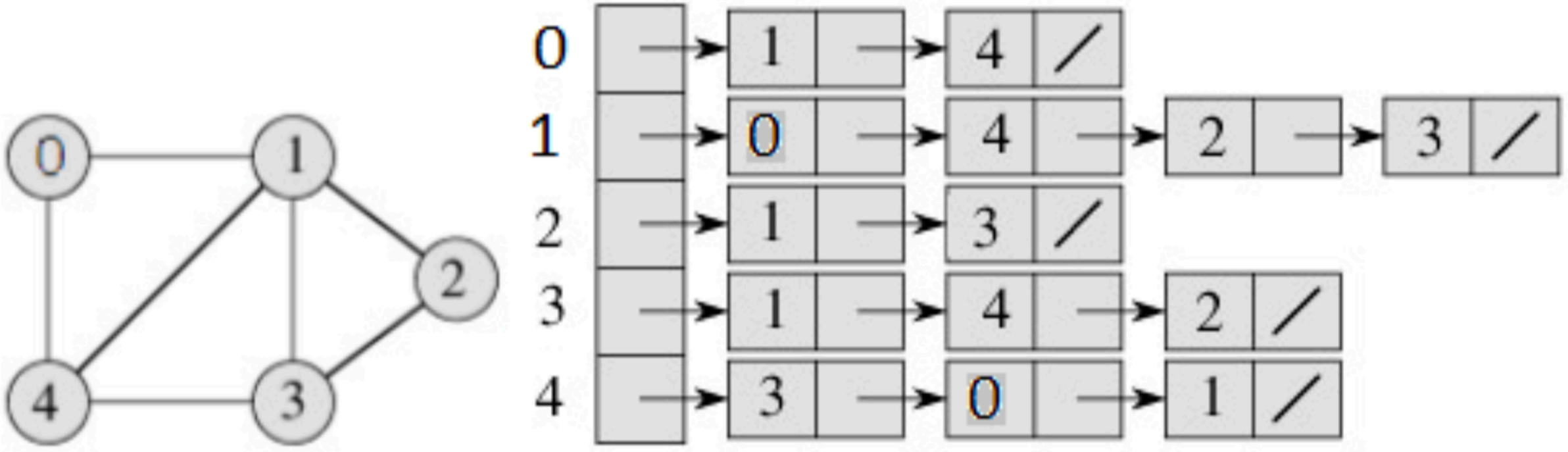
数组的每个元素的值代表了对应的边的数量。例如上面这张图就可以存储如下:

0 1 0 0
0 0 1 1
0 0 0 1
0 0 0 0

图的链表存储

图还可以用邻接表来存储。也就是每个结点维护一个链表，这个链表存储着以当前结点为开头的边。

通常情况下我们都是用这种方法来存储图的。



图的链表存储

```
#include <vector>
vector<int> e[N];
int main()
{
    cin >> n >> m;
    for(int i = 1; i <= m; i++)
    {
        int u, v;
        cin >> u >> v;
        e[u].push_back(v);
        e[v].push_back(u);
    }
    return 0;
}
```

图的遍历

- 一张无向图，所有边的长度都是 1
- 求 s 点到其它所有点的最短路径
- 可以对图进行广度优先搜索，得到分层图
- 如果只是遍历整个连通块，通常用深度优先搜索

广度优先搜索

```
vector<int> e[N];
int q[N], vis[N], dis[N];
void bfs(int x)
{
    int head = 0, tail = 1;
    q[0] = x; vis[x] = 1;
    while(head != tail)
    {
        int u = q[head]; head++;
        for(int i = 0; i < e[u].size(); i++)
        {
            int v = e[u][i];
            if(!vis[v])
            {
                dis[v] = dis[u] + 1;
                q[tail++] = v;
                vis[v] = 1;
            }
        }
    }
}
```

深度优先搜索

```
vector<int> e[N];
int vis[N];
void dfs(int x)
{
    vis[x] = 1;
    for(int i = 0; i < e[x].size(); i++)
        if(!vis[e[x][i]])
            dfs(e[x][i]);
}
```

- 特别注意：不要用深度优先搜索求最短路！

单源最短路 Bellman-Ford

给定了一个边带有权值(可以为负数)的有向图(不包含负环)和一个指定的起点 s。要求求出从 s 到其余各点的最短路径长度。

Bellman-Ford 算法是一个求解单源最短路问题的算法。

我们可以肯定最短路径包含的边的条数不会超过 n-1 个，或者说结点不超过 n 个。

如果超过这个数，那么肯定形成了一个环，又因为这个环权值是正的， 我们可以将路径上这个环删除，路径长度就会变小。

单源最短路 Bellman-Ford

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], min (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)。

单源最短路 SPFA

事实上，你会发现我们每次进行的计算可以看成是一次“松弛”。 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。

那么一个优化就是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，并且这个结点不在队列中，那么就将它加入队列。

这样的算法被称为 SPFA——一种带队列优化的 Bellman-Ford 算法。

单源最短路 SPFA

在竞赛中大多数人会选择用 SPFA 作为单源最短路的算法，主要原因在于它比较好写，而且通常情况下跑得比较快。但是 SPFA 的复杂度实际最坏情况和 Bellman-Ford 相同。

非常重要的一点就是 SPFA 的队列需要使用循环队列，虽然最多队列里只会有 n 个点，但是每个点可能会入队多次。

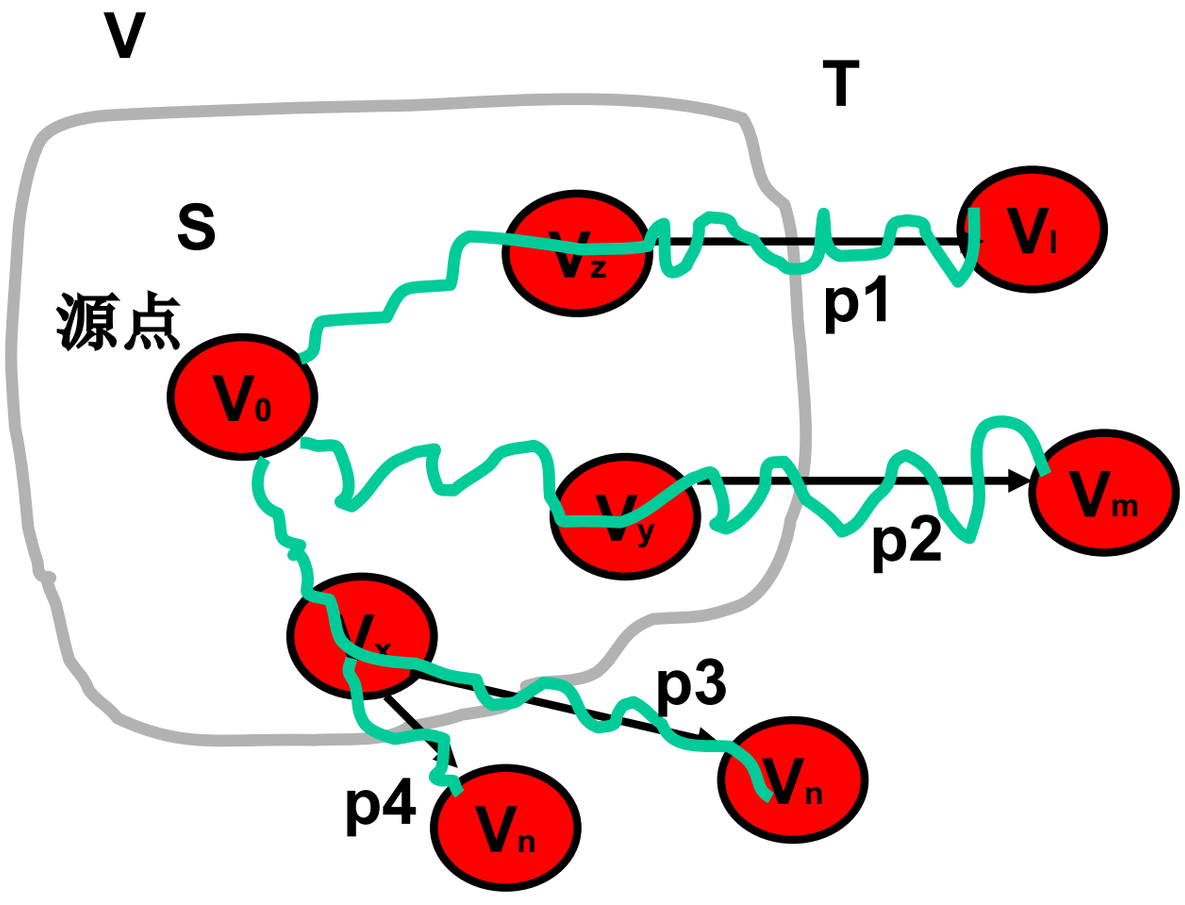
单源最短路 SPFA

```
vector<int> e[N], w[N];
int q[N], dis[N];
bool inq[N];
void spfa(int s){
    memset(dis, 127, sizeof(dis));
    int head = 0, tail = 1;
    q[0] = s; dis[s] = 0; inq[s] = 1;
    while(head != tail){
        int u = q[head]; head++; if(head == N)head = 0;
        for(int i = 0; i < e[u].size(); i++){
            int v = e[u][i];
            if(dis[u] + w[u][i] < dis[v]){
                dis[v] = dis[u] + w[u][i];
                if(!inq[v]){
                    q[tail++] = v;
                    if(tail == N)tail = 0;
                    inq[v] = 1;
                }
            }
        }
        inq[u] = 0;
    }
}
```

单源最短路 Dijkstra

如果有向图的边权值全为正数，那么有一种复杂度有保证的单源最短路算法——Dijkstra 算法。它的复杂度是 O(mlogn)。

Dijkstra 算法维护了一个未访问的结点集合 T 以及一个从 s 到结点 u 的当前距离 dist[u]。



单源最短路 Dijkstra

1 将除源外所有结点当前距离设置为无穷大，将源 s 的当前距离设置为 0，将当前节点设置为源 s。

2 从当前结点 u 开始，找出所有在未访问集合 T 中与 u 有边 (u,v) 的结点 v。如果 dist[u] + w[u][v] < dist[v]，那么就更新 dist[v]。

3 将当前节点从 T 中删除（打个标记），并且找到在 T 中 dist 最小的结点设置为新的当前节点。

4 重复 (2) 和 (3) 直到 T 成为空集。

单源最短路 Dijkstra

```
#include<queue>
#include <vector>
vector<int> e[N], w[N];
int dis[N];
bool vis[N];
#define pa pair<int, int>
priority_queue<pa, vector<pa>, greater<pa> >q;
void dijkstra(int s)
{
    memset(dis, 127, sizeof(dis));
    dis[s] = 0; q.push(make_pair(0, s));
    while(!q.empty())
    {
        int u = q.top().second; q.pop(); //找当前dis最小的结点
        if(vis[u])continue; vis[u] = 1; //vis=1表示这个点的最短路确定过了
        for(int i = 0; i < e[u].size(); i++)
        {
            int v = e[u][i];
            if(!vis[v] && dis[u] + w[u][i] < dis[v])
            {
                dis[v] = dis[u] + w[u][i];
                q.push(make_pair(dis[v], v));
            }
        }
    }
}
```

单源最短路 Dijkstra

它的正确性在于，在未访问集合 T 中结点的 dist 是从 s 开始经过已经访问集合中的结点到达它的最短路。

如果选出的当前结点 u 的 dist 不是最终的最小值，那么它最终的最短路一定是要经过一个此时 T 中的其它结点 v 再到 u。这时 dist[v] < dist[u]，这和 u 是 dist 最小的结点矛盾!

多源最短路 Floyd

如果要求所有顶点间的最短路，可以对每个顶点跑单源最短路，或者使用比较简单的 Floyd 算法。

我们设 d[k][i][j] 为除了 i 和 j 外只经过前 k 个结点，从 i 到 j 的最短路。

显然可以知道 d[0][i][j] = w[i][j]。 那么当加入了一个顶点 k 之后，最短路如果有变化的话一定是以 k 为中间顶点，那么可以得到

d[k][i][j] = min(d[k – 1][i][j], d[k – 1][i][k] + d[k – 1][k][j])

这个算法的复杂度是 O(n^3)。

多源最短路 Floyd

```
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[i][k] + dis[k][j]);
```

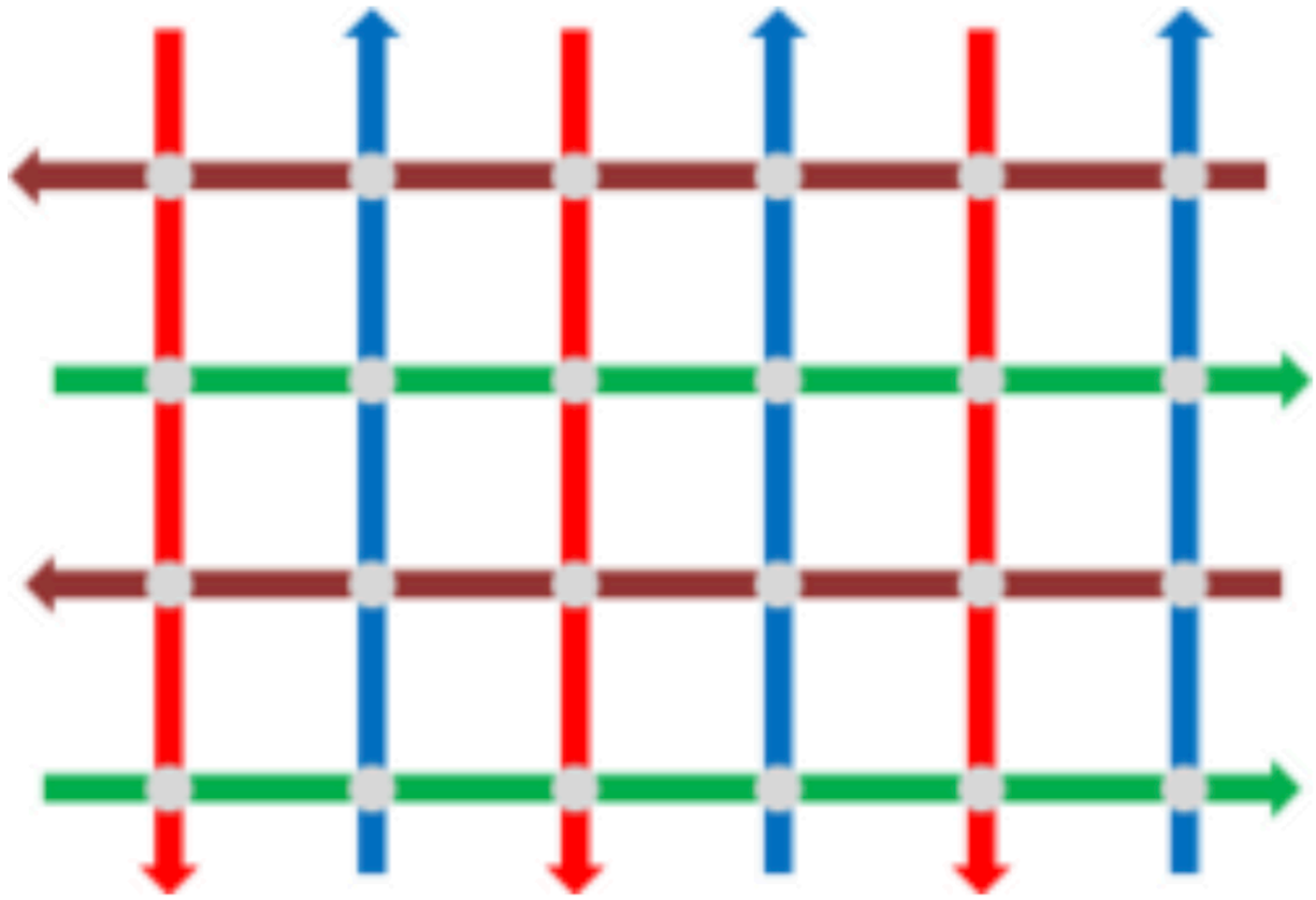
记得 k 要在最外层循环，本质上 Floyd 是一个滚动数组优化的 dp！

练习题

CF475B. Strongly Connected City

给 n 条水平和 m 条竖直的单向街道，其互相交叉为 n*m 个结点，问这些结点是否都能互相到达。

$n, m \le 20$



CF475B. Strongly Connected City

从每个结点开始 dfs，最后看每个结点是否最终被搜到 n*m 次。

实际上只需要判断外环路是否成环即可。

437C.The Child and Toy

n 个带权点，m 条边的无向图，删除一个点的代价是与这个点相邻的，且没有被删除的点权和。

求将所有点删除的最小代价。

$1 \le n \le 1000$, $1 \le m \le 2000$

437C.The Child and Toy

按照从大到小的次序删点。

发现每条边的贡献是连接的两个点的权值的最小值。

CERC1998. Invitation Cards

在 n 个点 m 条正权边的有向图中，共有 n 个人要执行任务，第 i 个人的路线为 1 → i → 1，求最小总路程。

1 ≤ n, m ≤ 10^6。

CERC1998. Invitation Cards

最小总路程？每个人来回路程都最小。

点 1 到每个点的最短路程？单源最短路。

每个点到点 1 的最短路程？把边反向，仍然是单源最短路。

时间复杂度？O(mlogn)

最小乘积路

在 n 个点 m 条正权边的有向图中，求一条 s → t 的路，使得各边权乘积最小。

2 ≤ n ≤ 10^6，1 ≤ m ≤ 10^6，0 < 边权 < 100。

最小乘积路

权值变成对数后，仿照 Dijkstra

CF295B. Greg and Graph

给 n 个点，两两之间均存在有向边。按顺序删除编号 n ~ 1 的点及其所有连边，每次删除点之后，询问图中现存任意两点间最短路之和。

n <= 500

CF295B. Greg and Graph

时光倒流，将删点改为加点

对于每次加点，如何更新最短路?

注意 Floyd 的原理中 dis[k][i][j] 中 k 的定义

设新加入的点为x，则我们枚举 u, v（都是从 1 到 n），然后更新

dis[u][v] = min(dis[u][v], dis[u][x] + dis[x][v])

这样一来只要计算答案时，枚举的 u，v 都是当前已经加入的点，就能保证计算的所有最短路经过的点都是当前存在的点。

CF1214D. Treasure Island

一个 n * m 的网格图，有一些格有障碍，小 A 想要从网格图的左上角走到右下角，每次只能向下或向右走。

现在你可以在一些格放障碍来阻止他，问最少放几个障碍?

n*m <= 1000000

CF1214D. Treasure Island

首先发现答案是 0，1 或者 2

暴力做法，枚举在某个位置放一个障碍，再判断一下连通性

如果把图看成分层图，其实就是问，某一层是不是有一个割点

预处理删掉和起点终点不连通的点，看一下每一层还剩几个

复杂度 O(nm)

权值变换

n 点 m 边的图，要求从 1 号点走到 n 号点的最短路
每当走了一步后，所有边权（假设为 e_i）都会变成 $\frac{1}{1-e_i}$

$n \le 1000$，$m \le 10^4$，$1 <$ 边权 < 100。

权值变换

边权只有三种情况

于是可以拆点，将每个点拆成 3 个点，分别对应走的总步数模 3 的余数

然后直接跑最短路算法

NOIP2009. 最优贸易

n 个点，m 条边的有向图。每个点有物品的买入价格和卖出价格。现在要从 1 号点走到 n 号点，途中只买一个物品并在这之后卖出，问最多能赚多少钱?

$1 \le n \le 10^5, 1 \le m \le 5 \times 10^5$

NOIP2009. 最优贸易

两次 SPFA 处理出到某个点的路径上的最小买入价格以及其到终点路径上的最大卖出价格

其实可以处理出到某个点的路径上的最小买入价格后，枚举卖出的结点

JLOI2011.飞行路线

在 n 个点 m 条正权边的有向图中，求一条 s → t 的路，使得去掉路上 k 条边的权值后边权和最小（或者说有 k 条可以免费）。

$n \le 10^4$，$m \le 5 \times 10^4$，$0 \le k \le 10$。

JLOI2011.飞行路线

f(i, j) 表示从起点 s 到点 i 去掉路上 j 条边的权值后的最短路。

时间复杂度? O(km log kn)。

449B.Jzzhu and Cities

n 个点，m 条带权边的无向图，另外还有 n - 1 条特殊边，每条边连接 1 和 i。求可以删除这 n - 1 条边中的多少条，使得每个点到 1 的最短距离不变。

1 ≤ n, m ≤ 10^5

449B.Jzzhu and Cities

求 1 到所有结点的最短路，对于每条特殊边 1 到 x：

若特殊边长度大于 dis(x)，删除。

若特殊边长度等于 dis(x)，考察所有非特殊边 (x, y), 如果存在 dis(y) + w = dis(x)，删除。（本质上就是判断最短路是否唯一）

小奇回地球

给一个 n 个点 m 条边的有向图，将有向图的所有边权加上一个值（可以是负数），使得 1 -> n 的最短路大于等于 0 且尽量小

n ≤ 100, m ≤ 10^4，边权的绝对值小于 10^5

小奇回地球

枚举答案，然后判断图中最短路是否存在，即 1 -> n 的路径上是否有负环存在。可以先用 Floyd 传递闭包或者 dfs，将与1，n不连通的结点从图中去掉，然后用 SPFA 算法来判负环。

复杂度上界为 O(v*n^2*m)（v是边权）

这个算法复杂度过高，考虑对其进行优化

枚举答案可以改成二分，负环可以用深搜版的 SPFA 判O（n*m），最终复杂度上界为 O(logv*n*m)

666B.World Tour

给定一张边权为 1 的有向图，求四个不同点 A, B, C, D 使得 dis(A, B) + dis(B, C) + dis(C, D) 取最大值

$1 \le n \le 3000, 1 \le m \le 5000$

666B.World Tour

先 bfs 出任意两点的距离，f(i, 0 – 2) 表示从 i 出发的最远的三个点，g(i, 0 – 2) 表示到 i 的最远的三个点枚举 B 和 C，再枚举 3 * 3 个点对，选出最优的 A 和 D

保留三个点的原因是保证 A, B, C, D 互不相同

复杂度 O(n^2)

545E.Paths and Trees

n 个点，m 条边的无向图和一个点 u，找出若干条边，组成一个子图，要求这个子图中 u 到其他点的最短距离与在原图中的相等，并且要求子图所有边的权重和最小，求出最小值。

$1 \le n, m \le 3 * 10^5$

545E.Paths and Trees

求 u 到所有结点的最短路

记录下到每个结点最短路上的最后一条边(若有多条，取最小的)

答案是以 u 为根的一棵最短路径树