目录

10.5 模拟赛

1. Spelling Check

【问题描述】

给定一个 n 个顶点，m 条边的有向图。你允许从其中去掉最多一条边。

你能够去掉最多一条边就让这个图无环吗？我们称一个有向图无环，当且仅当它不包含一个环(起点和终点相同的路径)。

【输入格式】

第一行两个正整数 n,m（2≤n≤500,1≤m≤min(n(n-1),10^5)），代表图的顶点数和边数。

接下来 m 行，每行两个数,u,v，表示有一条从 u 到 v 的有向边（1≤u,v≤n）。一对(u,v) 最多出现一次。

【输出格式】

判断是否成立。

【样例输入】

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

【样例输出】

```
YES
```

题目链接：https://www.luogu.com.cn/problem/CF915D

【题目分析】

根据题意，很容易想到拓扑排序，拓扑排序可以判断图里面有没有环，所以有一种朴素的做法：首先枚举所有边，然后然后把这条边去掉，再然后跑拓扑，最后判断有没有环，有环继续枚举，没环直接输出。

但这种方法复杂度为，枚举边 O(m),拓扑排序 O(n+m),总时间复杂度为 O（m（n+m)），不能接受。所以需要优化删边这个过程，考虑删边的本质是为了让某一个原本不能入队的点入队，删一条边让这个点入队，所以这个点的入度需要-1 才能入队，本质上是让某个点入度-1 后，判断能否无环，所以枚举边的操作可以转化为枚举点。对于同一个点来说，删掉任意一条指向这个点的边对拓扑排序都是相同的，这样复杂度就变成了 O（n（n+m)），可以通过。

【参考代码】

```
#include<bits/stdc++.h>
#define MAXM 100005
#define MAXN 505
using namespace std;

int n, m;
vector<int> e[MAXN];
int in[MAXN];
int IN[MAXN];

bool topsort()
{
    queue<int> q;
    int cnt = 0;

    for (int i = 1; i <= n; i++) in[i] = IN[i];

    for (int i = 1; i <= n; i++)
    {
        if (in[i] == 0) q.push(i);
    }
```

```
    while (!q.empty())
    {
        int u = q.front(); q.pop();
        cnt++;
        for (int i = 0; i < e[u].size(); i++)
        {
            int cur = e[u][i];
            in[cur]--;
            if (in[cur] == 0)
            {
                q.push(cur);
            }
        }
    }

    return (cnt == n);
}

int main()
{
    std::ios::sync_with_stdio(false);

    cin >> n >> m;

    for (int i = 1; i <= m; i++)
    {
        int u, v;
        cin >> u >> v;
        e[u].push_back(v);
        IN[v]++;
    }

    for (int i = 1; i <= n; i++)
    {
        bool flag = 0;
        if (IN[i] != 0)
        {
            IN[i]--;
            flag = true;
        }

        if (topsort())
        {
            cout << "YES" << endl;
            return 0;
        }

        if (flag == true) IN[i]++;
    }
```

```
    cout << "NO" << endl;
}
```

2. Almost Acyclic Graph

【问题描述】

Petya 发现，当他用键盘打字时，他经常多打出一个字母。他想要发明一个自动改正单词的程序，能够将他打出的单词删去一个字母，改为字典中对应的正确单词。请你帮助他写一个程序，从打出的单词中删去哪一个字母，才能改为字典中的那个单词？

【输入格式】

输入包含两个字符串，字符串中只含有小写字母。每个字符串长度不超过 10^6，第一个字符串总是比第二个的字符数多 1。

【输出格式】

输出的第一行应为改正的方案总数。在第二行以增序输出每一种方案被删除字母的位置（字母从 1 开始编号），如果不能通过删除一个字母的方式改正，则输出 0。

【样例输入】

```
abdrakadabra
abrakadabra
```

【样例输出】

```
1
3
```

题目链接：https://www.luogu.com.cn/problem/CF39J

【题目分析】

方法一：双指针模拟。由于题目中只能删掉一个字母，那么对于出现两个及以上不同字母的情况下，一定无解。那么有解一定是存在一个字母不同的情况。对于一个字母不同的情况，这些解一定是连在一起的。如果存在两个位置 x,y 都为解，那么删掉 x 后，s1[x+1] = s2[x]，s1[y]=s2[y]，删掉 y 后，s1[y+1] = s2[y]，s1[x] = s2[x]。以此类推，可以得到 s1[x] = s1[x+1]=s1[x+2]=…=s1[y]，所以解的方案一定是连在一起的相同字母，用双指针标记模拟即可。

方法二：哈希。枚举删除点，利用哈希 O(1)判断两个字符串是否相同。要求[l,R]字符串的哈希值，可以用前缀和来快速计算，hashR-hashL*base^（R-L+1），其中 base^(R-L+1)可以提前打表预处理。为了判断在[l,R]内，删掉某个点 p 后，两个字符串是否相同，可以用第一个字符串 hash（L，p-1）*base^(R-p)+hash(p+1,R)的值与第二个字符串的 hash 值比较是否相同。

【参考代码 1】

```
#include<bits/stdc++.h>
using namespace std;
char s1[1000010],s2[1000010];
int len1,len2;
int an;
int main()
{
    scanf("%s",s1+1);len1=strlen(s1+1);
    scanf("%s",s2+1);len2=strlen(s2+1);
    int t=1,bj=0,ans=0;
    for(int i=1;i<=len1;i++)
    {
        if(s1[i]==s2[t])
        {
            t++;
        }
        else
        {
            an=i;bj++;
        }
    }
```

```
    if(bj>1)
    {
        printf("0");
        return 0;
    }
    for(int i=an-1;i>=1;i--)
    {
        if(s1[i]==s1[an])
        {
            ans++;
        }
        else break;
    }
    printf("%d\n",ans+1);
    for(int i=an-ans;i<=an;i++) printf("%d ",i);
    return 0;
}
//指针模拟
```

【参考代码 2】

```
#include<bits/stdc++.h>
#define ull unsigned long long
using namespace std;
const int base=233,N=1000005;
string a,b;
ull hs1[N],hs2[N],power[N];//
int lena,lenb,tot,ans[N];
ull get(int L,int R){//得到字符串 a 在[L,R]范围的字串哈希值
    return hs1[R]-hs1[L-1]*power[R-L+1];
}
ull re_hash(int L,int R,int p){//字符串 a 中，[L,R]的区间内删除第 p 位的哈希值
    return get(L,p-1)*power[R-p]+get(p+1,R);
}
int main(){
    cin>>a>>b;
    lena=a.size();
    lenb=b.size();
    a="0"+a;//方便计算，在前面补个零
    b="0"+b;
    power[0]=1;//初始化 base 的 0 次方为 1
    for(int i=1;i<=max(a.size(),b.size());i++){//预处理 base 的幂
        power[i]=base*power[i-1];
    }
    for(int i=1;i<=lena;i++){//生成字符串 a 的哈希值
        hs1[i]=hs1[i-1]*base+a[i];
    }
    for(int i=1;i<=lenb;i++){//生成字符串 b 的哈希值
        hs2[i]=hs2[i-1]*base+b[i];
    }
    ull val=hs2[lenb];//字符串 b 的哈希值
    for(int i=1;i<=lena;i++){
        if(re_hash(1,lena,i)==val){//删掉字符串 a 的第 i 位得到字符串的哈希值是否与字符串 b 的哈希值相同
```

```
            ans[++tot]=i;//满足条件就记录下来
        }
    }
    cout<<tot<<endl;//输出
    for(int i=1;i<=tot;i++){
        cout<<ans[i]<<' ';
    }
    return 0;
}
//哈希
```

3. Creative Snap

【问题描述】

灭霸要摧毁复仇者们的基地！

我们可以将复仇者的基地看成一个序列，每个位置都有可能有多个复仇者；但是每个复仇者只能占据一个位置。

他们基地的长度刚好是 2 的整数幂，灭霸想要用最少的能量摧毁它们。他在摧毁过程中，可以选择：

如果这段基地长度≥2，他可以将其分为相等长度的两半。

烧掉这段基地。如果这段基地中没有复仇者，他需要消耗 A 的能量；如果有，则需要消耗 B*x*l 的能量。其中 l 是这段基地长度，x 是这段中的复仇者数量。

输出一个整数，表示他摧毁全部基地需要的最少能量。

【输入格式】

第一行四个整数 n,k,A,B；2n 为基地长度，k 是总共的复仇者数量，A,B 的意义如题目描述。

接下来一行 k 个整数，ai 表示第 i 个复仇者所在的位置

【输出格式】

一个整数，表示摧毁基地所需要的最少能量。

【样例输入】

```
2 2 1 2
1 3
```

【样例输出】

```
6
```

题目链接：https://www.luogu.com.cn/problem/CF1111C

【题目分析】

题意暗示了可以采用分治的思路统计答案，所以可以用 dfs 分治来统计每段的最少能量，而最少能力取决于每一段中复仇者的数量，所以需要统计在每一段分治区间内的复仇者数量。

考虑采用二分维护，对于[l,r]区间内的复仇者数量 cnt，可以使用二分查找确定第一个位置大于等于 l 的复仇者和最后一个位置小于等于 r 的复仇者。如果区间内没有复仇者，那么答案就是 A，不用继续递归分治，再分区间也不会更优。如果区间内有复仇者，考虑拆分与不拆分两种情况的更小值即可。

如果不拆分成两半，代价为 B*cnt*(r-l+1)。如果拆分成两半，就是递归到[l,(l+r)/2]和[(l+r)/2+1,r]这两部分区间，然后继续递归求两端区间的和。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
#define LL long long
LL n,k,A,B,s[100005];
```

```
LL solve(LL l,LL r){
    LL m1=lower_bound(s,s+k+2,l)-s,m2=upper_bound(s,s+k+2,r)-s-1;
    if(m2<m1)  return A;
    if(l==r)    return B*(m2-m1+1);
    LL mid=(l+r)>>1;
    return min(solve(l,mid)+solve(mid+1,r),B*(r-l+1)*(m2-m1+1));
}
int main(){
    cin>>n>>k>>A>>B;
    for(int i=1;i<=k;i++)   cin>>s[i];
    sort(s+1,s+k+1),s[0]=LONG_LONG_MIN,s[k+1]=LONG_LONG_MAX;
    cout<<solve(1ll,1ll<<n);
}
```

4. Rain

【问题描述】

给定一个 n，这 n 天会降雨，每一个降雨天有一个 xi 和一个 pi，表示以 xi 为中心，降雨量向左右两边递减，即对于所有的数轴上的点 x ，其降雨量都会增加 max(0,pi-|xi-x|)

现在你可以消除一天的降雨，问消除第 i 天的降雨是否可以使得没有任何地方的降雨大于 M

【输入格式】

第一行三个整数 n,m,q，分别表示 点的个数，边的个数和询问个数

接下来 m 行，每行两个整数 x,y，表示有一条链接点 x,y 的边

接下来 q 行，每行表示一条操作

操作 1：1 x

操作 2：2 x y

【输出格式】

输出行数为操作 1 的个数

每行一个整数表示对应的操作一的答案

【样例输入】

```
4
3 6
1 5
5 5
3 4
2 3
1 3
5 2
2 5
1 6
10 6
6 12
4 5
1 6
12 5
5 5
9 7
8 3
```

【样例输出】

```
001
11
00
100110
```

题目链接：https://www.luogu.com.cn/problem/CF1710B

【题目分析】

由于题目中要分析每个降雨点的情况，所以分析消除每一天前，可以想办法把每个点的降雨总量求出来。分析题目中降雨量的分布，为一次函数，所以可以利用差分数组维护斜率和截距或者其他数据结构，维护出每个点的总降雨量。

接下来考虑消除每个点对降雨量的影响，要想使得消除一个点使得所有点降雨量不超过 m，那么当然可以 n^2 的暴力判断每个点的情况，但时间复杂度过高。重点分析降雨点与当前点的关系，设总降雨量为序列 a，对于一个原先降水量 >m 的位置 u，若删去一个位置 v 后它的降水量≤m，则必然有 au-(pv-|xv-xu|)≤m。将绝对值展开，将 u 和 v 分别移项到不等式两边，可得

$$au - xu \leqslant pv - xv + m$$
$$au + xu \leqslant pv + xv + m$$

那么对于考虑删去的降雨点 v，只有所有的 au-xu 和 au+xu 都小于等于相应的条件（pv - xv + m 和 pv + xv + m）才能满足题目要求。求出所有的 au-xu 和 au+xu 的最大值 m1,m2，则满足条件的 v 应该使得 $m1 \leqslant pv - xv + m$ 并且 $m2 \leqslant pv + xv + m$。

时间复杂度(nlogn)，离散化的复杂度。

【参考代码】

```
#include <bits/stdc++.h>
using namespace std;
#define fi first
#define se second
using ll = long long;
void umx(ll &x, const ll &y) { x = max(x, y); }
const char nl = '\n';
const ll INF = 1e18;
const ll MXN = 1e6 + 5;

ll n, m;
map<ll, ll> delt;
ll x[MXN], p[MXN];

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int t;
    cin >> t;
    while (t--) {
        delt.clear();
        cin >> n >> m;
        for (ll i = 1; i <= n; i++) {
            cin >> x[i] >> p[i];
            delt[x[i] - p[i] + 1]++;
            delt[x[i] + 1] -= 2;
            delt[x[i] + p[i] + 1]++;
        }
        ll b = 0, k = 0, lastx = -INF;
        ll b_1 = -INF, b1 = -INF;
        for (auto it : delt) {
            b += k * (it.fi - lastx);
            k += it.se;
```

```
            if (b > m) {
                umx(b1, b - it.fi + 1);
                umx(b_1, b + it.fi - 1);
            }

            lastx = it.fi;
        }
        for (ll i = 1; i <= n; i++) cout << ((p[i] + m - x[i] >= b1) && (p[i] + m + x[i] >= b_1));
        cout << nl;
    }
    return 0;
}
```