哈爾濱工程大學
HARBIN ENGINEERING UNIVERSITY

图论基础

主讲人：李圳达

ACM校队专用

什么是图？

点用边连起来就叫图，严格意义上讲，图是一种数据结构

图 (Graph) 是由若干给定的顶点及连接两顶点的边所构成的图形，这种图形通常用来描述某些事物之间的某种特定关系。顶点用于代表事物，连接两顶点的边则用于表示两个事物间具有这种关系。

图的基本概念

有向图: 图的边有方向, 只能按箭头方向从一点到另一点

无向图: 图的边没有方向, 可以双向

结点的度: 无向图中与结点相连的边的数目, 成为结点的度

结点的入度: 在有向图中, 以这个结点为终点的有向边的数目

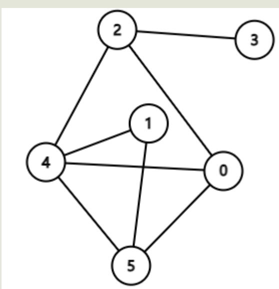
结点的出度: 在有向图中, 以这个结点为起点的有向边的数目

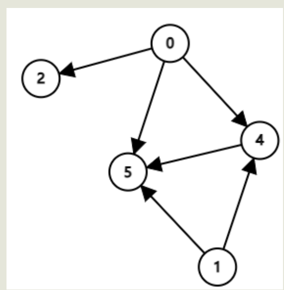
权值：边的”费用”或边的”长度”

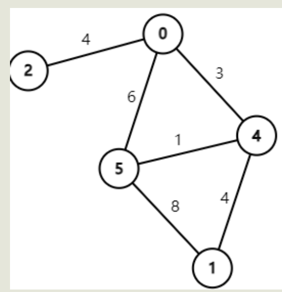
连通：如果图中结点U, V之间存在一条从U通过若干条边, 点到达V的通路, 则称U, V是连通的

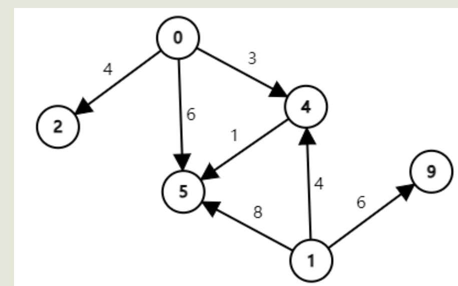
回路：起点和终点相同的路径, 成为回路, 或”环”

完全图：一个 n 阶的完全图含有 n * (n - 1) / 2 条边, 一个n 阶的完全有向图含有 n * (n - 1) 条边

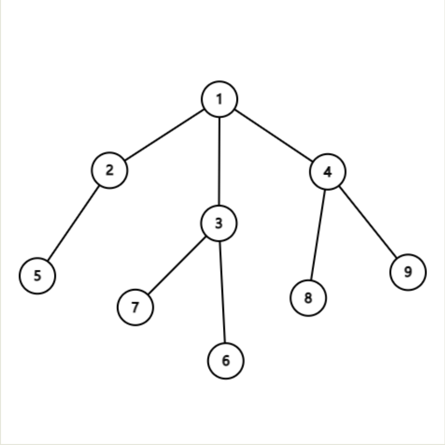






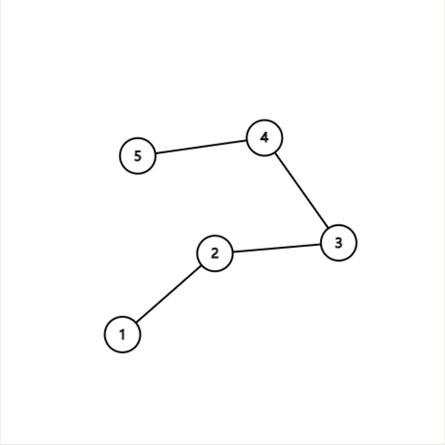


无向图　　有向图　　加权无向图　　加权有向图



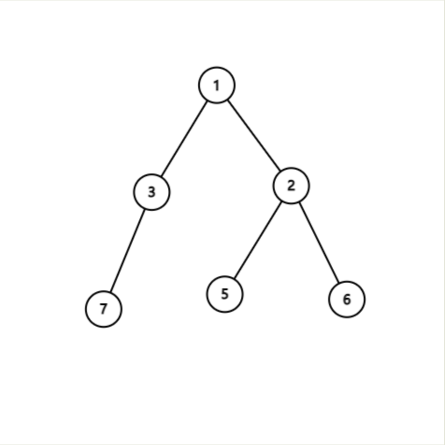
树

由 n 个结点n – 1条边构成的连通图



链

所有点度≤2



二叉树

所有点度≤3

图的存储

邻接矩阵

```
int M[105][105];
    int t;
    memset(M,0,sizeof(M));

    cin >> t;
    while(t--) {          //读入图
        int u, v;
        cin >> u >> v;
        M[u][v] = 1;      // 表现从结点 u 能到达结点 v
        //M[v][u] = 1;     若为无向图，则当作添加两条
有向边
    }
    int x, y;
    cin >> x >> y;
    if(M[x][y]) {     // 查询结点 x 与 y 是否连通
        cout << "Yes!" << endl;
    }
```

邻接矩阵

优点： • M[u][v]直接表示边(u,v),查询关系O(1)

• 更改或删除只要操作M[u][v] (O(1))

缺点： • 受限于顶点数n，需要n^2空间，稀疏图时浪费空间较多。

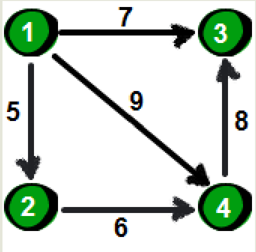
• 一个邻接矩阵中，只能记录顶点 w 到顶点 v 的一个关系（ 一个 基本型的二维数组中，无法 在同一对顶点之间添加两条边 ）

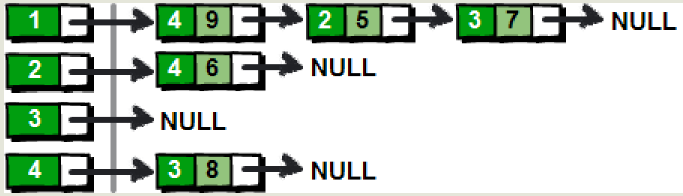
邻接表

核心思想:存边

把每个结点与其他相邻结点的边存下来(用链表)(但大多数时候用数组模拟)

在遍历一个结点所用连接的点时只需要按照链表逐一查找即可





邻接表的vector实现

```
vector<int> M[n];
    int u, v;
    int m;
    cin >> m;
    for(int i = 0; i < m; i++) {
        cin >> u >> v;
        M[u].pb(v);
    }
```

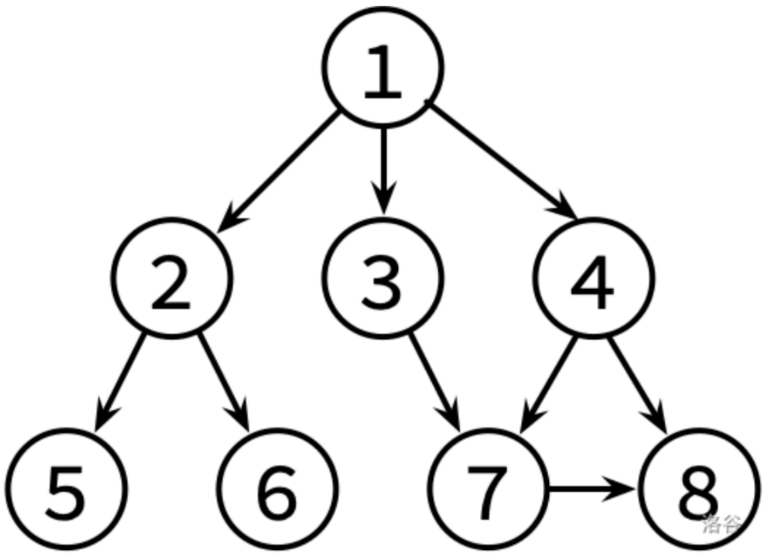
题目描述

展开

小K 喜欢翻看洛谷博客获取知识。每篇文章可能会有若干个（也有可能没有）参考文献的链接指向别的博客文章。小K 求知欲旺盛，如果他看了某篇文章，那么他一定会去看这篇文章的参考文献（如果他之前已经看过这篇参考文献的话就不用再看它了）。

假设洛谷博客里面一共有 $n(n \leq 10^5)$ 篇文章（编号为 1 到 n）以及 $m(m \leq 10^6)$ 条参考文献引用关系。目前小 K 已经打开了编号为 1 的一篇文章，请帮助小 K 设计一种方法，使小 K 可以不重复、不遗漏的看完所有他能看到的文章。

这边是已经整理好的参考文献关系图，其中，文献 X → Y 表示文章 X 有参考文献 Y。不保证编号为 1 的文章没有被其他文章引用。



请对这个图分别进行 DFS 和 BFS，并输出遍历结果。如果有很多篇文章可以参阅，请先看编号较小的那篇(因此你可能需要先排序)。

传送门

邻接表

优点： • 只需与边数成正比的内存空间

缺点： • 设u的相邻顶点数量为n，那么在调查顶点 u 与顶点 v 的关系时， 需要消耗 O(n) 来搜索邻接表。不过,大部分算法(如DFS和BFS等） 只需对特定顶点的相邻顶点进行一次遍历即可满足需求,因此影响 并不大

• 难以有效地删除边

链式前向星

链式前向星其实就是静态建立的邻接表，时间效率为O（m），空间效率也为O（m）。遍历效率也为O（m）。

链式前向星的实现

建边

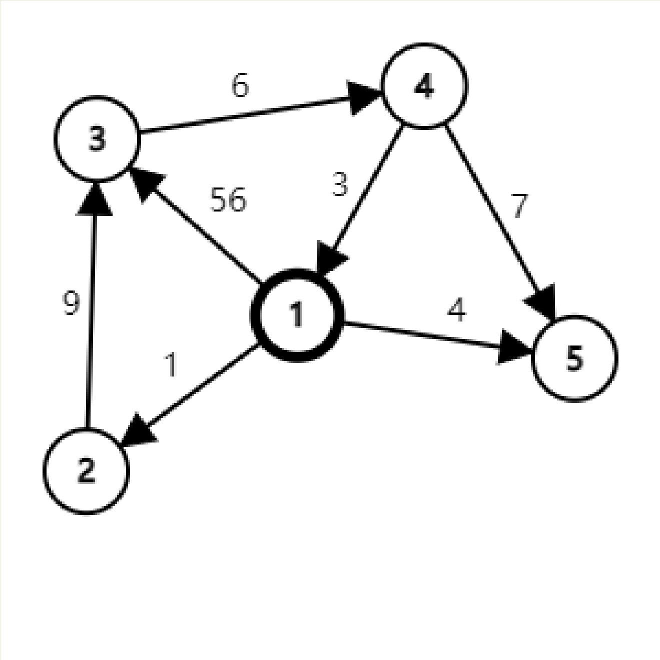
```
struct Edge {
    int to;   //终点
    int w;    // 边权
    int next;  // 同起点的上一条边编号
}edge[100100];//边集

int head[100100];  //head[i],表示以i为起点的最后一条边在边集数组的位置（编号）
int cnt;
```

链式前向星的实现

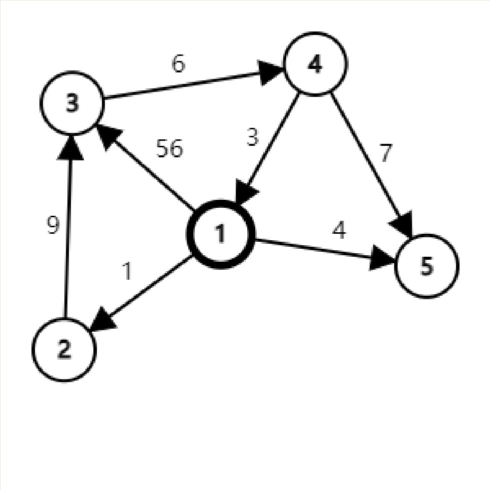
加边函数

```
void add_edge(int u, int v, int w) {  //u起点，v终点，w边权
    edge[++cnt].to = v; //终点
    edge[cnt].w = w; //权值
    edge[cnt].next = head[u];//以u为起点上一条边的编号，也就是与这个边起点相同的上一条边的编号
    head[u] = cnt;//更新以u为起点上一条边的编号
}
```

样例输入：




```
5 7

1 2 1
1 3 56
2 3 9
1 5 4
4 5 7
4 1 3
3 4 6
```



样例输入：

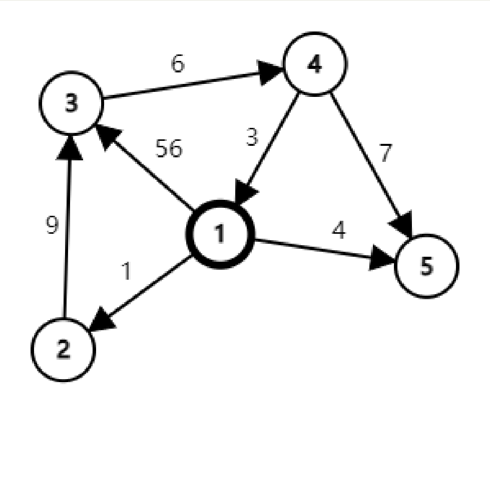
```
5 7

1 2 1
1 3 56
2 3 9
1 5 4
4 5 7
4 1 3
3 4 6
```

	v	w	nextt
edge (1)	2	1	0

head : 1 2 3 4 5

(1)

哈爾濱工程大學 HARBIN ENGINEERING UNIVERSITY



样例输入：

```
5 7

1 2 1
1 3 56
2 3 9
1 5 4
4 5 7
4 1 3
3 4 6
```

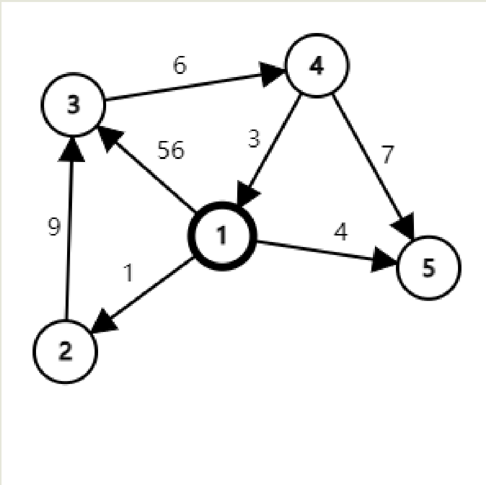
	v	w	nextt
edge (1)	2	1	0
edge (2)	3	56	1

head : 1 2 3 4 5

(1)

(2)

哈爾濱工程大學
HARBIN ENGINEERING UNIVERSITY



样例输入：

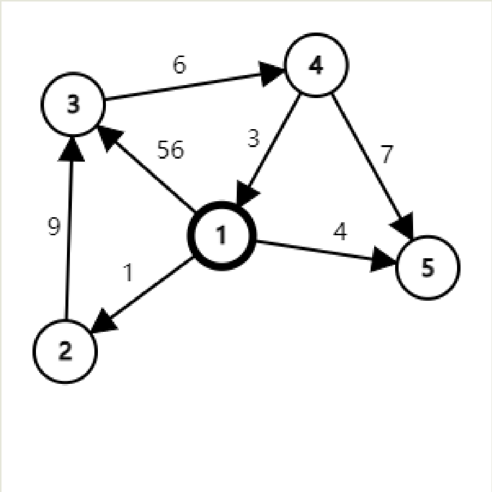
```
5 7

1 2 1
1 3 56
2 3 9
1 5 4
4 5 7
4 1 3
3 4 6
```

	v	w	nextt
edge (1)	2	1	0
edge (2)	3	56	1
edge (3)	3	9	0

head : 1 2 3 4 5

(1) (3)

(2)



样例输入：

```
5 7

1 2 1
1 3 56
2 3 9
1 5 4
4 5 7
4 1 3
3 4 6
```

	v	w	nextt
edge (1)	2	1	0
edge (2)	3	56	1
edge (3)	3	9	0
edge (4)	5	4	2

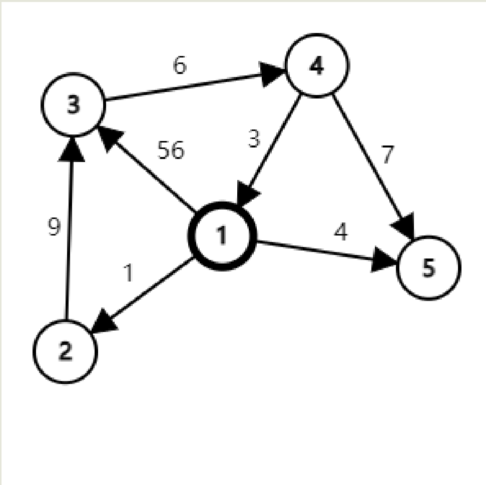
head : 1 2 3 4 5

(1) (3)

(2)

(4)

哈爾濱工程大學
HARBIN ENGINEERING UNIVERSITY



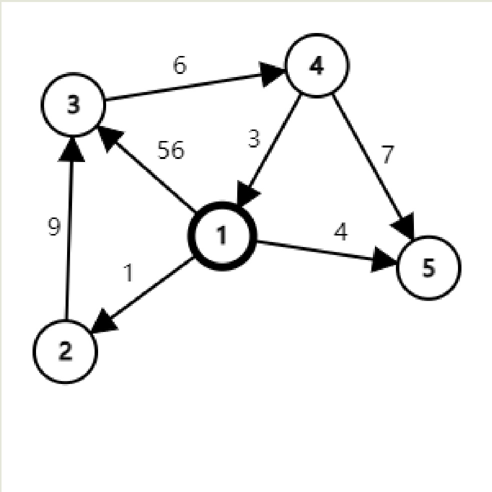
样例输入：

```
5 7

1 2 1
1 3 56
2 3 9
1 5 4
4 5 7
4 1 3
3 4 6
```

	v	w	nextt
edge (1)	2	1	0
edge (2)	3	56	1
edge (3)	3	9	0
edge (4)	5	4	2
edge (5)	5	4	0

head : 1 2 3 4 5

1	2	3	4	5
(1)	(3)		(5)	
(2)				
(4)				



样例输入：

```
5 7

1 2 1
1 3 56
2 3 9
1 5 4
4 5 7
4 1 3
3 4 6
```

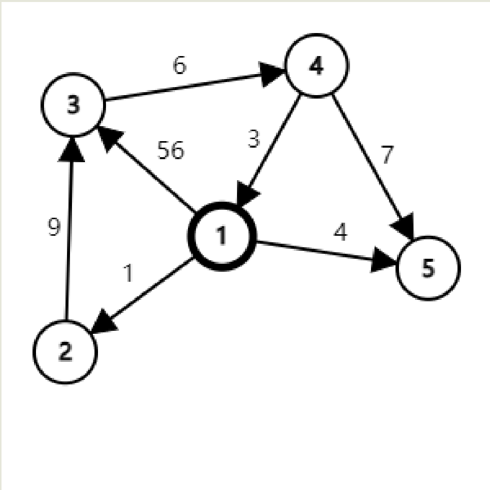
	v	w	nextt
edge (1)	2	1	0
edge (2)	3	56	1
edge (3)	3	9	0
edge (4)	5	4	2
edge (5)	5	4	0
edge (6)	1	4	5

head : 1 2 3 4 5

1 ← (1) ← (2) ← (4)

2 ← (3)

4 ← (5) ← (6)



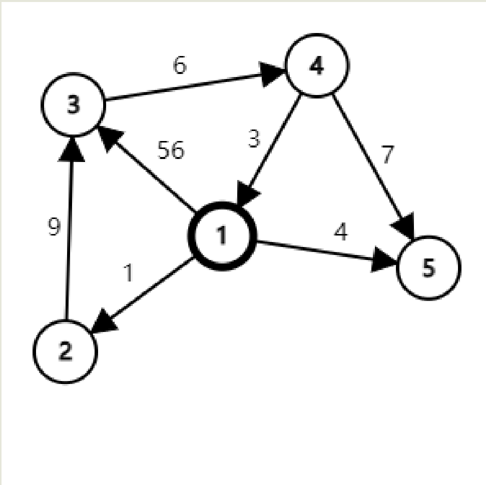
样例输入：

```
5 7

1 2 1
1 3 56
2 3 9
1 5 4
4 5 7
4 1 3
3 4 6
```

	v	w	nextt
edge (1)	2	1	0
edge (2)	3	56	1
edge (3)	3	9	0
edge (4)	5	4	2
edge (5)	5	4	0
edge (6)	1	4	5
edge (7)	4	6	0

head :

1	2	3	4	5
(1)	(3)	(7)	(5)	
(2)			(6)	
(4)				



样例输入：

```
5 7

1 2 1
1 3 56
2 3 9
1 5 4
4 5 7
4 1 3
3 4 6
```

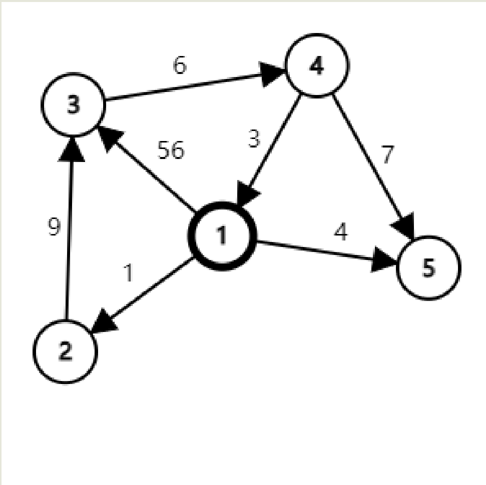
	v	w	nextt
edge (1)	2	1	0
edge (2)	3	56	1
edge (3)	3	9	0
edge (4)	5	4	2
edge (5)	5	4	0
edge (6)	1	4	5
edge (7)	4	6	0

1 2 3 4 5

(1)2 (5)5

(2)3

head : (4)5 (3)3 (7)4 (6)1 (0)



样例输入：

```
5 7

1 2 1
1 3 56
2 3 9
1 5 4
4 5 7
4 1 3
3 4 6
```

	v	w	nextt
edge (1)	2	1	0
edge (2)	3	56	1
edge (3)	3	9	0
edge (4)	5	4	2
edge (5)	5	4	0
edge (6)	1	4	5
edge (7)	4	6	0

	1	2	3	4	5
	(1)2			(5)5	
	(2)3				
head :	(4)5	(3)3	(7)4	(6)1	(0)

链式前向星的实现

遍历

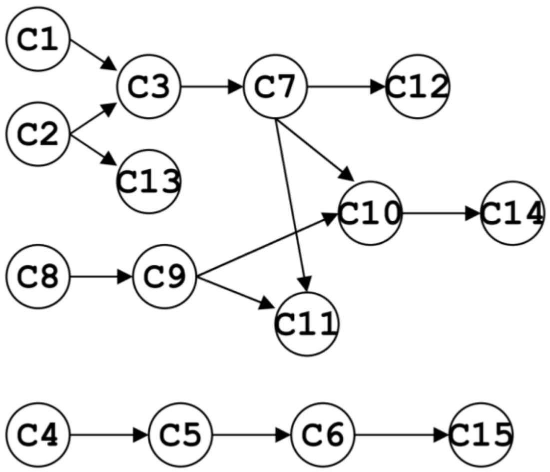
```
for(int i=head[x];i!=0;i=edge[i].next)
```

这个循环的结束条件是i等于0，因为最后一条边，也就是存边时第一条边，在把head值存进next时，head还没有更新过，也就是0。所以当next返回0时，就说明这些边遍历完毕了。

拓扑排序

例：计算机专业排课

课程号	课程名称	预修课程
C1	程序设计基础	无
C2	离散数学	无
C3	数据结构	C1, C2
C4	微积分（一）	无
C5	微积分（二）	C4
C6	线性代数	C5
C7	算法分析与设计	C3
C8	逻辑与计算机设计基础	无
C9	计算机组成	C8
C10	操作系统	C7, C9
C11	编译原理	C7, C9
C12	数据库	C7
C13	计算理论	C2
C14	计算机网络	C10
C15	数值分析	C6

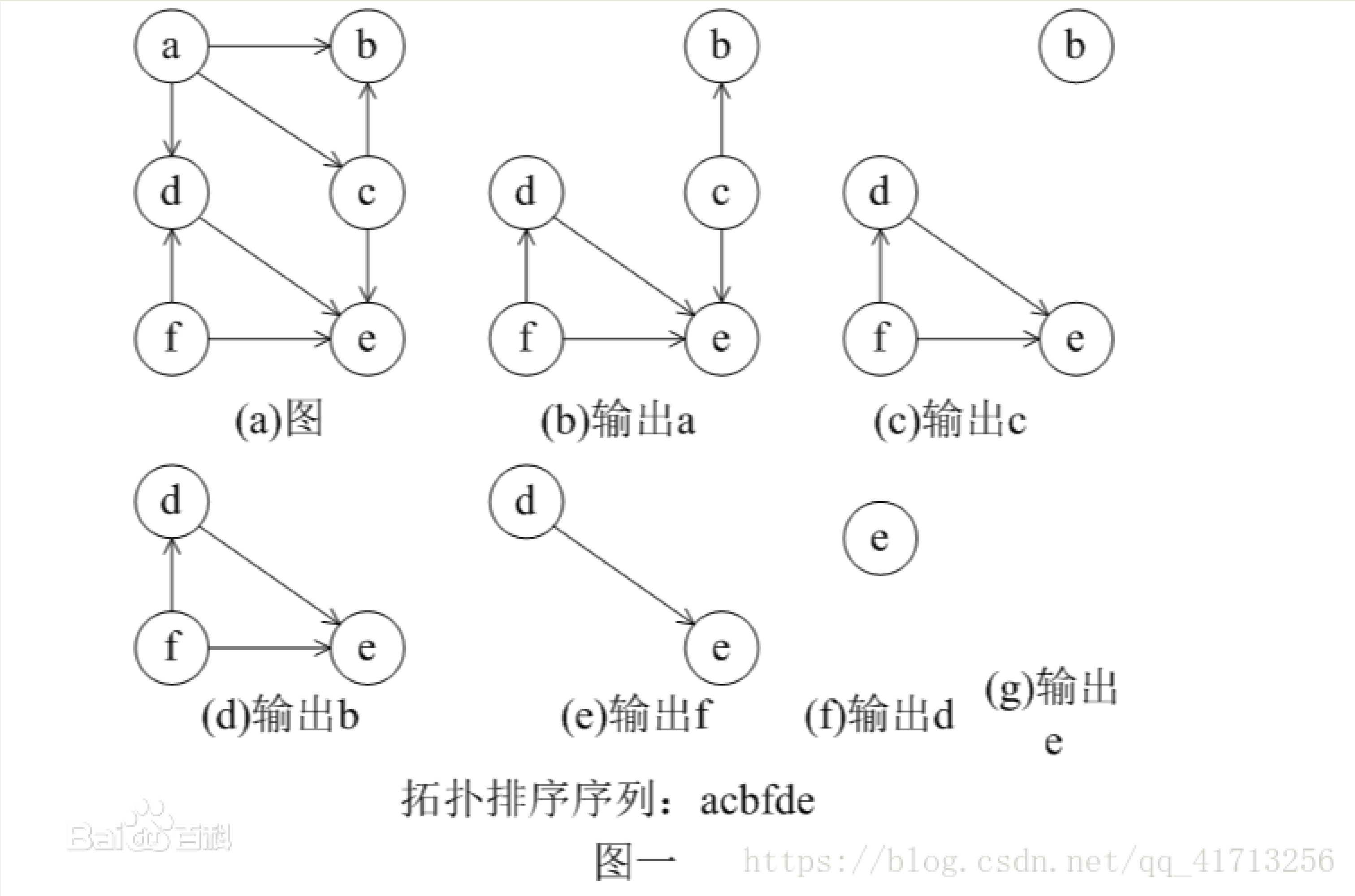


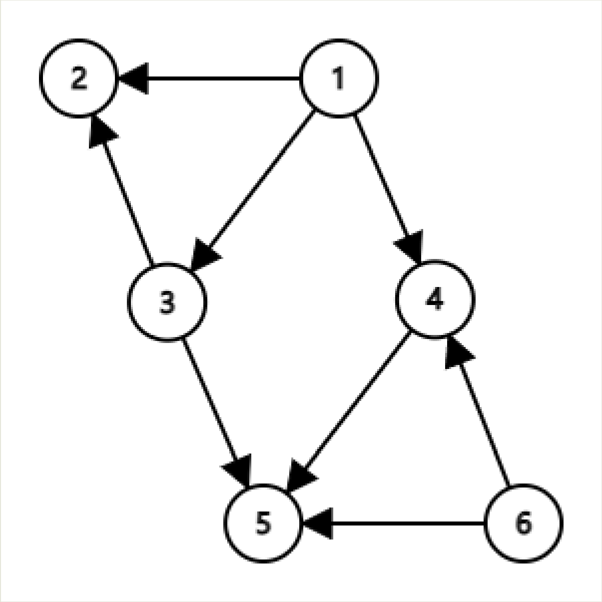
AOV (Activity On Vertex)

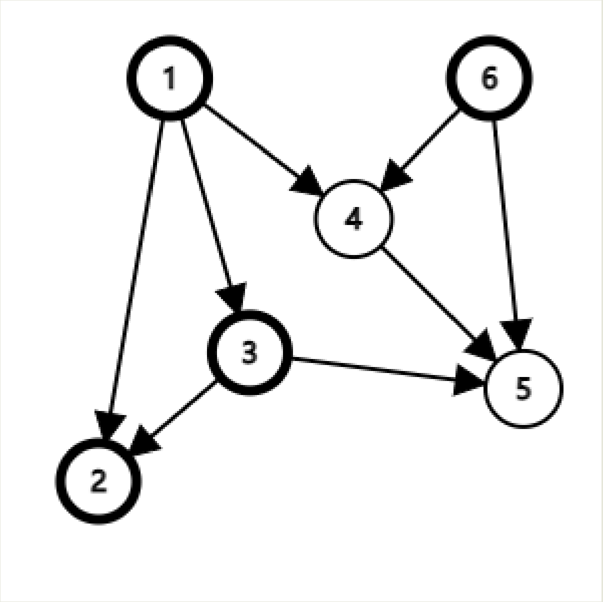
网络

拓扑排序

在一个有向图中，对所有的节点进行排序，**要求没有一个节点指向它前面的节点**。先**统计所有节点的入度**，对于入度为0的节点就可以分离出来，然后把**这个节点指向的节点的入度减一**。一直做该操作，直到所有的节点都被分离出来。如果最后不存在入度为0的节点，那就说明有环，不存在拓扑排序，也就是很多题目的无解的情况。


a
b
d
c
f
e
(a)图
(b)输出a
(c)输出c
(d)输出b
(e)输出f
(f)输出d
(g)输出
e
拓扑排序序列：acbfde
图一
Baidu百科
https://blog.csdn.net/qq_41713256


2
1
3
4
5
6


1
6
4
3
5
2

拓扑排序实现

```cpp
void topsort() {
    queue<int> q;
    for(int i = 1; i <= n; i++) {
        if(!ru[i]) {
            q.push(i);
        }
    }
    while(!q.empty()) {
        int x = q.front();
        q.pop();
        cout << x << " ";
        for(int i = head[x]; i; i = edge[i].nextt) {
            int y = edge[i].to;
            ru[y]--;
            if(!ru[y]) q.push(y);
        }
    }
}
```

哈爾濱工程大學
HARBIN ENGINEERING UNIVERSITY

P4017 最大食物链计数

提交答案 加入题单

传送门

题目背景

展开

你知道食物链吗？Delia 生物考试的时候，数食物链条数的题目全都错了，因为她总是重复数了几条或漏掉了几条。于是她来就来求助你，然而你也不会啊！写一个程序来帮帮她吧。

题目描述

给你一个食物网，你要求出这个食物网中最大食物链的数量。

（这里的“最大食物链”，指的是**生物学意义上的食物链**，即**最左端是不会捕食其他生物的生产者，最右端是不会被其他生物捕食的消费者。**）

Delia 非常急，所以你只有 1 秒的时间。

由于这个结果可能过大，你只需要输出总数模上 80112002 的结果。

输入格式

第一行，两个正整数 n、m，表示生物种类 n 和吃与被吃的关系数 m。

接下来 m 行，每行两个正整数，表示被吃的生物A和吃A的生物B。

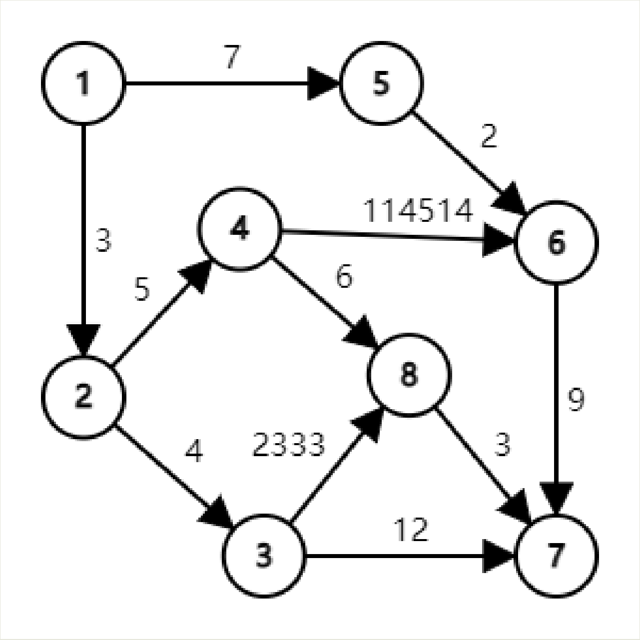
输出格式

一行一个整数，为最大食物链数量模上 80112002 的结果。

最短路问题

最短路径问题的抽象

- 在图中，两个不同顶点之间的所有路径中，边的权值之和最小的那一条路径
  - 这条路径就是两点之间的**最短路径**（Shortest Path）
  - 第一个顶点为**源点**（Source）
  - 最后一个顶点为**终点**（Destination）



给你一张图, 以及每条边的长度,
询问从u 到 v的最短距离

比如这张图从 1 到 7 的最短距离
就是17 (1 -> 2 -> 4 -> 8 -> 7)

问题分类

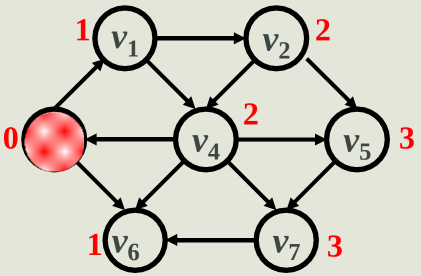
- 单源最短路径问题：从某固定源点出发，求其到所有其他顶点的最短路径
  - （有向）无权图
  - （有向）有权图
- 多源最短路径问题：求任意两顶点间的最短路径

单源最短路径

无权图？

BFS！

- 按照递增（非递减）的顺序找出到各个顶 点的最短路



0: ☞ v_3

1: ☞ v_1 and v_6

2: ☞ v_2 and v_4

3: ☞ v_5 and v_7

模板题传送门！

有权图单源最短路径 -> Dijkstra

带贪心的BFS？

- 开一个数组**dis,**表示从起点到能到达的点的**最短距离**。比如设s为起 点，dis[i]表示从结点s到结点i的最短距离。
- **初始化**dis[s]=0,其他结点dis[i]均赋一个很大的数。
- 循环进行以下操作，直至处理完所有结点：

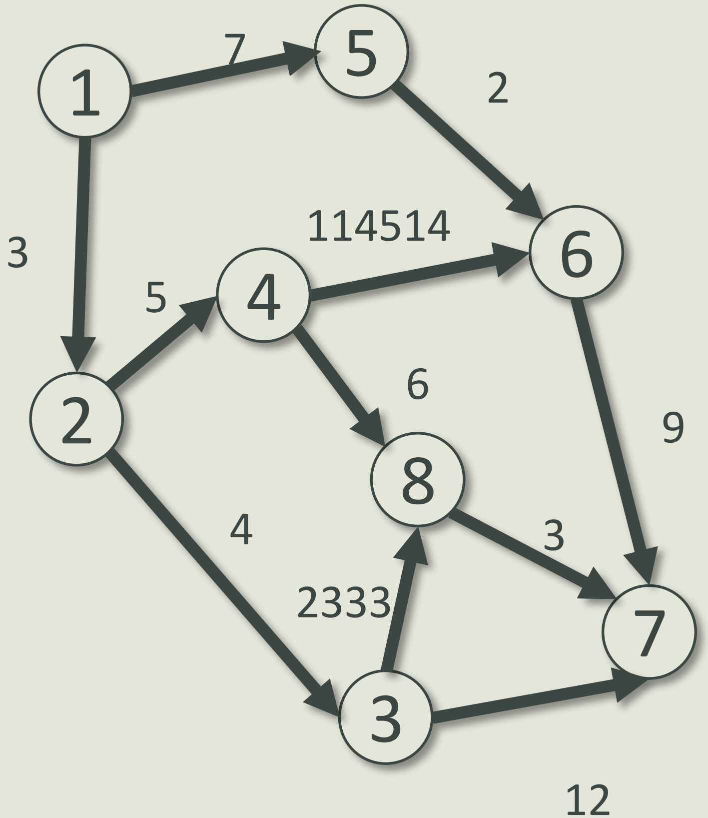
1）选择未处理且**dis[u]最小的点**u进行处理

2）将u相邻的所有未处理的结点**更新最短路**

```
if dis[v] > dis[u]+w[u][v]
    dis[v] = dis[u]+w[u][v];
```

如果当前结点v现在的dis值大于之前的结点u的dis值加边权 → 说明当前结点v的dis**显然不是**最优的，**将其更新**为dis[u] + w[u][v]

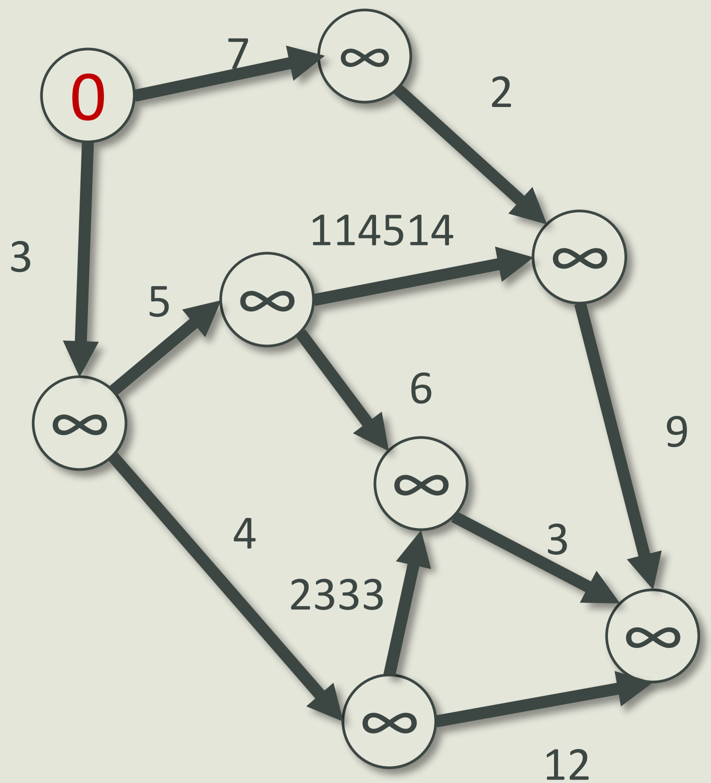
只能处理非负权图 !!!

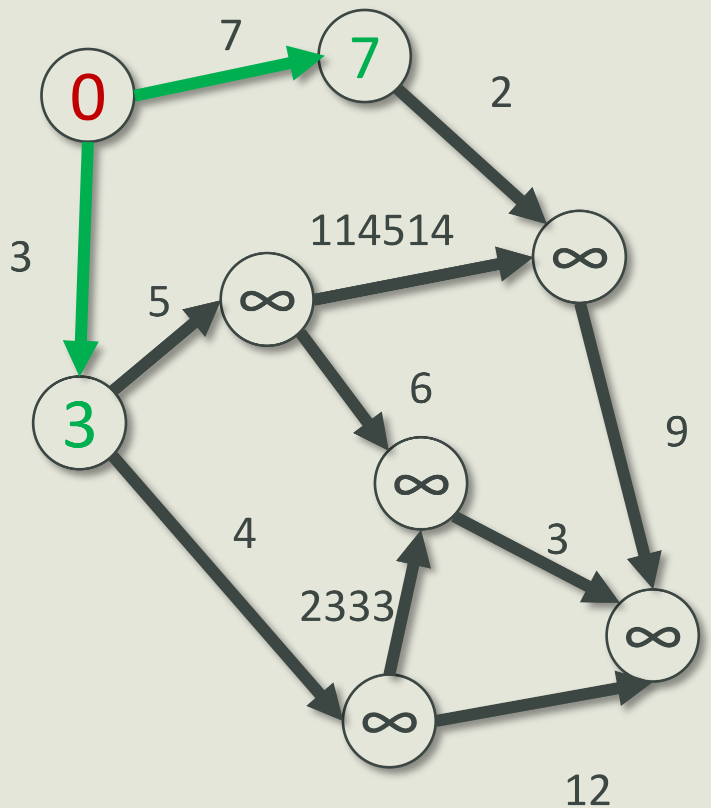


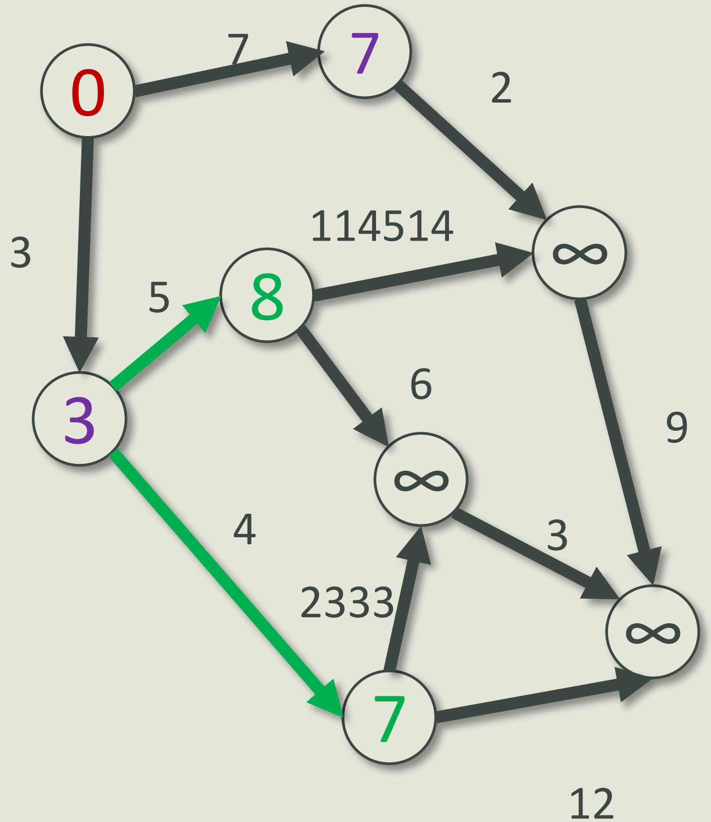
- 假设结点1为起点，求结点1到其余所有 结点的最短距离

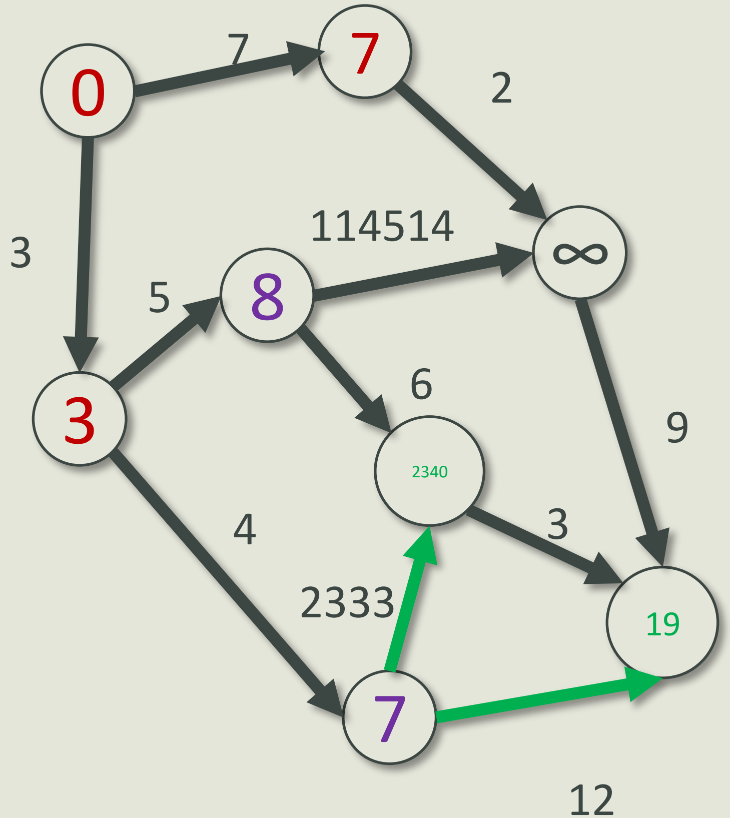
测试用例：

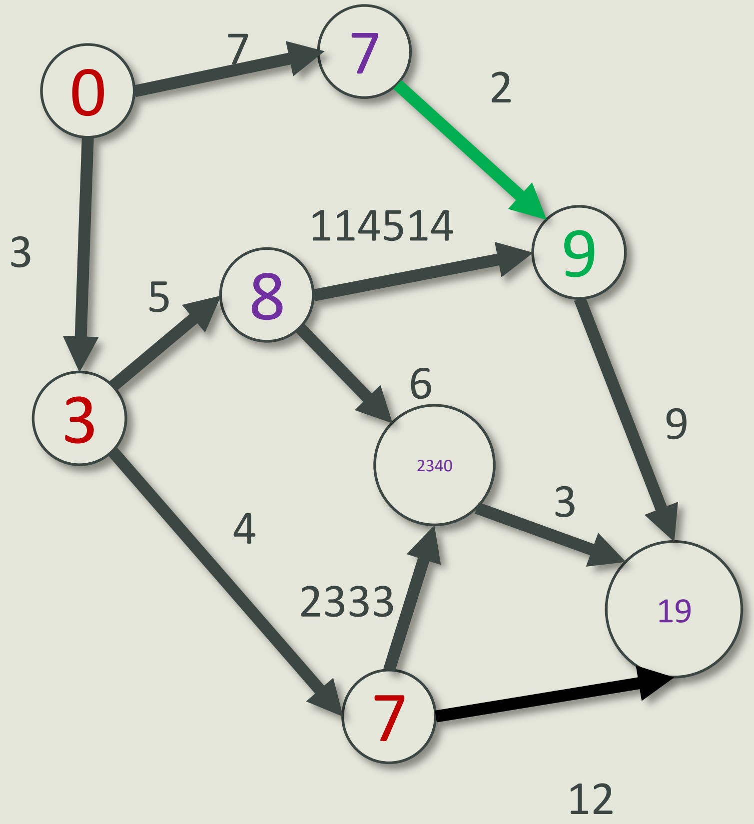
```
8 11 1
1 5 7
1 2 3
2 4 5
4 6 114514
4 8 6
2 3 4
6 7 9
3 8 2333
8 7 3
3 7 12
5 6 2
```

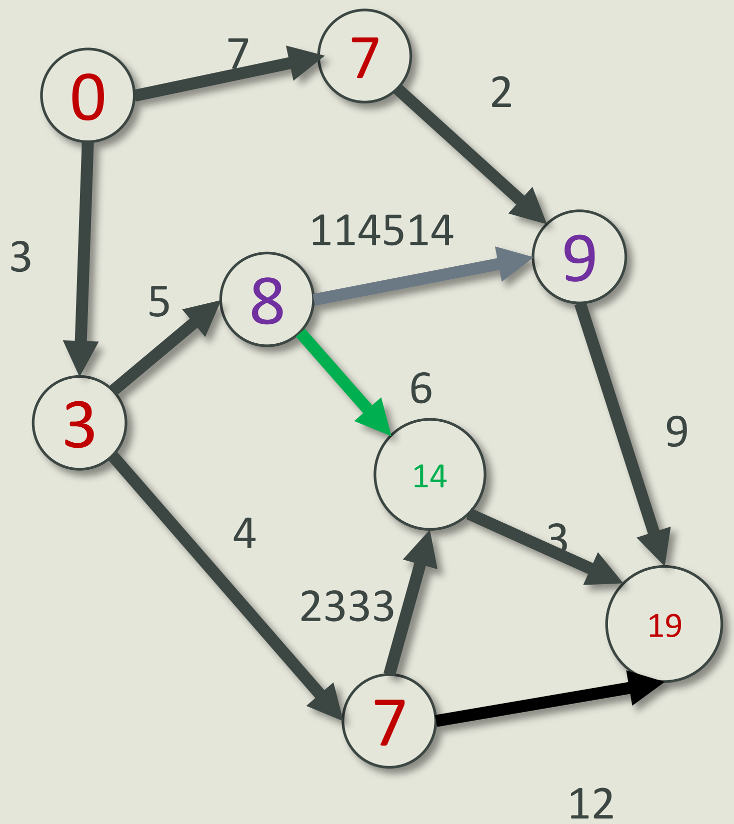

0
7
2
3
114514
5
6
9
4
3
2333
12

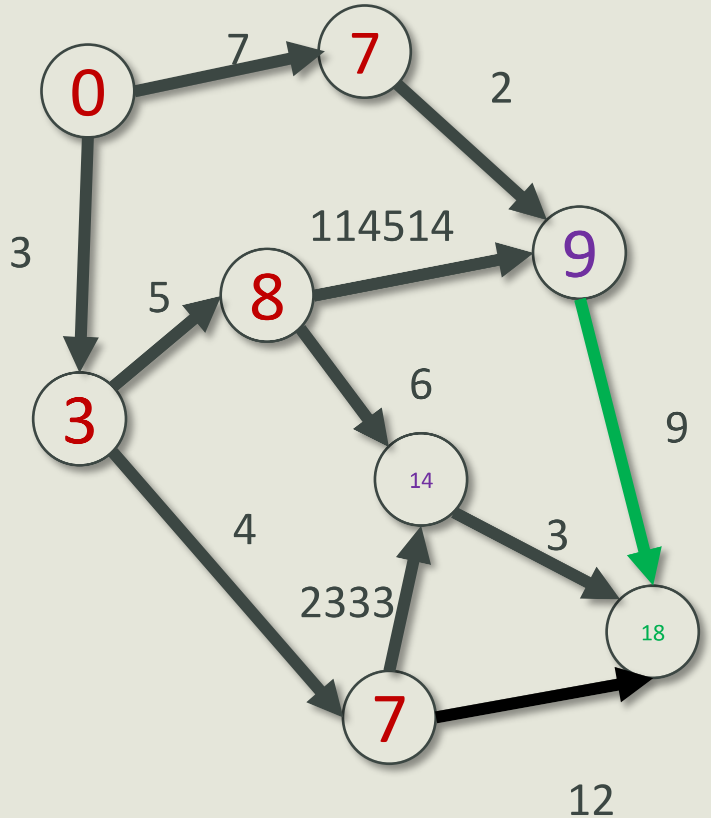

7
0
7
2
3
114514
5
∞
∞
3
6
9
∞
4
3
2333
∞
∞
12

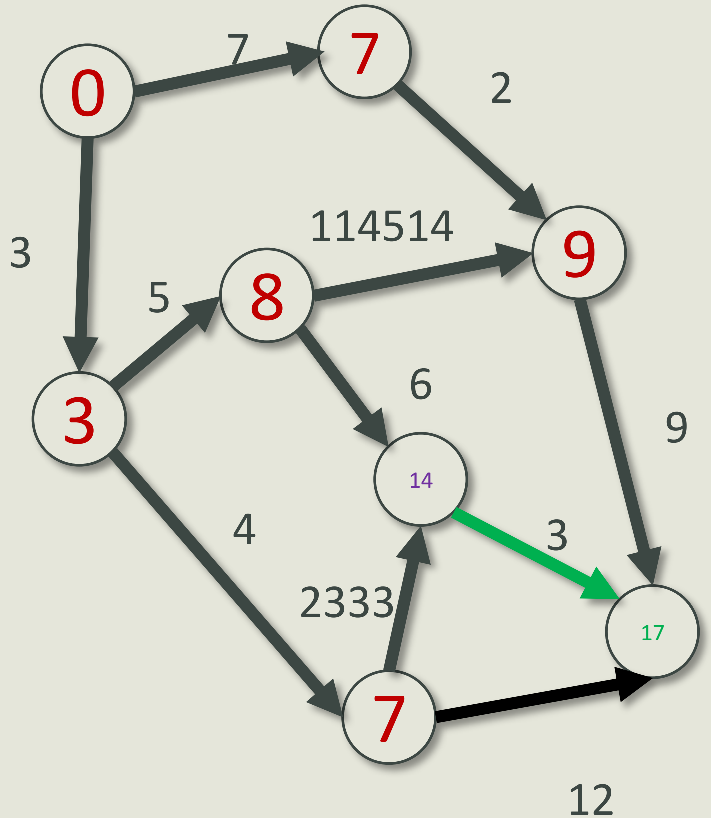

0
7
7
2
3
114514
∞
5
8
3
6
9
∞
4
3
2333
∞
7
12


0
7
7
2
3
114514
∞
5
8
6
3
9
2340
4
3
2333
19
7
12


0
7
7
2
3
114514
9
5
8
3
6
9
2340
3
4
2333
19
7
12


0
7
7
2
3
114514
9
5
8
9
6
3
14
9
4
3
2333
19
7
12


0
7
7
2
3
5
8
114514
9
6
9
3
14
4
3
2333
18
7
12

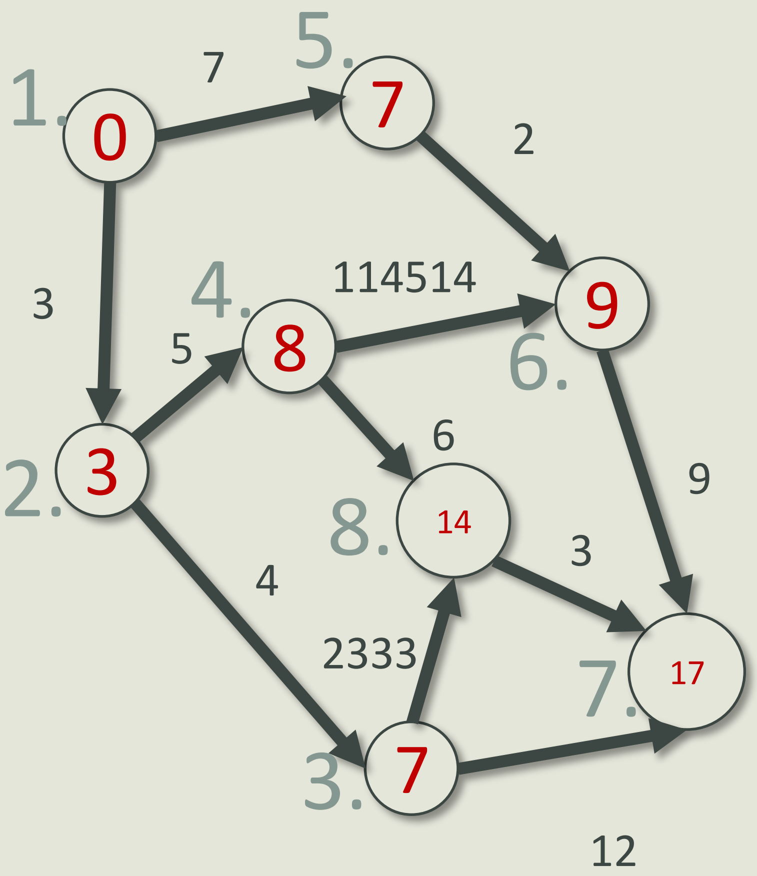

0
7
7
2
3
114514
9
5
8
3
6
9
14
4
3
2333
17
7
12

Dijkstra 代码实现

```cpp
#include<bits/stdc++.h>
using namespace std;

struct Edge{
    int nextt;
    int to;
    int w;
}edge[100100];

int head[100100];
int cnt;

int vis[100100];
int dis[100100];

void add(int u, int v, int w) {
    edge[++cnt].to = v;
    edge[cnt].w = w;
    edge[cnt].nextt = head[u];
    head[u] = cnt;
}

int n,m,s;
```

```cpp
int main() {
    cin >> n >> m >> s;
    for(int i = 1; i <= n; i++) {
        ans[i] = 0x7f7f7f7f;
    }
    for(int i = 1; i <= m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        add(u,v,w);
    }
    ans[s] = 0;
    int pos = s;
```

```
    while(vis[pos] == 0) {
        long can minn = 2147483647;
        vis[pos] = 1;
        for(int i = head[pos]; i != 0; i = edge[i].nextt) {
            if(!vis[edge[i].to] && ans[edge[i].to] > dis[pos] + edge[i].w) {
                dis[edge[i].to] = dis[pos] + edge[i].w;
            }
        }
        for(int i = 1; i <= n; i++) {
            if(!vis[i] && dis[i] < minn) {
                minn = dis[i];
                pos = i;
            }
        }
    }
    for(int i = 1; i <= n; i++) {
        cout << dis[i] << " ";
    }
    system("pause");
    return 0;
}
```


1.
0
7
5.
7
2
3
4.
8
114514
9
6.
5
3
2.
6
8.
14
9
4
3
2333
7.
17
3.
7
12


c:\Users\powerc\Code\jkshdfld.exe
8 11 1
1 5 7
1 2 3
2 4 5
4 6 114514
4 8 6
2 3 4
6 7 9
3 8 2333
8 7 3
3 7 12
5 6 2
0 3 7 8 7 9 17 14 请按任意键继续. . .

观察

遍历所有顶点以便求当前dis最小 ???

```
    while(vis[pos] == 0) {
        long long minn = 2147483647;
        vis[pos] = 1;
        for(int i = head[pos]; i != 0; i = edge[i].nextt) {
            if(!vis[edge[i].to] && ans[edge[i].to] >
ans[pos] + edge[i].w) {
                ans[edge[i].to] = ans[pos] + edge[i].w;
            }
        }
        for(int i = 1; i <= n; i++) {
            if(!vis[i] && ans[i] < minn) {
                minn = ans[i];
                pos = i;
            }
        }
    }
    for(int i = 1; i <= n; i++) {
        cout << ans[i] << " ";
    }
    system("pause");
    return 0;
}
```

优化

↓

小根堆!

STL：

```
priority_queue<node> q;
```

小根堆？

```cpp
struct node {
    int w;
    int pos;
    bool operator < (const node &x) const {
        return x.w < w;
    }
};
priority_queue<node> q;
```

堆优化的Dijkstra

```
#include<bits/stdc++.h>
using namespace std;
const int maxn = 1000100;
int vis[1001000];

struct edge {
    int to;
    int w;
    int nextt;
}edge[maxn];

int head[maxn];
int ans[maxn];
int cnt;
int n, m, s;

void add_edge(int x, int y, int z) {
    edge[++cnt].to = y;
    edge[cnt].w = z;
    edge[cnt].nextt = head[x];
    head[x] = cnt;
}
```

```
struct node {
    int w;
    int pos;
    bool operator < (const node &x) const {
        return x.w < w;
    }
};
priority_queue<node> q;

int main() {
    scanf("%d%d%d", &n, &m, &s);
    for(int i = 1; i <= n; i++) {
        ans[i] = 2147483647;
    }
    for(int i = 0; i < m; i++) {
        int u, v, d;
        scanf("%d %d %d", &u ,&v, &d);
        add_edge(u, v, d);
    }
    ans[s] = 0;
    q.push((node){0, s});
```

```
    while(!q.empty()) {
        node tmp = q.top();
        q.pop();
        int x = tmp.pos, d = tmp.w;
        if(vis[x]) continue;
        vis[x] = 1;
        for(int i = head[x]; i; i = edge[i].nextt) {
            int y = edge[i].to;
            if(!vis[y] && ans[y] > ans[x] + edge[i].w) {
                ans[y] = ans[x] + edge[i];
                q.push((node){ans[y], y});
            }
        }
    }
    for(int i = 1; i <= n; i++) {
        printf("%d ", ans[i]);
    }
    // system("pause");
    return 0;
}
```

有负权边？

放弃贪心，对每一条边都进行一次最短路更新(松弛操作)

Bellman-ford 算法！

Bellman-ford 算法

- 数组dis[i]记录**从源点s到顶点i的路径长度**，初始化数组dis[n]为∞，dis[s]为0；
- 以下操作循环执行至多**n-1**次，n为顶点数：

  对于每一条边e(u, v)，如果**dis[u] + w(u, v) < dis[v]**，则另**dis[v] = dis[u]+w(u, v)**。w(u, v)为边e(u,v)的权值；

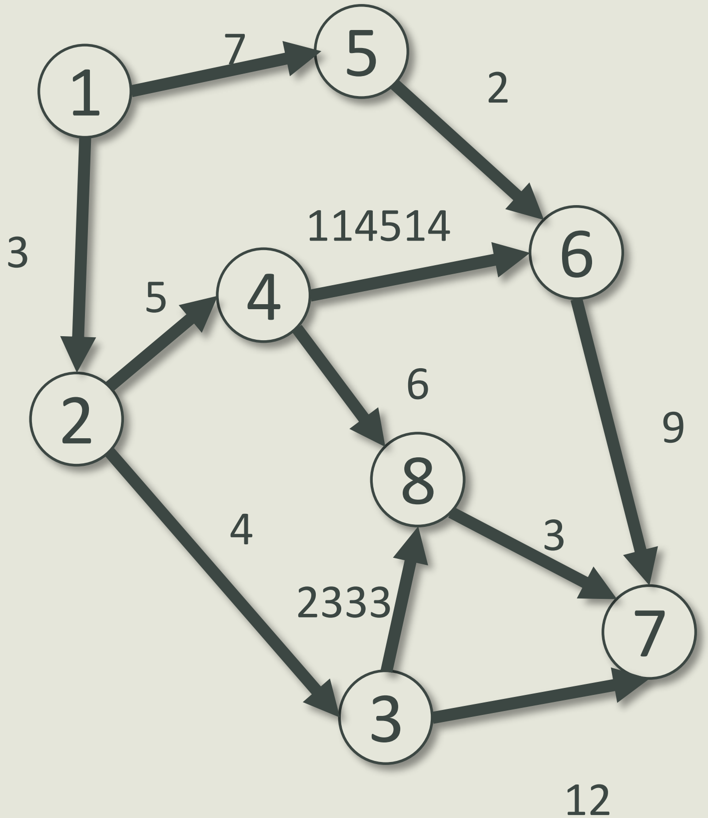
  若上述操作没有对Distant进行更新，说明最短路径已经查找完毕，或者部分点不可达，跳出循环。否则执行下次循环；
- 如果图中是否存在负环路，即权值之和小于0的环路。如果执行完n - 1次松弛操作后，对于每一条边e(u, v)，仍然存在dis[u] + w(u, v) < dis[v]的边，则图中存在负环路，即是说该图无法求出单源最短路径。否则数组dis[n]中记录的就是源点s到各顶点的最短路径长度。

时间复杂度过大？

Bellman-Ford 的队列优化 -> SPFA！

首先把起点放进队列里,然后扫队列的每一个结点,然后松弛操作更新结点放进队列中再重复松弛

BFS + 松弛？ 但SPFA 中一个点可能在出队列之后再次入队


1
5
4
6
2
8
7
3
7
2
3
114514
5
6
9
4
3
2333
12

SPFA 算法

```
queue<int> q;
```

```
q.push(s);
    while(!q.empty()) {
        int x = q.front();
        q.pop();
        vis[x] = 0;
        for(int i = head[x]; i; i = edge[i].nextt) {
            int y = edge[i].to; int w = edge[i].w;
            if(ans[y] > ans[x] + w) {
                ans[y] = ans[x] + w;
                if(!vis[y]) {
                    q.push(y);
                    vis[y] = 1;
                }
            }
        }
    }
```

多源最短路径：Floyd

Floyd算法适用于APSP(求任意两点最短路)，是一种动态规划算法，**稠密图**效果最佳，边权可正可负。此算法简单有效，由于三重循环结构紧凑，对于稠密图，效率要高于执行|V|次Dijkstra算法。

优点： 容易理解，可以算出任意两个节点之间的最短距离，代码编写简单

缺点： 时间复杂度比较高，不适合计算大量数据。

时间复杂度:O(n^3)；空间复杂度:O(n^2)；

任意节点i到j的最短路径两种可能：

1. 直接从i到j；
2. 从i经过若干个节点k到j。

核心思想： dis[i][j] = min{dis[i][j], dis[i][k] + dis[k][j] }

如果 I 能走到k, 且k 能走到 j, 我们直接比较从i直接到j和从i开始通过中继结点k到j的路径长度, 选较短的方案

主要作用：

1. 直接找出任意两点之间的最短路径(dis[u][v])
2. 找负环(dis[v][v] < 0)

哈爾濱工程大學 HARBIN ENGINEERING UNIVERSITY

邻接矩阵实现Floyd

```
#include<bits/stdc++.h>
using namespace std;

int dis[10010][10010];
int n,m;

int mm = 0x7f7f7f7f;
int main() {
    cin >> n >> m;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            if(i != j) dis[i][j] = 0x7f7f7f7f;
        }
    }
    for(int i = 1;i <= m; i++) {
        int u,v,w;
        cin >> u >> v >> w;
        dis[u][v] = w;
    }
```

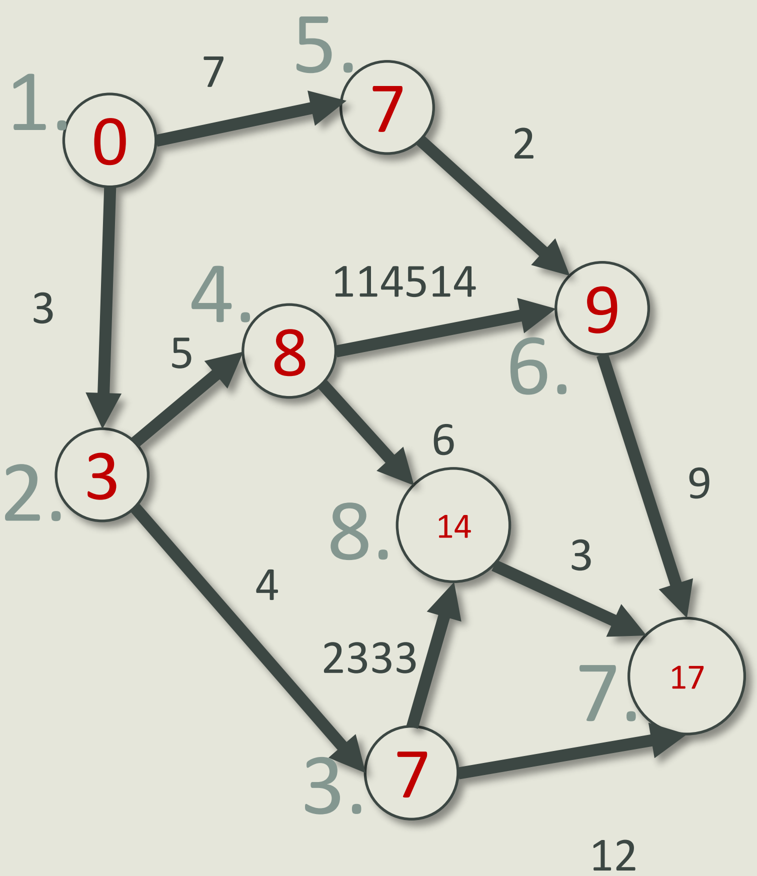
```
for(int k = 1; k <= n; k++) {
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            if(dis[i][k] != mm && dis[k][j] != mm) {
                dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
            }
        }
    }
}
```

判负环

```
int ok = 1;
    for(int i = 1; i <= n; i++) {
        if(dis[i][i] < 0) {
            ok = 0;break;
        }
    }
```

```
if(ok) {
        for(int i = 1; i <= n; i++) {
            if(dis[i][1] == mm) {
                cout << "INF";
            }
            else {
                cout << dis[i][1];
            }
            for(int j = 2; j <= n; j++) {
                if(dis[i][j] == mm) {
                    cout  << "INF";
                }
                else {
                    cout << " " << dis[i][j];
                }
            }
            cout << endl;
        }
    }
    else {
        cout << "存在负环 !" << endl;
    }

    system("pause");
    return 0;
}
```

Floyd 算法运行结果



c:\Users\powerc\Code\sdlkajfl.exe

```
8 11
1 5 7
1 2 3
2 4 5
4 6 114514
4 8 6
2 3 4
6 7 9
3 8 2333
8 7 3
3 7 12
5 6 2
0 3 7 8 7 9 17 14
INF 0 4 5INF 114519 14 11
INFINF 0INFINFINF 12 2333
INFINFINF 0INF 114514 9 6
INFINFINFINF 0 2 11INF
INFINFINFINFINF 0 9INF
INFINFINFINFINFINF 0INF
INFINFINFINFINFINF 3 0
请按任意键继续. . .
```

三种最短路算法性能比较

算法名称	时间复杂度(最好)	时间复杂度(最坏)	空间复杂度	用途
Dijkstra	O((m+n)logn)	O(n^2)	O(n)	单源
SPFA	O(km)(k为常数)	O(nm)	O(n)	单源
Floyd	O(n^3)	O(n^3)	O(n^2)	多源

Dijkstra不能处理含负权边的图，而SPFA可以，此外SPFA和Floyd均可找负环。

P1807 最长路

提交答案 加入题单

题目描述

展开

设 G 为有 n 个顶点的带权有向无环图，G 中各顶点的编号为 1 到 n，请设计算法，计算图 G 中 $1, n$ 间的最长路径。

输入格式

输入的第一行有两个整数，分别代表图的点数 n 和边数 m。

第 2 到第 $(m+1)$ 行，每行 3 个整数 u, v, w，代表存在一条从 u 到 v 边权为 w 的边。

输出格式

输出一行一个整数，代表 1 到 n 的最长路。

若 1 与 n 不连通，请输出 -1。

输入输出样例

输入 #1 复制

```
2 1
1 2 1
```

输出 #1 复制

```
1
```

说明/提示

【数据规模与约定】

- 对于 20%的数据，$n \le 100$，$m \le 10^3$。
- 对于 40% 的数据，$n \le 10^3$，$m \le 10^4$。
- 对于 100% 的数据，$1 \le n \le 1500$，$1 \le m \le 5 \times 10^4$，$1 \le u, v \le n$，$-10^5 \le w \le 10^5$。

传送门

求 1 -> n 最长路 ??

松弛每条边, 维护最长路 ?

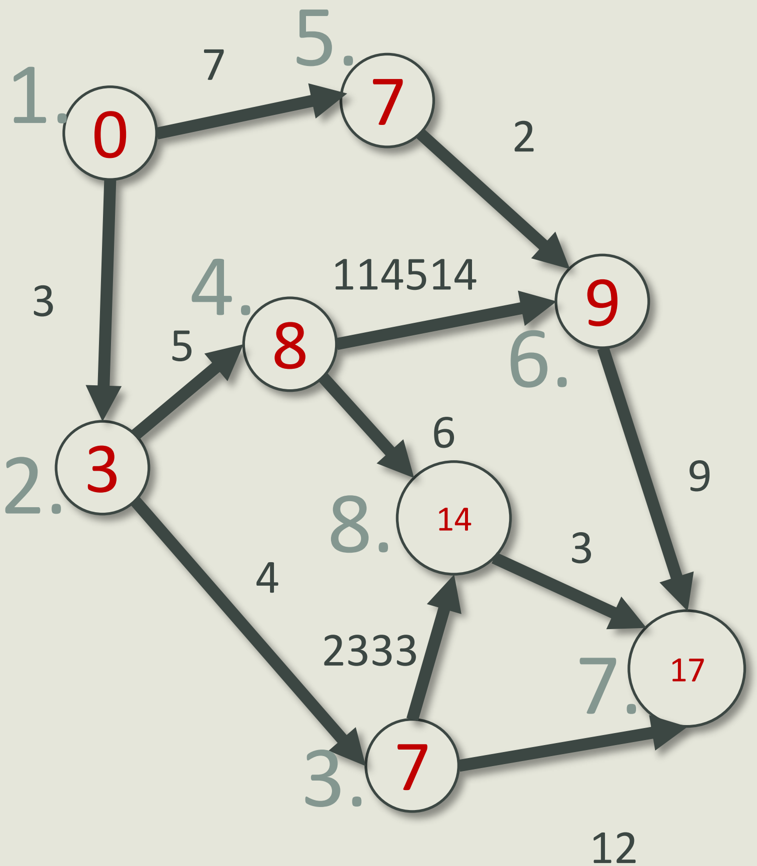
If dis[y] < dis[x] + w

dis[y] = dis[x] + w

SPFA

哈爾濱工程大學
HARBIN ENGINEERING UNIVERSITY

最小生成树



给你一张图。让你在这张图中找到一棵树，使得这棵树的边权之和最小

Prim O(n^2)

思想类似Dijkstra

- 首先先选择一个结点作为根结点
- 然后从MST中和图中剩余结点中找 一条最短

边，把这条边加入MST中

- 循环上述操作直至结束

代码实现

```
int prim() {
    //先把dis数组附为极大值
    for(re int i=2;i<=n;++i) {
        dis[i]=inf;
    }
    //这里要注意重边，所以要用到min
    for(re int i=head[1];i;i=e[i].next) {
        dis[e[i].v]=min(dis[e[i].v],e[i].w);
    }
    while(++tot<n){//最小生成树边数等于点数-1
        re int minn=inf;//把minn置为极大值
        vis[now]=1;//标记点已经走过
        //枚举每一个没有使用的点
        //找出最小值作为新边
        //注意这里不是枚举now点的所有连边，而是1~n
        for(re int i=1;i<=n;++i) {
            if(!vis[i]&&minn>dis[i]) {
                minn=dis[i];
                now=i;
            }
        }
```

```
        ans+=minn;
        //枚举now的所有连边，更新dis数组
        for(re int i=head[now];i;i=e[i].next) {
            re int v=e[i].v;
            if(dis[v]>e[i].w&&!vis[v]) {
                dis[v]=e[i].w;
            }
        }
    }
    return ans;
}
```

Kruskal O(mlogn)

利用了并查集

- 先把所有边升序排序，然后按照升 序把边上两个不在同一集合里的点 合并，全部合并完成后得到的边的 集合就是最小生成树。

HARBIN ENGINEERING UNIVERSITY

代码实现

```
for(re int i=1;i<=n;i++) {
        fa[i]=i;
}
//初始化并查集
```

```
void kruskal() {
    sort(edge,edge+m,cmp);
    //将边的权值排序
    for(re int i=0;i<m;i++) {
        eu=find(edge[i].u), ev=find(edge[i].v); {
            continue;
        }
        //若出现两个点已经联通了，则说明这一条边不需要了
        ans+=edge[i].w;
        //将此边权计入答案
        fa[ev]=eu;
        //将eu、ev合并
        if(++cnt==n-1) {
            break;
        }
        //循环结束条件，及边数为点数减一时
    }
}
```

拓扑排序题单 https://www.luogu.com.cn/training/42933#problems

零基础图论练习题单 https://www.luogu.com.cn/training/42489#problems

Thanks