A - Hiking trip

模拟

```
#include <cstdio>
#include <algorithm>
#include <iostream>
#define double long double

using namespace std;
const double ESP = 1e-10;

double d, v0, v1, v2, t, time, p0, p1 = 1, p2;
bool flag, st;

int main(){
    cin>>d>>v0>>v1>>v2>>t;
    while(true){
        double cost = 0;
        if(!flag){ // ->
            cost = (p1-p2)/(v2-v1);
            if(p1+cost*v1+ESP>=d) cost = (d-p2)/v2;
            if(time+cost+ESP>=t) cost = t - time, st = true;
            time += cost;
            p0 = min(d,p0+v0*cost);
            p1 = min(d,p1+v1*cost);
            if(!st) p2 = p1;
            else p2 = p2+cost*v2;
        }else{ // <-
            cost = (p2-p0)/(v0+v2);
            if(p0+cost*v0+ESP>=d) ; // impossible
            if(time+cost+ESP>=t) cost = t - time, st = true;
            time += cost;
            p0 = min(d,p0+v0*cost);
            p1 = min(d,p1+v1*cost);
            p2 = p2-cost*v2;
        }
        // cout<<p0<<" "<<p1<<" "<<p2<<" "<<cost<<endl;
        if(st||p0+ESP>=d) break;
        flag ^= 1;
    }
    printf("%.12LF\n",p2);
    return 0;
}
```

B - Matematical Transformation

树链剖分

操作 0 轻重链上查询

对于操作 1 要加的值

$$v + k * d$$

$$= v + k * (depth_i - depth_u)$$

$$= v - k * depth_u + k * depth_i$$

$v - k * depth_u$是个固定值，$depth_i$对于每个节点也是固定值，需要在线段树上增加懒标记和节点的深度和。

```
#include <cstdio>
#include <algorithm>
#define size sz

using namespace std;
const int MAXN = 5e5 + 5;

int head[MAXN], tot;

struct Edge{
    int to, next;
}G[MAXN<<1];

void addEdge(int u, int v){
    G[++tot].to = v;
    G[tot].next = head[u];
    head[u] = tot;
}

struct Node{
    __int128 depth;
    __int128 val;
    __int128 add, k; // lazytag
}tree[MAXN<<2];

int n, q;
int size[MAXN], depth[MAXN], father[MAXN], son[MAXN];
int top[MAXN], id[MAXN], reflect[MAXN], cnt;

void DFS1(int np, int fat = 0){
    size[np] = 1;
    depth[np] = depth[fat] + 1;
    father[np] = fat;
    for(int i = head[np], to ; i ; i = G[i].next){
        to = G[i].to;
        if(to==fat) continue;
        DFS1(to,np);
        if(size[to]>size[son[np]]) son[np] = to;
        size[np] += size[to];
    }
}

void DFS2(int np, int topf){
    id[np] = ++cnt;
    reflect[cnt] = np;
    top[np] = topf;
    if(!son[np]) return ;
    DFS2(son[np],topf);
    for(int i = head[np], to ; i ; i = G[i].next){
        to = G[i].to;
        if(to==father[np]||to==son[np]) continue;
        DFS2(to,to);
    }
}

void print(__int128 x){
    if(x<0){
        x = -x;
        putchar('-');
    }
    if(x>9) print(x/10);
    putchar(x%10+'0');
}

void build(int node, int l, int r){
    if(l==r){
        tree[node].depth = depth[reflect[l]];
        return ;
    }
    int left_node = node<<1, right_node = node<<1|1, mid = (l+r)>>1;
    build(left_node,l,mid);
    build(right_node,mid+1,r);
    tree[node].depth = tree[left_node].depth + tree[right_node].depth;
}

void pushdown(int node, int l, int r){
    int left_node = node<<1, right_node = node<<1|1, mid = (r+l)>>1;
    if(tree[node].add){
        tree[left_node].val += (mid-l+1)*tree[node].add;
        tree[right_node].val += (r-(mid+1)+1)*tree[node].add;
        tree[left_node].add += tree[node].add;
        tree[right_node].add += tree[node].add;
        tree[node].add = 0;
    }
    if(tree[node].k){
        tree[left_node].val += tree[node].k*tree[left_node].depth;
        tree[right_node].val += tree[node].k*tree[right_node].depth;
        tree[left_node].k += tree[node].k;
        tree[right_node].k += tree[node].k;
        tree[node].k = 0;
    }
}

__int128 query(int node, int l, int r, int s, int e){
    if(r<s||l>e) return 0;
    pushdown(node,l,r);
    if(l>=s&&r<=e) return tree[node].val;
    int left_node = node<<1, right_node = node<<1|1, mid = (l+r)>>1;
    return query(left_node,l,mid,s,e)+query(right_node,mid+1,r,s,e);
}

void update(int node, int l, int r, int s, int e, __int128 depth, __int128 v,
__int128 k){
    if(r<s||l>e) return ;
    if(l>=s&&r<=e){
        tree[node].val = tree[node].val+(r-l+1)*(v-k*depth)+k*tree[node].depth;
        tree[node].k += k;
        tree[node].add += v-k*depth;
        return ;
    }
    pushdown(node,l,r);
    int left_node = node<<1, right_node = node<<1|1, mid = (l+r)>>1;
    update(left_node,l,mid,s,e,depth,v,k);
    update(right_node,mid+1,r,s,e,depth,v,k);
    tree[node].val = tree[left_node].val + tree[right_node].val;
}


__int128 qrytree(int x, int y){
    __int128 res = 0;
    while(top[x]!=top[y]){
        if(depth[top[x]]<depth[top[y]]) swap(x,y);
        res = res + query(1,1,n,id[top[x]],id[x]);
        x = father[top[x]];
    }
    res = res+query(1,1,n,min(id[x],id[y]),max(id[x],id[y]));
    return res;
}

int main(){
    scanf("%d",&n);
    for(int i = 1, u, v ; i < n ; ++i){
        scanf("%d %d",&u,&v);
        addEdge(u,v);
        addEdge(v,u);
    }
    DFS1(1);
    DFS2(1,1);
    build(1,1,n);
    scanf("%d",&q);
    for(int i = 1, opt, u, v, k ; i <= q ; ++i){
        scanf("%d",&opt);
        if(opt){
            scanf("%d %d %d",&u,&v,&k);
            update(1,1,n,id[u],id[u]+size[u]-1,depth[u],v,k);
        }else{
            scanf("%d %d",&u,&v);
            print(qrytree(u,v));
            putchar('\n');
        }
    }
    return 0;
}
```

C-Infinite Operations

思维+数据结构

最后元素都会变成$\bar{A}$，但答案并不是简单的$\sum_{i=1}^{n}|A_i\text{-}\bar{A}|$，因为在小于（或大于）$\bar{A}$的两个数之间操作$\sum_{i=1}^{n}|A_i\text{-}\bar{A}|$不会减少，但对答案产生了贡献。

如果A_i要给A_j一个值，要是 optimal operation，那么得尽量选取一个A_k来周转一次，分数比A_i直接向A_j要高。

要使得周转次数最多，那么总是要在排名相邻的两个数字之间进行操作，于是把A排序，用最后元素都会变成$\bar{A}$的结论得到每个位置对答案的贡献为

$$\text{ans}_i = (A_i + \text{ans}_{i+1}\text{-}\bar{A})$$

把答案累加得到

$$\text{ans} = (A_n\text{-}\bar{A}) + (A_n + A_{n\text{-}1}\text{-}2{*}\bar{A}) + (A_n + A_{n\text{-}1} + A_{n\text{-}2}\text{-}3{*}\bar{A}) + \ldots + (A_n + A_{n\text{-}1} + A_{n\text{-}2} + \ldots + A_1\text{-}n{*}\bar{A})$$

$$= \sum_{i=1}^{n} i * A_i - \frac{(n+1)^{*}n}{2} * \bar{A}$$

$$=\sum_{i=1}^{n} i * A_i - \frac{n+1}{2} * \sum_{i=1}^{n} A_i$$

主要是维护$\sum_{i=1}^{n} i * A_i$，用各种数据结构可维护，这里用了离散化后三个树状数组。

```
#include <cstdio>
#include <algorithm>
#define data dt

using namespace std;
const __int128 MAXN = 3e5 + 5, MOD = 998244353, INV = (MOD+1)>>1;

struct data{
    int val, type, id, rank, x;
}a[MAXN<<1];

int n, q, tree_rk[MAXN<<1];
__int128 tree_val[MAXN<<1], tree_ans[MAXN<<1], sum;

void print(__int128 x){
    if(x<0){
        x = -x;
        putchar('-');
    }
    if(x>9) print(x/10);
    putchar(x%10+'0');
}

bool cmp(data a, data b){
    return a.val < b.val;
}

bool cmp1(data a, data b){
    if(a.type==b.type) return a.rank<b.rank;
    return a.type<b.type;
}

bool cmp2(data a, data b){
    return a.id<b.id;
}
```

```
inline int lowbit(int a){
    return a&(-a);
}

void add_rk(int x, __int128 val){
    for(; x <= n + q ; x += lowbit(x)) tree_rk[x] += val;
}

__int128 ask_rk(int x){
    __int128 res = 0;
    for(; x >= 1 ; x -= lowbit(x)) res += tree_rk[x];
    return res;
}

void add_val(int x, __int128 val){
    for(; x <= n + q ; x += lowbit(x)) tree_val[x] = tree_val[x]+val;
}

__int128 ask_val(int x){
    __int128 res = 0;
    for(; x >= 1 ; x -= lowbit(x)) res += tree_val[x];
    return res;
}

void add_ans(int x, __int128 val){
    for(; x <= n + q ; x += lowbit(x)) tree_ans[x] = tree_ans[x]+val;
}

__int128 ask_ans(int x){
    __int128 res = 0;
    for(; x >= 1 ; x -= lowbit(x)) res += tree_ans[x];
    return res;
}
signed main(){
    scanf("%d %d",&n,&q);
    for(int i = 1 ; i <= n ; ++i){
        scanf("%d",&a[i].val);
        a[i].type = 0;
        a[i].id = i;
    }
    for(int i = 1 ; i <= q ; ++i){
        scanf("%d %d",&a[i+n].x,&a[i+n].val);
        a[i+n].type = 1;
        a[i+n].id = i + n;
    }
    sort(a+1,a+1+n+q,cmp);
    for(int i = 1 ; i <= n + q ; ++i) a[i].rank = i;
    sort(a+1,a+1+n+q,cmp1);
    for(int i = 1 ; i <= n ; ++i){
        add_rk(a[i].rank,1);
        add_ans(a[i].rank,(a[i].val*ask_rk(a[i].rank)));
        add_val(a[i].rank,a[i].val);
        sum += a[i].val;
    }
    sort(a+1,a+1+n+q,cmp2);

    for(int i = n + 1, x, y ; i <= n + q ; ++i){
        x = a[i].x;
        y = a[i].val;
        sum -= a[x].val;
        add_val(a[x].rank,-a[x].val);
        add_ans(a[x].rank,-a[x].val*ask_rk(a[x].rank));
        add_ans(a[x].rank,-(ask_val(n+q)-ask_val(a[x].rank)));
        add_rk(a[x].rank,-1);
        a[x].rank = a[i].rank;
        a[x].val = y;
        sum += a[i].val;
        add_rk(a[i].rank,1);
        add_ans(a[i].rank,a[i].val*ask_rk(a[i].rank));
        add_ans(a[i].rank,ask_val(n+q)-ask_val(a[i].rank));
        add_val(a[i].rank,a[i].val);
        print((ask_ans(n+q)-(n+1)*sum%MOD*INV%MOD+MOD)%MOD);
        puts("");
    }
    return 0;
}
```

D-Shortest Good Path

BFS

无权图求最短路，应该想到用 BFS 来做。

记录当前点和路径的状态(状态压缩)，初始状态为$\{i, 1 << (i-1)\}$，跑 BFS，记录最短距离。

每个点每个状态最多走一次，因此可以在$O(2^N * N)$时间完成。

```
#include <cstdio>
#include <queue>
#include <algorithm>

using namespace std;
typedef pair<int,int> p;
const int MAXN = 20, INF = 0x3f3f3f3f;

int head[MAXN], tot;

struct Edge{
    int to, next;
}G[MAXN*MAXN];

void addEdge(int u, int v){
    G[++tot].to = v;
    G[tot].next = head[u];
    head[u] = tot;
}

int n, m, dis[1<<17][MAXN], ans;
queue<p> q;

int main(){
    scanf("%d %d",&n,&m);
    for(int i = 1, u, v ; i <= m ; ++i){
        scanf("%d %d",&u,&v);
        addEdge(u,v);
        addEdge(v,u);
    }
    for(int i = 1 ; i <= n ; ++i){
        for(int j = 1 ; j < (1<<n) ; ++j) dis[j][i] = INF;
        dis[1<<(i-1)][i] = 1;
        q.push({1<<(i-1),i});
    }
    while(!q.empty()){
        int ns = q.front().first, np = q.front().second; q.pop();
        for(int i = head[np], to ; i ; i = G[i].next){
            to = G[i].to;
            if(dis[ns^(1<<(to-1))][to]!=INF) continue;
            dis[ns^(1<<(to-1))][to] = dis[ns][np] + 1;
            q.push({ns^(1<<(to-1)),to});
        }
    }
    for(int i = 0 ; i < (1<<n) ; ++i)
        ans += *min_element(dis[i]+1,dis[i]+1+n);
    printf("%d\n",ans);
    return 0;
}
```

E-Range XOR

打表异或前缀和发现

① $\mathrm{pre}_{4i} = 0$

② $\mathrm{pre}_{4i+1} = 4i$

③ $\mathrm{pre}_{4i+2} = 1$

④ $\mathrm{pre}_{4i+3} = 4i + 3$

因此问题转换成有多少对(i,j)满足$\mathrm{pre}_i \oplus \mathrm{pre}_j = V$

分类讨论$4 * 4$种情况就能得到方案数，②和④组合要数位 DP，其他情况直接计算方案即可。

令$\mathrm{work}_{a,b}$为$0 \le i < a, 0 \le j < b$时，$\mathrm{pre}_i \oplus \mathrm{pre}_j = V$的方案数

那么容斥一下，答案=$\mathrm{work}_{r,r}$-$\mathrm{work}_{r,l}$-$\mathrm{work}_{l,r} + \mathrm{work}_{l,l}$

```
#include <cstdio>
#include <iostream>
```

```
using namespace std;
const int MOD = 998244353, INV = (MOD+1)>>1;
typedef long long ll;

ll l, r, v, l1[64], l2[64], tar[64], f[64];
bool vis[64];
int offset[4] = {0,1,3,0};

void qumo(ll &x){
    x+=(x>>31)&MOD;
}

ll DFS(int x, bool u1, bool u2){
    if(x<0) return 1;
    if(!u1&&!u2&&vis[x]) return f[x];
    ll p1 = u1?l1[x]:1, p2 = u2?l2[x]:1;
    ll res = 0;
    for(int i = 0 ; i <= p1 ; ++i){
        for(int j = 0 ; j <= p2 ; ++j){
            if((i^j)==tar[x]){
                qumo(res+=DFS(x-1,u1&&i==p1,u2&&j==p2)-MOD);
            }
        }
    }
    if(!u1&&!u2) f[x] = res, vis[x] = 1;
    return res;
}

int dp(ll a, ll b, ll t){
    for(int i = 59 ; i >= 0 ; --i){
        l1[i] = (a>>i)&1;
        l2[i] = (b>>i)&1;
        tar[i] = (t>>i)&1;
        vis[i] = f[i] = 0;
    }
    return DFS(59,1,1);
}

ll work(ll a, ll b){
    if(a<0||b<0) return 0;
    ll res = 0;
    for(int i = 0 ; i < 4 ; ++i){
        for(int j = 0 ; j < 4 ; ++j){
            if(i<=a&&j<=b&&(offset[i]^offset[j])==(v&3)){
                ll c = (a-i)>>2, d = (b-j)>>2, t = v>>2;
                if((i&1)&&(j&1)){
                    res = (res+!t*(c%MOD+1)*(d%MOD+1))%MOD;
                }else if(i&1){
                    if(t<=d) res = (res+c+1)%MOD;
                }else if(j&1){
                    if(t<=c) res = (res+d+1)%MOD;
                }else qumo(res+=dp(c,d,t)-MOD);
            }
        }
    }
    return res%MOD;
}

int main(){
    cin>>l>>r>>v;
    l--;
    /*
    for(int i = 0, res = 0 ; i <= 100 ; ++i){
        printf("i=%d res=%d\n",i,res);
        res ^= i;
    }
    */
    ll ans = (work(r,r)-work(r,l-1)*2+work(l-1,l-1)+2*MOD)%MOD;
    if(!v) qumo(ans-=(r-l+1)%MOD);
    printf("%lld\n",ans*INV%MOD);
    return 0;
}
```

F-Tree Painting

换根 DP

先 DFS 一次求根为 1 时的答案，换根。

换到的值=父节点答案-之前根节点大小+当前父节点大小

```
#include <cstdio>
#include <algorithm>

using namespace std;
typedef long long ll;
const int MAXN = 2e5 + 5;

int head[MAXN], tot;

struct Edge{
    int to, next;
}G[MAXN<<1];

void addEdge(int u, int v){
    G[++tot].to = v;
    G[tot].next = head[u];
    head[u] = tot;
}

int n, size[MAXN];
ll ans, dp[MAXN];

void DFS1(int np, int fat = 0){
    size[np] = 1;
    for(int i = head[np], to ; i ; i = G[i].next){
        to = G[i].to;
        if(fat==to) continue;
        DFS1(to,np);
        size[np] += size[to];
        dp[np] += dp[to];
    }
    dp[np] += size[np];
}

void DFS2(int np, int fat, ll val){
    ans = max(ans,val);
    for(int i = head[np], to ; i ; i = G[i].next){
        to = G[i].to;
        if(fat==to) continue;
        DFS2(to,np,val-n-size[to]+n+n-size[to]);
    }
}
```

```
int main(){
    scanf("%d",&n);
    for(int i = 1, u, v ; i < n ; ++i){
        scanf("%d %d",&u,&v);
        addEdge(u,v);
        addEdge(v,u);
    }
    DFS1(1);
    DFS2(1,0,dp[1]);
    printf("%lld\n",ans);
    return 0;
}
```

G-Flow of binary matrix

思维+模拟

用r_i和c_j记录第i行 1 的个数和第j列 1 的个数，用于$O(n)$查询

操作 1，$O(1)$更新

操作 2

① 要把整个矩阵后移一位，可以把矩阵转换成一维数组，操作时把整体的区间前移一位即可。

② 对于列：更新$i+1$列为第i列，用$O(n)$的时间更新第 1 列

③ 对于行：注意新加入的数和被挤掉的数

```
#include <cstdio>
#include <numeric>

using namespace std;
const int MAXN = 5005;

```

```
int n, q, r[MAXN], c[MAXN], offset = 5001;
bool mtx[MAXN*MAXN+MAXN];

int acc(){
    int res = 0;
    for(int i = 0 ; i < n ; ++i)
        res += (c[i]==n)+(r[i]==n);
    return res;
}

int main(){
    scanf("%d %d\n",&n,&q);
    for(int i = 0 ; i < n ; ++i){
        for(int j = 0 ; j < n ; ++j){
            mtx[i*n+j+offset] = getchar()=='1';
            if(mtx[i*n+j+offset]) ++r[i], ++c[j];
        }
        getchar();
    }
    for(int i = 1, opt, x, y, v ; i <= q ; ++i){
        scanf("%d",&opt);
        if(opt==1){
            scanf("%d %d %d",&x,&y,&v);
            --x; --y;
            if(mtx[offset+x*n+y]^v)
                r[x] -= mtx[offset+x*n+y]?1:-1, c[y] -= mtx[offset+x*n+y]?1:-1;
            mtx[offset+x*n+y] = v;
        }else{
            scanf("%d",&v);
            mtx[--offset] = v;
            for(int j = 0 ; j < n ; ++j){
                if(mtx[offset+j*n]) ++r[j];
                if(mtx[offset+j*n+n]) --r[j];
            }
            for(int j = n ; j > 0 ; --j)
                c[j] = c[j-1];
            c[0] = 0;
            for(int j = 0 ; j < n ; ++j)
                if(mtx[offset+j*n]) ++c[0];
        }
        printf("%d\n",acc());
    }
    return 0;
}
```

H-Foreign Friends

最短路

先不考虑“不同城市”的限制条件，那么新建一个源点，与L个名人连一条边权为 0 的边，跑 Dijktra 最后得到的$dis1_i$即为最小花费

加入“不同城市”的限制条件，用类似次短路的方法，Dijktra 时记录最开始名人点，就能求出最开始名人点在不同城市的花费$dis2_i$。

```
#include <cstdio>
#include <queue>
#include <tuple>
#include <algorithm>

using namespace std;
typedef long long ll;
typedef tuple<ll,int,int> tp;
const int MAXN = 2e5 + 5;

int head[MAXN], tot;

struct Edge{
    int to, next, val;
}G[MAXN<<1];

void addEdge(int u, int v, int w){
    G[++tot].to = v;
    G[tot].val = w;
    G[tot].next = head[u];
    head[u] = tot;
}

int n, m, k, l, belong[MAXN], b[MAXN], used[MAXN], city[MAXN];
ll dis1[MAXN], dis2[MAXN];
priority_queue<tp,vector<tp>,greater<tp> > pq;

int main(){
    scanf("%d %d %d %d",&n,&m,&k,&l);
    for(int i = 1 ; i <= n ; ++i)
        scanf("%d",&belong[i]);
    for(int i = 1 ; i <= l ; ++i)
        scanf("%d",&b[i]);
    for(int i = 1, u, v, w ; i <= m ; ++i){
        scanf("%d %d %d",&u,&v,&w);
        addEdge(u,v,w);
        addEdge(v,u,w);
    }
    for(int i = 1 ; i <= l ; ++i)
        addEdge(n+1,b[i],0);
    pq.push(tp{0,n+1,k+1});
    while(!pq.empty()){
        ll dis = get<0>(pq.top()), np = get<1>(pq.top()), belongs =
get<2>(pq.top());
        pq.pop();
        if(used[np]>=0&&used[np]!=belongs){
            if(!used[np]){
                dis1[np] = dis;
                used[np] = city[np] = belongs;
            }else{
                dis2[np] = dis;
                used[np] = -1;
            }
            for(int i = head[np], to ; i ; i = G[i].next){
                to = G[i].to;
                pq.push(tp{dis+G[i].val,to,(belongs==k+1)?belong[to]:belongs});
            }
        }
    }
    for(int i = 1 ; i <= n ; ++i){
        if(belong[i]!=city[i]) printf("%lld%c",dis1[i]?dis1[i]:-1," \n"[i==n]);
        else printf("%lld%c",dis2[i]?dis2[i]:-1," \n"[i==n]);
    }
    return 0;
```

```
}
```