图论-SPFA找负环

一、AcWing 904. 虫洞

【题目描述】

农夫约翰在巡视他的众多农场时，发现了很多令人惊叹的虫洞。

虫洞非常奇特，它可以看作是一条**单向**路径，通过它可以使你回到过去的某个时刻（相对于你进入虫洞之前）。

农夫约翰的每个农场中包含N片田地，M条路径（双向）以及W个虫洞。

现在农夫约翰希望能够从农场中的某片田地出发，经过一些路径和虫洞回到过去，并在他的出发时刻之前赶到他的出发地。

他希望能够看到出发之前的自己。

请你判断一下约翰能否做到这一点。

下面我们将给你提供约翰拥有的农场数量F，以及每个农场的完整信息。

已知走过任何一条路径所花费的时间都不超过10000秒，任何虫洞将他带回的时间都不会超过10000秒。

【输入格式】

第一行包含整数F，表示约翰共有F个农场。

对于每个农场，第一行包含三个整数N,M,W。

接下来M行，每行包含三个整数S,E,T，表示田地S和E之间存在一条路径，经过这条路径所花的时间为T。

再接下来W行，每行包含三个整数S,E,T，表示存在一条从田地S走到田地E的虫洞，走过这条虫洞，可以回到T秒之前。

【输出格式】

输出共F行，每行输出一个结果。

如果约翰能够在出发时刻之前回到出发地，则输出`YES`，否则输出`NO`。

【数据范围】

$1 \leq F \leq 5$

$1 \leq N \leq 500$

$1 \leq M \leq 2500$

$1 \leq W \leq 200$

$1 \leq T \leq 10000$

$1 \leq S, E \leq N$

【输入样例】

```
2
3 3 1
1 2 2
1 3 4
2 3 1
3 1 3
3 2 1
1 2 3
2 3 4
3 1 8
```

【输出样例】

```
NO
YES
```

【分析】

将虫洞的权值记为负数，然后直接判断图中是否存在负环即可。

求负环时dis数组的初值不管为多少都不影响结果，因为图中有负环那么做完SPFA求最短路后会存在一点i的$dis[i] == -\infty$，不管给初值赋多少都是有限值，做完SPFA后都会变成$-\infty$，且每条边的边权都是有限值，则存在负环时求最短路必然要更新无限次（更新次数$\geq n$）。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>
using namespace std;
```

```cpp

const int N = 510, M = 5210;
int e[M], ne[M], d[M], h[N], idx;
int dis[N], cnt[N];
bool st[N];
int n, m1, m2;

void add(int u, int v, int w)
{
    e[idx] = v, d[idx] = w, ne[idx] = h[u], h[u] = idx++;
}

bool spfa()
{
    memset(cnt, 0, sizeof cnt);
    memset(st, false, sizeof st);
    queue<int> Q;
    for (int i = 1; i <= n; i++) Q.push(i), st[i] = true;
    while (Q.size())
    {
        int t = Q.front();
        Q.pop();
        st[t] = false;
        for (int i = h[t]; ~i; i = ne[i])
            if (dis[t] + d[i] < dis[e[i]])
            {
                dis[e[i]] = dis[t] + d[i];
                cnt[e[i]] = cnt[t] + 1;
                if (cnt[t] >= n) return true;
                if (!st[e[i]]) Q.push(e[i]), st[e[i]] = true;
            }
    }
    return false;
}

int main()
{
    int T;
    cin >> T;
    while (T--)
    {
        memset(h, -1, sizeof h);

        idx = 0;
```

```
        cin >> n >> m1 >> m2;
        int a, b, c;
        while (m1--)
        {
            cin >> a >> b >> c;
            add(a, b, c), add(b, a, c);
        }
        while (m2--)
        {
            cin >> a >> b >> c;
            add(a, b, -c);
        }
        if (spfa()) puts("YES");
        else puts("NO");
    }
    return 0;
}
```

二、AcWing 361. 观光奶牛（01分数规划）

【题目描述】

给定一张L个点、P条边的有向图，每个点都有一个权值$f[i]$，每条边都有一个权值$t[i]$。

求图中的一个环，使“环上各点的权值之和”除以“环上各边的权值之和”最大。

输出这个最大值。

注意：数据保证至少存在一个环。

【输入格式】

第一行包含两个整数L和P。

接下来L行每行一个整数，表示$f[i]$。

再接下来P行，每行三个整数$a, b, t[i]$，表示点a和b之间存在一条边，边的权值为$t[i]$。

【输出格式】

输出一个数表示结果，保留两位小数。

【数据范围】

$2 \le L \le 1000$

$2 \le P \le 5000$

$1 \leq f[i], t[i] \leq 1000$

【输入样例】

```
5 7
30
10
10
5
10
1 2 3
2 3 2
3 4 5
3 5 2
4 5 5
5 1 3
5 2 2
```

【输出样例】

```
6.00
```

【分析】

我们可以使用二分来找出这个最大值，假设当前二分的结果为mid，那么判断一下是否存在一个环使得$\Sigma f[i]/\Sigma t[i] > mid$，如果满足，说明答案在$[mid, r]$中，否则答案在$[l, mid]$中。

将这个式子变形：

$\Sigma f[i] - mid * \Sigma t[i] > 0$

$\Sigma(f[i] - mid * t[i]) > 0$

因此我们需要判断是否存在一个环使得$\Sigma(f[i] - mid * t[i]) > 0$。将每条边的权值看成$f[i] - mid * t[i]$，那么上述式子等价于判断是否存在正环，而求正环的话只需要用SPFA求最长路即可找出是否存在正环。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>

#include <queue>
```

```cpp
using namespace std;

const int N = 1010, M = 5010;
int e[M], ne[M], t[M], h[N], idx;
int f[N], cnt[N];
double dis[N];
bool st[N];
int n, m;

void add(int u, int v, int w)
{
    e[idx] = v, t[idx] = w, ne[idx] = h[u], h[u] = idx++;
}

//判断是否存在正环
bool check(double mid)
{
    memset(cnt, 0, sizeof cnt);
    memset(st, false, sizeof st);
    queue<int> Q;
    for (int i = 1; i <= n; i++) Q.push(i), st[i] = true;
    while (Q.size())
    {
        int p = Q.front();
        Q.pop();
        st[p] = false;
        for (int i = h[p]; ~i; i = ne[i])
            if (dis[p] + f[p] - mid * t[i] > dis[e[i]])
            {
                dis[e[i]] = dis[p] + f[p] - mid * t[i];
                cnt[e[i]] = cnt[p] + 1;
                if (cnt[e[i]] >= n) return true;
                if (!st[e[i]]) Q.push(e[i]), st[e[i]] = true;
            }
    }
    return false;
}

int main()
{
    cin >> n >> m;
    memset(h, -1, sizeof h);

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

```
    while (m--)
    {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c);
    }
    double l = 0, r = 1000;
    while (r - l > 1e-4)//二分的精度比结果保留的精度多两位
    {
        double mid = (l + r) / 2;
        if (check(mid)) l = mid;
        else r = mid;
    }
    printf("%.2lf\n", r);
    return 0;
}
```

三、AcWing 1165. 单词环（01分数规划+玄学优化）

【题目描述】

我们有n个字符串，每个字符串都是由$a \sim z$的小写英文字母组成的。

如果字符串A的结尾两个字符刚好与字符串B的开头两个字符相匹配，那么我们称A与B能够相连（注意：A能与B相连不代表B能与A相连）。

我们希望从给定的字符串中找出一些，使得它们首尾相连形成一个环串（一个串首尾相连也算），我们想要使这个环串的平均长度最大。

如下例：

```
ababc
bckjaca
caahoynaab
```

第一个串能与第二个串相连，第二个串能与第三个串相连，第三个串能与第一个串相连，我们按照此顺序相连，便形成了一个环串，长度为$5+7+10=22$（重复部分算两次），总共使用了3个串，所以平均长度是$22/3 \approx 7.33$。

【输入格式】

本题有多组数据。

每组数据的第一行，一个整数n，表示字符串数量；

接下来n行，每行一个长度大于等于2且小于等于1000的字符串。

读入以$n=0$结束。

【输出格式】

若不存在环串，输出`No solution`，否则输出最长的环串的平均长度。

只要答案与标准答案的差不超过0.01，就视为答案正确。

【数据范围】

$1 \leq n \leq 10^5$

【输入样例】

```
3
intercommunicational
alkylbenzenesulfonate
tetraiodophenolphthalein
0
```

【输出样例】

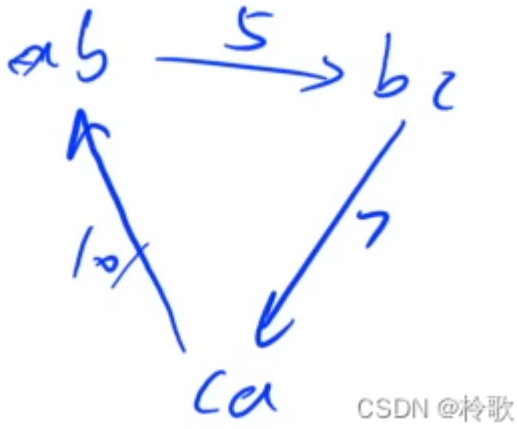
```
21.66
```

【分析】

1.建图：

一个比较直观的建图方式是将每个单词作为一个节点，如果这两个单词能够相连，则在这两个单词之间连接一条有向边，此时最多有10^5个点，10^{10}条边，不能接受。

考虑一个对偶的建图方式，将每一个单词看作一条边，其开头两个字符（用一个二十六进制数表示）和结尾两个字符为它两边的点，这样建图的话，节点数就缩小到了676个$(26*26)$，边数为10^5条。对样例进行建图如下图所示：



2.01分数规划：

我们所要求的答案为$\Sigma len/s$的最大值，其中$s=\Sigma 1$表示单词个数，len表示每个单词的长度。可以发现所求问题具有单调性，可以使用二分来求解。

设左端点为l，右端点为r，当前二分的中点为mid，则当$\Sigma len/\Sigma 1>mid$时，答案在$[mid,r]$中，否则在$[l,mid]$中。同第二题一样继续对式子进行变换：$\Sigma len-mid*\Sigma 1>0\rightarrow\Sigma(len-mid)>0$。

因此我们将每条边的权值看成$len-mid$，则在此基础上原问题可以转化为求当前图中有无正环，处理办法与第二题相同。

3.玄学优化：

在用SPFA求正环的过程中，可以采取一种比较取巧的方法：当求最短（长）路时，更新的点的总数量大于某一个值时，我们就可以武断地认为当前图中存在一个负（正）环。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <string>
#include <queue>
using namespace std;

const int N = 700, M = 100010;
int e[M], ne[M], d[M], h[N], cnt[N], idx;
double dis[N];
bool st[N];
int n;
string str;

```

```cpp
void add(int u, int v, int w)
{
    e[idx] = v, d[idx] = w, ne[idx] = h[u], h[u] = idx++;
}

bool check(double mid)
{
    memset(cnt, 0, sizeof cnt);
    memset(st, false, sizeof st);
    int count = 0;//表示所有点更新的总次数
    queue<int> Q;
    for (int i = 0; i < 676; i++) Q.push(i), st[i] = true;
    while (Q.size())
    {
        int t = Q.front();
        Q.pop();
        st[t] = false;
        for (int i = h[t]; ~i; i = ne[i])
            if (dis[t] + d[i] - mid > dis[e[i]])
            {
                dis[e[i]] = dis[t] + d[i] - mid;
                cnt[e[i]] = cnt[t] + 1;
                if (cnt[e[i]] >= N) return true;
                if (++count > 10 * N) return true;//经验值trick
                if (!st[e[i]]) Q.push(e[i]), st[e[i]] = true;
            }
    }
    return false;
}

int main()
{
    while (cin >> n, n)
    {
        memset(h, -1, sizeof h);
        idx = 0;
        for (int i = 0; i < n; i++)
        {
            cin >> str;
            int u = (str[0] - 'a') * 26 + str[1] - 'a';
            int v = (str[str.size() - 2] - 'a') * 26 + str[str.size() - 1] - 'a';

            add(u, v, str.size());
```

```
        }
        double l = 0, r = 1000;
        while (r - l > 1e-4)
        {
            double mid = (l + r) / 2;
            if (check(mid)) l = mid;
            else r = mid;
        }
        if (r - 0 < 1e-4) puts("No solution");
        else cout << r << endl;
    }
    return 0;
}
```