

2024暑期CSP-S&NOIP模拟赛第1套题解

T1 弹钢琴 (piano)

30-70%

如果音 a_i 能弹对，则说明 $a_i - a_1 = b_i k$ 。其中 $b_i = \begin{cases} b_{i-1} + 1 & a_i > a_{i-1} \\ b_{i-1} & a_i = a_{i-1} \text{ 且 } b_1 = 0 \\ b_{i-1} - 1 & a_i < a_{i-1} \end{cases}$ 。

枚举与 a_1 匹配的 a_i ，算出对应的 k ，然后对于剩余的去check算出的 k 是否符合要求取 max 即可，时间复杂度 $O(n^2)$

100%

观察 $a_i - a_1 = b_i k$ ，发现已知 i 时 k 是唯一确定的，那么其实可以从每个 k 考虑对答案的贡献即可避免之前做法的check，记 c_k 表示确定 k 后能弹对多少个音。初始 c_i 均为 1 (a_1 均能弹对)。

由 2 到 n 递推。如果音 a_i 弹对了，那么：

- 若 $b_i = a_i - a_1 = 0$ ，所有 c_k 均加一。
- 若 b_i 与 $a_i - a_1$ 异号或 $b_i = 0$ 且 $a_i \neq a_1$ ，那么对 c_k 无贡献。
- 若 $(a_i - a_1) \bmod b_i \neq 0$ ，也对 c_k 无贡献。
- 否则，记 $p = \frac{a_i - a_1}{b_i}$ ， c_p 加一，即 $k = p$ 时的贡献加一。

递推过程中更新并记录即可。

时间复杂度 $O(n)$ 。

```
1  #include<bits/stdc++.h>
2  #include<unordered_map>
3  #define int long long
4  using namespace std;
5  const int N=1e6+10;
6  int a[N],b[N];
7  int n,tag,ans, pos;
8  unordered_map<int,int> num;
9  signed main(){
10     freopen("piano.in", "r", stdin);
11     freopen("piano.out", "w", stdout);
12     cin>>n;
13     for(int i=1;i<=n;i++)
14         cin>>a[i];
15     for(int i=2;i<=n;i++){
16         if(a[i]>a[i-1])
17             b[i]=b[i-1]+1;
18         else if(a[i]==a[i-1])
19             b[i]=b[i-1];
```

```

20         else b[i]=b[i-1]-1;
21         int r=a[i]-a[1], q=b[i];
22         int npos;
23         if(q==0){
24             if(r==0) ++tag,npos=pos;
25             else continue;
26         }
27         else if(r==0) ++num[0],npos=0;
28         else if(r<0&&q<0){
29             r=-r;q=-q;
30             if(r%q==0) ++num[r/q],npos=r/q;
31         }
32         else if((r<0&&q>0)|| (r>0&&q<0))
33             continue;
34         else if(r%q==0) ++num[r/q],npos=r/q;
35         if(ans==num[npos]+tag) pos=min(pos,npos);
36         else if(ans<num[npos]+tag){
37             ans=num[npos]+tag;
38             pos=npos;
39         }
40     }
41     cout<<ans+1<<'\n'<<pos;
42     return 0;
43 }

```

考察知识点

桶，计数

T2 选彩笔 (rgb)

可以从一维二维的情况开始考虑

50%

发现RGB范围很小，可以考虑把RGB堪称三维空间，枚举一个长度（Colorfulness 值）再枚举答案组里RGB三个坐标的最大值，去这个空间里check是否存在k个点，时间复杂度约为 $O(R^7)$, R 为RGB的值域范围。

80%

枚举一个长度后发现check空间里是否存在k个点能通过三维前缀和+容斥维护得到，时间复杂度降为 $O(R^4)$, R 为RGB的值域范围。

100%

观察到题目要求差值最大值最小，注意到答案具有二分性，可以考虑二分 长度。枚举 R,G,B 的最大值，计算在 R 值在区间 $[maxr - mid, maxr]$ 内，G 值在区间 $[maxg - mid, maxg]$ 内，B 值在区间 $[maxb - mid, maxb]$ 内的笔的个数。定义 $sum[i][j][k]$ R 值在区间 $[1, i]$ 内，G 值在区间 $[1, j]$ 内，B 值在区间 $[1, k]$ 内的笔的个数，可以通过三维前缀和求解。时间复杂度降为 $O(R^3 \log R)$, R 为RGB的值域范围。

代码如下：

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  int n,m,a[100010][3],mx;
4  int sum[300][300][300];
5  int ct[3];
6  int rs(int x,int y,int z,int len){
7      int rt=sum[x][y][z];
8      rt-=(sum[x-len-1][y][z]+sum[x][y-len-1][z]+sum[x][y][z-len-1]);
9      rt+=(sum[x-len-1][y-len-1][z]+sum[x-len-1][y][z-len-1]+sum[x][y-len-1][z-len-1]);
10     rt-=sum[x-len-1][y-len-1][z-len-1];
11     return rt;
12 }
13 bool check(int l){
14     for(int i=l+1;i<=mx;i++){
15         for(int j=l+1;j<=mx;j++){
16             for(int k=l+1;k<=mx;k++){
17                 if(rs(i,j,k,l)>=m){
18                     ct[0]=i;
19                     ct[1]=j;
20                     ct[2]=k;
21                     return 1;
22                 }
23             }
24         }
25     }
26     return 0;
27 }
28 int main(){
29     freopen("rgb.in", "r", stdin);
30     freopen("rgb.out", "w", stdout);
31     scanf("%d%d",&n,&m);
32     for(int i=1;i<=n;i++){
33         scanf("%d%d%d",&a[i][0],&a[i][1],&a[i][2]);
34         a[i][0]++;//为了避免边界情况讨论，整体增加一
35         a[i][1]++;
36         a[i][2]++;
37         mx=max(mx,a[i][0]);
38         mx=max(mx,a[i][1]);
39         mx=max(mx,a[i][2]);
40         sum[a[i][0]][a[i][1]][a[i][2]]++;
41     }
42     for(int i=1;i<=mx;i++)for(int j=1;j<=mx;j++)for(int k=1;k<=mx;k++)sum[i][j][k]+=sum[i][j][k-1];
43     for(int i=1;i<=mx;i++)for(int k=1;k<=mx;k++)for(int j=1;j<=mx;j++)sum[i][j][k]+=sum[i][j-1][k];
44     for(int i=1;i<=mx;i++)for(int j=1;j<=mx;j++)for(int k=1;k<=mx;k++)sum[i][j][k]+=sum[i-1][j][k];
45     int l=1,r=mx,mid,ans;
46     while(l<=r){
47         mid=(l+r)/2;
48         if(check(mid)){
49             ans=mid;

```

```
50         r=mid-1;
51     }
52     else l=mid+1;
53 }
54 printf("%d\n",ans);
55 return 0;
56 }
```

考察知识点

前缀和，二分答案，容斥

T3 洪水拯救计划 (rescue)

50%

要求算出以每个关键点为根的答案，可以枚举根，然后进行一次树形dp计算每次进子树回来与否的代价即可

100%

注意到枚举每个关键点为根时不会错过最优答案，不难发现最优的走法就是dfs，而最优的dfs应该在最深的被拯救的点（以下称为标记点）结束。给出一个简单的证明：因为如果最后要回到原点，那么只要是dfs，不同的走法走过的路程一样，而在最深的点结束可以省下最多回程的路程。因此只要求出这个点走到其他关键点最远的距离是多少，减去最远距离即可

详细题解补充：

Part 0：引入

这道题目有一个很明显的特征：要求算出以每个点为根的答案，而且显然不能枚举，我们可以用运用了一种策略：根的翻转递推（其实是运用了树形dp的思想），也称为换根DP。

不妨回顾一下那道题目，我们通过dfs枚举根，并且把一个点为根的情况通过它的父亲递推得到，其实，这是一种树形dp的策略，只是把dp过程融入到dfs求解当中。这道题是否可以采用类似的策略呢？不妨按照那道题目的步骤一步一步来。

以下，是这道题的树形dp大方向下，从答案结构框架到某部分的计算实现的过程。

Part 1：状态转移结论的发现

如果要用这种方法解决问题，包括所有的树形dp问题，我们都必须要知道当根从u转移到u的儿子v时，答案是怎么变化的。

观察这道题目里的答案，不难发现最优的走法就是dfs，而最优的dfs应该在最深的被拯救的点（以下称为标记点）结束。给出一个简单的证明：因为如果最后要回到原点，那么只要是dfs，不同的走法走过的路程一样，而在最深的点结束可以省下最多回程的路程。而从这个证明中，我们实际上已经把答案拆分成两部分：回到原点走过的路程和，减去最深（在根为基地的意义下，下同）的标记点到根的距离，即深度。

Part 2：计算路程和

通过树形dp，最深的点的深度可以树形dp出来（下面再讲），而回到原点走过的路程len和是可以 $O(1)$ 递推的。我们知道，一条边被经过，当且仅当（当根为基地时）它的子树中有至少一个标记点。所以，如果根从u转移到它的儿子v，那么边u-v分3种情况讨论（以下情况中，根为1）：

1. v的子树中没有标记点，那么这条边本来不用走，现在要走了，所以len加上这条边长的两倍。
2. v的子树中有所有标记点，即在v的子树外没有标记点，那么这条边本来要走，现在不用走，len减去这条边长的两倍。
3. 其他情况，len不变。

除此之外，没有其它遗漏。我们解决了len的计算问题。

Part 3：树形dp求最深点深度

那么，本题的难点就只有一个，就是求最深点深度了。这是一个经典问题，而且我猜应该有不少解决的方法。这里提供我的一种树形dp思路。

在遇到这种求一个点和其它所有点关系的时候，有一种常见的套路，就是把这个问题拆分为（以某个固定点，如1为根的）子树内+子树外两种，先处理子树内，再用父亲的子树内和子树外算出儿子的子树外。这道题，子树内的最深点深度很好计算，一次dfs收集上来最深的深度就可以，关键是求这个点到子树外标记点的最远距离。回到那个套路中，这里我们可以从它的父亲节点的信息来推。

考察节点u，有c个儿子 $v_1, v_2, \dots, v_c$ 。那么，对于儿子 v_i ，它子树外的最远距离有两种来源：来自u的其他儿子的子树内和u的子树外。u的子树外已经算出来了，只要加上边u-v，而u的其他儿子的子树内信息也已经算好（因为先算子树内），只要一遍扫过来，记录最大值、次大值（如果最大值取在 v_i 上，要用次大值），然后加上边u-v。这样，两部分就都能够求解了。

另外注意初始值：对于子树内，如果这个点被标记取0，否则取-inf；对于子树外，如果父亲被标记取边u-v，否则取-inf。

Part 4：代码组织与实现

这个部分，直接上代码。

```
1  #include<cstdio>
2  #include<algorithm>
3  using namespace std;
4  const int M=5e5+5;
5  void gread(int &r) { //Io数据很大啊，用读入挂可能会好一点
6      r=0;
7      char c=0;
8      while(c<'0' || c>'9') c=getchar();
9      while(c>='0' && c<='9') r=(r<<1)+(r<<3)+(c^48), c=getchar();
10 }
11 int n,m;
12 struct edge {
13     int v,c,to;
14 } E[M<<1];
15 int head[M],tot;
16 void Link(int u,int v,int c) {
17     E[++tot]=(edge) {
18         v,c,head[u]
```

```

19     };
20     head[u]=tot;
21 }
22 bool mark[M];
23 long long dis[M],len,sz[M];
24 long long from_f[M],from_s[M];
25 void dfs_Init(int u,int f) { //这一次dfs预处理很多东西
26     if(mark[u])sz[u]=1,from_s[u]=0;
27     else sz[u]=0,from_s[u]=-2e18;
28     for(int i=head[u]; i!=0; i=E[i].to) {
29         int v=E[i].v,c=E[i].c;
30         if(v==f)continue;
31         dis[v]=dis[u]+c; //深度
32         dfs_Init(v,u);
33         sz[u]+=sz[v]; //子树内标记点数量
34         if(sz[v]>0)len+=c*2; //len在根为1时的初始值
35         from_s[u]=max(from_s[u],from_s[v]+c); //子树内深度
36     }
37 }
38 void re_dfs_Init(int u,int f) {
39     long long fir=-2e18,sec=-2e18; //最大值First, 次大值Second
40     for(int i=head[u]; i!=0; i=E[i].to) {
41         int v=E[i].v,c=E[i].c;
42         if(v==f)continue;
43         long long Dis=from_s[v]+c;
44         if(Dis>fir)sec=fir,fir=Dis;
45         else if(Dis>sec)sec=Dis;
46     }
47     for(int i=head[u]; i!=0; i=E[i].to) {
48         int v=E[i].v,c=E[i].c;
49         if(v==f)continue;
50         long long Dis=from_s[v]+c;
51         if(Dis==fir)from_f[v]=max(from_f[u],sec)+c; //最大值以外的最大值是次大值
52         else from_f[v]=max(from_f[u],fir)+c;
53         if(mark[u])from_f[v]=max(from_f[v],(long long)c); //这里注意父亲被标记的情况 (实
测, 比较难拍出来)
54         re_dfs_Init(v,u);
55     }
56 }
57 long long Ans[M];
58 void dfs_Solve(int u,int f,long long len) {
59     Ans[u]=len-max(from_f[u],from_s[u]);
60     for(int i=head[u]; i!=0; i=E[i].to) {
61         int v=E[i].v,c=E[i].c;
62         if(v==f)continue;
63         long long nxt_len=len;
64         if(sz[v]==0)nxt_len+=c*2; //这里是len的O(1)递推转移
65         else if(sz[v]==m)nxt_len-=c*2;
66         dfs_Solve(v,u,nxt_len);
67     }
68 }
69 void Solve() {

```

```

70     dfs_Init(1,-1);
71     if(!mark[1])from_f[1]=-2e18;
72     re_dfs_Init(1,-1);
73     dfs_Solve(1,-1,len);
74     for(int i=1; i<=n; i++)printf("%lld\n",Ans[i]);
75 }
76 int main() {
77     freopen("rescue.in", "r", stdin);
78     freopen("rescue.out", "w", stdout);
79     scanf("%d%d",&n,&m);
80     for(int i=1; i<n; i++) {
81         int u,v,c;
82         qread(u),qread(v),qread(c);
83         Link(u,v,c);
84         Link(v,u,c);
85     }
86     for(int i=1; i<=m; i++) {
87         int u;
88         qread(u);
89         mark[u]=true;
90     }
91     Solve();
92     return 0;
93 }

```

考察知识点

换根DP，树的直径

T4 最短路 (path)

题目大意

给定一张有 n 个点， m 条边的无向图，对于每个 $i \in (1, n]$ ，请求出在不过原来1点到 i 节点最短路最后一条边的前提下的新最短路。

特别地，保证1到任何一个点的最短路都是唯一的。

20%

注意到性质 特别地，保证1到任何一个点的最短路都是唯一的

此性质与树中：任意两点间路径唯一相似

因此我们可以先跑一次最短路找出每个点最短路最后一条边，对于每个点手动 ban 掉它，再跑 Dijkstra 算最短路，时间复杂度 $O(nm \log m)$ 。

100%

所以考虑建立**最短路树**，以1为根节点。（补充，若最短路不唯一则建立的是最短路图）

如果把点 i 到它父亲的边断了怎么办？为了到达，我们需要一条边连接 i 的子树内任意一节点，和子树外任意一节点。

这样思考，似乎很难处理。我们不妨转换角度：

对于任意一条非树边，它能给哪些点做贡献呢？

显然，要求 u, v 在不同子树内。于是，这个子树的根必须在 u, v 的路径上（不含 LCA）。

那，对路径上的这些点的贡献是多少呢？

我们思考，假设当前节点为 k ，本来它应该是 $1 \rightarrow k$ 的最短路，现在变成了 $1 \rightarrow u \rightarrow v \rightarrow k$ 了。

树上问题，我们常用的手段是把路径转化为到的路径。

于是， $1 \rightarrow u \rightarrow v \rightarrow k = (u \rightarrow v) + (u \rightarrow 1) + (v \rightarrow 1) - (k \rightarrow 1)$ 。其中 $u \rightarrow v$ ，是当前枚举的非树边边权。

这里推荐画图辅助理解。

我们发现， $(k \rightarrow 1)$ 是定值，而 $(u \rightarrow v), (u \rightarrow 1), (v \rightarrow 1)$ 均只和当前的非树边 (u, v) 有关。因此，我们可以对于每条非树边算出刚才的式子的值。

我们所求转化为求出删去一个点与他父亲的边，通过非树边走到这个点的新路径长度，每一条非树边会在原树上形成一个环，对于环上的每一个点 i ，可以选取的答案为 $W_{\text{环}} - dis_i$ ，依次处理非树边并更新环上所有的答案即可

于是，一个节点最终答案的大小与更新它的边的最小值有直接关系。

Solution 1

接下来，答案就很显然了：我们树链剖分，然后线段树维护区间最小值即可。时间复杂度 $O(n \log^2 n)$ 。

Solution 2

Solution 1 十分的粗鲁：直接用数据结构解决了，但是双log。能不能用更简单的方法呢？

我们发现，一个点被一条边权最小的边更新之后，就不会再被更新了。于是考虑对边按照其对应值进行从小到大排序，每条边暴力找路径，更新答案，但是每次更新完一个点，就把他删掉。删掉？就是下次访问它，直接跳到它的最近没被删的祖先。这让我们想到并查集路径压缩。于是，这道题被优雅解决了，时间复杂度 $O(n)$ 左右。

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  inline int read(){
4      int x=0,w=1;
5      char ch=0;
6      while(ch<'0' || ch>'9'){if(ch=='-')w=-1;ch=getchar();}
7      while(ch>='0' && ch<='9'){x=x*10+(ch-'0');ch=getchar();}
8      return x*w;
9  }
10 const int MAXN=2e5;
11 int n,m;
12 int dis[MAXN+10],vis[MAXN+10];
13 int fa[MAXN+10],dep[MAXN+10];
14 struct A{
15     int to,w;
```

```

16 };
17 vector<A> g[MAXN];
18 void dij(int s){
19     priority_queue<pair<int,int>,vector<pair<int,int> >,greater<pair<int,int> > > q;
20     q.push(make_pair(0,s));
21     memset(dis,0x3f3f3f3f,sizeof(dis));
22     dis[s]=0;dep[s]=0;
23     while(!q.empty()){
24         pair<int,int> ex=q.top();q.pop();
25         if(vis[ex.second])continue;
26         vis[ex.second]=1;
27         for(int i=0;i<g[ex.second].size();i++){
28             int v=g[ex.second][i].to,w=g[ex.second][i].w;
29             if(!vis[v]&&dis[v]>dis[ex.second]+w){
30                 dis[v]=dis[ex.second]+w;
31                 fa[v]=ex.second;
32                 dep[v]=dep[ex.second]+1;
33                 q.push(make_pair(dis[v],v));
34             }
35         }
36     }
37 }
38 int ans[MAXN+10];
39 void work(int x,int y,int w){
40     while(x!=y){
41         if(dep[x]<dep[y]){
42             swap(x,y);
43         }
44         ans[x]=min(ans[x],w-dis[x]);
45         x=fa[x];
46     }
47 }
48 int main(){
49     freopen("path.in", "r", stdin);
50     freopen("path.out", "w", stdout);
51     n=read();m=read();
52     for(int i=1;i<=m;i++){
53         int a=read(),b=read(),c=read();
54         g[a].push_back(A{b,c});
55         g[b].push_back(A{a,c});
56     }
57     dij(1);
58     memset(ans,0x3f,sizeof(ans));
59     for(int i=1;i<=n;i++){
60         for(int j=0;j<g[i].size();j++){
61             int v=g[i][j].to,w=g[i][j].w;
62             if(dis[v]!=dis[i]+w&&dis[i]!=dis[v]+w){
63                 work(i,v,dis[v]+dis[i]+w);
64             }
65         }
66     }
67     for(int i=2;i<=n;i++){

```

```
68         if(ans[i]==0x3f3f3f3f)printf("-1\n");
69         else printf("%d\n",ans[i]);
70     }
71     return 0;
72 }
```

考察知识点

最短路，树链剖分，并查集，贡献