Week7,8

1. <https://www.jianshu.com/p/92e46d990d17>最短路径问题：Dijkstra与Floyd算法

2. <http://keyblog.cn/article-21.htmlspfa>

<https://blog.csdn.net/weixin_43902449/article/details/88605417>spfa

3. <https://www.cnblogs.com/zzz-hhh/p/11200893.html>差分约束

4. <https://www.cnblogs.com/bigsai/p/11489260.html>拓扑排序

5. <https://www.cnblogs.com/five20/p/7594239.html>强连通分量

6. <https://blog.csdn.net/u012194956/article/details/79103843>二分法

# Week7

## [A - TT 的魔法猫](https://vjudge.net/problem/HDU-1704) [HDU - 1704](https://vjudge.net/problem/HDU-1704/origin)

众所周知，TT 有一只魔法猫。

这一天，TT 正在专心致志地玩《猫和老鼠》游戏，然而比赛还没开始，聪明的魔法猫便告诉了 TT 比赛的最终结果。TT 非常诧异，不仅诧异于他的小猫咪居然会说话，更诧异于这可爱的小不点为何有如此魔力？

魔法猫告诉 TT，它其实拥有一张游戏胜负表，上面有 N 个人以及 M 个胜负关系，每个胜负关系为 A B，表示 A 能胜过 B，且胜负关系具有传递性。即 A 胜过 B，B 胜过 C，则 A 也能胜过 C。

TT 不相信他的小猫咪什么比赛都能预测，因此他想知道有多少对选手的胜负无法预先得知，你能帮帮他吗？

**Input**

第一行给出数据组数。

每组数据第一行给出 N 和 M（N , M <= 500）。

接下来 M 行，每行给出 A B，表示 A 可以胜过 B。

**Output**

对于每一组数据，判断有多少场比赛的胜负不能预先得知。注意 (a, b) 与 (b, a) 等价，即每一个二元组只被计算一次。

**Sample Input**

3

3 3

1 2

1 3

2 3

3 2

1 2

2 3

4 2

1 2

3 4

**Sample Output**

0

0

4

**思路：**

使用floyd算法，win[a][b]=1表示a强于b，当win[a][b]=0表示胜负不确定·，当win[a][b]=0&& win[b][a]=0表示a和b结果无法预知，如果直接用floyd会超时，所以要进行剪枝，也就是在两人胜负关系确定时进行下一层循环，因为当win[i][k]=0时进入下一层循环没有意义；

for (int k = 0; k < N; k++) {

                for (int i = 0; i < N; i++) {

                    if (win[i][k])

                        for (int j = 0; j < N; j++) {

                            if (win[i][k] && win[k][j]) {

                                win[i][j] = 1;

                            }

                        }

                }

            }

在这之后，遍历二位数组，统计不可预先判断胜负数最后减半就是结果。

**代码：**

#include "cstring"

#include "iostream"

using namespace std;

int win[501][501];

int n, N, M, a, b, ans;

int main() {

    while (cin >> n) {

        for (int z = 0; z < n; z++) {

            memset(win, 0, sizeof(win));

            cin >> N >> M;

            for (int i = 0; i < M; i++) {

                cin >> a >> b;

                win[a - 1][b - 1] = 1;

            }

            ans = 0;

            for (int k = 0; k < N; k++) {

                for (int i = 0; i < N; i++) {

                    if (win[i][k])

                        for (int j = 0; j < N; j++) {

                            if (win[i][k] && win[k][j]) {

                                win[i][j] = 1;

                            }

                        }

                }

            }

            for (int i = 0; i < N; i++) {

                for (int j = 0; j < N; j++) {

                    if ((!win[i][j]) && (!win[j][i]) && i != j) {

                        ans++;

                    }

                    // cout<<win[i][j]<<" ";

                }

                // cout<<endl;

            }

            cout << ans / 2 << endl;

        }

    }

    return 0;

}

## [B - TT 的旅行日记](https://vjudge.net/problem/UVA-11374)

[UVA - 11374](https://vjudge.net/problem/UVA-11374/origin)

众所周知，TT 有一只魔法猫。

今天他在 B 站上开启了一次旅行直播，记录他与魔法猫在喵星旅游时的奇遇。 TT 从家里出发，准备乘坐猫猫快线前往喵星机场。猫猫快线分为经济线和商业线两种，它们的速度与价钱都不同。当然啦，商业线要比经济线贵，TT 平常只能坐经济线，但是今天 TT 的魔法猫变出了一张商业线车票，可以坐一站商业线。假设 TT 换乘的时间忽略不计，请你帮 TT 找到一条去喵星机场最快的线路，不然就要误机了！

**输入**

输入包含多组数据。每组数据第一行为 3 个整数 N, S 和 E (2 ≤ N ≤ 500, 1 ≤ S, E ≤ 100)，即猫猫快线中的车站总数，起点和终点（即喵星机场所在站）编号。

下一行包含一个整数 M (1 ≤ M ≤ 1000)，即经济线的路段条数。

接下来有 M 行，每行 3 个整数 X, Y, Z (1 ≤ X, Y ≤ N, 1 ≤ Z ≤ 100)，表示 TT 可以乘坐经济线在车站 X 和车站 Y 之间往返，其中单程需要 Z 分钟。

下一行为商业线的路段条数 K (1 ≤ K ≤ 1000)。

接下来 K 行是商业线路段的描述，格式同经济线。

所有路段都是双向的，但有可能必须使用商业车票才能到达机场。保证最优解唯一。

**输出**

对于每组数据，输出3行。第一行按访问顺序给出 TT 经过的各个车站（包括起点和终点），第二行是 TT 换乘商业线的车站编号（如果没有使用商业线车票，输出"Ticket Not Used"，不含引号），第三行是 TT 前往喵星机场花费的总时间。

**本题不忽略多余的空格和制表符，且每一组答案间要输出一个换行**

**输入样例**

4 1 4  
4  
1 2 2  
1 3 3  
2 4 4  
3 4 5  
1  
2 4 3

**输出样例**

1 2 4  
2  
5

**思路：**

从起点和终点跑两次dijkstra，分别记录单源最短路，再枚举经过一次商业线的情况（需要分别比较正反两个方向的距离大小。），与纯经济线进行比较。

两个dis1，dis2数组记录下起点和终点到每一个点的单源最短路，pre1和pre2记录前驱节点。使用优先级队列优化dijstra ，先将起点push进小根堆并将dis设为0，pre设为-1，每次取堆顶元素并进行标记，每个点只松弛一次，若未标记过将该点进行标记并遍历邻接点，判断dis[y] >dis[x] + w，若满足则更新dis[y]，表示松弛成功，同时将该点存入堆。

堆使用pair结构分别存距离和点，因为优先级队列默认为大根堆，所以取负变为小根堆。输出时需要注意pre记录的是从起点开始每个点的前驱节点，所以从起点到商业线起点之间的pre值需要倒序输出。

#include <cstring>

#include <algorithm>

#include <iostream>

#include <queue>

using namespace std;

const int inf = 0x3f3f3f3f;

int n, s, e, m, k;

priority\_queue<pair<int, int>> q;

bool vis[10001];

int path[10001];

int dis1[10001], dis2[10001];

int fre1[10001], fre2[10001];

struct edge {

    int to, next, w;

} edge[10001];

int head[10001], tot;

void dij(int s, int \*dis, int \*fre) {

    while (q.size()) q.pop();

    memset(vis, false, sizeof(vis));

    for (int i = 1; i <= n; i++) {

        dis[i] = inf;

    }

    fre[s] = -1;

    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 = head[x]; i != -1; i = edge[i].next) {

            int y = edge[i].to, w = edge[i].w;

            if (dis[y] > dis[x] + w) {

                dis[y] = dis[x] + w;

                fre[y] = x;

                q.push(make\_pair(-dis[y], y));

            }

        }

    }

}

int main() {

    int ct = 0;

    ios::sync\_with\_stdio(false);

    while (cin >> n >> s >> e) {

        if (ct) cout << endl;

        cin >> m;

        tot = 0;

        int a, b, c;

        // init

        tot = 0;

        memset(head, -1, sizeof(head));

        while (m--) {

            cin >> a >> b >> c;

            edge[tot].to = b;

            edge[tot].next = head[a];

            edge[tot].w = c;

            head[a] = tot;

            tot++;

            edge[tot].to = a;

            edge[tot].next = head[b];

            edge[tot].w = c;

            head[b] = tot;

            tot++;

        }

        dij(s, dis1, fre1);

        dij(e, dis2, fre2);

        int u, v, w;

        cin >> k;

        int ans = dis1[e], minu, minv, flag = 0;

        for (int i = 0; i < k; i++) {

            cin >> u >> v >> w;

            if (dis1[u] + dis2[v] + w < ans) {

                ans = dis1[u] + dis2[v] + w;

                minu = u;

                minv = v;

                flag = 1;

            }

            if (dis1[v] + dis2[u] + w < ans) {

                ans = dis1[v] + dis2[u] + w;

                minu = v;

                minv = u;

                flag = 1;

            }

        }

        if (flag == 1) {

            int cnt = 0;

            int temp = minu;

            path[cnt++] = temp;

            while (fre1[temp] != -1) {

                path[cnt++] = fre1[temp];

                temp = fre1[temp];

            }

            for (int i = cnt - 1; i >= 0; i--) cout << path[i] << " ";

            cnt = 0;

            temp = minv;

            path[cnt++] = temp;

            while (fre2[temp] != -1) {

                path[cnt++] = fre2[temp];

                temp = fre2[temp];

            }

            for (int i = 0; i < cnt - 1; i++) cout << path[i] << " ";

            cout << path[cnt - 1] << endl;

            cout << minu << endl;

        } else {

            int cnt = 0;

            int temp = e;

            path[cnt++] = temp;

            while (fre1[temp] != -1) {

                path[cnt++] = fre1[temp];

                temp = fre1[temp];

            }

            for (int i = cnt - 1; i > 0; i--) cout << path[i] << " ";

            cout << path[0] << endl;

            cout << "Ticket Not Used" << endl;

        }

        cout << ans << endl;

        ct++;

    }

    return 0;

}

## [C - TT 的美梦](https://vjudge.net/problem/LightOJ-1074)

[LightOJ - 1074](https://vjudge.net/problem/LightOJ-1074/origin)

这一晚，TT 做了个美梦！

在梦中，TT 的愿望成真了，他成为了喵星的统领！喵星上有 N 个商业城市，编号 1 ～ N，其中 1 号城市是 TT 所在的城市，即首都。

喵星上共有 M 条有向道路供商业城市相互往来。但是随着喵星商业的日渐繁荣，有些道路变得非常拥挤。正在 TT 为之苦恼之时，他的魔法小猫咪提出了一个解决方案！TT 欣然接受并针对该方案颁布了一项新的政策。

具体政策如下：对每一个商业城市标记一个正整数，表示其繁荣程度，当每一只喵沿道路从一个商业城市走到另一个商业城市时，TT 都会收取它们（目的地繁荣程度 - 出发地繁荣程度）^ 3 的税。

TT 打算测试一下这项政策是否合理，因此他想知道从首都出发，走到其他城市至少要交多少的税，如果总金额小于 3 或者无法到达请悄咪咪地打出 '?'。

**Input**

第一行输入 T，表明共有 T 组数据。（1 <= T <= 50）

对于每一组数据，第一行输入 N，表示点的个数。（1 <= N <= 200）

第二行输入 N 个整数，表示 1 ～ N 点的权值 a[i]。（0 <= a[i] <= 20）

第三行输入 M，表示有向道路的条数。（0 <= M <= 100000）

接下来 M 行，每行有两个整数 A B，表示存在一条 A 到 B 的有向道路。

接下来给出一个整数 Q，表示询问个数。（0 <= Q <= 100000）

每一次询问给出一个 P，表示求 1 号点到 P 号点的最少税费。

**Output**

每个询问输出一行，如果不可达或税费小于 3 则输出 '?'。

**Sample Input**

2

5

6 7 8 9 10

6

1 2

2 3

3 4

1 5

5 4

4 5

2

4

5

10

1 2 4 4 5 6 7 8 9 10

10

1 2

2 3

3 1

1 4

4 5

5 6

6 7

7 8

8 9

9 10

2

3 10

**Sample Output**

Case 1:

3

4

Case 2:

?

?

**思路：**

要求求出含有环的单源最短路，使用Bellman−Ford算法经队列优化的spfa算法，还是链式结构。

先将起点到所有点的距离初始化为inf并将起点入队，当队列不为空时，每次取队首元素，将其出队并将visited值修改为0，然后开始对邻接点进行松弛操作，其邻接点如果不在队列中将它加入队列。

由于负环可能存在，而不存在环时只可能点数大于边数。用num来存每个点的松弛次数，当num[i]>n时则说明存在负环。因为与负环连通的所有点都会受负环影响，故可以从i点开始dfs标记可到达的点。

dis记录距离，visited数组标记点是否在队列中， flag记录是否受负环影响

**代码：**

#include <cmath>

#include <cstring>

#include <iostream>

#include <queue>

using namespace std;

struct node {

    int to;

    int next;

    int weight;

};

int inf = 999999;

int n, m, t, total;

int num[100001], a[100001], dis[100001], head[100001];

queue<int> q;

bool visited[100001], flag[100001];

node edge[100001];

void dfs(int x) {

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

        if (!flag[edge[i].to]) {

            flag[edge[i].to] = 1;

            dfs(edge[i].to);

        }

}

int main() {

    int N, x, y, z;

    cin >> N;

    for (int nn = 1; nn <= N; nn++) {

        total = 1;

        for (int i = 0; i < 100001; i++) {

            head[i] = 0;

            num[i] = 0;

            visited[i] = 0;

            flag[i] = 0;

        }

        cin >> n;

        for (int i = 1; i <= n; i++) cin >> a[i];

        cin >> m;

        for (int i = 1; i <= m; i++) {

            cin >> x >> y;

            edge[total].next = head[x];

            edge[total].to = y;

            edge[total].weight = pow(a[y] - a[x], 3);

            head[x] = total++;

        }

        //spfa

        for (int i = 1; i <= n; i++) {

            dis[i] = inf;

        }

        dis[1] = 0;

        q.push(1);

        visited[1] = 1;

        while (!q.empty()) {

            int u = q.front();

            q.pop();

            visited[u] = 0;

            for (int i = head[u]; i; i = edge[i].next)

                if (dis[edge[i].to] > dis[u] + edge[i].weight) {

                    num[edge[i].to]++;

                    dis[edge[i].to] = dis[u] + edge[i].weight;

                    if (num[edge[i].to] > n) {

                        dfs(edge[i].to);

                        continue;

                    }

                    if (!visited[edge[i].to]) {

                        q.push(edge[i].to);

                        visited[edge[i].to] = 1;

                    }

                }

        }

        cout<<"Case "<<nn<<":"<<endl;

        cin >> t;

        for (int i = 1; i <= t; i++) {

            cin >> n;

            if (dis[n] < 3 || flag[n] || dis[n] == inf)

                cout << "?" << endl;

            else

                cout << dis[n] << endl;

        }

    }

    //system("pause");

    return 0;

}

# CCF-201604-3

**问题描述**

|  |  |
| --- | --- |
| 试题编号： | 201604-3 |
| 试题名称： | 路径解析 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　在操作系统中，数据通常以文件的形式存储在文件系统中。文件系统一般采用层次化的组织形式，由目录（或者文件夹）和文件构成，形成一棵树的形状。文件有内容，用于存储数据。目录是容器，可包含文件或其他目录。同一个目录下的所有文件和目录的名字各不相同，不同目录下可以有名字相同的文件或目录。 　　为了指定文件系统中的某个文件，需要用**路径**来定位。在类 Unix 系统（Linux、Max OS X、FreeBSD等）中，路径由若干**部分**构成，每个部分是一个目录或者文件的名字，相邻两个部分之间用 / 符号分隔。 　　有一个特殊的目录被称为**根目录**，是整个文件系统形成的这棵树的根节点，用一个单独的 / 符号表示。在操作系统中，有**当前目录**的概念，表示用户目前正在工作的目录。根据出发点可以把路径分为两类： 　　 **绝对路径**：以 / 符号开头，表示从根目录开始构建的路径。 　　 **相对路径**：不以 / 符号开头，表示从当前目录开始构建的路径。  　　例如，有一个文件系统的结构如下图所示。在这个文件系统中，有根目录 / 和其他普通目录 d1、d2、d3、d4，以及文件 f1、f2、f3、f1、f4。其中，两个 f1 是同名文件，但在不同的目录下。  　　对于 d4 目录下的 f1 文件，可以用绝对路径 /d2/d4/f1 来指定。如果当前目录是 /d2/d3，这个文件也可以用相对路径 ../d4/f1 来指定，这里 .. 表示上一级目录（注意，根目录的上一级目录是它本身）。还有 . 表示本目录，例如 /d1/./f1 指定的就是 /d1/f1。注意，如果有多个连续的 / 出现，其效果等同于一个 /，例如 /d1///f1 指定的也是 /d1/f1。 　　本题会给出一些路径，要求对于每个路径，给出**正规化**以后的形式。一个路径经过正规化操作后，其指定的文件不变，但是会变成一个不包含 . 和 .. 的绝对路径，且不包含连续多个 / 符号。如果一个路径以 / 结尾，那么它代表的一定是一个目录，正规化操作要去掉结尾的 /。若这个路径代表根目录，则正规化操作的结果是 /。若路径为空字符串，则正规化操作的结果是当前目录。  **输入格式**  　　第一行包含一个整数 *P*，表示需要进行正规化操作的路径个数。 　　第二行包含一个字符串，表示当前目录。 　　以下 *P* 行，每行包含一个字符串，表示需要进行正规化操作的路径。  **输出格式**  　　共 *P* 行，每行一个字符串，表示经过正规化操作后的路径，顺序与输入对应。  **样例输入**  7 /d2/d3 /d2/d4/f1 ../d4/f1 /d1/./f1 /d1///f1 /d1/ /// /d1/../../d2  **样例输出**  /d2/d4/f1 /d2/d4/f1 /d1/f1 /d1/f1 /d1 / /d2  **评测用例规模与约定**  　　1 ≤ *P* ≤ 10。 　　文件和目录的名字只包含大小写字母、数字和小数点 .、减号 - 以及下划线 \_。 　　不会有文件或目录的名字是 . 或 .. ，它们具有题目描述中给出的特殊含义。 　　输入的所有路径每个长度不超过 1000 个字符。 　　输入的当前目录保证是一个经过正规化操作后的路径。 　　对于前 30% 的测试用例，需要正规化的路径的组成部分不包含 . 和 .. 。 　　对于前 60% 的测试用例，需要正规化的路径都是绝对路径。 |

**思路：**

如果是相对路径，要先转化为绝对路径（当前绝对路径+读入路径），删去多余的“//”，这个时候就剩下“/./”和“/…/”,因为要处理“/…/”前面的目录，所以就需要先处理完“/./”再处理“/…/”以及前面的目录的情况。用string类函数find和erase找到对应字符进行相关操作。

注意路径为空字符串的情况。

最后输出处理过的字符串，

**代码：**

#include <iostream>

#include <cstring>

using namespace std;

string cur, path;

int main() {

    int p;

    cin >> p;

    getchar();

    getline(cin, cur);

    for (int i = 0; i < p; i++) {

        getline(cin, path);

        if (path[0] != '/') path = cur + "/" + path;  //转绝对路径

        int pos;

        int len;

        while ((pos = path.find("//")) != -1)  //处理"//"的情况

        {

            int num = 0;

            while (path[pos + 1 + num] == '/') num++;

            path.erase(pos + 1, num);

        }

        while ((pos = path.find("/./")) != -1)  // 处理 "./"的情况

        {

            path.erase(pos + 1, 2);

        }

        while ((pos = path.find("/../")) != -1)  //处理 /../ 的情况

        {

            if (pos == 0)

                path.erase(pos + 1, 3);  //起始位置

            else {

                int p = path.rfind("/", pos - 1);

                path.erase(p + 1, pos + 3 - p);

            }

        }

        len = path.length();  //更新长度

        //最后一个

        if (len != 1 && path[len - 1] == '/') path.erase(len - 1);

        cout << path << endl;

    }

}

# Week8

## [A - 区间选点 II](https://vjudge.net/problem/POJ-1201)

[POJ - 1201](https://vjudge.net/problem/POJ-1201/origin)

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 [ai, bi] 里至少有 ci 个点

使用差分约束系统的解法解决这道题  
使用差分约束系统的解法解决这道题  
使用差分约束系统的解法解决这道题  
使用差分约束系统的解法解决这道题  
使用差分约束系统的解法解决这道题

**Input**

输入第一行一个整数 n 表示区间的个数，接下来的 n 行，每一行两个用空格隔开的整数 a，b 表示区间的左右端点。1 <= n <= 50000， 0 <= ai <= bi <= 50000 并且 1 <= ci <= bi - ai+1。

**Output**

输出一个整数表示最少选取的点的个数

**Sample Input**

5

3 7 3

8 10 3

6 8 1

1 3 1

10 11 1

**Sample Output**

6

**思路：**

差分约束解决区间选点

sum[i]为区间[0，i]之间的点数，则应有：sum[bi]−sum[ai−1]≥ci；对每个点又有： sum[i]−sum[i−1]为0或1；转换为可用差分约束的不等式，也就是sum[max(bi)]，最后用spfa跑最长路。

**代码：**

#include <cstring>

#include <iostream>

#include <queue>

using namespace std;

#define inf -1e9

struct edge {

    int v, w, next;

} edge[200010];

int head[50010], tot;

int n, a, b, c, minLen, maxLen, dis[50010];

bool flag[50010];

queue<int> q;

void add(int x, int y, int w) {

    edge[tot].v = y;

    edge[tot].w = w;

    edge[tot].next = head[x];

    head[x] = tot;

    tot++;

}

void spfa() {

    while (!q.empty()) q.pop();

    for (int i = minLen; i <= maxLen; i++) {

        dis[i] = inf;

        flag[i] = 0;

    }

    dis[minLen - 1] = 0;

    flag[minLen - 1] = 1;

    q.push(minLen - 1);

    while (!q.empty()) {

        int x = q.front();

        q.pop();

        flag[x] = 0;

        for (int i = head[x]; i != -1; i = edge[i].next) {

            int y = edge[i].v;

            int w = edge[i].w;

            if (dis[y] < dis[x] + w) {

                dis[y] = dis[x] + w;

                if (!flag[y]) {

                    q.push(y);

                    flag[y] = 1;

                }

            }

        }

    }

}

int main() {

    cin.sync\_with\_stdio(false);

    while (cin >> n) {

        minLen = n;

        tot = 0;

        memset(head, -1, sizeof(head));

        for (int i = 0; i < n; i++) {

            cin >> a >> b >> c;

            add(a, b + 1, c);

            minLen = min(minLen, min(a, b));

            maxLen = max(maxLen, max(a, b));

        }

        minLen++;

        maxLen++;

        for (int i = minLen; i <= maxLen; i++) {

            add(i - 1, i, 0);

            add(i, i - 1, -1);

        }

        spfa();

        cout << dis[maxLen] << endl;

    }

    // system("pause");

    return 0;

}

## [B - 猫猫向前冲](https://vjudge.net/problem/HDU-1285)

[HDU - 1285](https://vjudge.net/problem/HDU-1285/origin)

众所周知， TT 是一位重度爱猫人士，他有一只神奇的魔法猫。  
有一天，TT 在 B 站上观看猫猫的比赛。一共有 N 只猫猫，编号依次为1，2，3，…，N进行比赛。比赛结束后，Up 主会为所有的猫猫从前到后依次排名并发放爱吃的小鱼干。不幸的是，此时 TT 的电子设备遭到了宇宙射线的降智打击，一下子都连不上网了，自然也看不到最后的颁奖典礼。  
不幸中的万幸，TT 的魔法猫将每场比赛的结果都记录了下来，现在他想编程序确定字典序最小的名次序列，请你帮帮他。

**Input**

输入有若干组，每组中的第一行为二个数N（1<=N<=500），M；其中N表示猫猫的个数，M表示接着有M行的输入数据。接下来的M行数据中，每行也有两个整数P1，P2表示即编号为 P1 的猫猫赢了编号为 P2 的猫猫。

**Output**

给出一个符合要求的排名。输出时猫猫的编号之间有空格，最后一名后面没有空格!  
  
其他说明：符合条件的排名可能不是唯一的，此时要求输出时编号小的队伍在前；输入数据保证是正确的，即输入数据确保一定能有一个符合要求的排名。

**Sample Input**

4 3

1 2

2 3

4 3

**Sample Output**

1 2 4 3

**思路：**

求字典序最小的拓扑序列，

首先求出每个点的入度，每次从第一个点开始遍历，找到第一个入度为0的点取出，并将它到达的各点的入度减一，重复遍历直到最后。

**代码：**

#include <iostream>

using namespace std;

int a[501][501], inDegree[501];

int n, m, p1, p2;

void topsort() {

    for (int i = 1; i <= n; i++) {

        for (int j = 1; j <= n; j++) {

            if (inDegree[j] == 0) {

                inDegree[j]--;

                cout << j;

                if (i != n)

                    cout << " ";

                else

                    cout << endl;

                for (int k = 1; k <= n; k++) {

                    if (a[j][k] == 1) inDegree[k]--;

                }

                break;

            }

        }

    }

}

int main() {

    while (cin >> n >> m) {

        for (int i = 0; i < n + 1; i++) {

            inDegree[i] = 0;

            for (int j = 0; j < n + 1; j++) {

                a[i][j] = 0;

            }

        }

        for (int i = 0; i < m; i++) {

            cin >> p1 >> p2;

            if (a[p1][p2] == 0) {

                a[p1][p2] = 1;

                inDegree[p2]++;

            }

        }

        topsort();

    }

    return 0;

}

## [C - 班长竞选](https://vjudge.net/problem/HDU-3639)

[HDU - 3639](https://vjudge.net/problem/HDU-3639/origin)

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 勤劳的 TT 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？

**Input**

本题有多组数据。第一行 T 表示数据组数。每组数据开始有两个整数 N 和 M (2 <= n <= 5000, 0 <m <= 30000)，接下来有 M 行包含两个整数 A 和 B(A != B) 表示 A 认为 B 合适。

**Output**

对于每组数据，第一行输出 “Case x: ”，x 表示数据的编号，从1开始，紧跟着是最高的票数。 接下来一行输出得票最多的同学的编号，用空格隔开，不忽略行末空格！

**Sample Input**

2

4 3

3 2

2 0

2 1

3 3

1 0

2 1

0 2

**Sample Output**

Case 1: 2

0 1

Case 2: 2

0 1 2

**思路：**

首先dfs求出每个点所在的连通分量，

然后缩点并记录缩点之后每个点的入度（要将图转为反图），

接下来dfs遍历得到每个点所在连通分量的点数，然后-1即为得票数

结果需要排序输出（cmp数组辅助）。

**代码：**

#include <cstring>

#include <iostream>

#include <stack>

#include <vector>

using namespace std;

int t, n, m;

const int inf = 0x3f3f3f3f;

int inDegree[5010];

int support[5010];

int vis[5010],  low[5010], cmp[5010], pre[5010], sz[5010], dfs\_ct, k;

vector<int> G[5010], rG[5010];

stack<int> s;

void dfs(int u) {

    pre[u] = low[u] = ++dfs\_ct;

    s.push(u);

    for (int i = 0; i < G[u].size(); i++) {

        int v = G[u][i];

        if (!pre[v]) {

            dfs(v);

            low[u] = min(low[u], low[v]);

        } else if (!cmp[v])

            low[u] = min(low[u], low[v]);

    }

    if (pre[u] == low[u]) {

        k++;

        while (1) {

            int x = s.top();

            s.pop();

            cmp[x] = k;  //每个点所在连通分量编号

            sz[k]++;

            if (x == u) break;

        }

    }

}

int dfs1(int u) {

    int ans = sz[u];

    vis[u] = true;

    for (int i = 0; i < rG[u].size(); i++) {

        int v = rG[u][i];

        if (vis[v]) continue;

        ans += dfs1(v);

    }

    return ans;

}

int main() {

    cin.sync\_with\_stdio(false);

    cin >> t;

    for (int ca = 1; ca <= t; ca++) {

        cin >> n >> m;

        for (int i = 0; i <= n; i++) {

            G[i].clear();

            rG[i].clear();

        }

        int x, y;

        for (int i = 0; i < m; i++) {

            cin >> x >> y;

            G[x].push\_back(y);

        }

        dfs\_ct = k = 0;

        memset(pre, 0, sizeof(pre));

        memset(cmp, 0, sizeof(cmp));

        memset(sz, 0, sizeof(sz));

        for (int i = 0; i < n; i++) {

            if (pre[i] == 0) dfs(i);

        }

        memset(inDegree, 0, sizeof(inDegree));

        // suodian

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < G[i].size(); j++) {

                int u = cmp[i], v = cmp[G[i][j]];

                if (u == v) continue;

                inDegree[u]++;

                rG[v].push\_back(u);

            }

        }

        int mx = 0;

        for (int i = 1; i <= k; i++) {

            if (inDegree[i] == 0) {

                memset(vis, 0, sizeof(vis));

                support[i] = dfs1(i);

                mx = max(mx, support[i]);

            }

        }

        cout << "Case " << ca << ": " << mx - 1 << endl;

        bool flag = false;

        for (int i = 0; i < n; i++) {

            int u = cmp[i];

            if (inDegree[u] == 0 && support[u] == mx) {

                if (flag) cout << " ";

                cout << i;

                flag = true;

            }

        }

        cout << endl;

    }

    return 0;

}

# CSP-M2



**思路：**

对于一个序列，满足题目要求时有两种情况：

1.整个序列只有三个数，而且中间数的两倍是另外两个数之和（开始写代码的时候想的好好的，最后忘了判断这个）；

2.序列含有小于三种不同数；

从头遍历序列判断，不满足上面两种情况直接输出NO，满足输出YES

**代码：**

#include "algorithm"

#include "cmath"

#include "iostream"

using namespace std;

long long t, n;

long long a[10001];

int main() {

    while (cin >> t) {

        for (long long ii = 0; ii < t; ii++) {

            long long cha[10];

            bool c1, c2, c3;

            c1 = false;

            c2 = false;

            c3 = false;

            cin >> n;

            for (long long i = 0; i < n; i++) {

                cin >> a[i];

            }

            cha[0] = a[0];

            bool NO = false;

            c1 = true;

            for (int i = 1; i < n; i++) {

                if (c1 && !c2 && !c3) {

                    if (a[i] != cha[0]) {

                        cha[1] = a[i];

                        c2 = true;

                    }

                } else if (c1 && c2 && !c3) {

                    if (a[i] != cha[0] && a[i] != cha[1]) {

                        cha[2] = a[i];

                        c3 = true;

                        sort(cha, cha + 3);

                        long long flag = cha[0] + cha[2] - cha[1] \* 2;

                        if (flag != 0) {

                            cout << "NO" << endl;

                            NO = true;

                            break;

                        }

                    }

                } else if (c1 && c2 && c3) {

                    if (a[i] != cha[0] && a[i] != cha[1] && a[i] != cha[2]) {

                        cout << "NO" << endl;

                        NO = true;

                        break;

                    }

                }

            }

            if (!NO) {

                cout << "YES" << endl;

            }

        }

    }

    return 0;

}





**思路：**

遇到字母题首先就想到ascii码了

从字符串首开始向后遍历，再遍历26个连续字符之前，遇到“?”跳过并计数，遇到已存在的则更新起始位置（起始位置后移到该字母上次出现位置的下一个位置，并将弹出字母的状态从true设为false），继续遍历直到存入26个连续字符，然后遍历visited数组，找到未标记的位置，利用ASCII码转字符的办法确定缺失的字母，最后将字母升序填充到空白位置，输出最后的序列。

如果遍历完序列没有达到26则无法找到要求序列；

**代码：**

#include <iostream>

#include <string>

using namespace std;

string str;

int main() {

    cin >> str;

    bool visited[26] = {false};

    long long location[26];

    long long \_left = 0, \_right = -1;

    long long len = str.size();

    for (long long i = 0; i < len; i++) {

        if (visited[str[i] - 'A'] == true) {

            for (long long j = \_left; j < location[str[i] - 'A']; j++) {

                visited[str[j] - 'A'] = false;

            }

            \_left = location[str[i] - 'A'] + 1;

            location[str[i] - 'A'] = i;

        }

        if (str[i] == '?') {

            if ((i - \_left + 1) == 26) {

                \_right = i;

                break;

            } else

                continue;

        } else {

            visited[str[i] - 'A'] = true;

            location[str[i] - 'A'] = i;

        }

        if ((i - \_left + 1) == 26) {

            \_right = i;

            break;

        }

    }

    if (\_right == -1)

        cout << -1;

    else {

        long long now = 0;

        for (long long i = \_left; i <= \_right; i++) {

            if (str[i] == '?')

                for (long long j = now; j < 26; j++) {

                    if (visited[j] == false) {

                        cout << (char)(j + 'A');

                        now = j + 1;

                        break;

                    }

                }

            else

                cout << str[i];

        }

    }

    str.clear();

}



**思路：**

数字实际上也是字符串。

算是一道关于数列的数学题

1

12

123

…

123456789

-----------------

12345678910

1234567891011

123456789101112

…

123456789………………..979899

------------------------

…

----------------

可以看出规律

假设 n 代表第 n 组数据：

1、1<=n <= 9时，其是一个公差d = 1的等差数列

2、10 <= n <= 99时，公差d = 2；

3、100 <= n <=999时，公差d = 3；

......

i、10^i <= n <= 10^i-1，公差 d = i+1；

**n=10^i时，长度为1\*9+2\*90+3\*900…+i\*9\*10^(i-1)+i+1**

**(可以根据这个式子加上等差数列求和公式获得每个等差数列的范围)**

对于第n组来说，其合又等于前一项的和加上新增的数，类似于：Sn = Sn-1 + an；所以可以用前缀和来求每一组的长度。

根据以上规律，首先二分找到目标位置在上述哪个n的范围中，然后再次二分在那个范围中找到准确位置。

**代码：**

#include <iostream>

#include <string>

using namespace std;

int q;

long long k, pos;

long long getNum(long long x) {

    long long a = 1, d = 1, n = 0, judge = 10, sn = 0;

    for (; judge <= x; d++, judge \*= 10) {

        n = judge - judge / 10;

        sn += a \* n + n \* (n - 1) \* d / 2;

        a = a + (n - 1) \* d + (d + 1);

    }

    long long nn = x - judge / 10 + 1;

    return sn + a \* nn + nn \* (nn - 1) \* d / 2;

}

long long getNum1(long long x) {

    long long d = 1, n = 0, judge = 10, sn = 0;

    for (; judge <= x; d++, judge \*= 10) {

        n = judge - judge / 10;

        sn += n \* d;

    }

    return sn + (x - judge / 10 + 1) \* d;

}

int main() {

    while (cin >> q) {

        for (int i = 0; i < q; ++i) {

            cin >> k;

            long long l = 0, r = 1e9;

            int ans = 0;

            while (l <= r) {

                long long mid = (l + r) >> 1;

                if (getNum(mid) < k) {

                    l = mid + 1;

                    ans = mid;

                } else

                    r = mid - 1;

            }

            k = k - getNum(ans);

            l = 0, r = 1e9;

            pos = 0;

            while (l <= r) {

                long long mid = (l + r) >> 1;

                if (getNum1(mid) < k) {

                    l = mid + 1;

                    pos = mid;

                } else

                    r = mid - 1;

            }

            string outs = to\_string(pos + 1);

            cout << outs[k - getNum1(pos) - 1] << endl;

        }

    }

    return 0;

}