A.B.D. 签到题

C.

暴力 O(8!)枚举排列，判断是否符合条件

或者贪心， 从字典序小的开始枚举，是否能作为下一个奶牛，分类讨论即可

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;typedef double db;

map<string,int>mp;
map<int,string>rev;
vector<int>to[9];
bool vis[9];
void work()
{
    mp["Beatrice"]=0;mp["Sue"] = 0;mp["Belinda"] = 0;mp["Bessie"] = 0;
    mp["Betsy"] = 0;mp["Bella"] = 0;mp["Blue"] = 0;mp["Buttercup"] = 0;
    int cnt=0;
    for(auto &x:mp)x.second=++cnt,rev[x.second]=x.first;
    int n;cin>>n;
    for(int i=1;i<=n;i++)
    {
        string A,B;cin>>A;cin>>B;
        cin>>B;cin>>B;cin>>B;cin>>B;
        to[mp[A]].push_back(mp[B]);
        to[mp[B]].push_back(mp[A]);
    }
    for(int i=1;i<=8;i++)
    {
        if(vis[i])continue;
        if(to[i].size()==0)
        {
            cout<<rev[i]<<'\n';
            vis[i]=1;
        }
        else if(to[i].size()==1)
        {
            cout<<rev[i]<<'\n';vis[i]=1;
            int nxt=to[i][0],pre=i;
            while(1)
            {
                cout<<rev[nxt]<<'\n';
                vis[nxt]=1;
                if(to[nxt].size()==1)break;
                else
                {
                    int tmp=nxt;
                    nxt=to[nxt][0]+to[nxt][1]-pre;
                    pre=tmp;
                }
```

```
            }
        }
    }

}
int main(){
        work();
}
```

--ztc

E.

> 球<=>牛 QAQ

问题可以分成两部分，首先要二分求出题目中的T，如果两只牛没有差别，就可以假设两只牛相遇后不会相撞，而是会穿过去，这样可以轻松的计算某一时刻掉下了几个球以及相撞的次数。但由于每只球的重量是不同的，于是还得知道哪几个球掉下去了。

一个小结论就是若某一时刻，给小球从左到右标号$[1,n]$，那么一段时间后，还剩在桌子上的球的顺序还是不变，并且记左边掉了 l 个球，右边掉了 r 个球，那么剩下的球就是$[l+1,n-r]$，由此可以得到某一时刻掉下的球的重量，然后T也得到了。

接下来要求T时刻内发生了几次碰撞。记第i个向右的球在坐标$posr_i$，第j个向做的球在$posl_j$,那么只有满足以下两个条件两个球才会碰撞：

$$
\begin{aligned}
(1).\quad posr_i < posl_j \quad &,\text{否则永远不会碰撞} \\
(2).\quad posl_j \le posr_i + 2T \quad &,\text{否则还没发生碰撞}
\end{aligned}
$$

这个二分一下边界即可

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll; typedef double db;

#define Fi first
#define Se second
#define _Out(a) cerr<<#a<<" = "<<(a)<<endl
const int INF = 0x3f3f3f3f, MAXN = 1e5 + 50;
const ll LINF = 0x3f3f3f3f3f3f3f3f, MOD = 1004535809;
const db Pi = acos(-1), EPS = 1e-6;
struct COW
{
    ll pos, w, v, id;
    bool operator<(const COW& R)const
    {
        return pos < R.pos;
    }
}cow[MAXN], cow1[MAXN], cow2[MAXN];
int n, L, totw;
bool check(ll x)
{
    int l = 0, r = 0;
```

```
    for (int i = 1; i <= n; i++)
    {
        if (cow[i].v == 1)
        {
            if (L - cow[i].pos <= x)r++;
        }
        else
        {
            if (cow[i].pos <= x)l++;
        }
    }
    int nw = 0;
    for (int i = 1; i <= l; i++)
    {
        nw += cow[i].w;
    }
    for (int i = n - r + 1; i <= n; i++)
    {
        nw += cow[i].w;
    }
    if (nw >= (totw + 1) / 2)return 1;
    else return 0;
}
void work()
{
    scanf("%d%d", &n, &L); int c1 = 0, c2 = 0;
    for (int i = 1; i <= n; i++)
    {
        int w, x, v;
        scanf("%d%d%d", &w, &x, &v);
        cow[i] = { x,w,v,i }; totw += w;
        if (v == 1)cow1[++c1] = cow[i];
        else cow2[++c2] = cow[i];
    }
    sort(cow + 1, cow + 1 + n);
    sort(cow1 + 1, cow1 + 1 + c1);
    sort(cow2 + 1, cow2 + 1 + c2);
    ll l = 1, r = L;
    while (l < r){
        ll mid = l + r >> 1;
        if (check(mid)) r = mid;
        else l = mid + 1;
    }
    ll ans = 0;
    for (int i = 1; i <= c1; i++)
    {
        COW tmp1 = { cow1[i].pos,0,0,0 };
        COW tmp2 = { cow1[i].pos + 2ll * l,0,0,0 };
        int pos1 = upper_bound(cow2 + 1, cow2 + 1 + c2, tmp1) - cow2;
        int pos2 = upper_bound(cow2 + 1, cow2 + 1 + c2, tmp2) - cow2;
        ans += pos2 - pos1;
    }
    printf("%lld\n", ans);
}
int main(){
    work();
}
```

--ztc

F.

答案的可能范围很小，可以枚举答案，即假设当前答案是F,那么只要求最小的花费C就行了,那么所有流量大于等于F的边都可以用，用这些边跑一个最短路，就得到最小花费的C了，这样真实的F只会比假设的更大，枚举完所有可能的F后最小的那个比值就是答案了.

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;typedef double db;
typedef pair<int, int> pii;typedef pair<ll, ll> pll;
typedef pair<int,ll> pil;typedef pair<ll,int> pli;
#define Fi first
#define Se second
#define _Out(a) cerr<<#a<<" = "<<(a)<<endl
const int INF = 0x3f3f3f3f, MAXN = 2e3 + 50;
const ll LINF = 0x3f3f3f3f3f3f3f3f, MOD = 1004535809;
const db Pi = acos(-1), EPS = 1e-6;

struct Edge
{
    int from,to,cost,nxt;

}E[MAXN*2];
int head[MAXN],cntE;
void addE(int a,int b,int c)
{
    E[cntE]={a,b,c,head[a]};
    head[a]=cntE++;
}
int dist[MAXN],n;
struct PNode
{
    int dis,id;
    bool operator <(const PNode &R)const
    {
        return dis>R.dis;
    }
};
int dij()
{
    memset(dist,0x3f,sizeof(dist));
    priority_queue<PNode>pq;
    pq.push({0,1});dist[1]=0;
    while(!pq.empty())
    {
        PNode tmp=pq.top();pq.pop();
        if(tmp.dis>dist[tmp.id])continue;
        for(int ei=head[tmp.id];ei!=-1;ei=E[ei].nxt)
        {
            int v=E[ei].to;
            if(dist[v]>dist[tmp.id]+E[ei].cost)
            {
                dist[v]= dist[tmp.id] + E[ei].cost;
                pq.push({dist[v],v});
```

```
            }
        }
    }
    return dist[n];
}
struct EQAQ
{
    int a,b,c,f;
    bool operator<(const EQAQ &R)const
    {
        return f>R.f;
    }
}eQAQ[MAXN];
void work()
{
    int m;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)head[i]=-1;
    for(int i=1;i<=m;i++)
    {
        scanf("%d%d%d%d",&eQAQ[i].a,&eQAQ[i].b,&eQAQ[i].c,&eQAQ[i].f);
    }
    sort(eQAQ+1,eQAQ+m+1);
    ll ans=0;
    for(int i=1;i<=m;i++)
    {
        addE(eQAQ[i].a, eQAQ[i].b, eQAQ[i].c);
        addE(eQAQ[i].b, eQAQ[i].a, eQAQ[i].c);
        int d=dij();
        if(d==INF)continue;
        ll F=eQAQ[i].f*1000000ll;
        ans=max(ans,(ll)(F/((db)d)));
    }
    printf("%lld\n",ans);
}
int main(){
        work();
}
```

--ztc

G.

这个问题就是把树上路径的询问转化成，某种更好求的子问题。

询问可以拆成：

- lca在某一端点，当作两个查询。
- lca不在端点，当作三个查询。

我的做法就是维护根到某个结点的各种类型统计情况，然后dfs一遍。

dfs模拟一下想下面这样：

```
有如下的一棵树：
1-2-3-4
    ┣3-4
```

```
    └1-2-1
有如下的路径：
    ┌3-4
    └1
设一个全局数组cnt[i]表示到根节点某个数字出现次数。
开始dfs：
*-2-3-4
    ├3-4
    └1-2-1
cnt[] = {0, 1, 0, 0, 0};
-----
1-*-3-4
    ├3-4
    └1-2-1
cnt[] = {0, 1, 1, 0, 0};
-----
1-2-*-4
    ├3-4
    └1-2-1
cnt[] = {0, 1, 1, 1, 0};
-----
1-2-3-*
    ├3-4
    └1-2-1
cnt[] = {0, 1, 1, 1, 1};
-----
// 开始回退，走向另一个分支。
1-2-3-4
    ├*-4
    └1-2-1
cnt[] = {0, 1, 1, 0, 0};
-----
// 以此类推
```

```
#include <bits/stdc++.h>
#define STOPSYNC ios::sync_with_stdio(false);cin.tie(nullptr)
#define MULTIKASE int Kase=0;cin>>Kase;for(int kase=1;kase<=Kase;kase++)
typedef long long ll;
const int MAXN = 2e5 + 59;
const int MOD = 1e9 + 7;
const int INF = 0x3F3F3F3F;
const ll llINF = 0x3F3F3F3F3F3F3F3F;
using namespace std;
using pii = pair<int, int>;
using vint = vector<int>;

namespace LCA {
    int rmq[MAXN << 1];

    struct ST {
        int mm[MAXN << 1];
        int dp[MAXN << 1][20];

        void init(int n) {
            mm[0] = -1;
```

```
        for (int i = 1; i <= n; ++i) {
            mm[i] = mm[i >> 1] + 1;
            dp[i][0] = i;
        }
        for (int j = 1; j <= mm[n]; ++j) {
            for (int i = 1; i + (1 << j) - 1 <= n; ++i) {
                dp[i][j] =
                        rmq[dp[i][j - 1]] <
                        rmq[dp[i + (1 << (j - 1))][j - 1]] ?
                        dp[i][j - 1] :
                        dp[i + (1 << (j - 1))][j - 1];
            }
        }
    }

    int query(int a, int b) {
        if (a > b)swap(a, b);
        int k = mm[b - a + 1];
        return rmq[dp[a][k]] <=
               rmq[dp[b - (1 << k) + 1][k]] ?
               dp[a][k] :
               dp[b - (1 << k) + 1][k];
    }
};

struct Edge {
    int to, nx;
    ll w;
};
Edge edge[MAXN << 1];
int totEd, head[MAXN];

int F[MAXN << 1];
int P[MAXN];
int cntStp;

ST st;

void init() {
    totEd = 0;
    memset(head, -1, sizeof(head));
}

void addedge(int u, int v, int w) {
    edge[totEd] = {v, head[u], w};
    head[u] = totEd++;
    edge[totEd] = {u, head[v], w};
    head[v] = totEd++;
}

int CrcA, CrcB, CrcW;
bool vis[MAXN];
ll dis[MAXN];

void dfs(int u, int fa, int dep) {
    F[++cntStp] = u;
    rmq[cntStp] = dep;
    P[u] = cntStp;
```

```
        vis[u] = true;
        for (int v = 0, i = head[u]; ~i; i = edge[i].nx) {
            v = edge[i].to;
            if (vis[v]) {
                if (v != fa) {
                    CrcA = u;
                    CrcB = v;
                    CrcW = edge[i].w;
                }
                continue;
            }
//          _debug(v);
            dis[v] = dis[u] + edge[i].w;
            dfs(v, u, dep + 1);
            F[++cntStp] = u;
            rmq[cntStp] = dep;
        }
    }
    void LCA_init(int root, int node_num) {
        cntStp = 0;
        memset(vis, 0, sizeof vis);
        memset(dis, 0, sizeof dis);
        dfs(root, -1, 0);
        st.init(2 * node_num - 1);
    }
    inline int LCA_query(int u, int v) {
        return F[st.query(P[u], P[v])];
    }
}

int node_type[MAXN];
struct Query {
    int val;
    int query_type;     // 1.simple path, 2.split path
    int target_type;
    int result;
};

struct SplitQuery {
    int qid;
    bool is_end;
};

vector<Query> qry;
vector<SplitQuery> g[MAXN];
int count_ci[MAXN];

void dfs(int u, int f) {

    count_ci[node_type[u]]++;

    while (!g[u].empty()) {
        int qid = g[u].back().qid;
        bool is_end = g[u].back().is_end;
        g[u].pop_back();

        int qtype = qry[qid].query_type;
        int target = qry[qid].target_type;
```

```
        if (qtype == 1) {
            if (is_end) {
                qry[qid].val += count_ci[target];
            } else { // is_begin
                qry[qid].val -= count_ci[target] - int(node_type[u] == target);
            }
        } else if (qtype == 2) {
            if (is_end) {
                qry[qid].val += count_ci[target];
            } else { // is_begin
                qry[qid].val -= count_ci[target];
                qry[qid].val -= count_ci[target];
                qry[qid].val += int(node_type[u] == target);
            }
        } /*else {
            debug(u);
            exit(0);
        }*/
    }

    for (int v = 0, i = LCA::head[u]; i != -1; i = LCA::edge[i].nx) {
        v = LCA::edge[i].to;
        if (v == f) continue;
        dfs(v, u);
    }

    count_ci[node_type[u]]--;
}


void solve(int kaseId = -1) {
    int n, m;
    cin >> n >> m;

    LCA::init();

    for (int i = 1; i <= n; ++i) {
        cin >> node_type[i];
    }

    for (int i = 1; i < n; ++i) {
        int u, v;
        cin >> u >> v;
        LCA::addedge(u, v, 1);
    }

    LCA::LCA_init(1, n);

//    debug(LCA::LCA_query(1, 2));
//    debug(LCA::LCA_query(4, 6));
//    debug(LCA::LCA_query(3, 6));
//    debug(LCA::LCA_query(5, 3));
//    debug(LCA::dis[1], LCA::dis[2], LCA::dis[6]);

    for (int i = 0; i < m; ++i) {
        int a, b, c;
        cin >> a >> b >> c;
```

```cpp
        int lca = LCA::LCA_query(a, b);
        if (lca != a && lca != b) {
            qry.push_back({0, 2, c, -1});
            g[a].push_back({int(qry.size()) - 1, true});
            g[b].push_back({int(qry.size()) - 1, true});
            g[lca].push_back({int(qry.size()) - 1, false});
        } else {
            qry.push_back({0, 1, c, -1});
            if (LCA::dis[a] > LCA::dis[b]) swap(a, b);
            g[a].push_back({int(qry.size()) - 1, false});
            g[b].push_back({int(qry.size()) - 1, true});
        }
    }

    dfs(1, -1);

    string ans(m, '0');
    for (int i = 0; i < qry.size(); ++i) {
        if (qry[i].val > 0)
            ans[i] = '1';
    }

    cout << ans << endl;
}

void solves() {
    MULTIKASE {
        solve(kase);
    }
}

int main() {
#ifdef DEBUG
    freopen("input.txt", "r+", stdin);
#endif
    STOPSYNC;
    solve();
    return 0;
}
/*

 */
```

--wtw

H.

题意

给出一个字符串,使它通过一系列操作变成合法串,求最小花费

将某个字符从a变为b需要花费$w[a][b]$的代价

如果一个串存在一种划分,使得每一段的字符全部相同并且每一段的长度都不小于K,那么这就是一个合法串

反之,若不存在该种划分,即为非法串

思路

元素的限制具有连续性,考虑对每一个前缀进行分析

令$dp[i]$为将第i个前缀$prefix[i]$变为合法串的最小代价

考虑转移过程

每一段的字符相同且长度不小于K,那么转移十分明显,为:

$$dp[i] = min_0^{i-K}(dp[j] + cost(j+1..i,ch));$$

其中$cost(j+1..i,ch)$为将从$j+1$个字符到第i个字符全部替换为ch的最小代价

时间复杂度为$O(26n^2)$,显然会超时

考虑优化,对dp方程进行分析

发现$cost(j+1..i,ch) = cost(1..i,ch) - cost(1..j,ch)$

那么就有

$dp[i] = min_0^{i-k}(dp[j] + cost(1..i,ch) - cost(1..j,ch)) = min_0^{i-k}(dp[j] - cost(1..j,ch)) + cost(1..i,ch)$

发现$cost(1..i,ch)$与j无关,那么等式右边的大小只与决策j有关,将常数$cost(1..i,ch)$移至左侧得

$$f(i) = dp[i] - cost(1..i,ch) = min_0^{i-k}(dp[j] - cost(1..j,ch)) = min_0^{i-k}f(j)$$

那么对不同的ch维护一个可以快速得到合法且最小的$f(j)$的数据结构即可

~~建议用$deque$,当然我只是建议~~

然后只要用前缀和什么的预处理一下$cost(1..i,ch)$就可以了

时空复杂度都是$O(26n)$,非常优秀

以下代码时间复杂度是$O(26^2n)$，请读者自行优化

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;
const ll inf = 1LL<<60;
const int MOD = 998244353;
const int N = 100005;
ll power(const ll & x,const ll & mi)
{
    ll s1=1LL,s2=x,m=mi;
    while (m)
    {
        if (m&1) s1=s1*s2%MOD;
        s2=s2*s2%MOD;
        m>>=1;
    }
    return s1;
}
struct Node_
{
    int x;ll dis;
};
```

```
char st[N];
int a[N];
ll dp[N],id[N];
ll s[N][50];
ll co[N][50];
int c[50][50];
int n,m,K;
deque <Node_> q[50];
void solve()
{
    ll ans=-inf;
    cin>>n>>m>>K;
    scanf("%s",st);
    for (int i=1;i<=n;++i)
    {
        dp[i]=inf;
        a[i]=st[i-1]-'a'+1;
        for (int j=1;j<=m;++j) s[i][j]=s[i-1][j];
        ++s[i][a[i]];
    }
    for (int i=1;i<=m;++i)
    {
        for (int j=1;j<=m;++j) scanf("%d",&c[i][j]);
    }
    for (int k=1;k<=m;++k)
        for (int i=1;i<=m;++i)
        for (int j=1;j<=m;++j)
        c[i][j]=min(c[i][j],c[i][k]+c[k][j]);
    for (int i=1;i<=n;++i)
    {
        for (int j=1;j<=m;++j)
        {
            for (int k=1;k<=m;++k) co[i][j]+=1LL*c[k][j]*s[i][k];
        }
    }
    for (int i=1;i<=m;++i)
    {
        q[i].push_back((Node_){0,0});
    }
    for (int i=K;i<=n;++i)
    {
        for (int ch=1;ch<=m;++ch)
        {
            while (!q[ch].empty())
            {
                Node_ t=q[ch].front();
                q[ch].pop_front();
                if (q[ch].empty() || (q[ch].front().x)+K>i)
                {
                    q[ch].push_front(t);
                    break;
                }
            }
        }
        for (int ch=1;ch<=m;++ch)
        {
            dp[i]=min(dp[i],q[ch].front().dis+co[i][ch]);
        }
```

```
        for (int ch=1;ch<=m;++ch)
        {
            if (q[ch].back().dis>dp[i]-co[i][ch])
            {
                q[ch].push_back((Node_){i,dp[i]-co[i][ch]});
            }
        }
    }
    cout<<dp[n]<<endl;
    return;
}
int main()
{
    int T=1;
    while (T--) solve();
    return 0;
}
```

--yzh

方法二

记$dp[i][j]$为前i个字符已经合法了并且最后一个字符是j时最小的代价，那么可以写出转移方程：

$$dp[i+1][j]=\min(dp[i+1][j],dp[i][j]+w[str[i]][j])$$
$$dp[i+k][j]=\min(dp[i+k][j],\min_{l=1}^{m}(dp[i][l])+\{\text{将子串}str[i+1,\ldots,i+k]\text{都转化为}j\text{的代价}\})$$

其中花括号内的可以预处理做到O(1) 查询，$\min_{l=1}^{m}(dp[i][l])$ 也可以边dp边预处理

时空复杂度为$O(26n)$

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll; typedef double db;
typedef pair<int, int> pii; typedef pair<ll, ll> pll;
#define Fi first
#define Se second
#define _Out(a) cerr<<#a<<" = "<<(a)<<endl
const int INF = 0x3f3f3f3f, MAXN = 1e5 + 50;
const ll LINF = 0x3f3f3f3f3f3f3f3f, MOD = 1004535809;
const db Pi = acos(-1), EPS = 1e-6;
char str[MAXN];
int dp[MAXN][28],QAQ[MAXN];
int w[33][33];
int sum[MAXN][28];
void work()
{
    int n, m, k;
    scanf("%d%d%d%s", &n, &m, &k, str + 1);
    memset(dp, 0x3f, sizeof(dp));
    memset(w, 0x3f, sizeof w);
    for(int i=0;i<m;i++)for(int j=0;j<m;j++)scanf("%d",&w[i][j]);
    for(int j=0;j<m;j++)for(int i=0;i<m;i++)for(int k=0;k<m;k++)
            w[i][k] = min(w[i][k], w[i][j] + w[j][k]);
    for(int i=1;i<=n;i++)for(int j=0;j<m;j++)sum[i][j]=sum[i-1][j]+w[str[i]-'a']
[j];
    memset(QAQ,0x3f,sizeof QAQ);
```

```
    for (int i = 0; i < m; i++){
        dp[k][i] = 0;
        for (int pos = 1;pos <= k;pos++)dp[k][i]+=w[str[pos]-'a'][i];
        QAQ[k]=min(QAQ[k],dp[k][i]);
    }
    for (int i = k + 1; i <= n; i++){
        for (int j = 0; j < m; j++){
            dp[i][j] = min(dp[i][j], dp[i - 1][j] + w[str[i] - 'a'][j]);
            QAQ[i]=min(QAQ[i],dp[i][j]);
            if (i + k - 1 > n)continue;
            int ned = sum[i + k - 1][j] - sum[i - 1][j];
            dp[i + k - 1][j] = min(dp[i + k - 1][j], QAQ[i-1] + ned);
        }
    }
    printf("%d\n", QAQ[n]);
}
int main() {
    work();
}
```

--ztc

I.

题意:

现在有一张长为n的饼（n<=300），现在有m头牛，每头牛都喜欢吃这个饼的一个区间，并且每头牛的区间都不相同。每头牛都有一个重量wi。现在你要选出一些牛，按照一个次序让他们去吃这个饼，到第i头牛的时候，他会把这个饼的li~ri部分全部吃光，如果到一头牛的时候，已经没有它喜欢吃的区间的饼了，他就会不开心，你需要避免这种情况选出一个序列，并且使得选出的牛的体重之和最大。

题解:

可以的，要检讨一下，300的数据范围为什么想不到区间dp，不过这道题就算我想到区间dp也不一定能够做出来。
我们在枚举区间断点的时候，需要同时维护断点在区间内的牛的最大重量，然后更新。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;
const ll inf = 1LL<<60;
const int MOD = 998244353;
const int N = 305;
int dp[N][N],a[N][N],f[N][N][N],ff[N][N];
int n,m;
void solve()
{
    cin>>n>>m;
    for (int i=1;i<=m;++i)
    {
        int w,x,y;
        scanf("%d%d%d",&w,&x,&y);
        a[x][y]=dp[x][y]=ff[x][y]=w;
        for (int j=x;j<=y;++j) f[x][y][j]=max(f[x][y][j],w);
    }
```

```
    for (int l=2;l<=n;++l)
    for (int i=1;i+l-1<=n;++i)
    {
        int j=i+l-1;
        for (int k=i;k<=j;++k)
        {
            f[i][j][k]=max(max(f[i+1][j][k],f[i][j-1][k]),f[i][j][k]);
            dp[i][j]=max(dp[i][j],dp[i][k-1]+dp[k+1][j]+f[i][j][k]);
        }
    }
    cout<<dp[1][n]<<endl;
}
int main()
{
    int T=1;
    while (T--) solve();
    return 0;
}
```

--yf