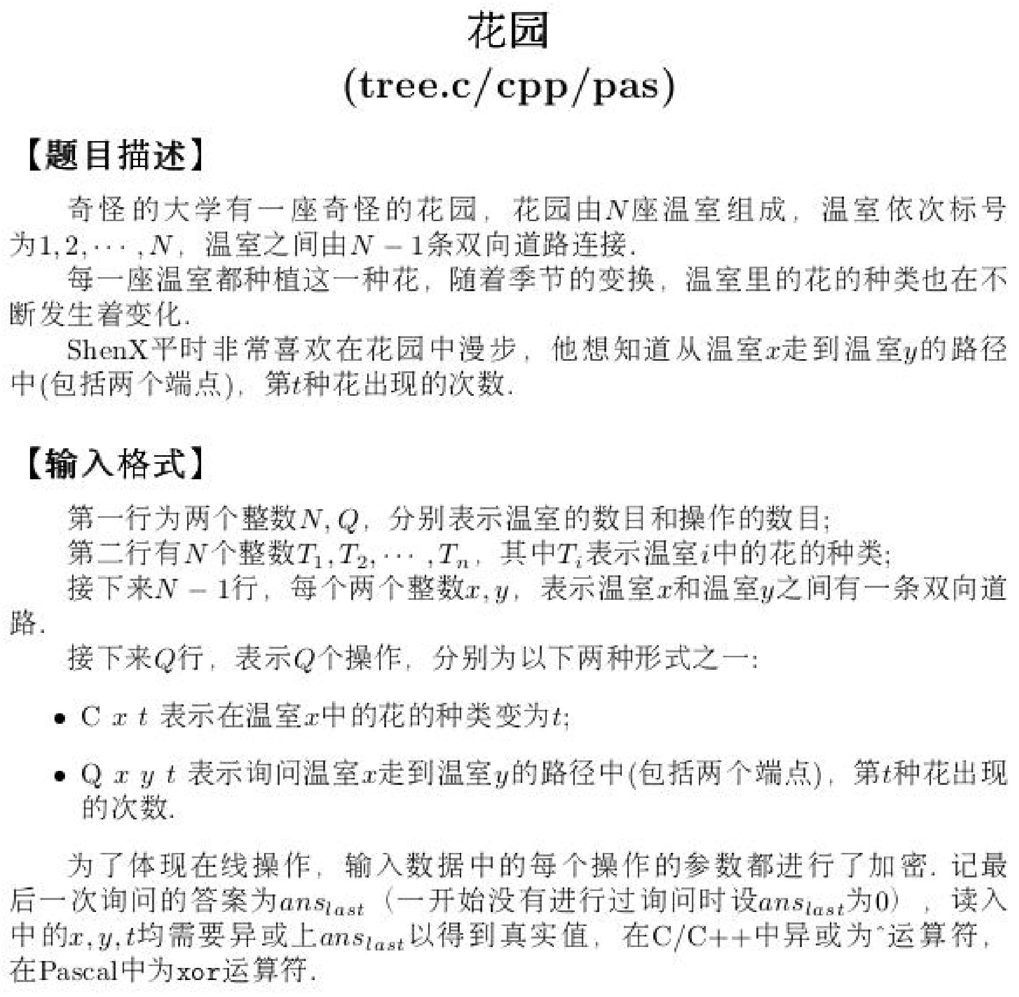
1 代码填空

题目背景



花园
(tree.c/cpp/pas)

【题目描述】

奇怪的大学有一座奇怪的花园，花园由N座温室组成，温室依次标号为$1, 2, \cdots, N$，温室之间由$N-1$条双向道路连接.

每一座温室都种植这一种花，随着季节的变换，温室里的花的种类也在不断发生着变化.

ShenX平时非常喜欢在花园中漫步，他想知道从温室x走到温室y的路径中(包括两个端点)，第t种花出现的次数.

【输入格式】

第一行为两个整数N, Q，分别表示温室的数目和操作的数目;

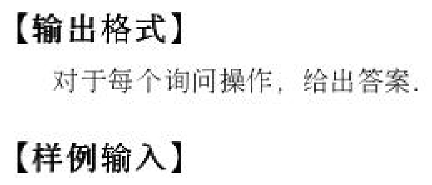
第二行有N个整数$T_1, T_2, \cdots, T_n$，其中T_i表示温室i中的花的种类;

接下来$N-1$行，每个两个整数x, y，表示温室x和温室y之间有一条双向道路.

接下来Q行，表示Q个操作，分别为以下两种形式之一:

- C x t 表示在温室x中的花的种类变为t;
- Q x y t 表示询问温室x走到温室y的路径中(包括两个端点)，第t种花出现的次数.

为了体现在线操作，输入数据中的每个操作的参数都进行了加密. 记最后一次询问的答案为ans_{last}（一开始没有进行过询问时设ans_{last}为0），读入中的x, y, t均需要异或上ans_{last}以得到真实值，在C/C++中异或为^运算符，在Pascal中为xor运算符.



【输出格式】

对于每个询问操作，给出答案.

【样例输入】

```
5 8
10 20 30 40 50
1 2
1 3
3 4
3 5
Q 2 5 10
C  2 21
```

```
Q 3 4 21
C 6 22
Q 1 7 28

C 5 20
Q 2 5 20
Q 2 0 9
```

【样例输出】

```
1
2
0
3
1
```

【样例解释】

这是加密前的操作:

```
Q 2 5 10
C 3 20
Q 2 5 20
C 4 20
Q 3 5 30
C 5 20
Q 2 5 20
Q 1 3 10
```

思路描述&代码

对于每一种颜色建一颗线段树，求一边 dfs 序把点修改链上询问转化成区间修改点询问的问题，线段树动态开节点防止爆炸。

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cmath>
#include <stack>
#include <vector>
#include <queue>
#include <cstring>
#include <string>
#include <map>
#include <set>
using namespace std;

int read() {
	int x = 0, f = 1; char c = getchar();
	while(!isdigit(c)){ if(c == '-') f = -1; c = getchar(); }
	while(isdigit(c)){ x = x * 10 + c - '0'; c = getchar(); }
	return x * f;
}

#define maxn 100010
#define maxm 200010
#define maxnode 10000010
int n, m, head[maxn], next[maxm], to[maxm], T[maxn];

#define MOD 233333
int Th, hd[MOD], nxt[maxn*3], Val[maxn*3];
int Find(int x) {
	int v = (x % MOD + MOD) % MOD;
	for(int e = hd[v]; e; e = nxt[e]) if(Val[e] == x) return e;
	return 0;
}
void insert(int x) {
	int v =         ①         ;
	Val[++Th] = x; nxt[Th] = hd[v]; hd[v] = Th;
	return ;
}

void AddEdge(int a, int b) {
	to[++m] = b; next[m] = head[a]; head[a] = m;
	swap(a, b);
	to[++m] = b; next[m] = head[a]; head[a] = m;
	return ;
```

```
}
int fa[maxn], son[maxn], siz[maxn], dep[maxn], top[maxn], W[maxn], ww, dl[maxn],
dr[maxn];
void build(int u) {
	siz[u] = 1;
	for(int e = head[u]; e; e = next[e]) if(to[e] != fa[u]) {
		fa[to[e]] = u;
		dep[to[e]] =      ②      ;
		build(to[e]);
		siz[u] +=      ③      ;
		if(!son[u] || siz[son[u]] < siz[to[e]]) son[u] = to[e];
	}
	return ;
}
void gett(int u, int tp) {
	dl[u] = W[u] = ++ww; top[u] = tp;
	if(son[u])      ④      ;
	for(int e = head[u]; e; e = next[e]) if(to[e] != fa[u] && to[e] != son[u])
		gett(to[e], to[e]);
	dr[u] = ww;
	return ;
}
int lca(int a, int b) {
	int f1 = top[a], f2 = top[b];
	while(      ⑤      ) {
		if(dep[f1] < dep[f2]) swap(f1, f2), swap(a, b);
		a = fa[f1]; f1 = top[a];
	}
	return dep[a] < dep[b] ? a : b;
}

int ToT, rt[maxn*3], lc[maxnode], rc[maxnode], S[maxnode], addv[maxnode];
void pushdown(int o, int L, int R) {
	if(!addv[o]) return ;
	S[o] += addv[o];
	if(L < R) {
		if(!lc[o]) lc[o] = ++ToT;
		if(!rc[o]) rc[o] = ++ToT;
		addv[lc[o]] += addv[o];
		addv[rc[o]] += addv[o];
```

```
        }
        addv[o] = 0;
        return ;
}
void update(int& o, int L, int R, int ql, int qr, int v) {
        if(        ⑥        ) o = ++ToT;
        pushdown(o, L, R);
        if(ql <= L && R <= qr) addv[o] += v;
        else {
                int M = L + R >> 1;
                if(ql <= M) update(lc[o], L, M, ql, qr, v);
                if(qr > M)                 ⑦                ;
        }
        return ;
}
int query(int o, int L, int R, int x) {
        if(!x || !o) return 0;
        pushdown(o, L, R);
        if(L == R) return S[o];
        int M = L + R >> 1;
        if(x <= M) return query(lc[o], L, M, x);
        return query(rc[o], M+1, R, x);
}

int main() {
        n = read(); int q = read();
        for(int i = 1; i <= n; i++) {
                T[i] = read();
                if(!Find(T[i])) insert(T[i]);
        }
        for(int i = 1; i < n; i++) {
                int a = read(), b = read();
                AddEdge(a, b);
        }
        build(1); gett(1, 1);
        for(int i = 1; i <= n; i++) update(rt[Find(T[i])], 1, n, dl[i], dr[i], 1);
        int lst = 0;
        while(q--) {
                char Tp[2]; scanf("%s", Tp);
                if(Tp[0] == 'C') {
                        int x = read() ^ lst, t = read() ^ lst;
                        update(rt[Find(T[x])], 1, n, dl[x], dr[x], -1);
```

```
            T[x] = t; if(!Find(t)) insert(t);
            update(rt[Find(t)], 1, n, dl[x], dr[x], 1);
        }
        if(Tp[0] == 'Q') {
            int a = read() ^ lst, b = read() ^ lst, t = rt[Find(read()^lst)], c = lca(a, b);
            int A = query(t, 1, n, W[a]), B = query(t, 1, n, W[b]), C = query(t, 1, n,
W[c]), fC = query(t, 1, n, W[fa[c]]);
            lst =           ⑧           ;
            printf("%d\n", lst);
        }
    }

    return 0;
}
```

2 代码填空

题目背景

第二题 （island）
Farmer John has taken the cows to a vacation out on the ocean! The cows are living on N (1 <= N <= 15) islands, which are located on an R x C grid (1 <= R, C <= 50). An island is a maximal connected group of squares on the grid that are marked as 'X', where two 'X's are connected if they share a side. (Thus, two 'X's sharing a corner are not necessarily connected.) Bessie, however, is arriving late, so she is coming in with FJ by helicopter. Thus, she can first land on any of the islands she chooses. She wants to visit all the cows at least once, so she will travel between islands until she has visited all N of the islands at least once. FJ's helicopter doesn't have much fuel left, so he doesn't want to use it until the cows decide to go home. Fortunately, some of the squares in the grid are shallow water, which is denoted by 'S'. Bessie can swim through these squares in the four cardinal directions (north, east, south, west) in order to travel between the islands. She can also travel (in the four cardinal directions) between an island and shallow water, and vice versa. Find the minimum distance Bessie will have to swim in order to visit all of the islands. (The distance Bessie will have to swim is the number of distinct times she is on a square marked 'S'.) After looking at a map of the area, Bessie knows this will be possible.

PROBLEM NAME: island
INPUT FORMAT: * Line 1: Two space-separated integers: R and C. * Lines 2..R+1: Line i+1 contains C characters giving row i of the grid. Deep water squares are marked as '.', island squares are marked as 'X', and shallow water squares are marked as 'S'.
SAMPLE INPUT (file island.in):

```
5 4
XX.S
.S..
SXSS
S.SX
..SX
```

INPUT DETAILS: There are three islands with shallow water

paths connecting some of them. OUTPUT FORMAT: * Line 1: A single integer representing the minimum distance Bessie has to swim to visit all islands.
SAMPLE OUTPUT (file island.out): 3
OUTPUT DETAILS: Bessie can travel from the island in the top left to the one in the middle, swimming 1 unit, and then travel from the middle island to the one in the bottom right, swimming 2 units, for a total of 3 units.

题意农给你一张 r*c 的地图，有’S’,’X’,’.’三种地形，所有判定相邻与行走都是四连通的。我们设’X’为陆地，一个’X’连通块为一个岛屿，’S’为浅水，’.’为深水。刚开始你可以降落在任一一块陆地上，在陆地上可以行走，在浅水里可以游泳。并且陆地和浅水之间可以相互通行。但无论如何都不能走到深水。你现在要求通过行走和游泳使得你把所有的岛屿都经过一边。Q：你最少要经过几个浅水区？保证有解。

思路描述&代码

这道题测试的时候，看到只有 15 个岛屿，想到了状压 DP 但是在预处理的时候卡住了，所有就只写了一个 DFS 的暴力，刚开始加了一些减枝优化，但是可能是处理不当，加上之后就卡死在里面了，最后没调出来就直接交了一个小暴力。然后考完之后，看了看题解就顿悟了。
首先先由 DFS 搜索出所有的连通块（即岛屿），然后给每个岛屿中的点相同的边号，然后用 spfa 预处理求出每两个连通块之间最少需要经过的浅水区，进行状压 dp。dp 方程如下：dp[i|(1<<k)][k]=min(dp[i|(1<<k)][k],dp[i][j]+dis[j+1][k+1]);
dp[i][j]表示处在 i 这种状态，最后到达 j 这个岛。
用二进制表示状态，0 表示处在当前位置的岛屿没被访问过，1 表示访问过。
上面之所以是 dis[j+1][k+1] 因第一个岛屿处在第一位，需要左移 0 位。当然转移之前需要判断状态 i 中是否已经访问过 j,然后进行转移就可以。

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<queue>
#define N 67
using namespace std;
int n,m,dis[N][N],len[N][N][N],p1[100000],p2[100000];
int map[N][N],c[N][N],can[N][N],dp[1<<15][15];
int x[10]={0,-1,0,1},y[10]={-1,0,1,0},tot;
const int inf=1e9;
struct data
{
    int x,y;
};
void dfs(int x1,int y1,int k)
{
    for (int i=0;i<4;i++)
    {
      int xx=x1+x[i]; int yy=y1+y[i];
      if (xx>0&&yy>0&&xx<=n&&yy<=m&&!map[xx][yy]&&!can[xx][yy])
        {
            can[xx][yy]=1; c[xx][yy]=k;
                      ⑨           ;
        }
    }
}
void spfa(int fx,int fy,int cnt)
{
    memset(can,0,sizeof(can));
    int k=c[fx][fy]; len[k][fx][fy]=0; can[fx][fy]=1;
    memset(p1,0,sizeof(p1)); memset(p2,0,sizeof(p2));
    int h=0; int t=0;
    t++; p1[t]=      ⑩      ; p2[t]=        ⑪        ;
    while (      ⑫      )
    {
      h++;
      int nowx=p1[h]; int nowy=p2[h];
      for (int i=0;i<4;i++)
      {
        int xx=nowx+x[i]; int yy=nowy+y[i];
        if (xx>0&&yy>0&&xx<=n&&yy<=m&&map[xx][yy]!=-1)
          {
            if (len[k][xx][yy]>len[k][nowx][nowy]+map[xx][yy])
            {
              len[k][xx][yy]=len[k][nowx][nowy]+map[xx][yy];
              if (!can[xx][yy])
                {
                can[xx][yy]=1;
                t++; p1[t]=xx; p2[t]=yy;
                }
            }
          }
      }
      can[nowx][nowy]=0;
    }
    for (int i=1;i<=n;i++)
      for (int j=1;j<=m;j++)
        if (k!=c[i][j])
          dis[k][c[i][j]]=dis[c[i][j]][k]=          ⑬          ;
}
int main()
{
  scanf("%d%d",&n,&m);
  for (int i=1;i<=n;i++)
    {
      char s[100]; scanf("%s",s);
      for (int j=1;j<=m;j++)
        switch(s[j-1])
        {
            case '.':map[i][j]=-1; break;
            case 'S':map[i][j]=1; break;
            case 'X':map[i][j]=0; break;
        }
    }
  memset(dis,127/3,sizeof(dis));
  memset(len,127/3,sizeof(len));
  for (int i=1;i<=n;i++)
      for (int j=1;j<=m;j++)
        if (!map[i][j]&&!can[i][j])
          {
              can[i][j]=1; tot++;    c[i][j]=tot;
              dfs(          ⑭          );
          }
    for (int i=1;i<=n;i++)
      for (int j=1;j<=m;j++)
        if (!map[i][j])
          spfa(i,j,c[i][j]);
    memset(dp,127/3,sizeof(dp));
    for (int i=0;i<tot;i++)     dp[1<<i][i]=0;
    for (int i=0;i<(1<<tot);i++)
      for (int j=0;j<tot;j++)
      if ((i>>j)&1)
        for (int k=0;k<tot;k++)
          dp[i|(1<<k)][k]=min(            ⑮             );
    int ans=inf;
    for (int i=0;i<tot;i++)
      ans=min(ans,dp[(1<<tot)-1][i]);
    printf("%d",ans);
}
```

3 代码填空

题目背景

grass

In an effort to better manage the grazing patterns of his cows, Farmer John has installed one-way cow paths all over his farm. The farm consists of N fields, conveniently numbered 1..N, with each one-way cow path connecting a pair of fields. For example, if a path connects from field X to field Y, then cows are allowed to travel from X to Y but not from Y to X. Bessie the cow, as we all know, enjoys eating grass from as many fields as possible. She always starts in field 1 at the beginning of the day and visits a sequence of fields, returning to field 1 at the end of the day. She tries to maximize the number of distinct fields along her route, since she gets to eat the grass in each one (if she visits a field multiple times, she only eats the grass there once). As one might imagine, Bessie is not particularly happy about the one-way restriction on FJ's paths, since this will likely reduce the number of distinct fields she can possibly visit along her daily route. She wonders how much grass she will be able to eat if she breaks the rules and follows up to one path in the wrong direction. Please compute the maximum number of distinct fields she can visit along a route starting and ending at field 1, where she can follow up to one path along the route in the wrong direction. Bessie can only travel backwards at most once in her journey. In particular, she cannot even take the same path backwards twice.

INPUT: (file grass.in) The first line of input contains N and M, giving the number of fields and the number of one-way paths (1 <= N, M <= 100,000). The following M lines each describe a one-way cow path. Each line contains two distinct field numbers X and Y, corresponding to a cow path from X to Y. The same cow path will never appear more than once.

SAMPLE INPUT: 7 10 1 2 3 1 2 5 2 4 3 7 3 5 3 6 6 5 7 2 4 7

OUTPUT: (file grass.out) A single line indicating the maximum number of distinct fields Bessie can visit along a route starting and ending at field 1, given that she can follow at most one path along this route in the wrong direction.

SAMPLE OUTPUT: (file grass.out) 6 SOLUTION NOTES: Here is an ASCII drawing of the sample input:

```
v---3-->6
7   |\   |
^\  v \  |
| \ 1  \ |
|  \|    v
|   v    5
4<--2----^
```

Bessie can visit pastures 1, 2, 4, 7, 2, 5, 3, 1 by traveling backwards on the path between 5 and 3. When she arrives at 3 she cannot reach 6 without following another backwards path.

给一个有向图，然后选一条路径起点终点都为 1 的路径出来，有一次机会可以沿某条边逆方向走，问最多有多少个点可以被经过？（一个点在路径中无论出现多少正整数次对答案的贡献均为 1）

思路描述&代码

边是可以重复走的，所以在同一个强连通分量里，无论从那个点进入从哪个点出，所有的点一定能被一条路走到。
那么首先用 tarjan 将所有的强连通分量缩成一个点，每个点的权为该强连通分量中点的个数。然后我们考虑将一条边反置。 强连通分量里的边反置是没有价值的，所以只需要考虑 DAG 里的边。枚举一条要反置的边，如果将这条边反置的话，答案应该为它的起点到 1 最多能走到的点数和 1 到它的终点最多能走到的点数，也就是相当于这条边将一个环连通起来了。而从它到 1 最多能走到的点数和 1 到它的终点最多能走到的点数可以用两遍 spfa 最长路预处理好，这个问题就被解决了。

```
#include<iostream>
#include<cstring>
#include<cstdio>
#include<queue>
using namespace std;
#define N 100005

int n,m,dfs_clock,scc,ans;
struct hp{int x,y;}e[N];
int dfn[N],low[N],stack[N],tmp,belong[N],size[N],dis[N],_dis[N];
int tot,point[N],nxt[N],v[N];
bool vis[N];
queue <int> q;

void add(int x,int y)
{
	++tot; nxt[tot]=point[x]; point[x]=tot; v[tot]=y;
}
void tarjan(int x)
{
	dfn[x]=low[x]=          ⑯          ;vis[x]=true;stack[++tmp]=x;
	for (int i=point[x];i;i=nxt[i])
		if (!dfn[v[i]])
		{
			tarjan(v[i]);
			low[x]=          ⑰          ;
		}
		else if (vis[v[i]])
			low[x]=min(low[x],dfn[v[i]]);
	if (dfn[x]==low[x])
	{
		++scc;int now=0;
		while (now!=x)
		{
			now=          ⑱          ;
			vis[now]=false;
			belong[now]=scc;
			size[scc]++;
		}
	}
}
```

```
void build()
{
	tot=0;memset(point,0,sizeof(point));
	for (int i=1;i<=m;++i)
		if (          ⑲          )
			add(belong[e[i].x],belong[e[i].y]);
}
void rebuild()
{
	tot=0;memset(point,0,sizeof(point));
	for (int i=1;i<=m;++i)
		if (belong[e[i].x]!=belong[e[i].y])
			add(belong[e[i].y],belong[e[i].x]);
}
void spfa(int *dis)
{
	memset(vis,0,sizeof(vis));
	dis[belong[1]]=          ⑳          ;vis[belong[1]]=true;q.push(belong[1]);
	while (!q.empty())
	{
		int now=q.front();q.pop();
		vis[now]=false;
		for (int i=point[now];i;i=nxt[i])
			if (dis[v[i]]<dis[now]+size[v[i]])
			{
				dis[v[i]]=dis[now]+size[v[i]];
				if (!vis[v[i]])
				{
					vis[v[i]]=true;
					q.push(v[i]);
				}
			}
	}
}
int main()
{
	scanf("%d%d",&n,&m);
	for (int i=1;i<=m;++i)
	{
		scanf("%d%d",&e[i].x,&e[i].y);
		add(e[i].x,e[i].y);
	}
	for (int i=1;i<=n;++i)
		if (!dfn[i]) tarjan(i);
	build();
	spfa(dis);
	rebuild();
	spfa(_dis);
	ans=size[belong[1]];
	for (int i=1;i<=m;++i)
		if (belong[e[i].x]!=belong[e[i].y])
			if (_dis[belong[e[i].x]]&&dis[belong[e[i].y]])
			  ans=max(        ㉑          );
	printf("%d\n",ans);
}
```