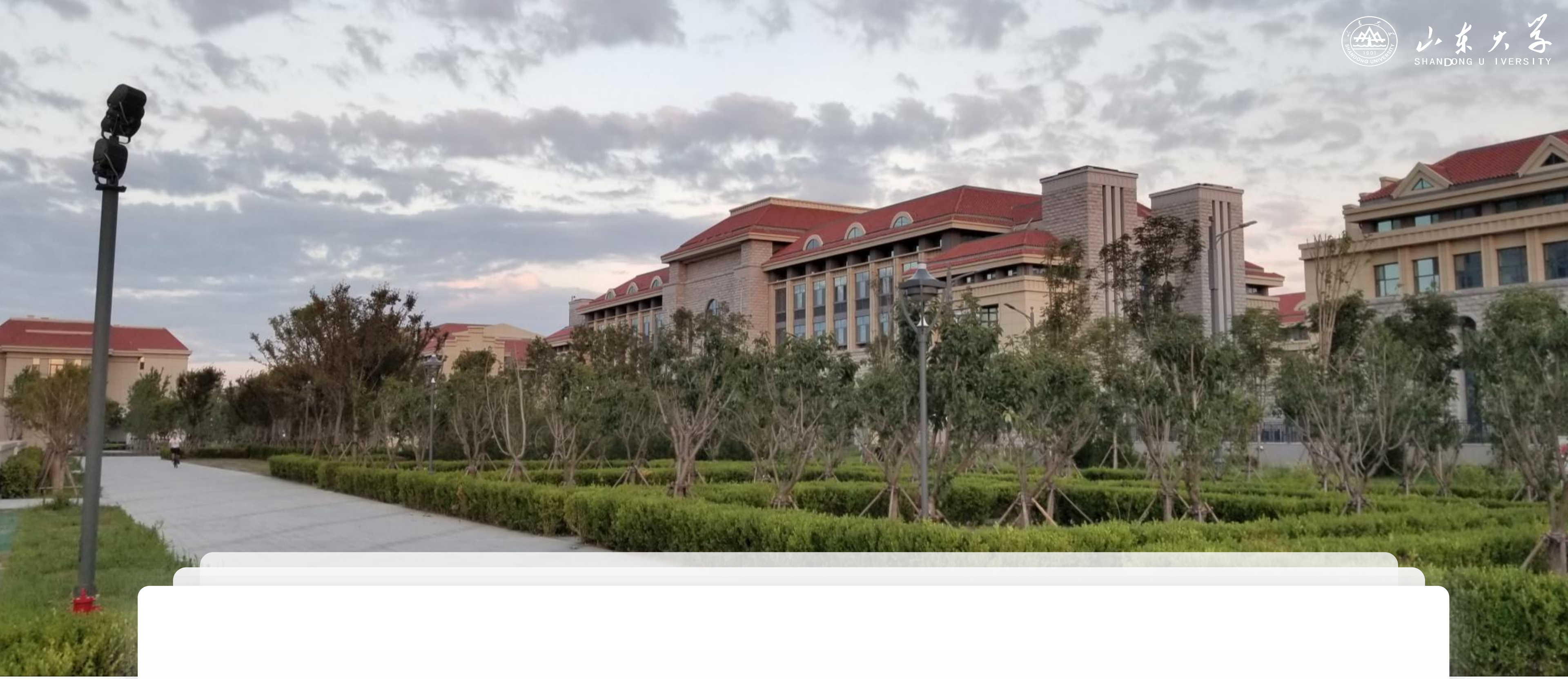
程序设计思维与实践

Thinking and Practice in Programming

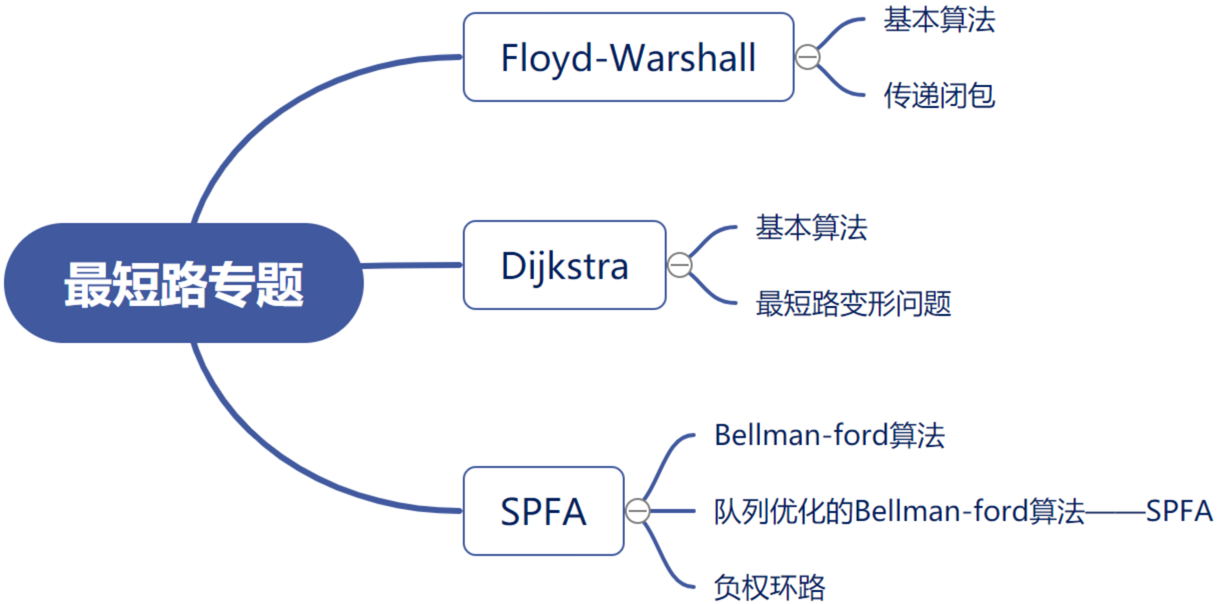
图和树的性质与应用（中） | 内容负责：袁胜利



1 课 程 脉 络

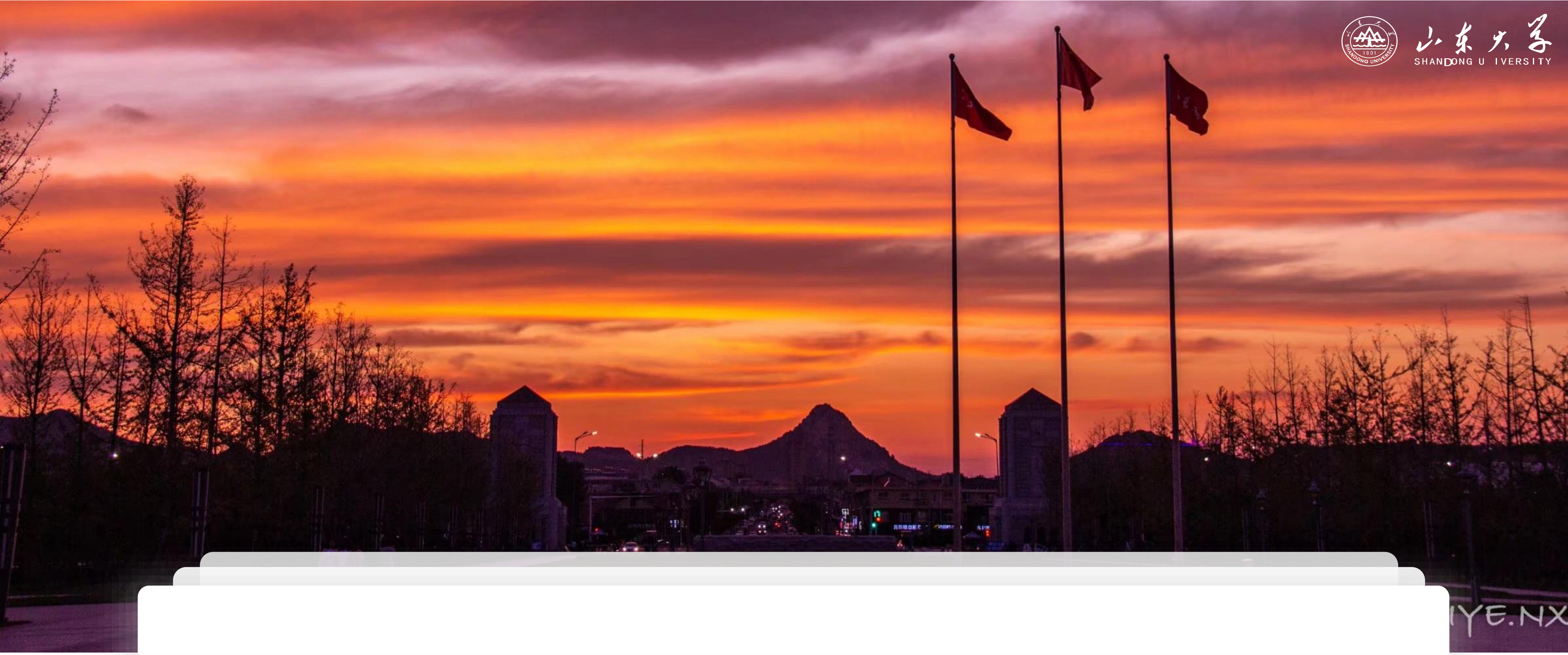
The outline of course

课程脉络



什么是“路”？简单路呢？最短路是边权和最小的路。

注意我们默认图是有向图，无向图也可以把一条双向边拆成两条单向边来转化为有向图



2 弗洛伊德算法

Floyd Algorithm

Floyd–Warshall

- 求取图中任意两点之间的距离
  - $dis[k][x][y]$，只允许中间经过(不包含起点和终点)节点 1 到 k， 节点 x 到节点 y 的最短路长度
  - $dis[n][x][y]$，即节点 x 到节点 y 的最短路长度
  - 初始化
    - 当$x = y$时，$dis[0][x][y] = 0$
    - 当存在边(x, y)时, $dis[0][x][y] = w(x, y)$,
    - 其他情况$dis[0][x][y] = \inf$ (无穷)
  - $dis[k][x][y] = \min(dis[k-1][x][y], dis[k-1][x][k] + dis[k-1][k][y])$

注意最外层枚举k

Floyd–Warshall

- 改进
  - 不难发现我们可以把第一维优化掉
    - 即 $dis[x][y] = \min(dis[x][y], dis[x][k] + dis[k][y])$

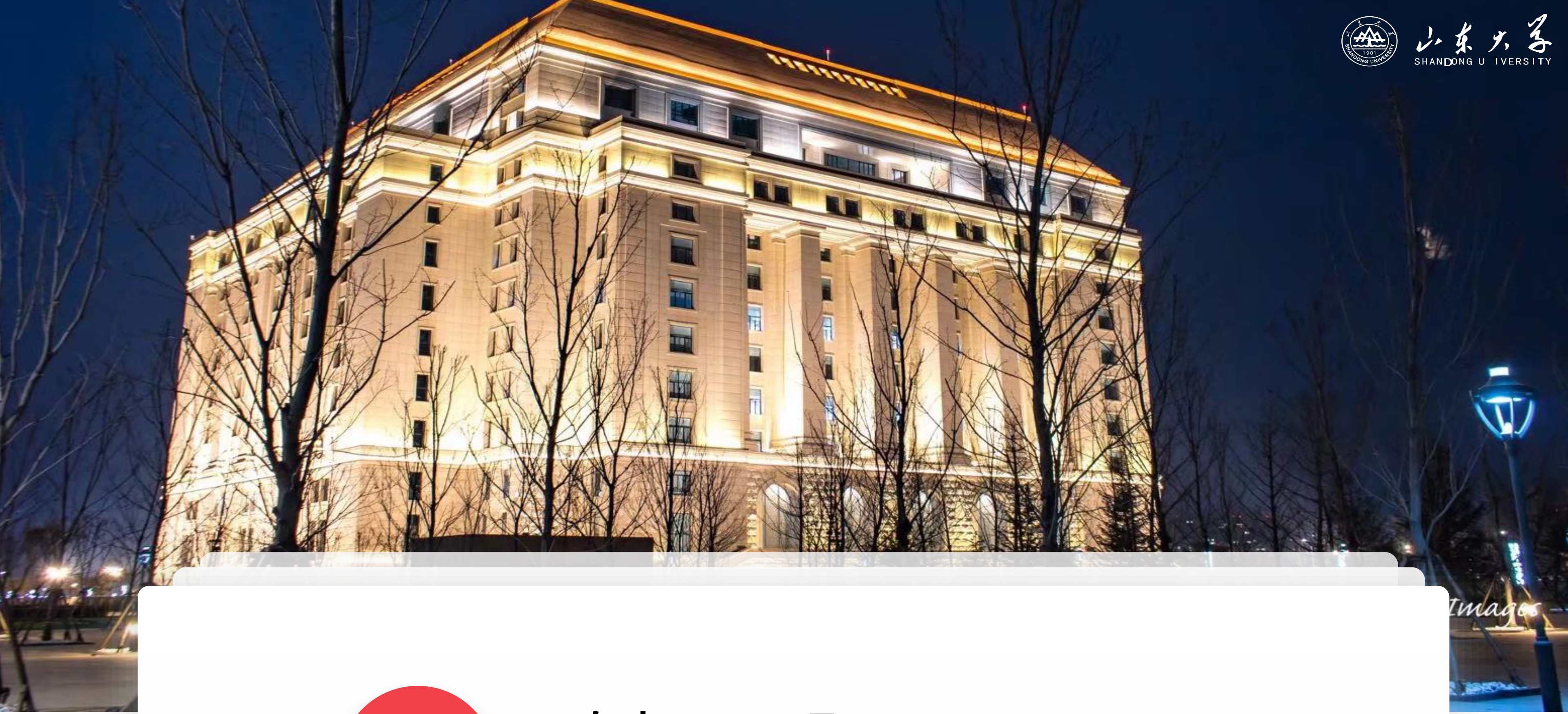
f 数组的第一维可以被优化掉

    - 因为在阶段 k 时， 和 $dis[k][y]$ 不会被更新

Floyd–Warshall

- Floyd–Warshall 算法应用
  - 多源最短路，即任意两点的距离关系
  - 图上的传递闭包，即任意两点的连通关系
  - 复杂度 $O(n^3)$
- Floyd–Warshall 算法实现

```
// n: 点个数, dis: 距离数组, dis[i][j]: 点i到点j的距离
void Floyd(int n, int **dis){
  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]);
}
```



3

例　　题　　1

Example 1

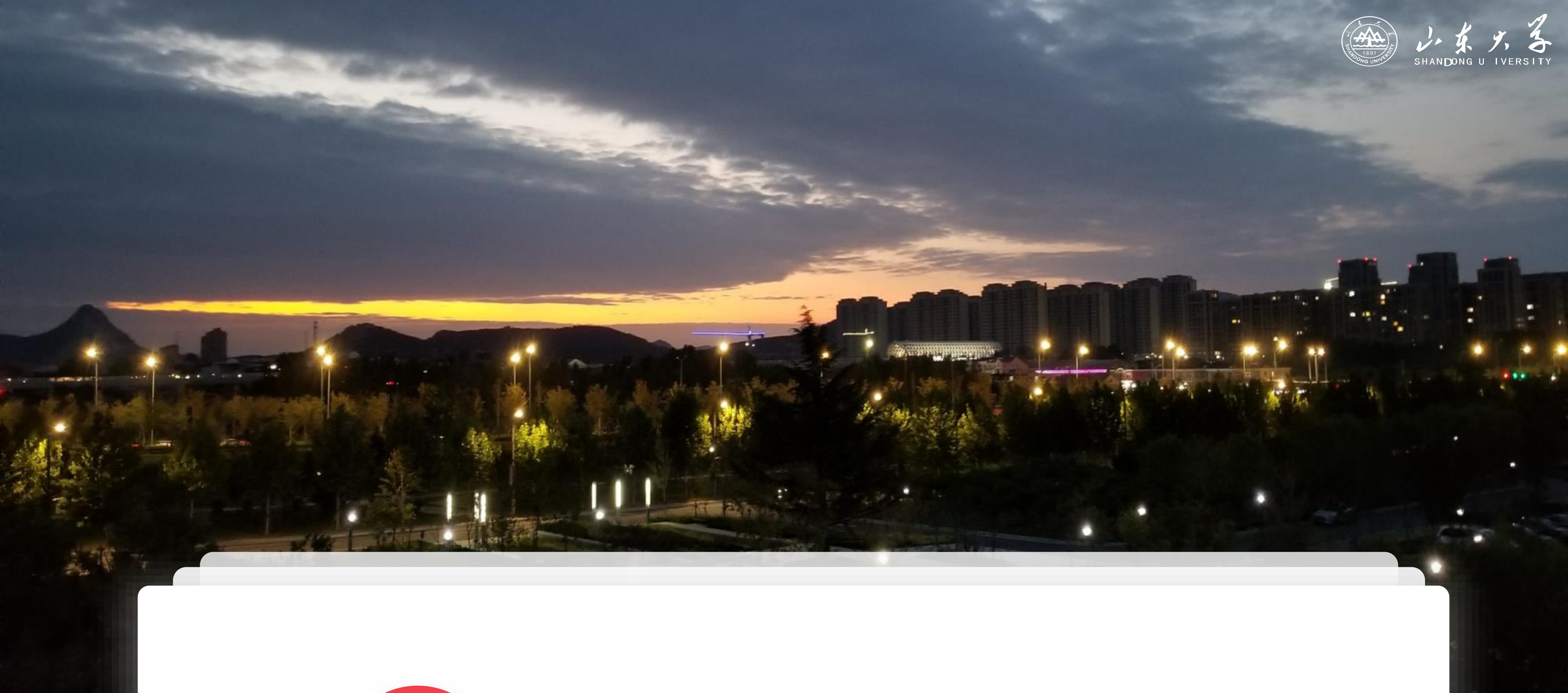
例题 1

- 题意
  - N 个人玩一个游戏，每两个人都要进行一场比赛
  - 已知 M 个胜负关系，每个关系为 A B，表示 A 比 B 强，胜负关系具有传递性，保证关系不会出现矛盾
  - 试问有多少场比赛的胜负无法预先得知?
  - $1 \leq N, M \leq 500$

例题 1

小结论：A必胜B当且仅当在建的胜负关系图中A可达B

- 思路
  - 由 $1 \leq N, M \leq 500$ 数据规模可以得知本题要求复杂度为 $O(n^3)$
  - 因为胜负关系具有传递性，因此可以用 Floyd 算法求出任意两点的胜负关系（传递闭包），即可求出答案
    - $dis[a][b] = 1$ 表示 a 比 b 强
    - $dis[a][b] = 0$ 且 $dis[b][a] = 0$ 即表示 a 与 b 的胜负关系无法预先判断
    - 在更新时$dis[i][j] = dis[i][j] | (dis[i][k] \& dis[k][j])$



4 迪杰斯特拉算法

Dijkstra Algorithm

Dijkstra

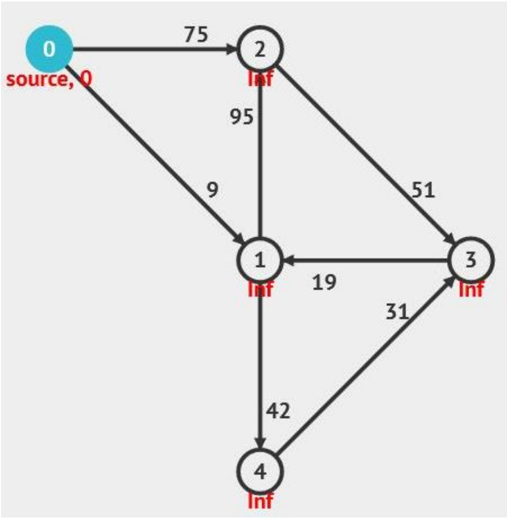
- Dijkstra
  - 该算法主要用于解决图中没有**负边**的单源最短路问题

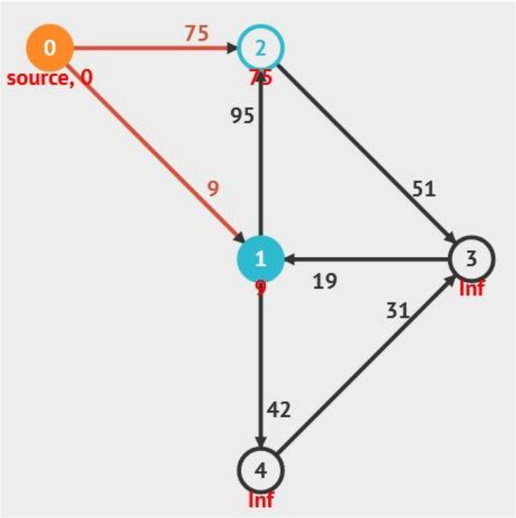
- Dis(x)表示源点到x的最短路径,dis[x]表示我们目前找到的从源点到x的所有“路”中的最短的路。
- 显然dis[x]>=Dis(x)
- 定义“松弛”操作：
  - 对边(u, v)，边权为w(u, v)，若$dis[v] > dis[u] + w(u, v)$,则更新$dis[v] = dis[u] + w(u, v)$

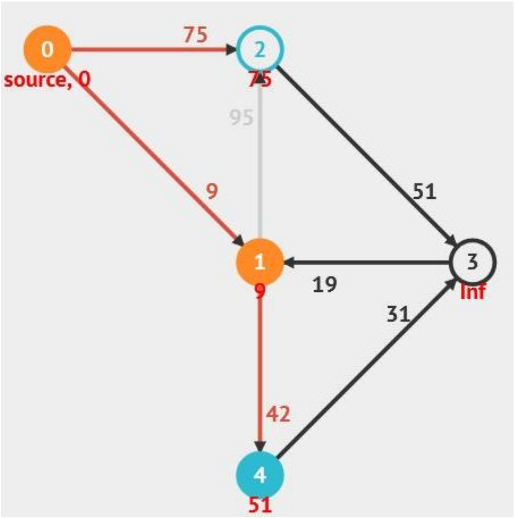
Dijkstra

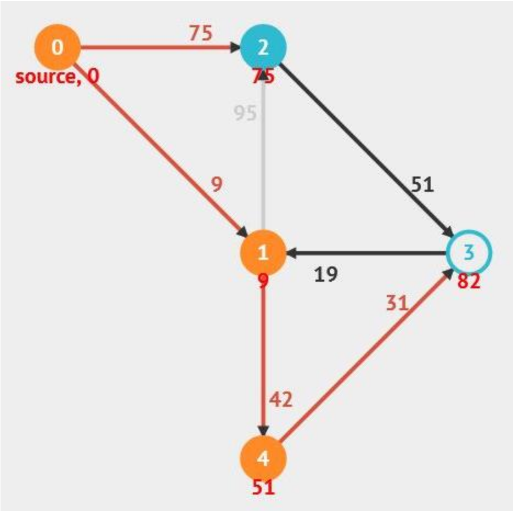
- 算法实现大体流程
  - 假设已经确定最短路长度的点集为S集合，未确定最短路长度的点集为T集合。源点为s，$dis[\mathrm{i}]$表示的含义见上页
  - 初始化S集合为空，T集合为图中所有的点。$dis[s] = 0$，$dis[\mathrm{i}] = \mathrm{inf}$ (无穷大)
  - 重复以下操作
    - 从T集合选取dis数组值最小的结点u，<u>一定有Dis(u)=dis[u]</u>，将u移动到S集合
      - 对刚选取的结点的所有出边进行松弛操作
  - 直到T集合为空，算法结束。
- 复杂度为$O(n^2+m)$

Dijkstra

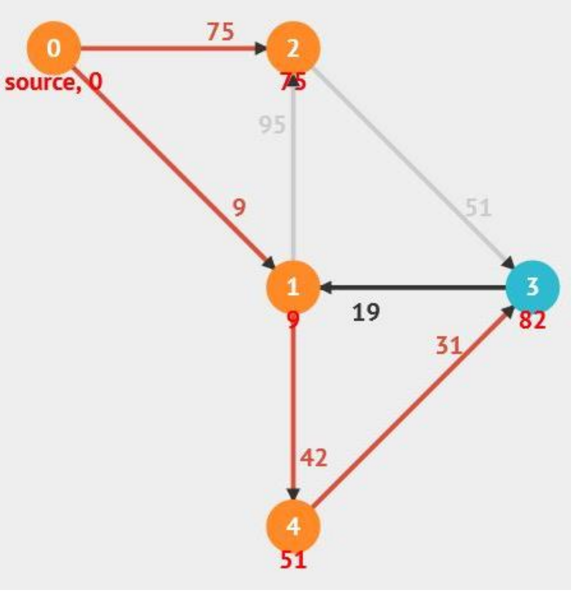

0
source, 0
75
2
Inf
95
9
51
1
Inf
19
3
Inf
31
42
4
Inf

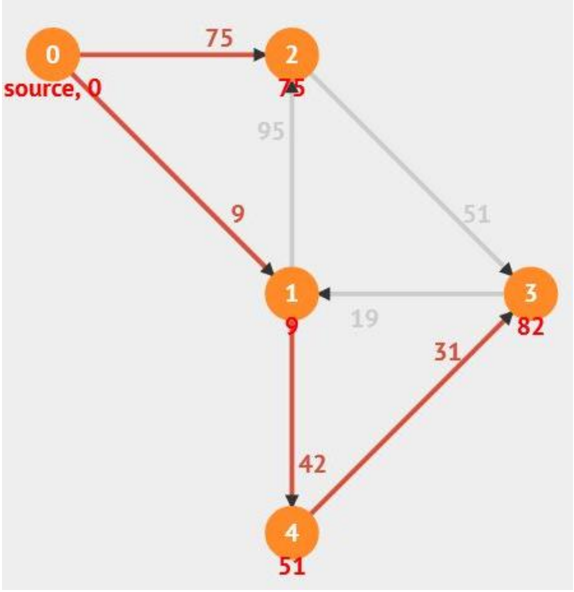

0
source, 0
75
2
75
95
9
51
1
9
19
3
Inf
31
42
4
Inf


0
source, 0
75
2
75
95
9
51
1
9
19
3
Inf
31
42
4
51


0
source, 0
75
2
75
95
9
51
1
9
19
3
82
31
42
4
51

Dijkstra





最短路树

Dijkstra

- 思考：从T集合选点过程是否可以进行优化
- 使用优先队列(堆)或者set(平衡树)进行维护

Dijkstra

- 优化后的算法实现流程
  - 源点为s, $dis[\mathrm{i}]$ 表示源点 s 到点 i 的最短距离。
  - 初始化$dis[s] = 0$，$dis[\mathrm{i}] = \mathrm{inf}$(无穷大)，将s加入优先队列中
  - 重复以下操作
    - 从优先队列中取出一个点u，对u的所有出边(u, v)进行松弛操作
    - 若松弛成功，则将v加入到优先队列中
  - 直到优先队列为空，算法结束。

Dijkstra

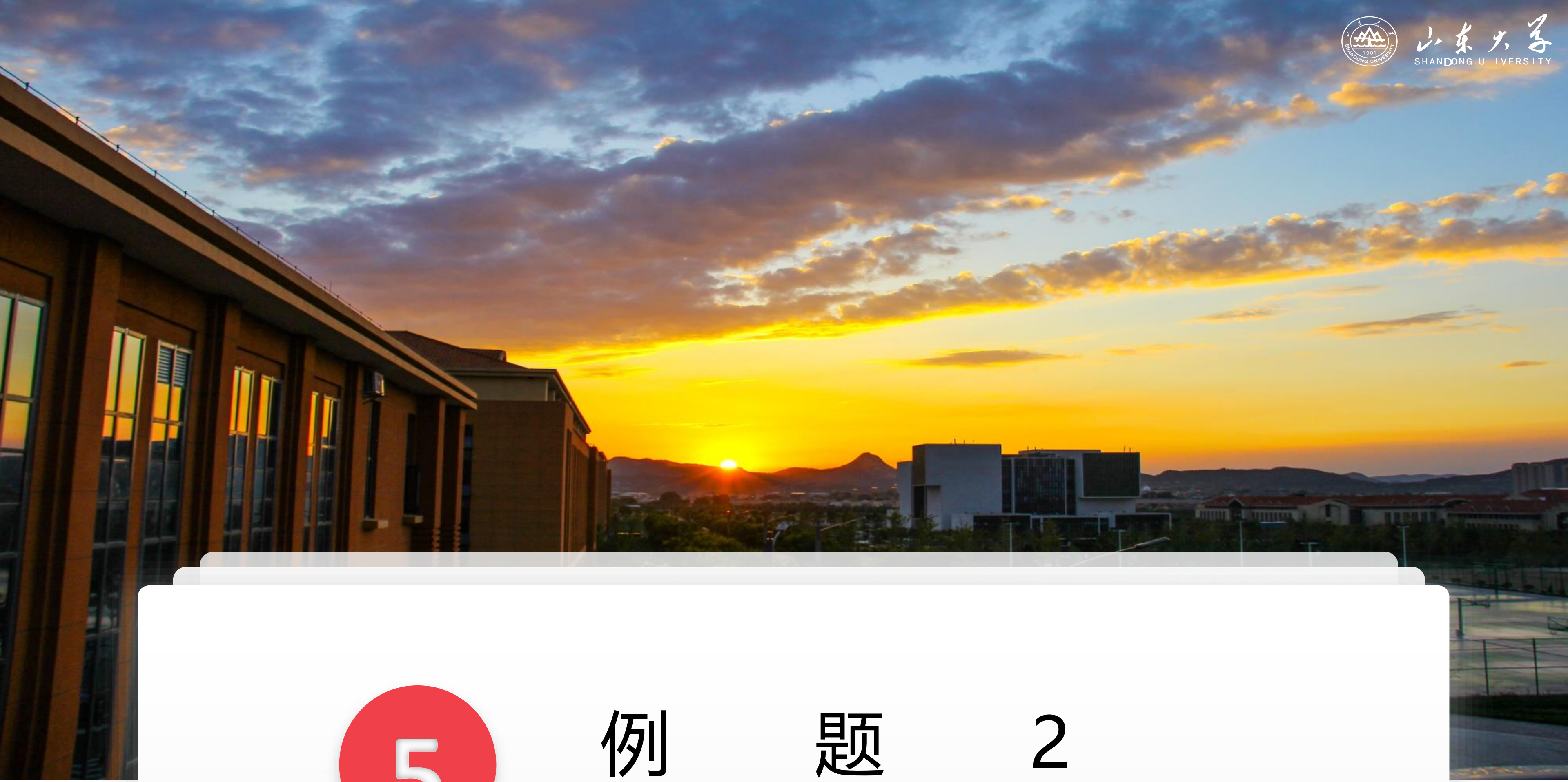
- 一个点可能重复入队，如何解决？
- 设置vis数组，记录是否被弹出
  - 一旦某个点被优先队列弹出，则不会再被松弛，dis 的值为最短路
- 如何记录最短路的路径？
- 复杂度为$O((n+m)logn)$

代码

```
const int N=100010,M=400010;
struct node {
    int dis,pos;
    bool operator < ( const node &x ) const
    {
        return x.dis<dis;
    }
}e;
priority_queue < node > q;
struct tnode{
    int nxt,to,w;
}a[M];
int dis[N],h[N],vis[N];
```

```
void dijkstra(int s)
{
    memset(dis,0x3f,sizeof(dis)); dis[s]=0;
    q.push((node){0,s});
    while(!q.empty())
    {
        node e=q.top(); q.pop();
        int u=e.pos;
        if(vis[u]) continue;
        vis[u]=1;
        for(int i=h[u];i;i=a[u].nxt)
        {
            int v=a[i].to;
            if(dis[u]+a[i].w<dis[v])
            {
                dis[v]=dis[u]+a[i].w;
                q.push((node){dis[v],y});
            }
        }
    }
}
```

```
cin>>n>>m>>s;
for(int i=1;i<=m;++i)
{
    cin>>u>>v>>w;
    add(u,v,w); //add(v,u,w);
}
dijkstra(s);
for(int i=1;i<=n;++i) cout<<dis[i]<<" ";
return 0;
```



5 例题2

Example 2

例题 2

- 题意
  - 在喵星中，猫猫快线是市民从市内去喵星机场的首选交通工具。猫猫快线分为经济线和商业线两种，线路、速度和价钱都不同。TT 有一张商业线车票，可以坐一站商业线，而其他时候只能乘坐经济线。假设换乘时间忽略不计，你的任务是找一条去喵星机场最快的线路。
  - 输入猫猫快线中的车站总数，起点和终点，以及商业线与经济线连接的两个车站和单程花费的时间
  - $1 \leq$ 车站数量 $\leq 1e5$，$1 \leq$ 商业线数量 $\leq 1e5$， $1 \leq$ 经济线数量 $\leq 1e5$

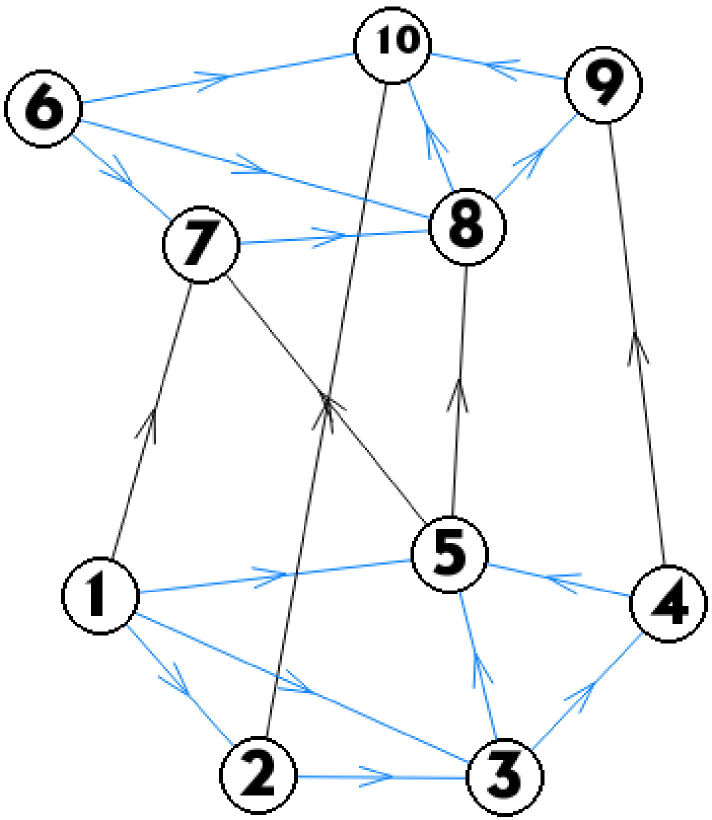
例题 2

- 思路 1
  - 题目给定了起点与终点，而且要求商业线最多乘坐一次
  - 可以枚举每一条商业线，计算起点到u的最短路以及v到终点的最短路再加上该商业线所花费的时间

- 以起点为源点求单源最短路，得到 dis1 数组
- 再以终点为源点求反图的单源最短路，得到 dis2 数组
- 枚举商业线(u, v, w)，取 min{dis1[u]+dis2[v]+w, dis1[v]+dis2[u]+w}，最终再与不走商业线的答案取min

例题 2

- 思路 2
  - 建立分层图，如右图所示
  - 在分层图上直接使用Dijkstra算法，最终结果为 $\min(dis[n], dis[2*n])$

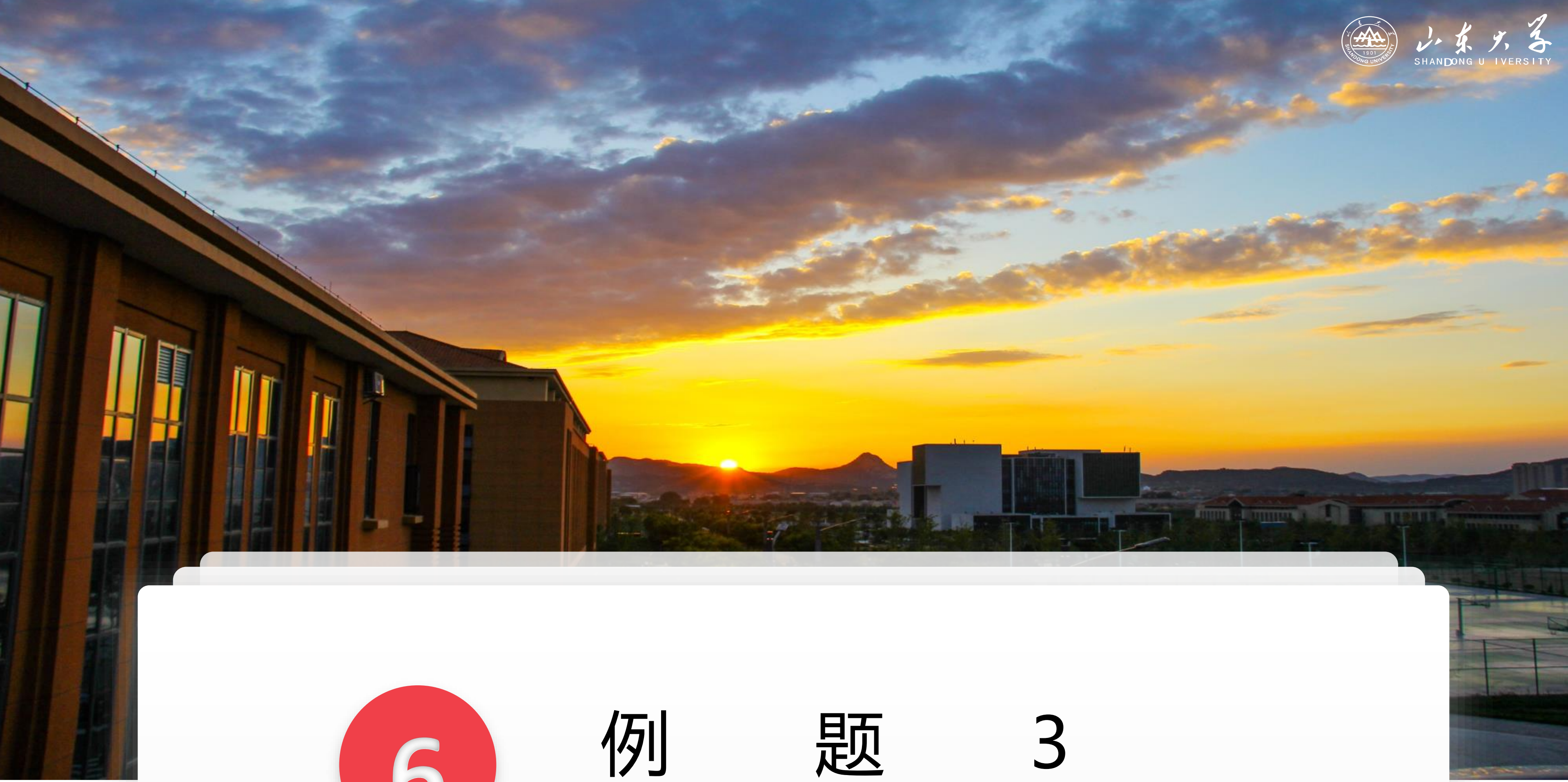


当然，分层图可以虚建图

例题 2

- 思路 3
  - 跑一次单源最短路（变形），记录答案 dis[u][0/1]
  - dis[u][0] 表示从起点到结点 u 没有经过商业线时的最短路，在松弛的时候可以选择商业线或者经济线
  - dis[u][1] 表示从起点到结点 u 经过商业线后的最短路，在松弛的时候只能选择经济线

此时我们回到week6的一道题
BFS求最短路
将优先队列变成桶



6 例 题 3

E x a m p l e 3

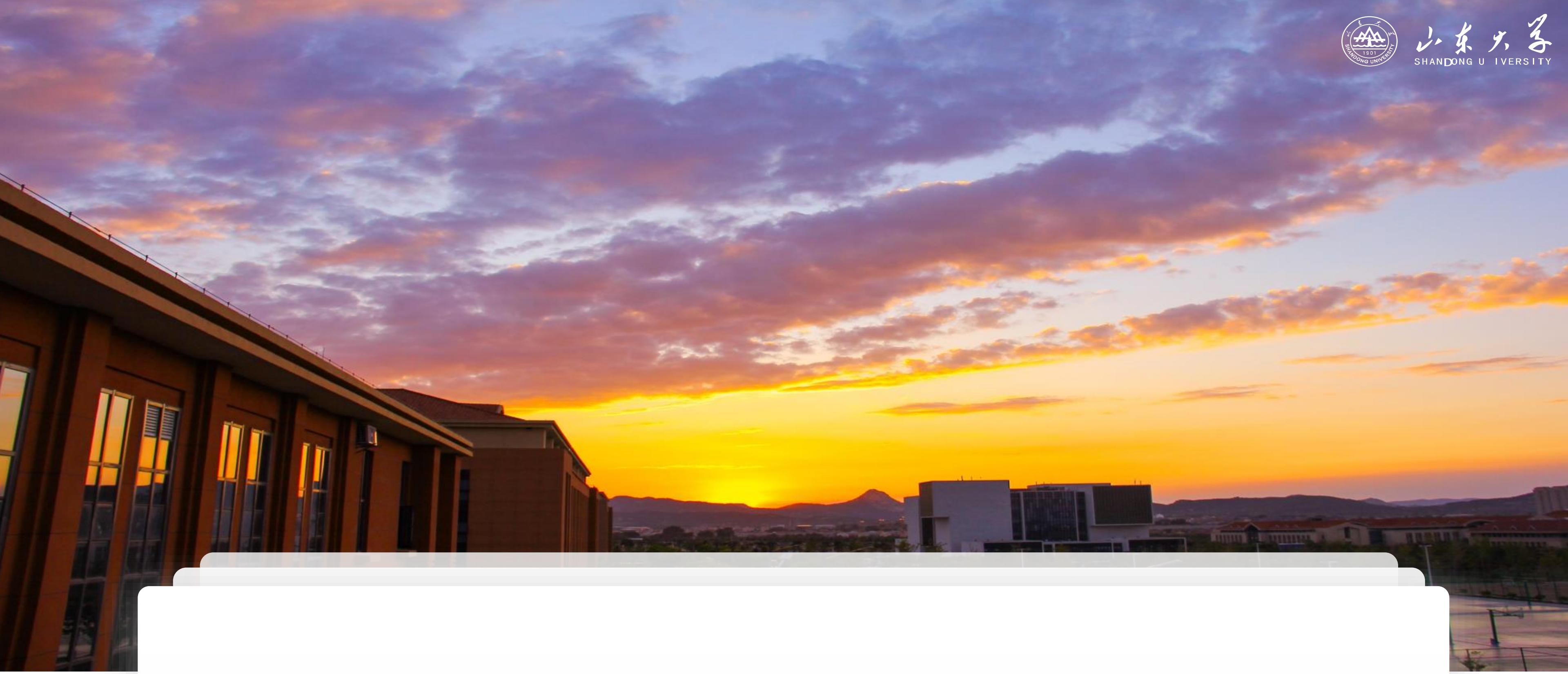
例题 3

- 对于一个n个点、m条边的有向无环图，1号点为起点，n号点为终点，每条边连接两个不同的顶点并拥有两个参数，分别为最大承重量C和通行时间D。
- 现要从起点向终点运送货物，一条路径所能承受的最大重量为路径经过的边中最大承重量C的最小值。
- 你只有T时间可以运送货物，即从起点到终点的运输时间（经过边的通行时间之和）必须小于等于T，求可以运送的最大重量。

- $1 \le n \le 1e4$， $1 \le \mathrm{m} \le 5e4$， $1 \le T \le 5e5$
- $1 \le C \le 1e7$， $1 \le D \le 5e4$

例题 3

- 能运送货物的最大重量取决于路径上边最大承重量的最小值minC
- 若已知最小值minC,则可以对图求单源最短路，得到1到n的最短路（只能走C大于等于minC的边）
- 对于两个值minC1,minC2且minC1<minC2，minC1对应的最短路大于T,则minC2对应的最短路也一定大于T
- 也就是说问题是满足单调性的，所以我们可以二分minC然后判定是否可行即可



7 贝尔曼-福特

Bellman-Ford

Bellman-ford

- 前面讲解的 Floyd 算法用解决了多源最短路径问题，Dijkstra 算法也给出了对于单源最短路径问题一种不错的解法。

- 思考一下它们的局限性
  - Floyd 算法解决的是多源最短路径问题，对于单源最短路径问题有些许小题大做(主要是复杂度不允许)
  - Dijkstra 算法在图中存在负权边时不能保证结果的正确性

- 在这种情况下，一种新的单源最短路径算法呈现在我们眼前

- Bellman-ford 算法

Bellman-ford

- 一句话：重复n轮，每轮按任意顺序松弛所有边
- Bellman-ford 算法可以给出源点 S 至图内其他所有点的最短路。(或发现最短路是负无穷)
- Bellman-ford 算法的正确性基于以下事实
  - 最短路经过的边的条数小于图中点的个数
    - 若大于等于点的个数，则意味着存在某点被重复经过
  - 当松弛边 (u, v) 时，如果 dis[u] 已经是最短路且 u 是 v 的最短路树的父亲，则松弛结束后 dis[v] 也是最短路，并且从此以后其dis值不会发生变化

Bellman-ford

- Bellman-ford 算法的整体思路比较简单暴力
  - 对图中的每一条边进行松弛操作，由于最短路的路径条数小于点的个数，故松弛 点数-1 轮即可
  - 第 i 轮松弛完毕后，所有经过 i 条边(即最短路树上深度为i+1)的最短路均被确定

- 思考：如何记录路径
- 答：记录前驱(得到最短路树)

Bellman-ford

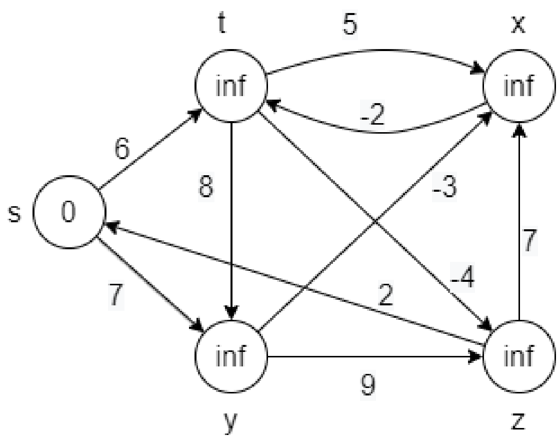
- 代码实现

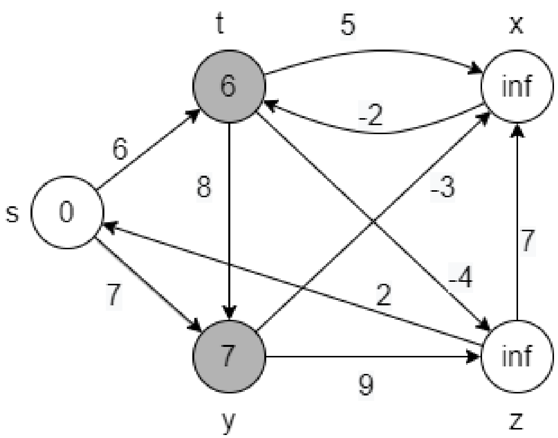
```
for(int i = 1; i <= n; ++i) {
    dis[i] = INF;
    pre[i] = 0;
}
dis[s] = 0;
for(int k = 1; k < n; ++k)
    for(int i = 1; i <= m; ++i)
        if(dis[v[i]] > dis[u[i]] + w[i]) {
            dis[v[i]] = dis[u[i]] + w[i];
            pre[v[i]] = u[i];
        }
```

- 时间复杂度为 $O(nm)$

Bellman-ford

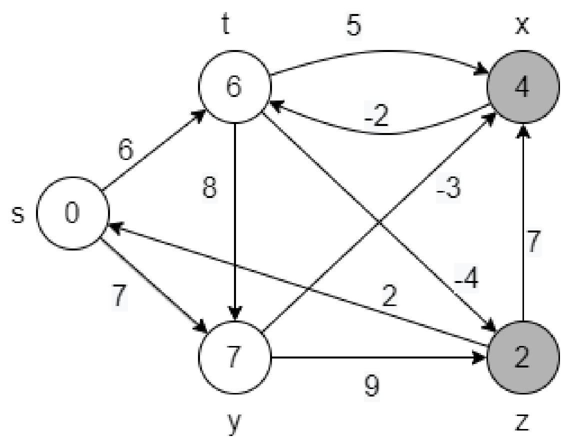
源点为s，松弛的顺序为(t, x),(t, y),(t, z),(x, t),(y, x),(y, z),(z, x),(z, s),(s, t),(s, y)

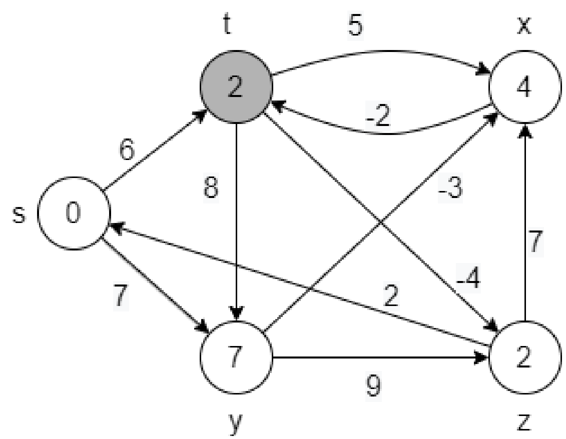


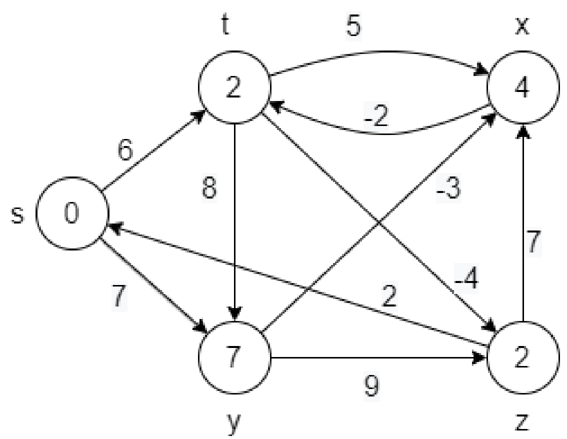


Bellman-ford

源点为s， 松弛的顺序为(t, x),(t, y),(t, z),(x, t),(y, x),(y, z),(z, x),(z, s),(s, t),(s, y)







SPFA

- Bellman-ford 算法完美地解决了负权边的问题，但是它的复杂度过高，堪比复杂度为 $O(n^3)$ 的 Floyd 算法，令人无法接受。

- 观察 Bellman-ford 算法中的松弛过程。
- 在第一轮松弛的时候，最短路上的第一条边被确定;
- 在第二轮松弛时，最短路上的第二条边被确定;
- 在第三轮松弛的时候，最短路上的第三条边被确定。
- 。。。

- 在 Bellman-ford 算法中，每一轮有很多无效的松弛操作，怎样才能避免?

- 解决了这个问题就得到了 Bellman-ford 的队列优化算法 —— SPFA

SPFA

- SPFA算法的全称是：Shortest Path Faster Algorithm

- 在之前观察 Bellman-ford 算法的松弛过程中，我们发现，松弛操作仅仅发生在最短路径前导结点中已经成功松弛过的结点上

- 第一轮，与 S 邻接的点被松弛 -> 最短路径上的第一条边
- 第二轮，与第一轮被松弛的点相邻接的点被松弛 -> 最短路径上的第二条边
- 。。。

- 这样我们不妨每次只做有效的松弛操作
  - 建立一个队列
  - 队列中存储被成功松弛的点
  - 每次从队首取点并松弛其邻接点
  - 如果邻接点成功松弛则将其放入队列

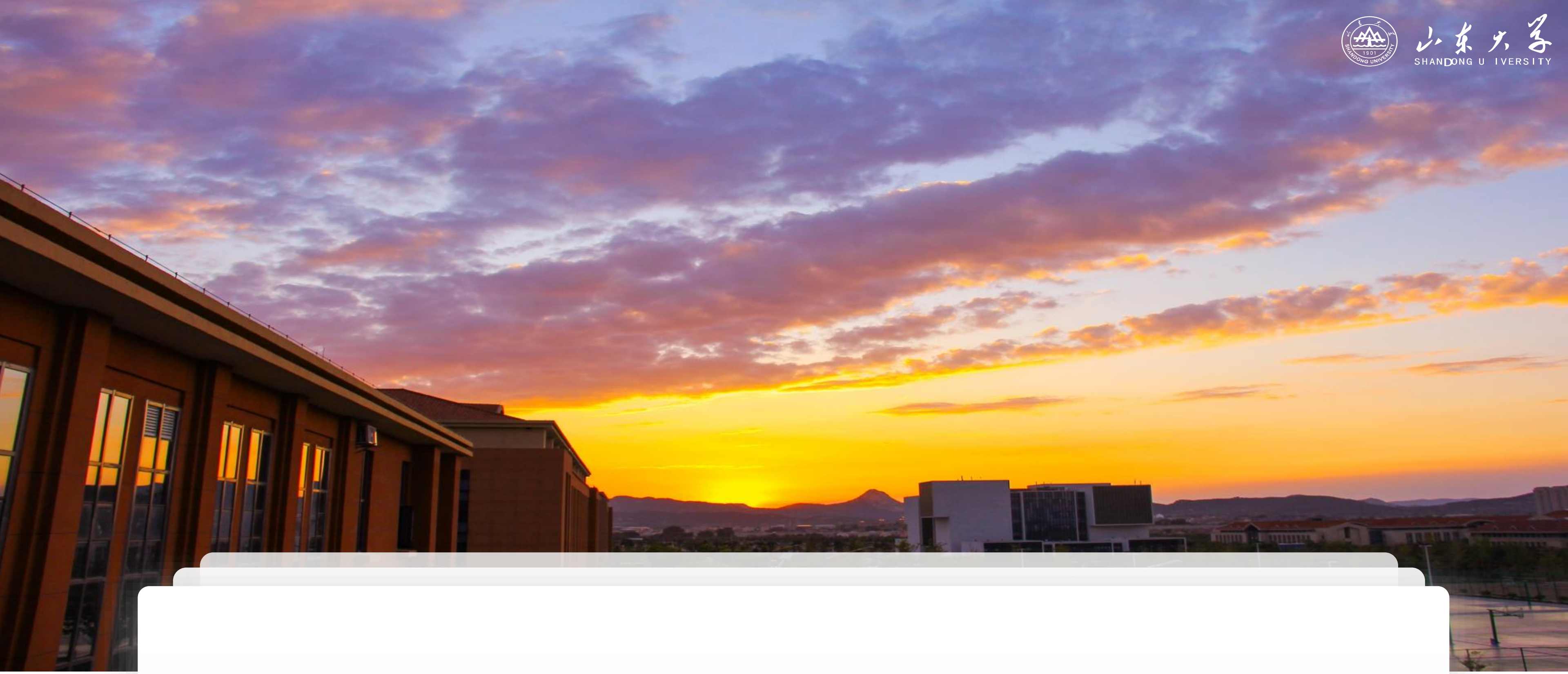
- 思考：
  - 队列如何初始化？　源点 S 入队
  - 重复入队？　数组记录是否在队列中

SPFA

- 算法代码

```
void spfa(int s){
    for(int i = 1;i <= n;++i) dis[i] = inf,vis[i] = false;
    dis[s] = 0;vis[s] = true;
    queue<int> q;
    q.push(s);
    while (!q.empty()){
        int x = q.front();q.pop();
        for(auto i:edge[x]){
            if(dis[i.first] > dis[x]+i.second){
                dis[i.first] = dis[x] + i.second;
                pre[i.first] = x;
                if(!vis[i.first]){
                    vis[i.first] = true;
                    q.push(i.first);
                }
            }
        }
        vis[x] = false;
    }
}
```

- 时间复杂度平均为 km，**注意这里不能写O**
- K是一个远小于 n 的甚至2左右的小常数
- 但是在特殊情况下 k 可能很大
- 即队列优化的 Bellman-ford 算法在特殊情况下时间复杂度会退化为$O(nm)$



8 负权环路

Negative Loop

负权环路

- Bellman-ford 算法及其队列优化可以解决负权边的问题，而且 SPFA 的时间复杂度较为优秀

- 思考:
  - 最短路一定存在吗?
  - 什么情况下最短路不存在

- S 不可达
- 图中存在负权环路

负权环路

- 当解决单源最短路径问题时，如果图中边的权值非负时，不需要考虑额外的问题。
- 如果图中含有负权边时，则需要考虑图中最短路是否存在。也就是我们应用 Bellman-ford 算法及其队列优化所求出的最短路是否有效。
- 如果存在负环，那么求出来的最短路是无效的!
- 我们需要判断图中是否含有负权环路（负环）！

负权环路

- 在 Bellman-ford 算法中如何判断负环的存在?
- 如果存在负环，那么最短路经过的边数会大于等于 n
- 一些边被松弛的次数会大于等于 n
- 如果在第 n 次松弛操作时还存在边能够被成功松弛，那么图中存在负环

负权环路

- 修改后的 Bellman-ford 算法代码

```
for(int i = 1; i <= n; ++i) {
    dis[i] = INF;
    pre[i] = 0;
}
dis[s] = 0;
for(int k = 1; k < n; ++k)
    for(int i = 1; i <= m; ++i)
        if(dis[v[i]] > dis[u[i]] + w[i]) {
            dis[v[i]] = dis[u[i]] + w[i];
            pre[v[i]] = u[i];
        }
for(int i = 1; i <= m; ++i) {
    if(dis[v[i]] > dis[u[i]] + w[i]) {
        // 存在负环
    }
}
```

负权环路

- SPFA 算法是 Bellman-ford 算法的队列优化，负环存在的判断条件与 Bellman-ford 算法一致

- 思考：如何判断一条边被松弛的次数大于等于 n？

- SPFA 中将点加入队列
  - 方法1：判断点的成功松弛次数，如果某一点成功松弛次数超过n 次则说明有负环
  - 方法2：判断最短路的边数，如果到某一点的最短路的边数超过了n-1则说明有负环

负权环路

- 修改后的 SPFA 算法
- cnt[x] 表示 x 当前松弛的次数
- 每次松弛成功时更新cnt[v]，若某一时刻cnt[v]大于等于n的话说明图中存在负环。

```
bool spfa(int s){
    for(int i = 1;i <= n;++i) dis[i] = inf,vis[i] = false,cnt[i] = 0;
    dis[s] = 0;vis[s] = true;
    queue<int> q;
    q.push(s);
    while (!q.empty()){
        int x = q.front();q.pop();
        for(auto i:edge[x]){
            if(dis[i.first] > dis[x]+i.second){
                cnt[i.first]++;
                if(cnt[i.first] >= n)return true;
                dis[i.first] = dis[x] + i.second;
                pre[i.first] = x;
                if(!vis[i.first]){
                    vis[i.first] = true;
                    q.push(i.first);
                }
            }
        }
        vis[x] = false;
    }
    return false;
}
```

9 例题4

Example 4

例题 4

- 有n个点，m条边，边是有向边，每条边有两个权值V_i, P_i，图不保证连通
- 求出在所有的回路中 $\frac{\sum V_i}{\sum P_i}$ 的最大值是多少。
- 回路不一定包括所有的点，但是不可以包含重复的点

- $1 \le n \le 2e3$，$1 \le \mathrm{m} \le 5e3$，$1 \le V_i, P_i \le 1e3$，结果保留一位小数，保证答案在200以内

例题 4

- 设ans为答案，则有$\frac{\sum V_i}{\sum P_i} \le ans$
- $ans * \sum P_i - \sum V_i \ge 0$
- $\sum(ans * P_i - V_i) \ge 0$
- 如果一个解可以，比这个解大的都可以，考虑二分ans，跑SPFA
- 在进行SPFA过程中，将边权改为$ans * P_i - V_i$，判断负环即可

例题 4

- 如何解决图不连通问题?
- 设一个超级源点S，建立S到其余n个点的边，边权为0

感谢收听

Thank You For Your Listening