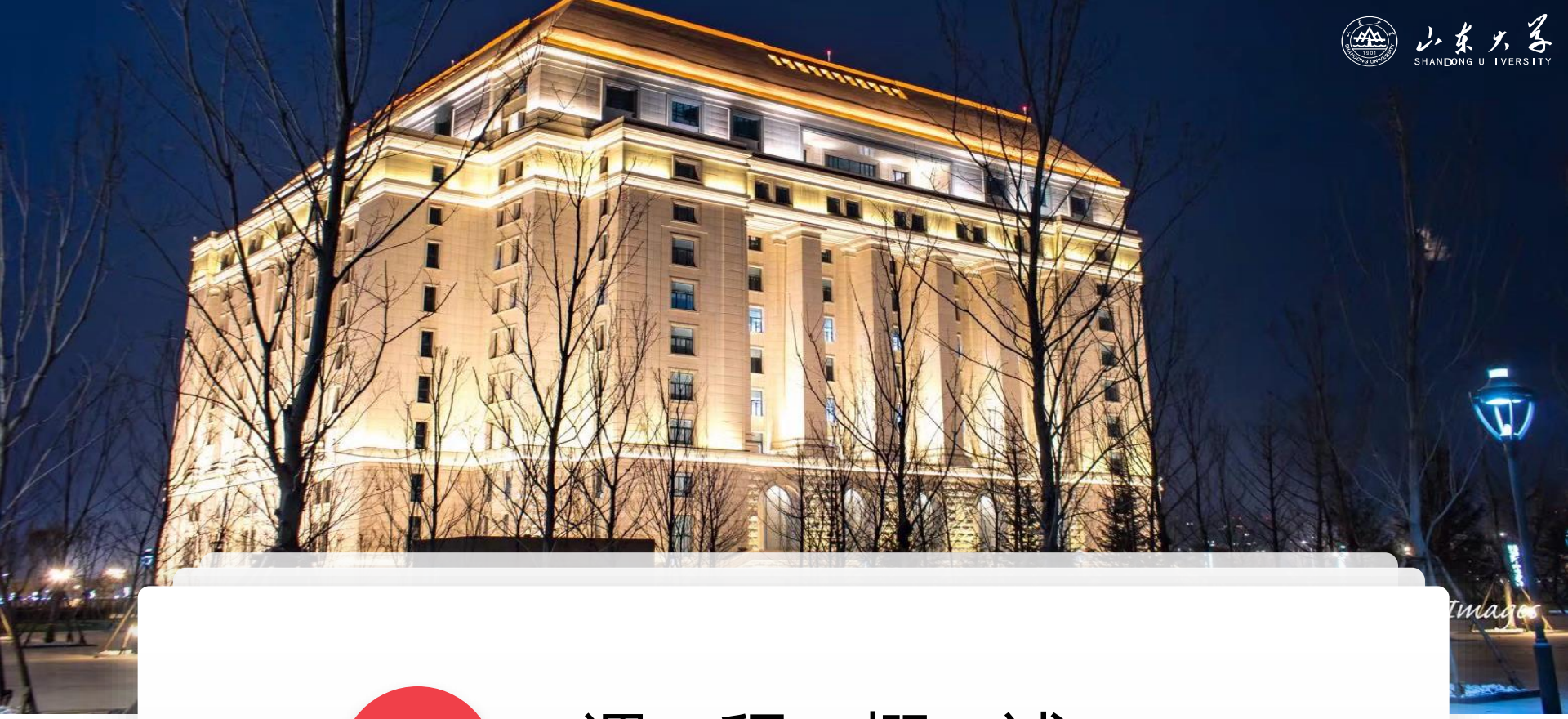
程序设计思维与实践

Thinking and Practice in Programming

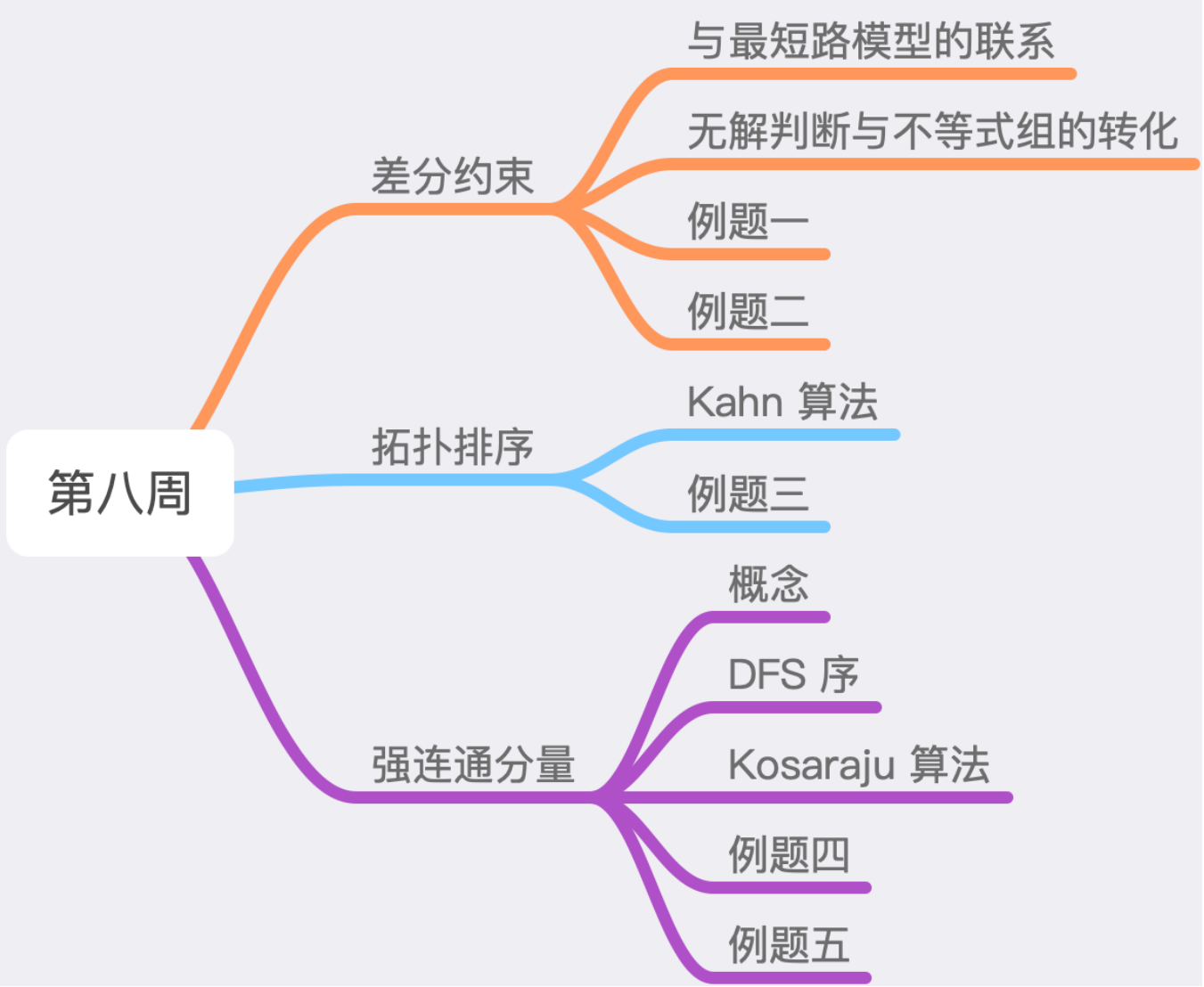
图和树的性质与应用（下） | 内容负责：李子晗/魏安然

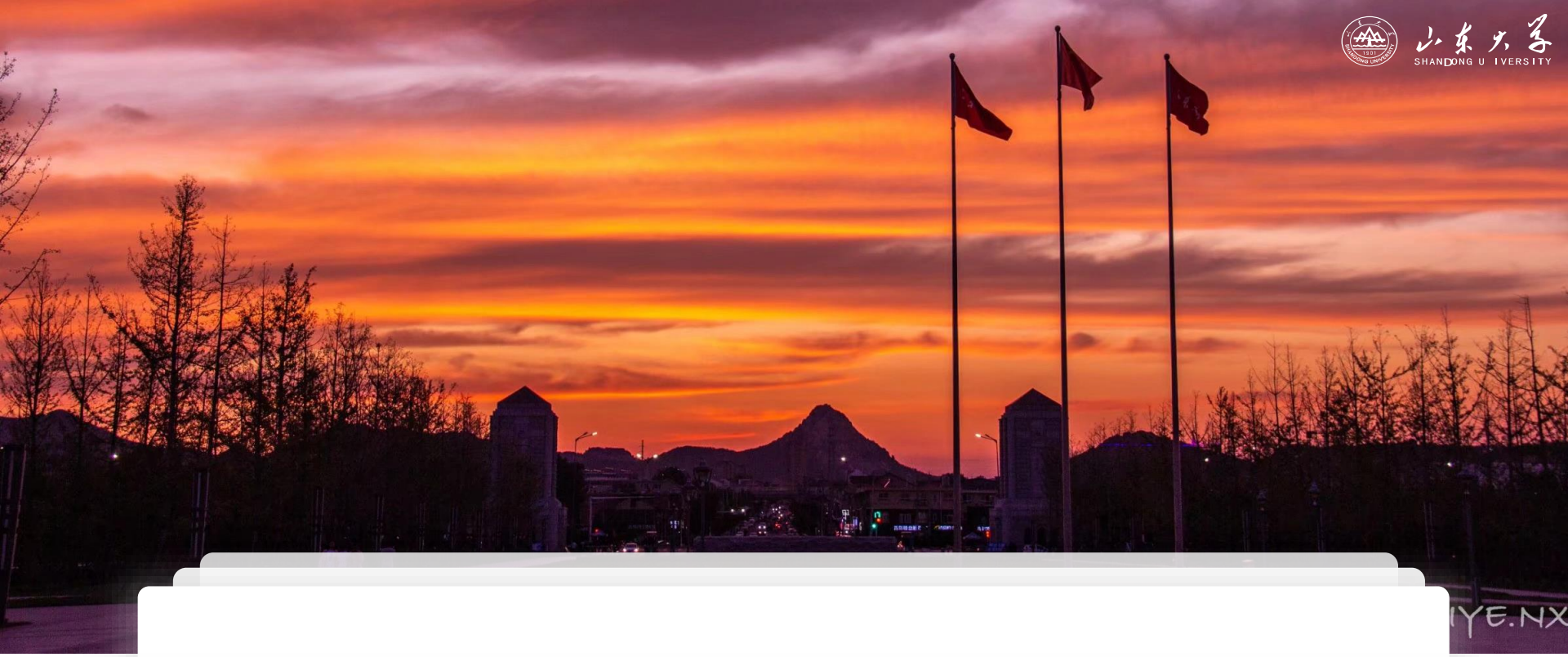


1 课 程 概 述

The outline of course

课程概述


第八周
差分约束
与最短路模型的联系
无解判断与不等式组的转化
例题一
例题二
拓扑排序
Kahn 算法
例题三
强连通分量
概念
DFS 序
Kosaraju 算法
例题四
例题五



2 差分约束系统

Difference constraints

差分约束

- **差分约束系统**
  - 一种特殊的 n 元一次不等式组，它包含 n 个变量以及 m 个约束条件。
  - 每个约束条件是由两个其中的变量做差构成的，形如 $x_i - x_j \leq c_k$，其中 c_k 是常数（可以是非负数，也可以是负数）。
  - 我们要解决的问题是：求一组解 $x_1 = a_1, x_2 = a_2, \ldots, x_n = a_n$，使得所有的约束条件得到满足，否则判断出无解。
- 注意到，如果 $\{a_1, a_2, a_3, \ldots, a_n\}$ 是该差分约束系统的一组解，那么对于任意的常数 d，显然 $\{a_1 + d, a_2 + d, \ldots, a_n + d\}$ 也是该差分约束系统的一组解，因为这样做差后 d 刚好被消掉。
- 一般的求解思路是尝试把几个不等式组合在一起得到我们想要的式子，那么式子中最小的就是我们的答案。

差分约束

- $x_1 - x_0 \leq 1$
- $x_2 - x_0 \leq 2$
- $x_3 - x_0 \leq 4$
- $x_2 - x_1 \leq 3$
- $x_3 - x_2 \leq 1$

- 在这个例子中，根据第3条可以知道 $x_3 - x_0 \leq 4$
- 根据第1、4、5条可以知道 $x_3 - x_0 \leq 5$
- 而根据第2、5条可以知道 $x_3 - x_0 \leq 3$
- 那么此时如果规定 $x_0 = 0$ 则 x_3 的最大值是3

差分约束

- 结论：求解差分约束系统，都可以转化为图论中单源最短路问题
- 对于差分约束中的每一个不等式约束 $x_i - x_j \leq c_k$ 都可以移项变形为 $x_i \leq c_k + x_j$
- 如果令 $c_k = w(i,j)$，$dis[i] = x_i$，$dis[j] = x_j$，那么原式变为 $dis[i] \leq dis[j] + w(i,j)$，与最短路问题中的松弛操作相似

```
if(dis[v] > dis[u] + w(u, v)) {
    dis[v] = dis[u] + w(u, v);
}
```

差分约束

- 我们可以把变量 x_i 看作图中的一个结点，对于每个不等式约束 $x_i - x_j \leq c_k$，从结点 j 到节点 i 连一条长度为 c_k 的有向边
- 路径与不等式约束是一一对应的
- 对于一条从 i 走到 j 路径而言，假设其长度为 C，其对应的不等式约束即为 $x_i - x_j \leq C$
- 即假设起点为 s，对于当前已经求出来的 $dis[i]$，其对应的不等式即为 $x_i - x_s \leq dis[i]$
- 于是求解差分约束问题变成了求解最短路问题
- 如果最终令 $x_s = 0$，那么 $x_i = \mathrm{dis}[i]$ 便是差分约束问题的一组解

差分约束

- 解的存在性
- 由于在求解最短路的过程中出现存在负环或者终点不可达的情况，那么在求解差分约束的过程中也会存在这样的问题
- 存在负环
  - 如果路径中存在负环则表现为最短路的无限小，即最短路不存在
  - 在不等式约束上表现为 $x_i - x_1 \leq T$ 中的 T 为无限小
  - 得出的结论就是 x_i 的结果不存在
  - 可以通过 Bellman 算法或者队列优化的 Bellman 算法（SPFA）来判断负环是否存在
- 终点不可达
  - 这种情况表明变量 x_i 与变量 1 之间没有约束关系
  - x_i 的结果可以是无限大，对应代码中为 dis[i]=inf

差分约束

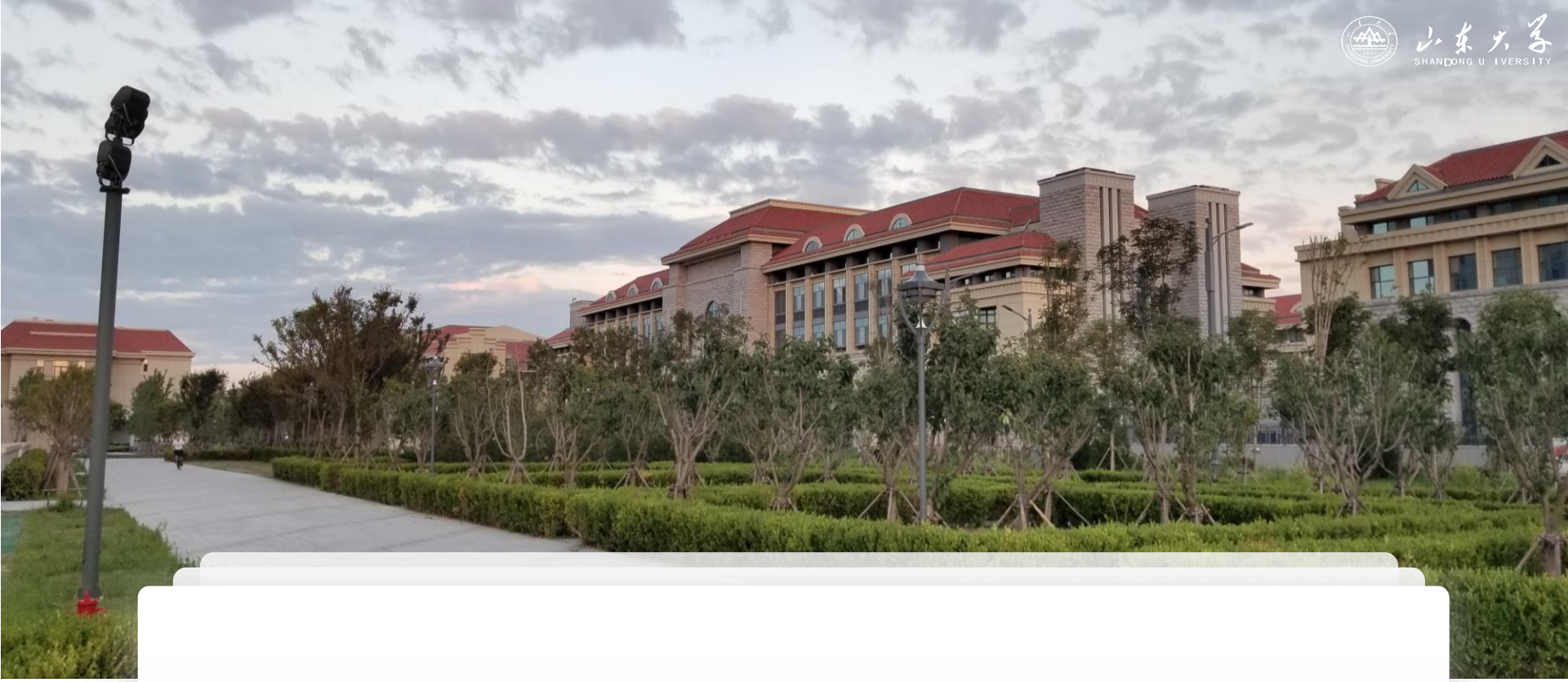
- 不等式组的转化
- 给出 $x_i - x_1 \geq T$，可以移项转化为 $x_1 - x_i \leq -T$
- 给出 $x_i - x_1 < T$，在整数域上可以转化为 $x_i - x_1 \leq T - 1$
- 给出 $x_i - x_1 = \mathrm{T}$，可以转化为 $x_i - x_1 \leq T$ 且 $x_1 - x_i \geq T$

- 思考：$\frac{x_i}{x_j} \leq k$ 的转化？

- 在求解最短路的时候，程序是按照等于号来计算的，此时跑最短路得到的是一组最大解（上界），如果想要求最小解（下界）则将≤变成≥然后跑最长路即可，原理都是一样的

差分约束

- **最长路**
  - SPFA
    - 初始化的时候将 $dis[x]$ 初始化成 $-\inf$
    - 松弛部分把 $dis[x] > dis[y] + w$ 改成 $dis[x] < dis[y] + w$
  - Dijkstra
    - 如果图中存在负权边，那么不能用 Dijkstra 求最短路
    - 如果图中存在正权边，那么不能用 Dijkstra 求最长路



4

例　　题　　1

Example Two

例题一

- ~~区间选点~~
- 给定一个数轴上的 n 个区间，要求在数轴上选取最少的点使得第 i 个区间里至少有 c_i 个点

- 1 <= n <= 50000
- 区间端点不超过 50000

例题一

- ~~区间选点~~
- ~~按右端点排序然后贪心~~
- 构造不等式组
  - 记 $sum[i]$ 表示数轴上 $[0, i]$ 之间选点的个数
  - 对于第 i 个区间 $[a_i, b_i]$ 需要满足 $sum[b_i] - sum[a_i - 1] \geq c_i$

- 需要保证 sum 是有意义的
  - $0 \leq sum[i] - sum[i-1] \leq 1$

- 求该差分约束系统的最小解，转化为 ≥ 不等式组跑最长路，答案为 $sum[\max\{b_i\}]$

3 例　题　2

Example One

例题二 CSP201809-4

问题描述

在一条街上有n个卖菜的商店，按1至n的顺序排成一排，这些商店都卖一种蔬菜。

第一天，每个商店都自己定了一个正整数的价格。店主们希望自己的菜价和其他商店的一致，第二天，每一家商店都会根据他自己和相邻商店的价格调整自己的价格。具体的，每家商店都会将第二天的菜价设置为自己和相邻商店第一天菜价的平均值（用去尾法取整）。

注意，编号为1的商店只有一个相邻的商店2，编号为n的商店只有一个相邻的商店n–1，其他编号为i的商店有两个相邻的商店i–1和i+1。

给定第二天各个商店的菜价，可能存在不同的符合要求的第一天的菜价，请找到符合要求的第一天菜价中字典序最小的一种。

字典序大小的定义：对于两个不同的价格序列($a_1, a_2, ..., a_n$)和($b_1, b_2, b_3, ..., b_n$)，若存在i (i>=1), 使得$a_i<b_i$，且对于所有j<i，$a_j=b_j$，则认为第一个序列的字典序小于第二个序列。

输入格式

输入的第一行包含一个整数n，表示商店的数量。

第二行包含n个正整数，依次表示每个商店第二天的菜价。

输出格式

输出一行，包含n个正整数，依次表示每个商店第一天的菜价。

样例输入

```
8
2 2 1 3 4 9 10 13
```

样例输出

```
2 2 2 1 6 5 16 10
```

数据规模和约定

对于30%的评测用例，2<=n<=5，第二天每个商店的菜价为不超过10的正整数；

对于60%的评测用例，2<=n<=20，第二天每个商店的菜价为不超过100的正整数；

对于所有评测用例，2<=n<=300，第二天每个商店的菜价为不超过100的正整数。

请注意，以上都是给的第二天菜价的范围，第一天菜价可能会超过此范围。

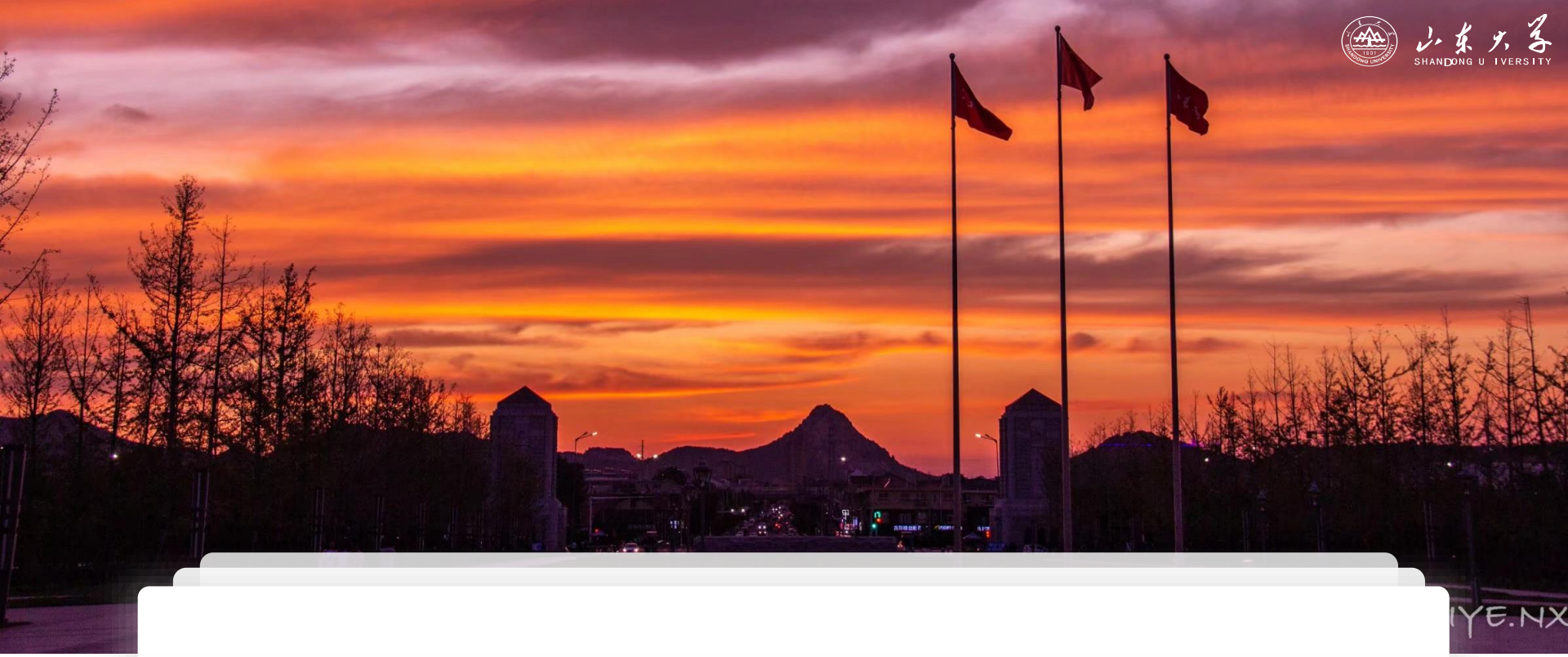
例题二

- 构造不等式组
- $b[i] = (a[i-1] + a[i] + a[i+1])/3$
- $3b[i] + 0/1/2 = a[i-1] + a[i] + a[i+1]$
- $3b[i] \leq a[i-1] + a[i] + a[i+1] \leq 3b[i] + 2, i \geq 2$
- 特别的有
  - $2b[1] \leq a[1] + a[2] \leq 2b[1] + 1$
  - $2b[n] \leq a[n-1] + a[n] \leq 2b[n] + 1$
- 记 $s[i] = a[1] + a[2] + \cdots + a[i], s[0] = 0$
- 则有 $3b[i] \leq s[i+1] - s[i-2] \leq 3b[i] + 2, i \geq 2$
- $2b[1] \leq s[2] - s[0] \leq 2b[1] + 1$
- $2b[n] \leq s[n] - s[n-2] \leq 2b[n] + 1$

例题二

- 保证最终的结果为正数?
  - $a[i] \geq 1 \Rightarrow s[i] - s[i-1] \geq 1$
- 题目要求字典序最小?
  - 求最小解，构造 ≥ 的不等式组并跑最长路
- 对最长路求得的dis数组做差分即为答案
- 部分代码

```
add(0, 2, b[1] * 2);
add(2, 0, -(b[1] * 2 + 1));
add(n - 2, n, b[n] * 2);
add(n, n - 2, -(b[n] * 2 + 1));
for(int i = 2; i < n; ++i) {
    add(i - 2, i + 1, b[i] * 3);
    add(i + 1, i - 2, -(b[i] * 3 + 2));
}
for(int i = 1; i <= n; ++i) {
    add(i - 1, i, 1);
}
spfa(0);
for(int i = 1; i <= n; ++i) printf("%d ", dis[i] - dis[i - 1]);
```



2 拓 扑 排 序

Topological Sorting

拓扑排序

- 拓扑排序要解决的问题是给一个图的所有节点排序。
- 我们可以拿大学选课的例子来描述这个过程，比如学习大学课程中有：「高等数学」，「线性代数」，「离散数学」，「概率论」，「算法导论」，「机器学习」。当我们想要学习「算法导论」的时候，就必须先学会「离散数学」和「概率论」，不然在课堂就会听的一脸懵逼。当然还有一个更加前的课程「高等数学」。

- 这些课程就相当于几个顶点 u，顶点之间的有向边 (u, v) 就相当于学习课程的顺序。
- 显然拓扑排序不是那么的麻烦，不然你是如何选出合适的学习顺序?

拓扑排序

- 但是如果某一天排课的老师打瞌睡了，说想要学习「算法导论」，还得先学「机器学习」，而「机器学习」的前置课程又是「算法导论」，然后你就一万脸懵逼了，我到底应该先学哪一个？
- 当然我们在这里不考虑什么同时学几个课程的情况。在这里，「算法导论」和「机器学习」间就出现了一个环，显然你现在没办法弄清楚你需要学什么了，于是你也没办法进行拓扑排序了。

- 因而如果有向图中存在环路，那么我们就没办法进行 拓扑排序 了。
- 但同时，我们也可以用拓扑排序判断图中是否存在环

拓扑排序

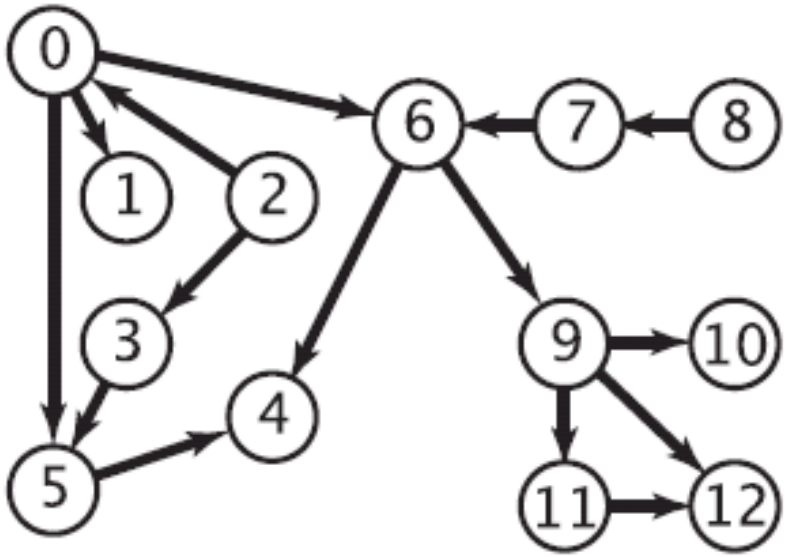
- 因此我们可以说 在一个 DAG（有向无环图） 中，我们将图中的顶点以线性方式进行排序，使得对于任何的顶点 u 到 v 的有向边 (u, v)，都可以有 u 在 v 的前面。
- 还有给定一个 DAG，如果从 u 到 v 有边，则认为 v 依赖于 u 。如果 u 到 v 有路径（ u 可达 v ），则称 v 间接依赖于 u 。

- 拓扑排序的目标是将所有节点排序，使得排在前面的节点不能依赖于排在后面的节点。

Kahn算法

- 将入度为 0 的点组成一个集合 S
- 每次从 S 里面取出一个顶点 u （可以随便取）放入 L，然后遍历顶点 u 的所有边 (u, v)，并删除之，并判断如果该边的另一个顶点 v，如果在移除这一条边后入度为 0，那么就将这个顶点放入集合 S 中。不断地重复取出顶点然后重复这个过程……
- 最后当集合为空后，就检查图中是否存在任何边。如果有，那么这个图一定有环路，否者返回 L，L 中顺序就是拓扑排序的结果

Kahn算法



排序结果:

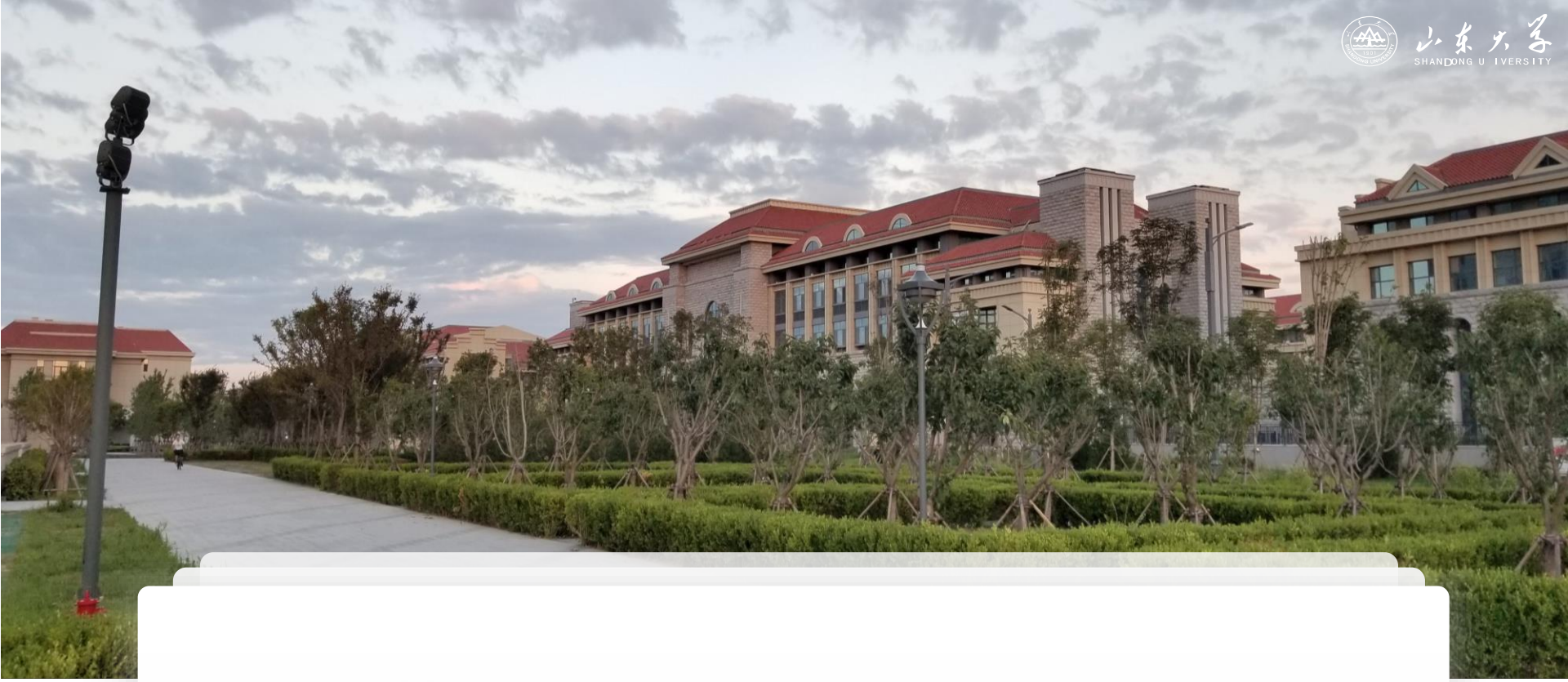
2 -> 8 -> 0 -> 3 -> 7 -> 1 -> 5 -> 6 -> 9 -> 4 -> 11 -> 10 -> 12

拓扑排序并不一定唯一:

8 -> 7 -> 2 -> ... 也是合法的拓扑序

Kahn算法

```
bool toposort() {
    q = new queue();
    for (i = 0; i < n; i++)
        if (in_deg[i] == 0) q.push(i);
    ans = new vector();
    while (!q.empty()) {
        u = q.pop();
        ans.push_back(u);
        for each edge(u, v) {
            if (--in_deg[v] == 0) q.push(v);
        }
    }
    if (ans.size() == n) {
        for (i = 0; i < n; i++)
            std::cout << ans[i] << std::endl;
        return true;
    } else {
        return false;
    }
}
```



4

例　　题　　3

Example Three

例题三

- 现在有 n 个同学参加模测，现在知道的信息有类似 A 比 B 考得多。现在想要根据已有信息，输出一个排名，分数高的在前，想让学号小的同学尽可能靠前。
- $1 \leq N, M \leq 10^5$

N=4 M=4
1 2
1 3
2 3
2 4

答案为1 2 3 4

例题三

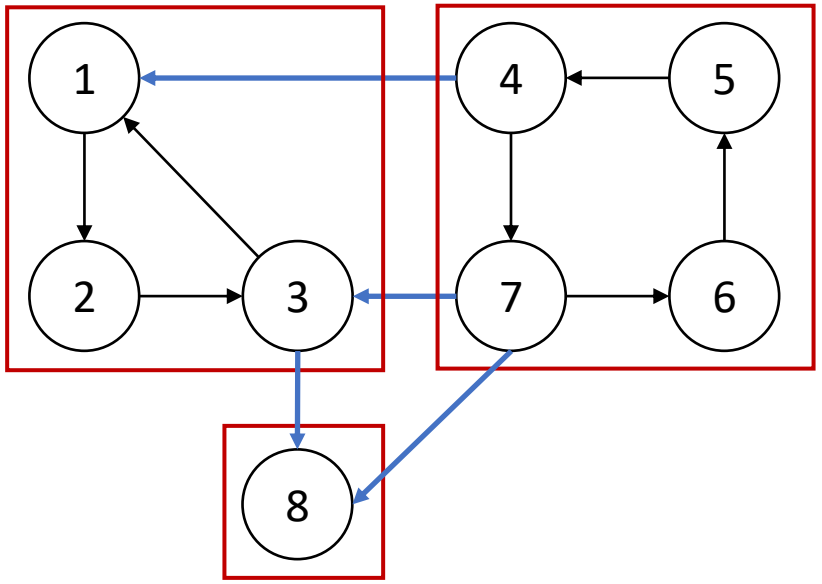
- 首先可以根据分数高低关系建立一张图，如果A比B高，则建立一个 $A \to B$ 的有向边
- 由于有要求同分数学号小的在前，所以我们用优先队列存储同学的学号。

4 强 连 通 分 量

Strong Connected Component

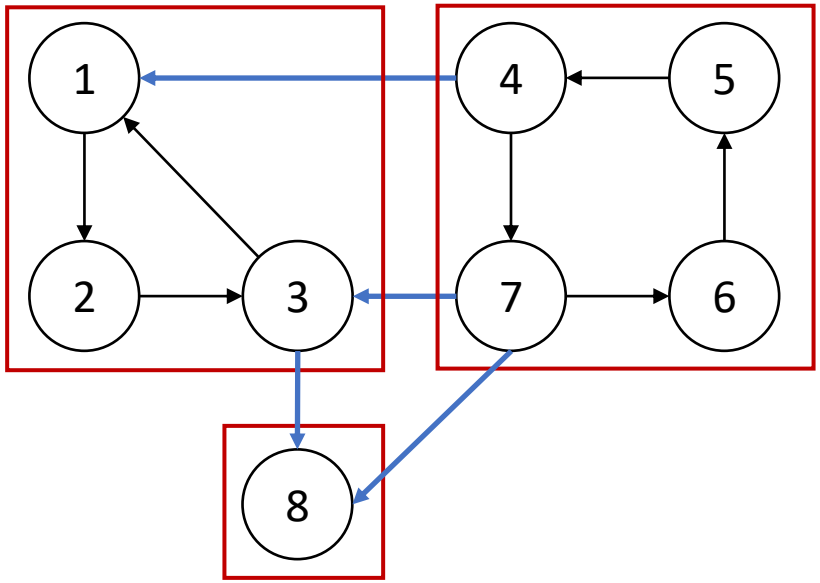
强连通分量

- 强连通分量（SCC）
  - 有向图中的概念
  - 强连通：有向图 G 中任意两个结点连通
  - 强连通分量（SCC）：极大的强连通子图
- 例子
  - 蓝线为 SCC 间的边
  - 红框为 SCC
    - {1, 2, 3}
    - {4, 5, 6, 7}
    - {8}



DFS 序

- DFS 序分类
  - 前序：第一次达到点 x 的次序，用 d[x] 表示
  - 后序：x 点遍历完成的次序，即回溯时间，用 f[x] 表示
- DFS 序列
  - 前序序列
  - 后序序列
  - 逆后序序列
    - 后序序列的逆序



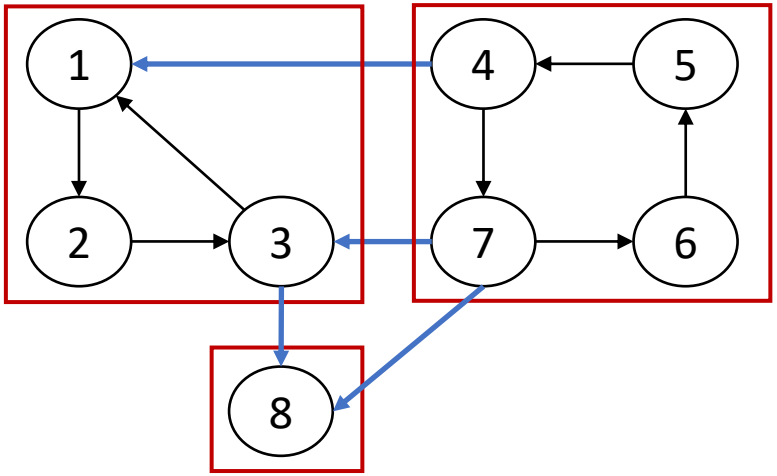
DFS 模拟

使用的边	当前点	数组更新
开始点	1	$d[1]=1$
$1\to 2$	2	$d[2]=2$
$2\to 3$	3	$d[3]=3$
$3\to 1$	1 已经被访问过了	
$3\to 8$	8	$d[8]=4$
8 号点遍历结束	8	$f[8]=1$
3 号点遍历结束	3	$f[3]=2$
2 号点遍历结束	2	$f[2]=3$
1 号点遍历结束	1	$f[1]=4$

```
int n,d[N],f[N],vis[N],dcnt,fcnt;
vector<int> G[N];

void dfs(int x){
    d[x] = ++dcnt; vis[x] = 1;
    for(auto y:G[x])
        if(!vis[y]) dfs(y);
    f[x] = ++fcnt;
}

void solve(){
    dcnt = fcnt = 0;
    memset(vis,0,sizeof vis);
    for(int i = 1; i <= n; i++)
        if(!vis[i]) dfs(i);
}
```



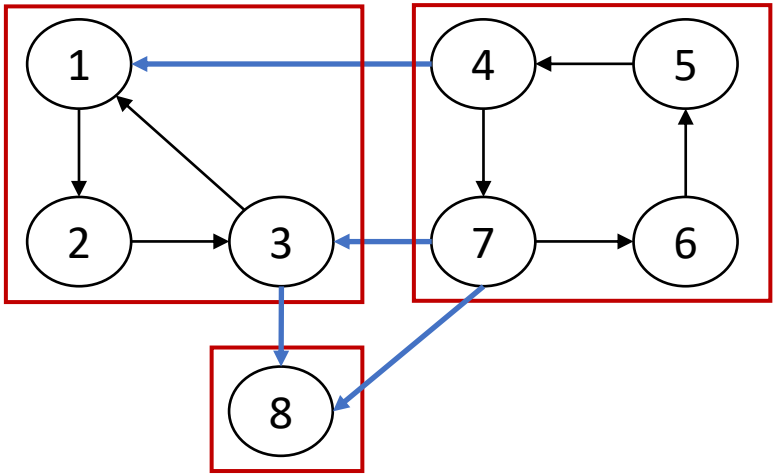
DFS 模拟

使用的边	当前点	数组更新
开始点	4	$d[4]=5$
$4\to 1$	1 已经被访问过了	
$4\to 7$	7	$d[7]=6$
$7\to 3$	3 已经被访问过了	
$7\to 8$	8 已经被访问过了	
$7\to 6$	6	$d[6]=7$
$6\to 5$	5	$d[5]=8$
$5\to 4$	4 已经被访问过了	
5 号点遍历结束	5	$f[5]=5$
6 号点遍历结束	6	$f[6]=6$
7 号点遍历结束	7	$f[7]=7$
4 号点遍历结束	4	$f[4]=8$

```
int n,d[N],f[N],vis[N],dcnt,fcnt;
vector<int> G[N];

void dfs(int x){
    d[x] = ++dcnt; vis[x] = 1;
    for(auto y:G[x])
        if(!vis[y]) dfs(y);
    f[x] = ++fcnt;
}

void solve(){
    dcnt = fcnt = 0;
    memset(vis,0,sizeof vis);
    for(int i = 1; i <= n; i++)
        if(!vis[i]) dfs(i);
}
```



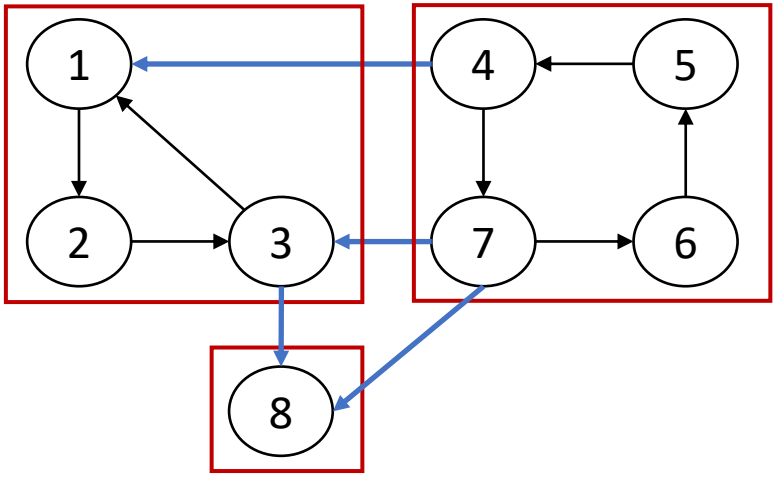
DFS 模拟

当前点 (x)	开始次序 $(d[x])$	结束次序 $(f[x])$
1	1	4
2	2	3
3	3	2
4	5	8
5	8	5
6	7	6
7	6	7
8	4	1

```
int n,d[N],f[N],vis[N],dcnt,fcnt;
vector<int> G[N];

void dfs(int x){
    d[x] = ++dcnt; vis[x] = 1;
    for(auto y:G[x])
        if(!vis[y]) dfs(y);
    f[x] = ++fcnt;
}

void solve(){
    dcnt = fcnt = 0;
    memset(vis,0,sizeof vis);
    for(int i = 1; i <= n; i++)
        if(!vis[i]) dfs(i);
}
```

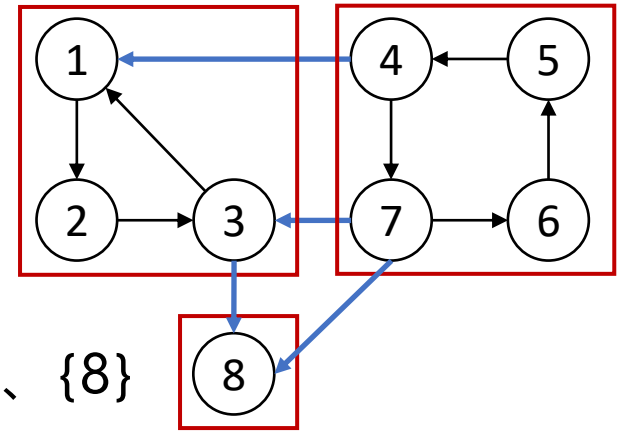


前序序列：1 2 3 8 4 7 6 5

后序序列：8 3 2 1 5 6 7 4

逆后序序列：4 7 6 5 1 2 3 8

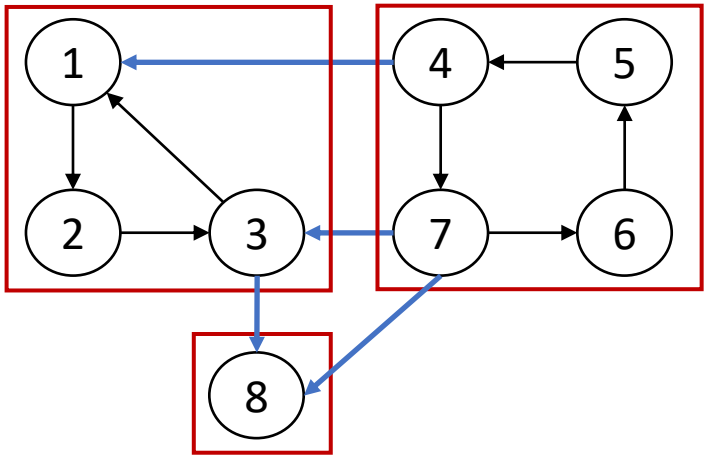
Kosaraju



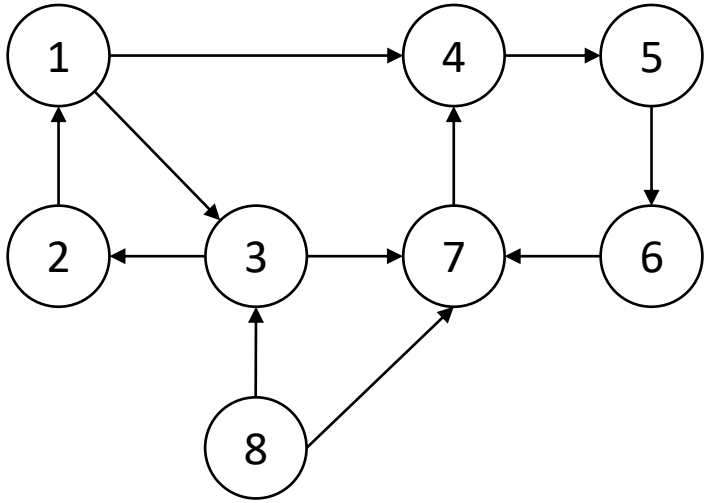
- 算法解决的问题
  - 找到有向图中所有的 SCC
  - 即对于右图找到 {1, 2, 3}、{4, 5, 6, 7}、{8}
- 算法步骤
  - 第一遍 dfs 确定原图的逆后序序列，即 4 7 6 5 1 2 3 8
  - 第二遍 dfs 在反图中按照逆后序序列进行遍历
    - 反图即将原图中的有向边反向
    - 每次由起点遍历到的点即构成一个 SCC

Kosaraju 模拟

- 第一遍 dfs 确定原图的逆后序序列
  - 4 7 6 5 1 2 3 8
- 第二遍 dfs 在反图中按照逆后序序列进行遍历
  - 每次由起点遍历到的点即构成一个 SCC

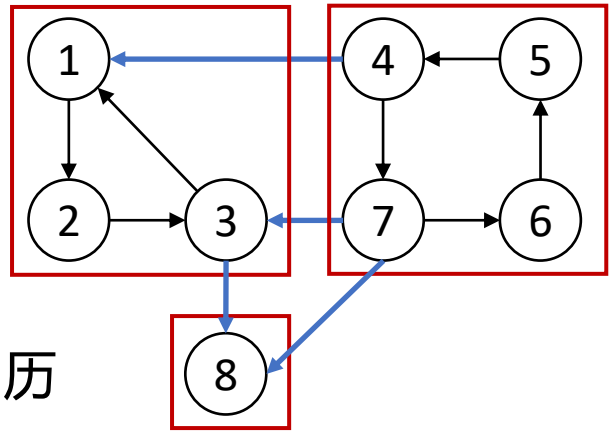


原图

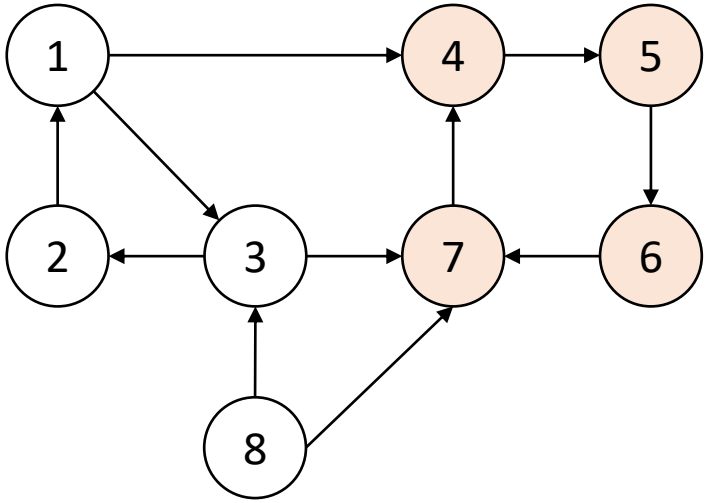


反图

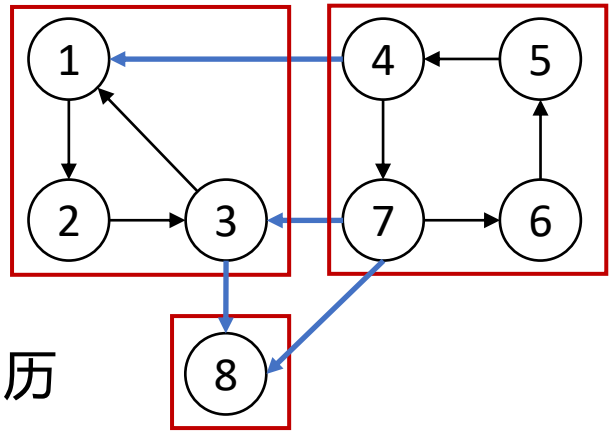
Kosaraju 模拟



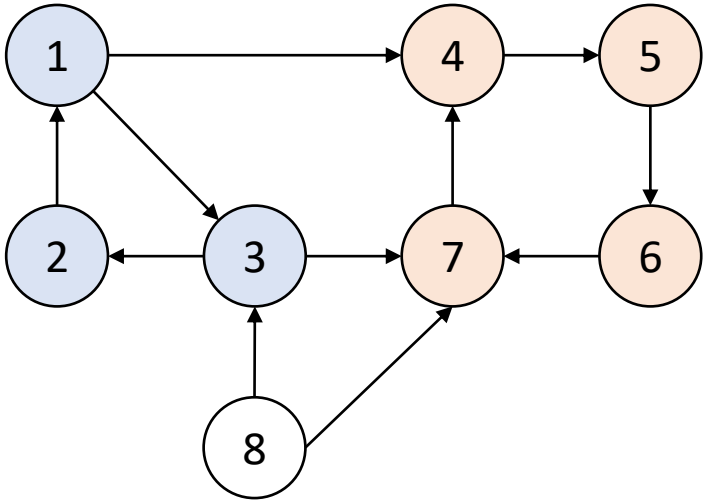
- 第一遍 dfs 确定原图的逆后序序列
  - 4 7 6 5 1 2 3 8
- 第二遍 dfs 在反图中按照逆后序序列进行遍历
  - 每次由起点遍历到的点即构成一个 SCC
- 第二遍 dfs 模拟
  - dfs(4)
    - 4 5 6 7 被染色，属于同一 SCC



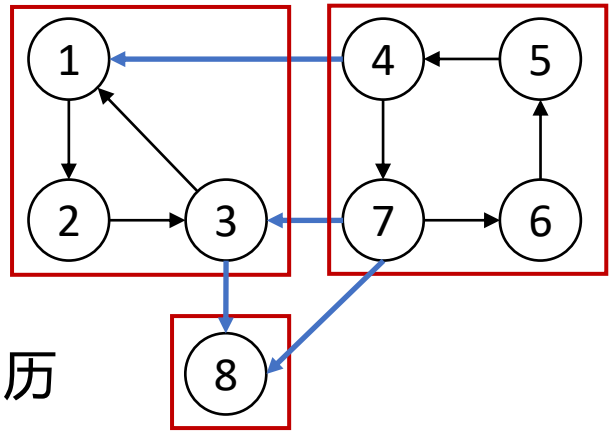
Kosaraju 模拟



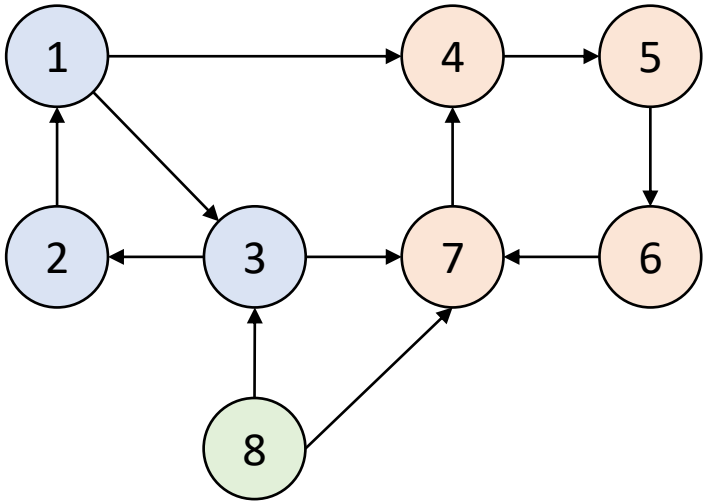
- 第一遍 dfs 确定原图的逆后序序列
  - 4 7 6 5 1 2 3 8
- 第二遍 dfs 在反图中按照逆后序序列进行遍历
  - 每次由起点遍历到的点即构成一个 SCC
- 第二遍 dfs 模拟
  - dfs(4)
    - 4 5 6 7 被染色，属于同一 SCC
  - dfs(1)
    - 1 2 3 被染色，属于同一 SCC



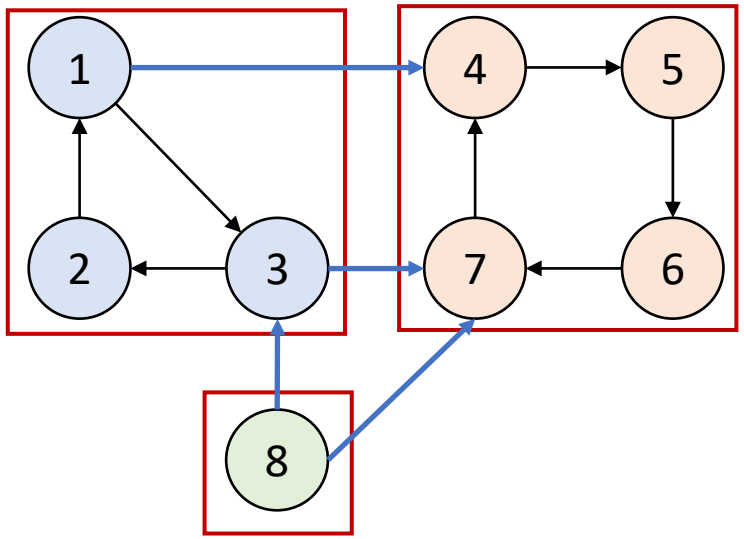
Kosaraju 模拟



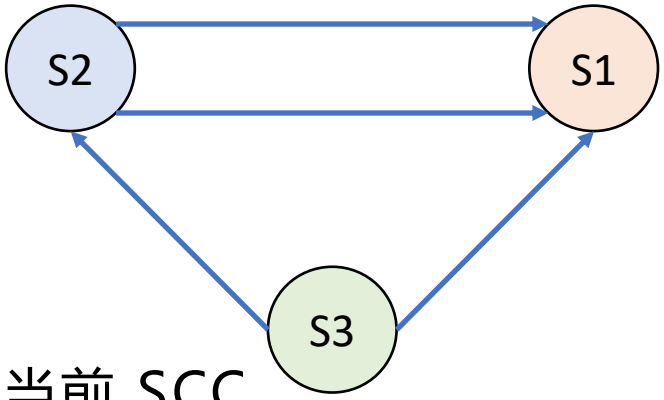
- 第一遍 dfs 确定原图的逆后序序列
  - 4 7 6 5 1 2 3 8
- 第二遍 dfs 在反图中按照逆后序序列进行遍历
  - 每次由起点遍历到的点即构成一个 SCC
- 第二遍 dfs 模拟
  - dfs(4)
    - 4 5 6 7 被染色，属于同一 SCC
  - dfs(1)
    - 1 2 3 被染色，属于同一 SCC
  - dfs(8)



Kosaraju 模拟



- 至此我们得到了原图 3 个 SCC
  - {4 5 6 7}, {1 2 3}, {8}
- 如何理解这个算法
  - 反图与原图具有一样的 SCC
  - 逆后序序列起到了类似于拓扑排序的作用
    - 先遍历 S1
    - 再遍历 S2
    - 最后遍历 S3
    - 优先遍历出边连向的 SCC，再遍历当前 SCC



- 严格数学证明较繁琐，感兴趣可以课后查阅《算法导论》

Kosaraju 代码

```
int n,c[N],dfn[N],vis[N],dcnt,scnt;
// dcnt - dfs序计数, scnt - scc计数
// dfn[i] - dfs后序列中第i个点
// c[i] - i 号点所在 scc 编号
vector<int> G1[N], G2[N]; // G1-原图, G2-反图
```

```
void dfs1(int x){
    vis[x] = 1;
    for(auto y:G1[x])
        if(!vis[y]) dfs1(y);
    dfn[++dcnt] = x;
}

void dfs2(int x){
    c[x] = scnt;
    for(auto y:G2[x])
        if(!c[y]) dfs2(y);
}

void kosaraju(){
    // 初始化
    dcnt = scnt = 0;
    memset(c,0,sizeof c);
    memset(vis,0,sizeof vis);
    // 第一遍 dfs
    for(int i = 1; i <= n; i++)
        if(!vis[i]) dfs1(i);
    // 第二遍 dfs
    for(int i = n; i >= 1; i--)
        if(!c[dfn[i]]) ++scnt, dfs2(dfn[i]);
}
```

5 例　题　4

Example 4

例题 4 (CSP 201509-4)

问题描述

某国有n个城市，为了使得城市间的交通更便利，该国国王打算在城市之间修一些高速公路，由于经费限制，国王打算第一阶段先在部分城市之间修一些单向的高速公路。

现在，大臣们帮国王拟了一个修高速公路的计划。看了计划后，国王发现，有些城市之间可以通过高速公路直接（不经过其他城市）或间接（经过一个或多个其他城市）到达，而有的却不能。如果城市A可以通过高速公路到达城市B，而且城市B也可以通过高速公路到达城市A，则这两个城市被称为便利城市对。

国王想知道，在大臣们给他的计划中，有多少个便利城市对。

输入格式

输入的第一行包含两个整数n, m，分别表示城市和单向高速公路的数量。

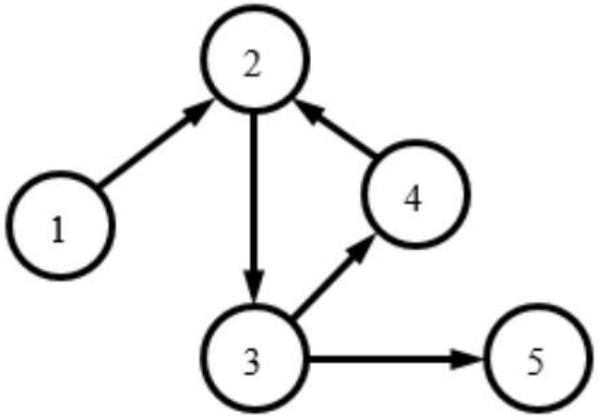
接下来m行，每行两个整数a, b，表示城市a有一条单向的高速公路连向城市b。

输出格式

输出一行，包含一个整数，表示便利城市对的数量。

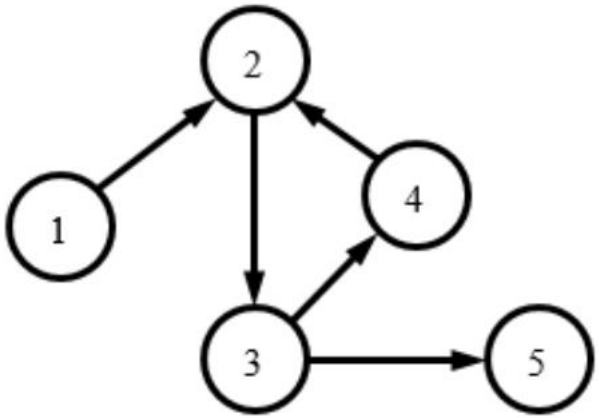
例题 4 (CSP 201509-4)

- 题意
  - N 个点，M 条边的有向图
  - 问有多少对 pair(x, y) 满足 x 到 y 可达，y 到 x 也可达
  - 注意 (3,4) 与 (4,3) 等价
  - $1 \leq N \leq 10^4,\ 1 \leq M \leq 10^5$
- 样例
  - ans = 3
  - (2, 4) (2, 3) (3, 4)



例题 4 (CSP 201509-4)

- 思路
  - pair(x, y) 即 x、y 两两可达
  - 只有 scc 中的点才满足两两可达的性质
  - 因此找出原图中的所有 scc，计算每个 scc 中的 pair 数
    - scc[i] 为第 i 个 scc 中点的个数
    - ans += scc[i] * (scc[i] – 1) / 2



6 例　题　5

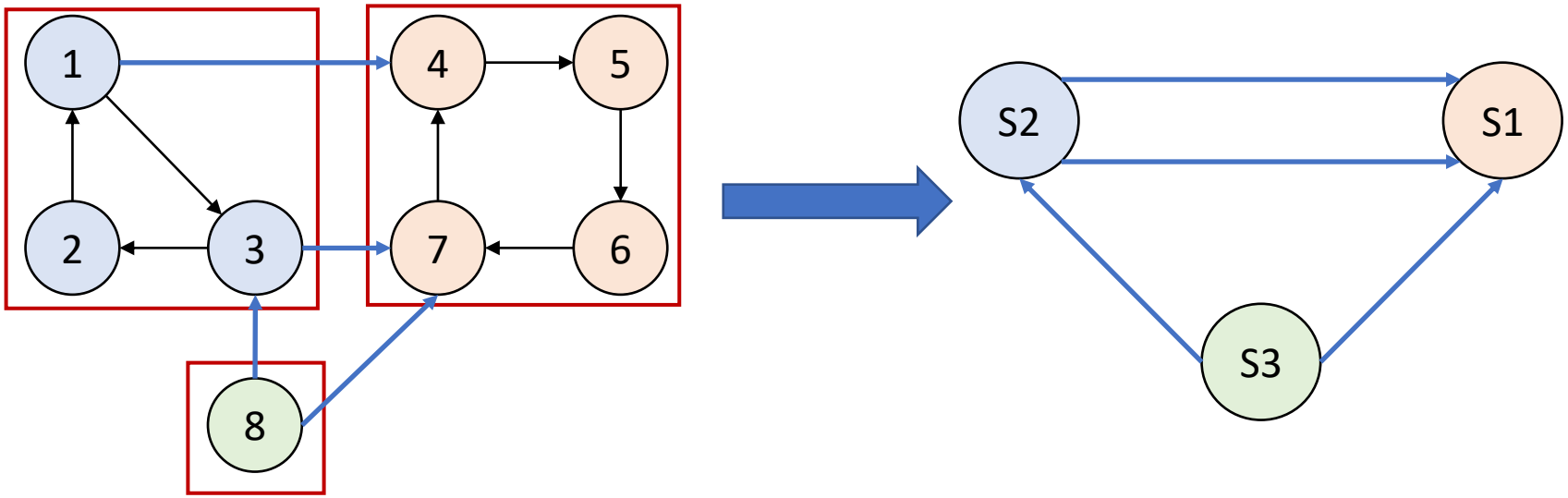
Example 5

例题 5

- 题意
  - 大学班级选班长， N个同学均可以发表意见
  - 若意见为 A B 则表示 A 认为 B 合适，意见具有传递性，即 A 认为 B 合适，B 认为 C 合适，则 A 也认为 C 合适
  - 收集了M条意见，想要知道最高票数，并给出一份候选人名单，即所有得票最多的同学，你能帮帮他吗?
  - $1 \leq N \leq 5 * 10^3$, $1 \leq M \leq 3 * 10^4$
- 样例
  - 4 3
  - （3 2） （2 0） （2 1）
  - ans = 2, {0 1}

例题 5

- 思路
  - 有向图，通常考虑求出 SCC 并缩点，即将互相可达与单向可达分开考虑



例题 5

- 思路
  - 有向图，通常考虑求出 SCC 并缩点，即将互相可达与单向可达分开考虑
  - 缩点后，不难发现对于属于第 i 个 SCC 的点来说，答案分为两部分，令 SCC[i] 表示第 i 个 SCC 中点的个数
    - 当前 SCC 中的点，ans += SCC[i] – 1（去除自己）
    - 其它 SCC 中的点
      - SUM（SCC[j]），其中 j 可到达 i

例题 5

- 思路
  - 缩点后，不难发现对于属于第 i 个 SCC 的点来说，答案分为两部分，令 SCC[i] 表示第 i 个 SCC 中点的个数
    - 当前 SCC 中的点，ans += SCC[i] – 1（去除自己）
    - 其它 SCC 中的点
      - SUM（SCC[j]），其中 j 可到达 I
  - 稍加思考，可以发现最后答案一定出现在出度为 0 的 SCC 中，可以用反证法证明

例题 5

- 思路
  - 缩点后，不难发现对于属于第 i 个 SCC 的点来说，答案分为两部分，令 SCC[i] 表示第 i 个 SCC 中点的个数
    - 当前 SCC 中的点，ans += SCC[i] – 1（去除自己）
    - 其它 SCC 中的点
      - SUM（SCC[j]），其中 j 可到达 I
  - 稍加思考，可以发现最后答案一定出现在出度为 0 的 SCC 中，可以用反证法证明
  - 因此我们将边反向，对每个入度为 0 的点进行 dfs，计算其能到达的点的 SUM(SCC[j])，即可得到答案

感谢收听

Thank You For Your Listening