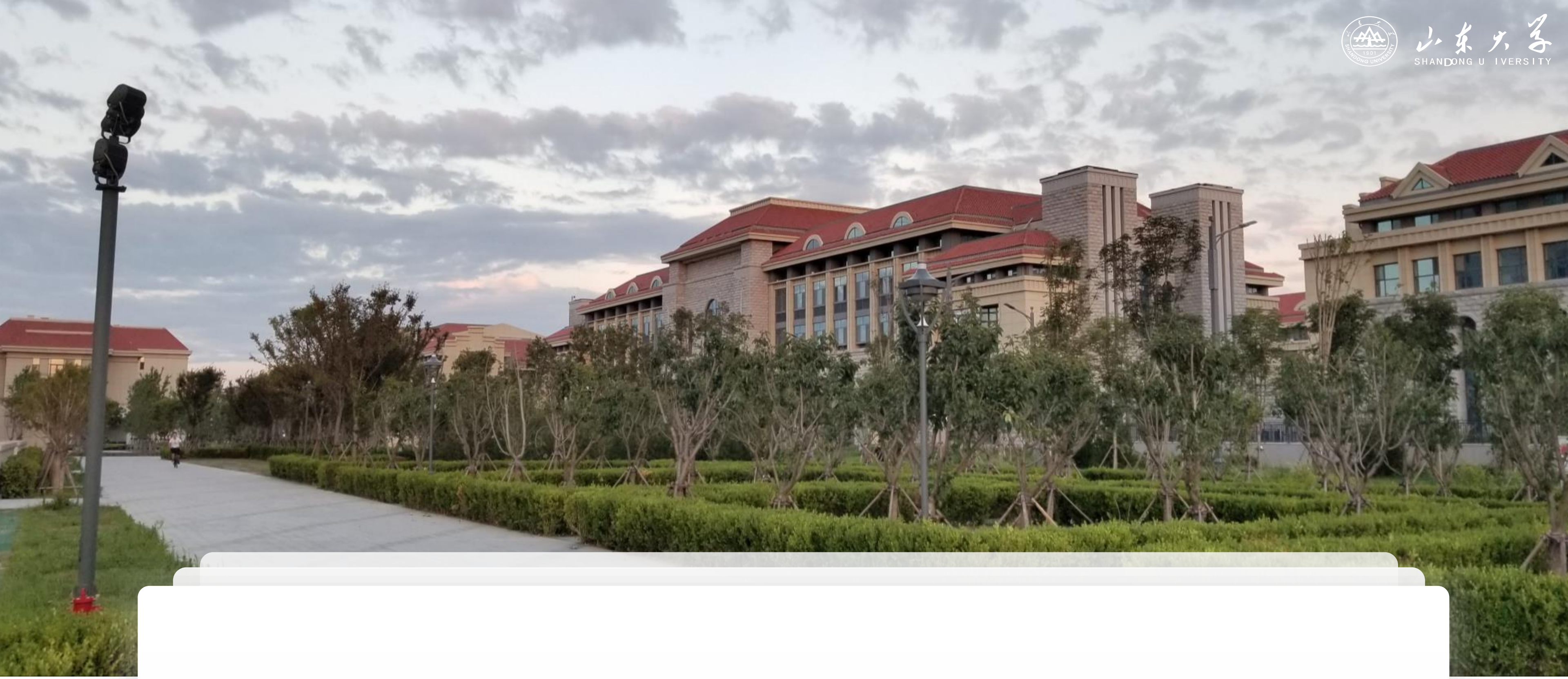

# 程序设计思维与实践

Thinking and Practice in Programming

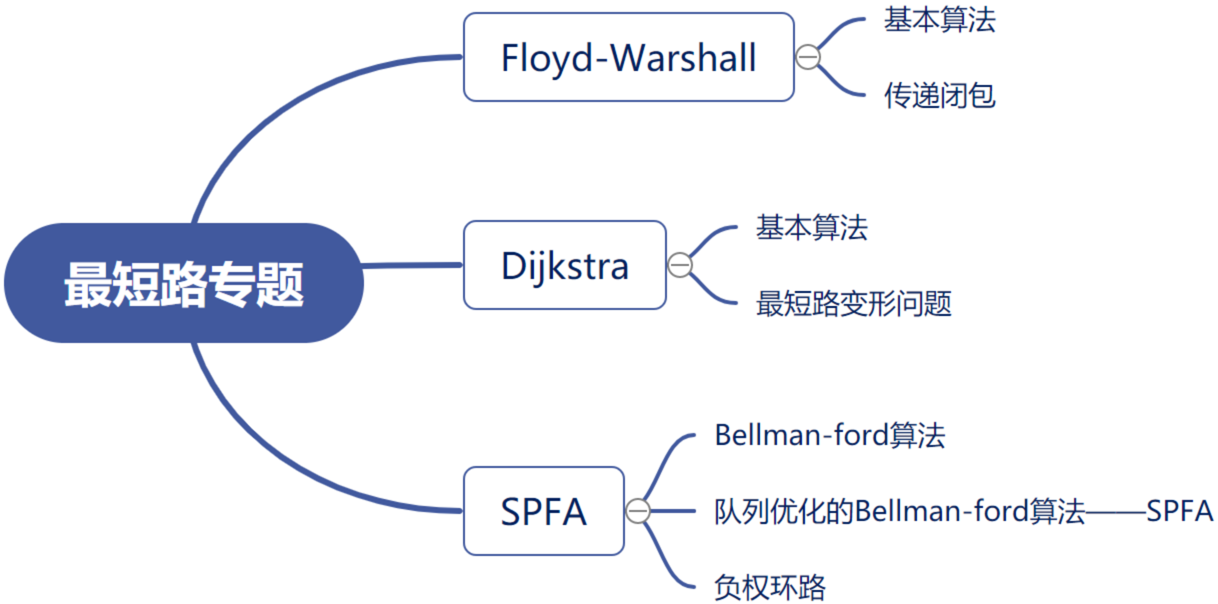
图和树的性质与应用（中） I 内容负责：韩宵玥/李子晗

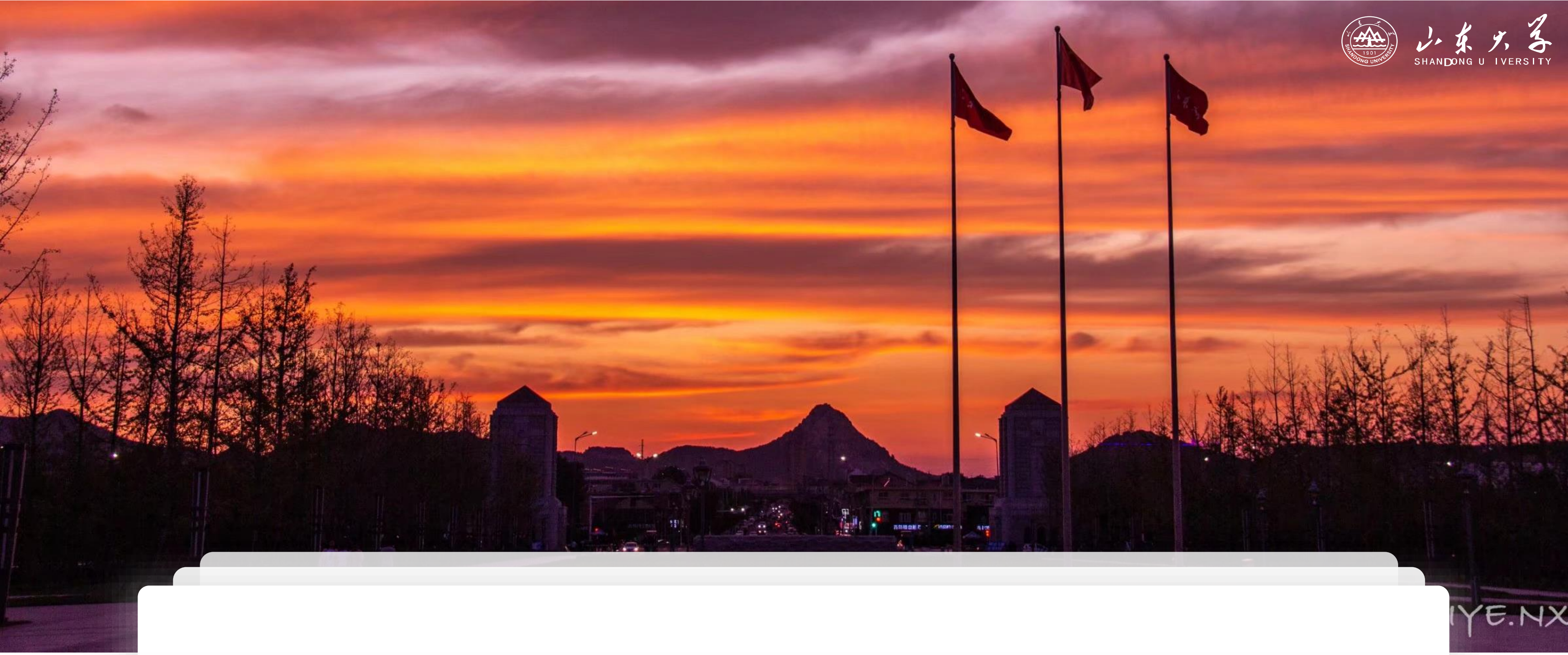


# 1 课 程 脉 络

The outline of course

# 课程脉络


最短路专题
Floyd-Warshall
基本算法
传递闭包
Dijkstra
基本算法
最短路变形问题
SPFA
Bellman-ford算法
队列优化的Bellman-ford算法——SPFA
负权环路



# 2 弗洛伊德算法

Floyd Algorithm

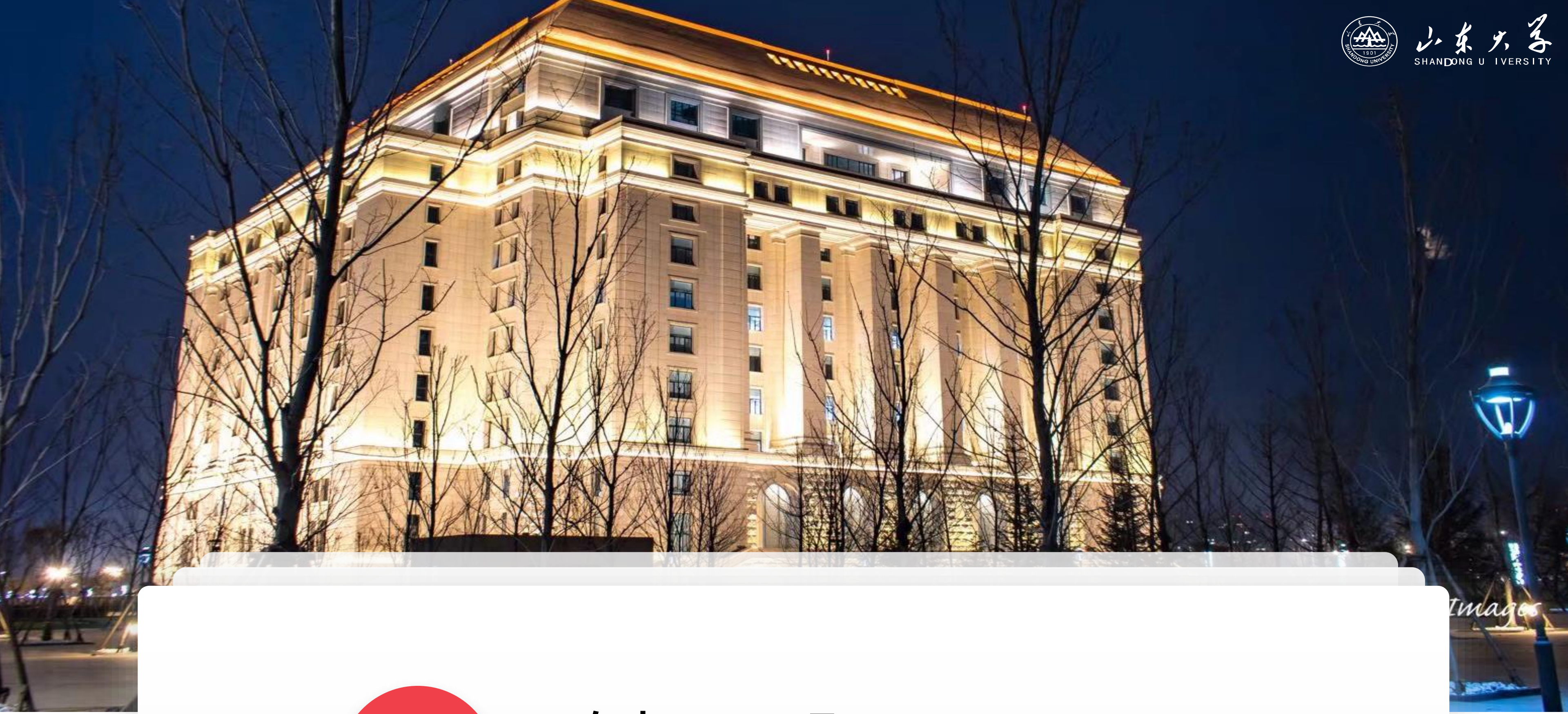
# Floyd–Warshall

- 求取图中任意两点之间的距离
  - $f[k][x][y]$，只允许经过节点 1 到 k，节点 x 到节点 y 的最短路长度
  - $f[n][x][y]$，即节点 x 到节点 y 的最短路长度
  - $f[k][x][y] = \min(f[k-1][x][y], f[k-1][x][k] + f[k-1][k][y])$
- 改进
  - 不难发现我们可以把第一维优化掉
    - 即 $f[x][y] = \min(f[x][y], f[x][k] + f[k][y])$
    - 因为在阶段 $k$ 时， $f[x][k]$ 和 $f[k][y]$ 不会被更新

# Floyd–Warshall

- Floyd–Warshall 算法应用
  - 用于求取图中任意两点之间的关系
  - 多源最短路，任意两点的距离关系
  - 图上的传递闭包，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

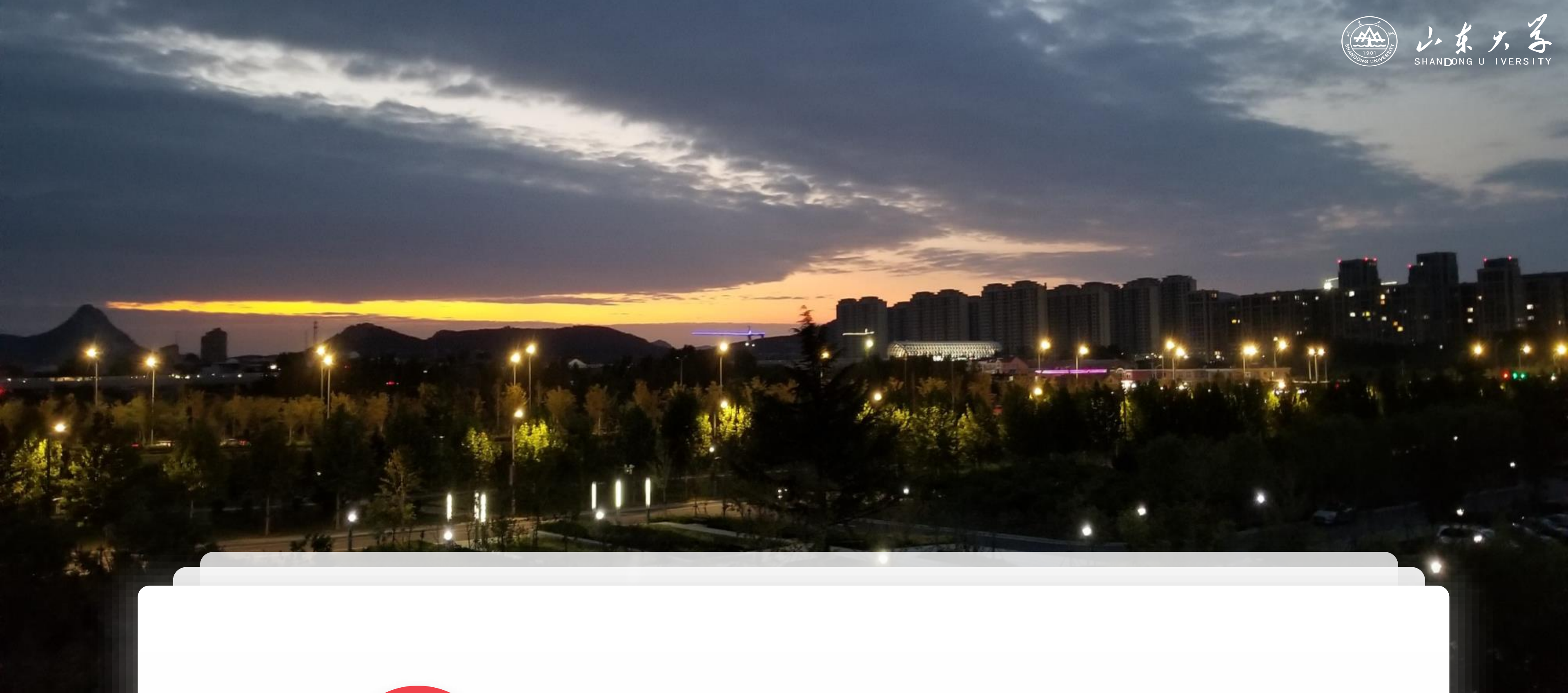
Example 1

# 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
  - 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

- 思路
  - 由 $1 \le N, M \le 500$ 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
    - $dis[a][b] = 0$ 表示 a 与 b 的胜负关系不明
    - $dis[a][b] = 0$ 且 $dis[b][a] = 0$ 即表示 a 与 b 的胜负关系无法预先判断




# 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
  - 复杂度 $O((n+m)logn)$
- 算法实现大体流程
  - 设置 s 为源点，$dis[a]$ 表示源点 s 到点 a 的最短距离，初始化 $dis[s]=0, dis[i]=inf$ (无穷大)，将 s 加入最小堆
  - 每次从堆中取出一个点 x，遍历 x 的所有邻接边 $(x\ y\ w)$，比较 $dis[y]$ 与 $dis[x]+w$ 的大小
  - 这一比较过程称为松弛操作

# Dijkstra

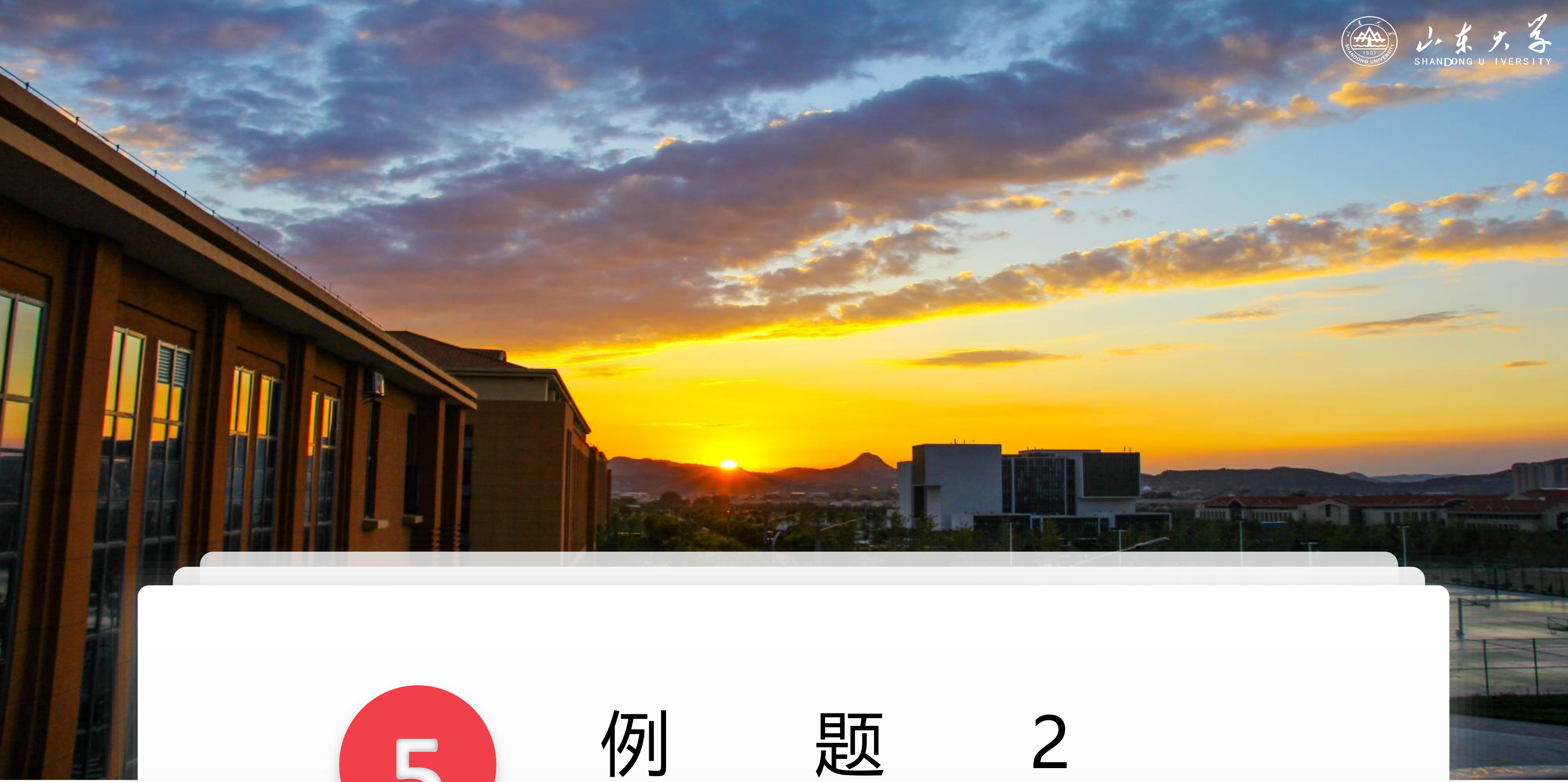
- 比较 $dis[y]$ 与 $dis[x]+w$ 的大小这一过程称为松弛操作
- 若 $dis[y]>dis[x]+w$，则松弛成功
  - 更新 $dis[y]$ 的大小
  - 将 y 加入最小堆中
- 不断执行上述操作，直至最小堆为空，最后得到的 $dis$ 数组即为所求单源最短路值
- 注意
  - 由于图中只有正边，因此每个点只会被最小堆弹出一次
  - 即一旦某个点被最小堆弹出，则不会再被松弛，dis 的值为最短路

# 代码

```
#include<bits/stdc++.h>
#define N 200003
#define pa pair<int,int>
#define inf 1e9
using namespace std;
int dis[N],vis[N],n,m,s;
int tot,point[N],w[N],v[N],nxt[N];
```

```
int main()
{
    scanf("%d%d%d",&n,&m,&s);
    for (int i=1;i<=m;i++) {
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        add(x,y,z);
    }
    dijkstra(s);
    for (int i=1;i<=n;i++) printf("%d ",dis[i]);
    printf("\n");
    return 0;
}
```

```
void add(int x,int y,int z)
{
    tot++; nxt[tot]=point[x]; point[x]=tot; v[tot]=y; w[tot]=z;
}
int dijkstra(int s)
{
    priority_queue<pa,vector<pa>,greater<pa> > q;
    for (int i=1;i<=n;i++) dis[i]=inf,vis[i]=0;
    dis[s]=0;
    q.push(make_pair(0,s));
    while(!q.empty()) {
        int x=q.top().second;
        q.pop();
        if (vis[x]) continue;
        vis[x]=1;
        for (int i=point[x];i;i=nxt[i])
            if(dis[v[i]]>dis[x]+w[i]) {
                dis[v[i]]=dis[x]+w[i];
                q.push(make_pair(dis[v[i]],v[i]));
            }
    }
}
```



# 5 例 题 2

Example 2

# 例题 2 (CSP 201703-4)

**问题描述**

A市有n个交通枢纽，其中1号和n号非常重要，为了加强运输能力，A市决定在1号到n号枢纽间修建一条地铁。

地铁由很多段隧道组成，每段隧道连接两个交通枢纽。经过勘探，有m段隧道作为候选，两个交通枢纽之间最多只有一条候选的隧道，没有隧道两端连接着同一个交通枢纽。

现在有n家隧道施工的公司，每段候选的隧道只能由一个公司施工，每家公司施工需要的天数一致。而每家公司最多只能修建一条候选隧道。所有公司同时开始施工。

作为项目负责人，你获得了候选隧道的信息，现在你可以按自己的想法选择一部分隧道进行施工，请问修建整条地铁最少需要多少天。

**输入格式**

输入的第一行包含两个整数$n$, $m$，用一个空格分隔，分别表示交通枢纽的数量和候选隧道的数量。

第2行到第$m$+1行，每行包含三个整数$a$, $b$, $c$，表示枢纽$a$和枢纽$b$之间可以修建一条隧道，需要的时间为$c$天。

**输出格式**

输出一个整数，修建整条地铁线路最少需要的天数。

**样例输入**

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

**样例输出**

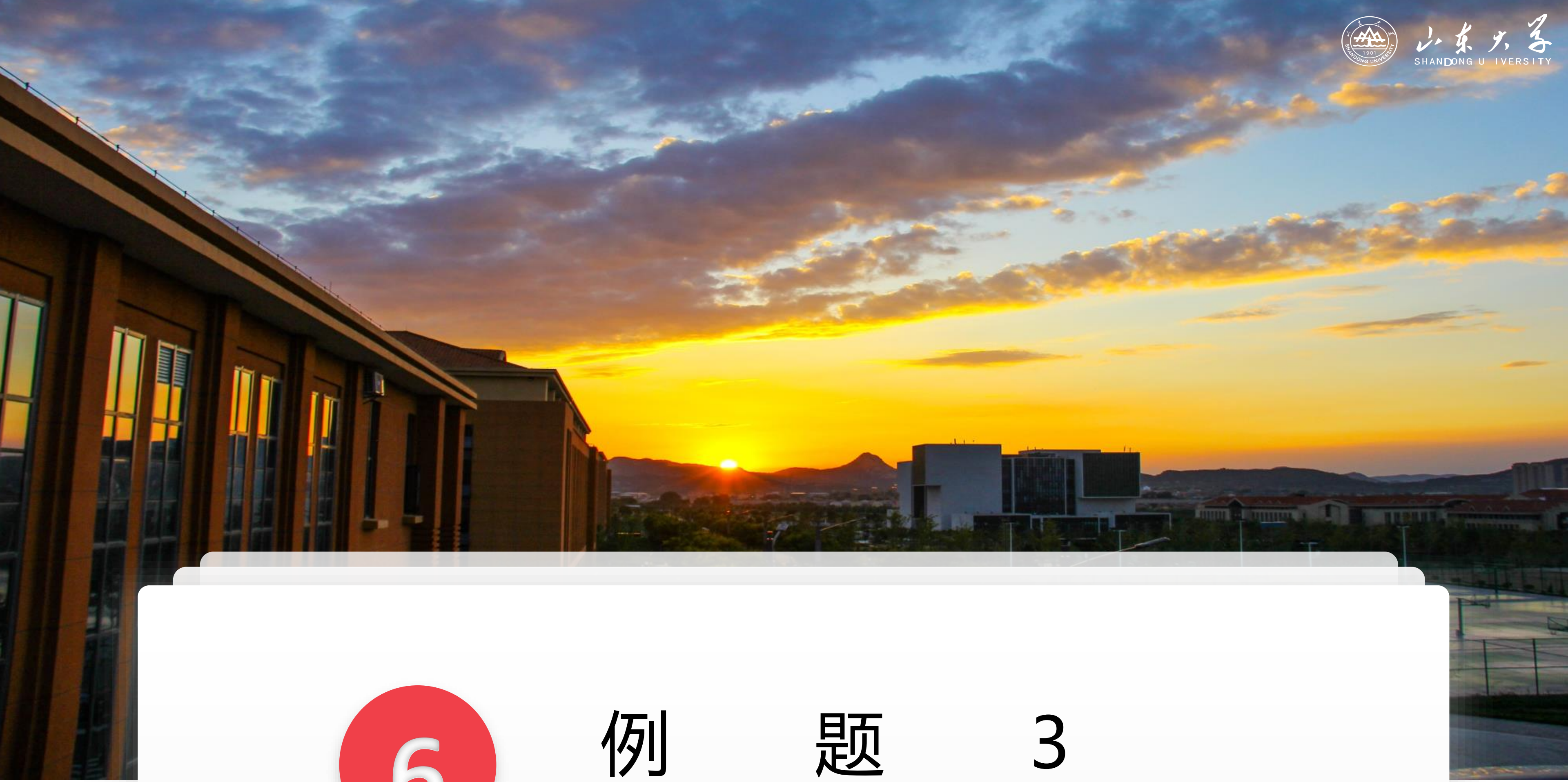
```
6
```

# 例题 2 (CSP 201703-4)

- 题意
  - N 个点，M 条边的有向图，每条边有一个边权
  - 现需要选择若干条边，使得 1 号点与 n 号点连通，选择代价为这若干条边的边权最大值
  - 问选择代价最小为多少?
  - $1 \leq N \leq 10^5$，$1 \leq M \leq 2 * 10^5$

# 例题 2

- 思路
  - 常规最短路问题
    - 询问 1 ~ n 路径总长度最小值
    - $dis[x]$ 表示 1 ~ x 的距离
    - 松弛条件，$dis[y] > dis[x] + w$
  - 最短路变形问题
    - 询问 1 ~ n 经过的边权最大值的最小值
    - $dis[x]$ 表示 1 ~ x 边权最大值的最小值
    - 松弛条件，$dis[y] > \max(dis[x], w)$
  - 在 Dijkstra 模板上修改松弛条件即可




# 6 例题3

Example 3

# 例题 3

- 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
  - $1 \le$ 车站数量 $\le 500$，$1 \le$ 商业线数量 $\le 1000$，$1 \le$ 经济线数量 $\le 1000$

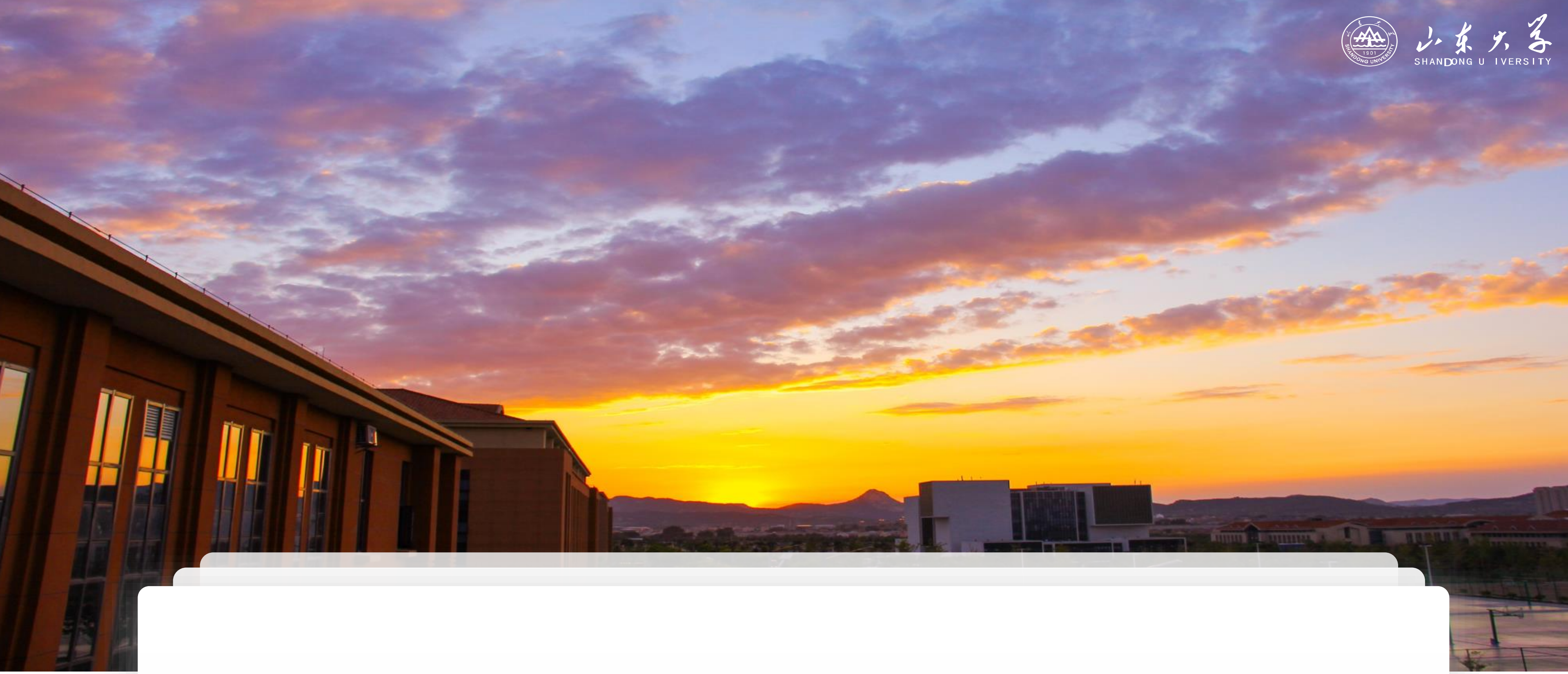
# 例题 3

- 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 3

- 思路 2
  - 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



# 7 SPFA

Shortest Path Faster Algorithm

# SPFA

- 前面讲解的 Floyd 算法用动态规划的思想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
  - 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 算法及其队列优化 (SPFA)

# SPFA

- Bellman-ford 算法可以给出源点 S 至图内其他所有点的最短路以及对应的前驱子图。
- Bellman-ford 算法的正确性基于以下事实
  - 最短路经过的路径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 在 v 的最短路上，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

# SPFA

- 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 条边的最短路均被确定

```
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 $O(nm)$

# 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

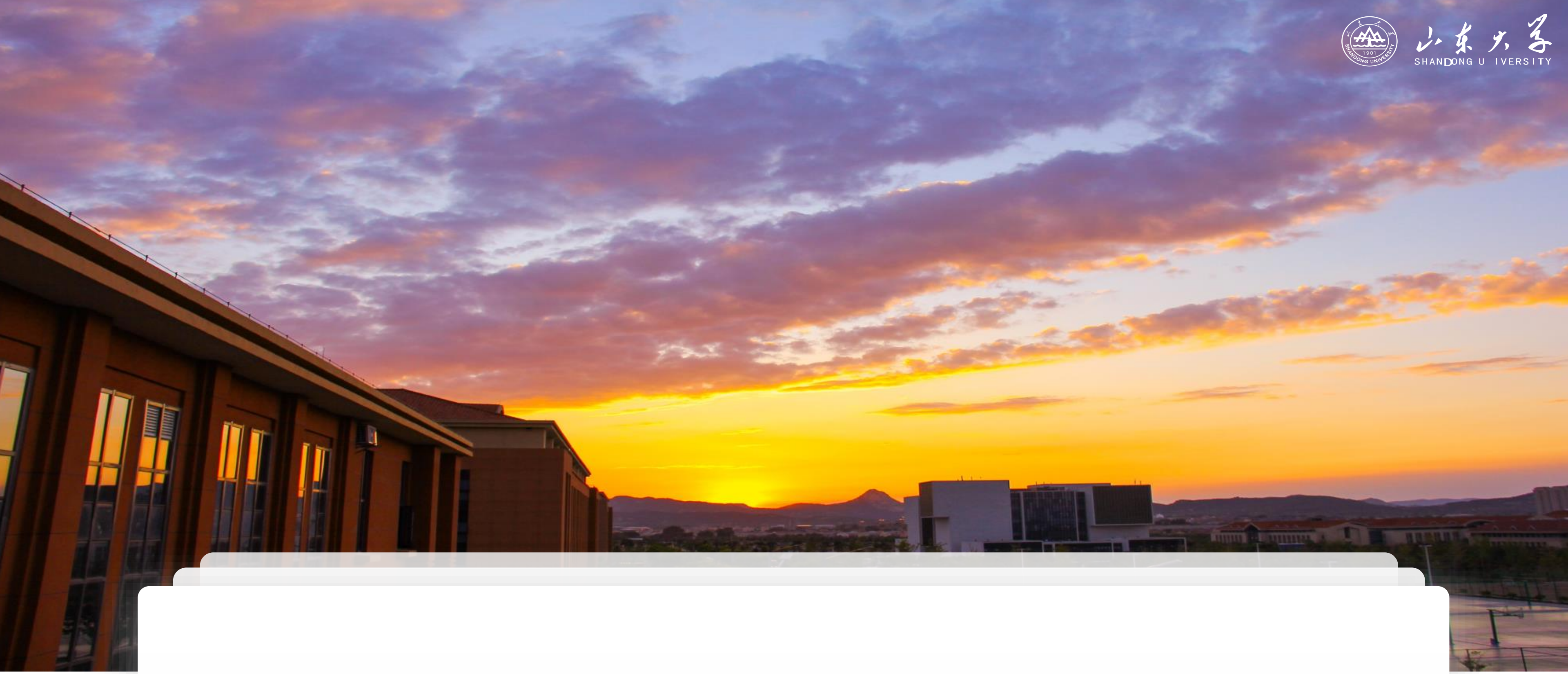
- 思考：
  - 队列如何初始化？ 源点 S 入队
  - 重复入队？ 数组记录是否在队列中

# SPFA

- 算法代码

```
void spfa(int s)
{
    for (int i=1;i<=n;i++)  can[i]=0,dis[i]=inf;
    dis[s]=0; can[s]=1;
    queue<int> p;
    p.push(s);
    while (!p.empty()){
        int now=p.front(); p.pop();
        for (int i=point[now];i;i=nxt[i])
         if (dis[v[i]]>dis[now]+len[i]){
            dis[v[i]]=dis[now]+len[i];
            pre[v[i]]=now;
            if (!can[v[i]]){
                can[v[i]]=1;
                p.push(v[i]);
            }
         }
        can[now]=0;
    }
}
```

- 时间复杂度平均为 $O(km)$
- K是一个小于 n 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

- 当点 u 是 S 不可达时，显然 S 到 u 的最短路不存在，dis[u]=INF
- 当图中存在负环时，那么可以沿着负环不断走下去，那么最短路长度为负无穷，没有意义。

# 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，不容易判断边的松弛次数

  - 方法1：放宽条件，判断点的入队次数，如果某一点入队 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 当前最短路上的边数
- 每次更新最短路时更新cnt[v]，若某一时刻cnt[v]大于等于n的话说明图中存在负环。

```
void spfa(int s)
{
    for (int i=1;i<=n;i++)  can[i]=0,dis[i]=inf,cnt[i]=0;
    dis[s]=0; can[s]=1;
    queue<int> p;
    p.push(s);
    while (!p.empty()){
        int now=p.front(); p.pop();
        for (int i=point[now];i;i=nxt[i])
         if (dis[v[i]]>dis[now]+len[i]){
            dis[v[i]]=dis[now]+len[i];
            cnt[v[i]]=cnt[now]+1;
            if (cnt[v[i]]>=n) {
                //负环
            }
            pre[v[i]]=now;
            if (!can[v[i]]){
                can[v[i]]=1;
                p.push(v[i]);
            }
         }
        can[now]=0;
    }
}
```

# 9 例 题 4

Example 4

# 例题 4

- 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。

- $1 \le n \le 10000$， $1 \le \mathrm{m} \le 50000$，$1 \le T \le 500000$
- $1 \le C \le 10000000$， $1 \le D \le 50000$

# 例题 4

- 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


# 感谢收听

Thank You For Your Listening