愉悦度(delight,1s,256MB)

【算法分析】

把问题抽象化：n个0/1变量$x_{1\ldots n}$，在对于所有的$1 \le s \le n-k+1$都满足$E \le \sum_{i=s}^{s+k-1} x_i \le k-S$的约束下，最大化$\sum_{i=1}^{n} \begin{cases} s_i, x_i = 0 \\ e_i, x_i = 1 \end{cases}$也就是$\sum_{i=1}^{n} s_i + \sum_{i=1}^{n} (e_i - s_i)[x_i = 1]$。

先不考虑≥ E的限制。

对每个不等式建立虚拟变量$y_{1\ldots n-k+1}$，每个不等式转化为等式：

$x_1 + x_2 + x_3 + \cdots + x_k + y_1 = k - S$

$x_2 + x_3 + x_4 + \cdots + x_{k+1} + y_2 = k - S$

…

$x_{n-k+1} + x_{n-k+2} + x_{n-k+3} + \cdots + x_n + y_{n-k+1} = k - S$

相邻两个等式作差：

$x_{k+1} + y_2 = x_1 + y_1$

$x_{k+2} + y_3 = x_2 + y_2$

…

$x_n + y_{n-k+1} = x_{n-k} + y_{n-k}$

把每个等式看作一个点，那么等式成立相当于每个点流量守恒。

而加上了≥ E的限制实际上就是限制$y \le k - S - E$。

由于每个变量有限制，所以把每个变量看作一条边。

(1) 由源点src向虚拟点src′连容量为$k-S$费用为0的边。点src′表示等式$x_1 + x_2 + x_3 + \cdots + x_k + y_1 = k - S$。

(2) 由src′发出k条边，其中第i条边指向点i，容量为1费用为$e_i - s_i$（表示x_i）。然后连边$(src', 1)$，容量为$k-S-E$费用为0（表示y_1），这时点1表示上面的第1个等式。

(3) 若要让第i个点表示第i个等式，则对于所有的$1 \le i \le n-k$，需要满足这些点的两条入边分别表示变量x_i和y_i，出边表示变量x_{k+i}和y_{i+1}。于是对于所有的$1 \le i \le n-k$，连边$< i, i+k, 1, e_{i+k} - s_{i+k} >$表示$x_{k+i}$，连边$< i, i+1, 1, 0 >$表示$y_{i+1}$。

(4) 注意到对于所有的$n-k < i \le n$，流入点i的流量无法正常流出。所以把这些点向汇点des连容量为∞费用为0的边即可。

答案为$\sum_{i=1}^{n} s_i$加上最大费用最大流。

【参考程序】

```
// 陈栉旷
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
#define For(i, a, b) for (i = a; i<= b; i++)
#define Edge(u) for (int e = adj[u], v = go[e]; e; e = nxt[e],
v = go[e])
#define CostFlow(u) for (u = T; u != S; u = st[pre[u]])
using namespace std;
```

```
inline int read() {
   int res = 0; bool bo = 0; char c;
   while (((c = getchar()) < '0' || c > '9') && c != '-');
   if (c == '-') bo = 1; else res = c - 48;
   while ((c = getchar()) >= '0' && c <= '9')
   res = (res << 3) + (res << 1) + (c - 48);
   return bo ? ~res + 1 : res;
}
typedef long longll;
const int N = 4005, M = 7e5 + 5, INF = 0x3f3f3f3f;
const llINFll = 1ll << 62;
int n, k, Ms, Me, Type, ecnt = 1, nxt[M], adj[M], go[M],
cap[M],
st[M], cost[M], S, T, len, que[M], pre[M], _s[N], _e[N],
sp[M];
ll dis[M], sum;
bool vis[M];
void add_edge(int u, int v, int w, int x) {
   nxt[++ecnt] = adj[u]; adj[u] = ecnt; st[ecnt] = u;
   go[ecnt] = v; cap[ecnt] = w; cost[ecnt] = x;
   nxt[++ecnt] = adj[v]; adj[v] = ecnt; st[ecnt] = v;
   go[ecnt] = u; cap[ecnt] = 0; cost[ecnt] = -x;
}
bool spfa() {
   int i;
   For (i, S, T) dis[i] = INFll;
   dis[que[len = 1] = S] = 0;
   For (i, 1, len) {
      int u = que[i]; vis[u] = 0;
      Edge(u) if (cap[e] && dis[u] + cost[e] < dis[v]) {
         dis[v] = dis[u] + cost[pre[v] = e];
         if (!vis[v]) vis[que[++len] = v] = 1;
      }
   }
   return dis[T] <INFll;
}
llcalc() {
   int u, flow = INF;
   CostFlow(u) flow = min(flow, cap[pre[u]]);
   CostFlow(u) cap[pre[u]] -= flow, cap[pre[u] ^ 1] += flow;
   return dis[T] * flow;
}
llMinCostMaxFlow() {
   ll res = 0;
```

```
    while (spfa()) res += calc();
    return res;
}
int main() {
    freopen("delight.in", "r", stdin);
    freopen("delight.out", "w", stdout);

    int i, tot = 0;
    n = read(); k = read(); Ms = read();
    Me = read();
    S = 1; T = n + 3;
    For (i, 1, n) _s[i] = read();
    For (i, 1, n) _e[i] = read(), sum += _s[i];
    add_edge(S, 2, k - Ms, 0);
    add_edge(2, 3, k - Ms - Me, 0);
    For (i, 1, k) add_edge(2, i + 2, 1, _s[i] - _e[i]),
        sp[++tot] = ecnt;
    For (i, 1, n - 1) add_edge(i + 2, i + 3, k - Ms - Me, 0);
    For (i, 1, n - k)
        add_edge(i + 2, i + k + 2, 1, _s[i + k] - _e[i + k]),
            sp[++tot] = ecnt;
    For (i, n - k + 1, n) add_edge(i + 2, T, INF, 0);
    cout<< sum - MinCostMaxFlow() <<endl;
    return 0;
}
```

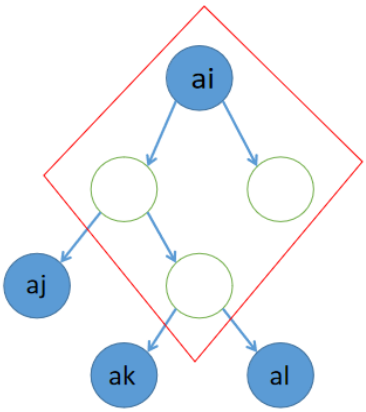
红蓝树(w,1s,256MB)

【算法分析】

题目保证同一棵树的条件不会自相矛盾，因此一个点的激活对所有存在条件的祖先的影响就可以转变为对最近的存在条件的祖先的影响。

具体地说，原本我们的限制是 a_i 的子树中恰好有 b_i 个点被激活，现在我们可以找出 a_i 的子树中所有存在限制且与 a_i 相邻的点 a_j，令 b_i 减去所有的 b_j，就得到了新的限制形式，即限制以 a_i 为深度最小的点且不包含其它有限制的点的连通块内恰好有 b_i 个点被激活。

例如蓝树中的一个点 a_i 所限制的范围即为下图中红线所圈出的区域。



这样一棵树上的每个点都会被划分到某一个区域内，也就只对这一区域的点 a_i 有影响。

由此我们来考虑费用流的建图，通过上述处理得到新的 b_i 后，考虑把红树和蓝树中所有存在限制的点看做二分图中的两个点集，从源点向红树中所有存在限制的点连一条流量为 b_i、费用为 0 的边，从蓝树中所有存在限制的点向汇点连一条流量为 b_i，费用为 0 的边。

对于任意一个点 x，我们分别找到它在红树中所属区域的点 a_i 和在蓝树中所属区域的点 a_j，此时点 x 产生的影响为这两个区域所包含的点数加 1，因此我们从 a_i 到 a_j 连一条流量为 1，费用为-w_x 的边，求最小费用最大流，得到的答案取反就是最大收益。若源汇点所连边存在不满流的情况，则不存在满足条件的方案，输出-1。

由于点数、边数和最大流的规模均为 $O(n)$，总的时间复杂度 $O(n^3)$，实际情况下运行较快。

【参考程序】

```
//陈贤
#include <bits/stdc++.h>

template <class T>
inline void read(T &res)
{
    char ch; bool flag = false; res = 0;
    while (ch = getchar(), !isdigit(ch) && ch != '-');
    ch == '-' ? flag = true : res = ch ^ 48;
    while (ch = getchar(), isdigit(ch))
        res = res * 10 + ch - 48;
    flag ? res = -res : 0;
}

using std::vector;
const int N = 1005;
const int M = 1e6 + 5;
const int Maxn = 0x3f3f3f3f;
int T = 1, n, src, des, ans, Q1, Q0, rtx, rty; bool vis[N], walk[N];
int fa1[N], fa2[N], w[N], dis[N];
int h[M], lim1[N], lim2[N];
```

```
int tag1[N], tag2[N], lst[N], nxt[M], cst[M], to[M], flw[M];
vector<int> v1[N], v2[N];

inline void Link(int x, int y, int z, int w)
{
    nxt[++T] = lst[x]; lst[x] = T; to[T] = y; flw[T] = z; cst[T] = w;
    nxt[++T] = lst[y]; lst[y] = T; to[T] = x; flw[T] = 0; cst[T] = -w;
}

inline bool SPFA()
{
    for (int i = 1; i <= des; ++i)
        dis[i] = Maxn, walk[i] = false;
    dis[h[1] = src] = 0;
    int t = 0, w = 1, x, y;
    while (t < w)
    {
        vis[x = h[++t]] = false;
        for (int e = lst[x]; e; e = nxt[e])
            if (y = to[e], flw[e] > 0 && dis[y] > dis[x] + cst[e])
            {
                dis[y] = dis[x] + cst[e];
                if (!vis[y])
                    vis[h[++w] = y] = true;
            }
    }
    return dis[des] < Maxn;
}

template <class T>
inline T Min(T x, T y) {return x < y ? x : y;}

inline int Dinic(int x, int flow)
{
    if (x == des)
    {
        ans += flow * dis[des];
        return flow;
    }

    walk[x] = true;
    int y, del, res = 0;
    for (int e = lst[x]; e; e = nxt[e])
        if (y = to[e], !walk[y] && flw[e] > 0 && dis[y] == dis[x] + cst[e])
```

```
        {
            del = Dinic(y, Min(flow - res, flw[e]));
            if (del)
            {
                flw[e] -= del;
                flw[e ^ 1] += del;
                res += del;
                if (flow == res)
                    break;
            }
        }
    return res;
}

inline void MCMF()
{
    int res = 0;
    while (SPFA())
        res += Dinic(src, Maxn);
}

inline void Dfs1(int x)
{
    int tot = 0;
    for (int i = 0, im = v1[x].size(); i < im; ++i)
    {
        int y = v1[x][i];
        if (y == fa1[x])
            continue;
        fa1[y] = x;
        Dfs1(y);
    }
}

inline void Dfs2(int x)
{
    for (int i = 0, im = v2[x].size(); i < im; ++i)
    {
        int y = v2[x][i];
        if (y == fa2[x])
            continue;
        fa2[y] = x;
        Dfs2(y);
    }
```

```
}

inline void Dfs3(int x, int &v) //通过 DFS 得到新的 bi
{
    for (int i = 0, im = v1[x].size(); i < im; ++i)
    {
        int y = v1[x][i];
        if (y == fa1[x])
            continue;
        if (tag1[y] != Maxn)
        {
            v -= tag1[y];
            continue;
        }
        Dfs3(y, v);
    }
}

inline void Dfs4(int x, int &v)
{
    for (int i = 0, im = v2[x].size(); i < im; ++i)
    {
        int y = v2[x][i];
        if (y == fa2[x])
            continue;
        if (tag2[y] != Maxn)
        {
            v -= tag2[y];
            continue;
        }
        Dfs4(y, v);
    }
}

int main()
{
    freopen("w.in", "r", stdin);
    freopen("w.out", "w", stdout);

    read(n); read(rtx); read(rty);
    src = n << 1 | 1;
    des = src + 1;
    for (int i = 1; i <= n; ++i)
        read(w[i]);
```

```
for (int i = 1, x, y; i < n; ++i)
{
    read(x); read(y);
    v1[x].push_back(y);
    v1[y].push_back(x);
}
for (int i = 1, x, y; i < n; ++i)
{
    read(x); read(y);
    v2[x].push_back(y);
    v2[y].push_back(x);
}

for (int i = 1; i <= n; ++i)
    tag1[i] = tag2[i] = Maxn;
read(Q0);
for (int i = 1, x, y; i <= Q0; ++i)
{
    read(x); read(y);
    tag1[x] = y;
}
read(Q1);
for (int i = 1, x, y; i <= Q1; ++i)
{
    read(x); read(y);
    tag2[x] = y;
}

Dfs1(rtx);
Dfs2(rty);
for (int i = 1; i <= n; ++i)
    lim1[i] = tag1[i], lim2[i] = tag2[i];
for (int i = 1; i <= n; ++i)
    if (lim1[i] != Maxn)
        Dfs3(i, lim1[i]);
for (int i = 1; i <= n; ++i)
    if (lim2[i] != Maxn)
        Dfs4(i, lim2[i]);

for (int i = 1; i <= n; ++i)
    if (lim1[i] < 0 || lim2[i] < 0)
        return puts("-1"), 0;

for (int i = 1; i <= n; ++i)
```

```
    {
        int u, v;
        for (u = i; tag1[u] == Maxn; u = fa1[u]);
        for (v = i; tag2[v] == Maxn; v = fa2[v]);
        Link(u, v + n, 1, -w[i]);
    }
    for (int i = 1; i <= n; ++i)
        if (lim1[i] != Maxn)
            Link(src, i, lim1[i], 0);
    for (int i = 1; i <= n; ++i)
        if (lim2[i] != Maxn)
            Link(i + n, des, lim2[i], 0);
    MCMF();

    bool flag = false; //判断无解情况
    for (int e = lst[src]; e; e = nxt[e])
        if (flw[e] > 0)
        {
            flag = true;
            break;
        }
    for (int e = lst[des]; e; e = nxt[e])
        if (flw[e ^ 1] > 0)
        {
            flag = true;
            break;
        }
    printf("%d\n", flag ? -1 : -ans);

    fclose(stdin); fclose(stdout);
    return 0;
}
```

木棍问题（trouble，1.5s，512MB）

【算法分析】

我们考虑在每个木球上计算答案。

若 $A=B$，则对于一个度数为 $k(0\le k\le 4)$ 的木球，其贡献就是 $\frac{k(k-1)}{2}\cdot A$。

那么我们考虑对木球黑白染色，对于黑点x，我们从 S向 x连四条边：

$$(1,0),(1,A),(1,2A),(1,3A)$$

对于白点，我们从 x 向 T 连四条边：

$$(1,0),(1,A),(1,2A),(1,3A)$$

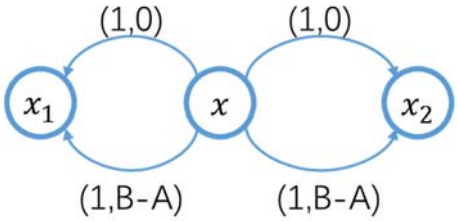
对于相邻的黑白点，由于一对相邻木球只能连一条边，所以我们就从黑点的 x 向白点的 x 连边 $(1,0)$。

当 $B>A$ 时，对于每个直线型，我们要多支付 $B-A$ 的费用，所以我们考虑拆点。

把每个点拆成 x,x_1,x_2 三个点。

与之前不同的是，我们不直接把相邻木球的 x 相连，而是把x_1 和纵向相邻木球的 x_1 连边，x_2和横向相邻木球的 x_2 连边，流量和费用都是 $(1,0)$，方向由黑白染色决定。

那么问题在于，当x_1 或 x_2连接的相邻点的总流量达到 2时，我们要多支付$B-A$的费用，这个我们可以通过在 $x\to x_1,x\to x_2$ 之间分别连两条边实现，如下图。



同时，每个木球的 x同样和源点汇点连那样的四条边。

这样建出的图点数和边数的规模就都是 $O(nm)$ 的。

值得注意的是，题目要求我们输出 k 为所有合法的值的答案，一般的思路是每次只增广 1个流量，每次记录总费用。但是实际上，因为源点汇点所连的边流量都是 1，我们可以发挥 $Dinic$的多路增广的优势，每次增广到汇点时，表示流到汇点的流量为 1，我们可以直接记录下 k 增加 1 的答案变化，这样可以提高效率。

【参考程序】

```
//陈煜翔
#include <bits/stdc++.h>

template <class T>
inline void read(T &x)
{
    static char ch;
    while (!isdigit(ch = getchar()));
    x = ch - '0';
    while (isdigit(ch = getchar()))
        x = x * 10 + ch - '0';
}

template <class T>
inline bool tense(T &x, const T &y)
{
    return x >y ? x = y, true : false;
}

#define trav(u) for (halfEdge * e = adj[u]; e; e = e->next)
#define flow(u) for (halfEdge *&e = cur[u]; e; e = e->next)//当前弧优化
```

```
constint INF = 0x3f3f3f3f;
constintMaxN = 45;
constintMaxNV = MaxN * MaxN * 4;
constintMaxNE = MaxNV * 10;

structhalfEdge
{
    int v, w, c;
    halfEdge *next;
};

halfEdge *adj[MaxNV], *cur[MaxNV];
halfEdgeadj_pool[MaxNE], *adj_tail = adj_pool;

intnV, src, des;
int dis[MaxNV];
bool walk[MaxNV];

int n, m, typ, A, B;
intpos[MaxN][MaxN][3];
bool bo[MaxN][MaxN];

char s[MaxN];
intres_cost;

inline void addEdge(int u, int v, int w, int c)
{
    adj_tail->v = v, adj_tail->w = w, adj_tail->c = +c, adj_tail->next = adj[u], adj[u] = adj_tail++;
    adj_tail->v = u, adj_tail->w = 0, adj_tail->c = -c, adj_tail->next = adj[v], adj[v] = adj_tail++;
}

inline void insEdge(int u, int v, int w, int c, bool opt)
{
    if (opt) //根据黑白染色来判断连边的方向
        addEdge(u, v, w, c);
    else
        addEdge(v, u, w, c);
}

inline halfEdge *oppo(halfEdge *e) //取 e 的反边
{
    return adj_pool + ((e - adj_pool) ^ 1);
}
```

```
inline bool SPFA()
{
    static bool inq[MaxNV];
    static intque[6000010], qr;
    for (inti = 1; i<= nV; ++i)
    {
        cur[i] = adj[i];
        dis[i] = INF;
        walk[i] = false; //walk 数组表示某个点是否已经不能再增广
    }
    inq[src] = true;
    dis[que[qr = 1] = src] = 0;
    for (intql = 1; ql<= qr; ++ql)
    {
        int u = que[ql];
        inq[u] = false;

        trav(u)
            if (e->w && tense(dis[e->v], dis[u] + e->c) && !inq[e->v])
                inq[que[++qr] = e->v] = true;
    }
    return dis[des] < INF;
}

inline intDinic(int u, intflw)
{
    if (u == des)
    {
        res_cost+=dis[des];//每次增广到汇点，流量肯定恰好增加 1，直接记录答案
        printf("%d\n", typ>= 8 &&typ<= 12 ? (bool)res_cost :res_cost);
        return flw;
    }
    walk[u] = true;
    int res = 0;
    flow(u)
        if (e->w && dis[e->v] == dis[u] + e->c && !walk[e->v])
        {
            int delta = Dinic(e->v, std::min(e->w, flw - res));
            if (delta)
            {
                e->w -= delta;
                oppo(e)->w += delta;
                res += delta;
```

```
            if (res == flw)
                break;
        }
    }
    if (res == flw)
        walk[u] = false; //流入 u 的流量全部被增广出去，从 u 可能还可以增广
    return res;
}

inline void MCMF()
{
    while (SPFA())
        Dinic(src, INF);
}

intmain()
{
    freopen("trouble.in", "r", stdin);
    freopen("trouble.out", "w", stdout);
    read(typ), read(n), read(m), read(A), read(B);
    src = ++nV, des = ++nV;
    for (inti = 1; i<= n; ++i)
    {
        scanf("%s", s + 1);
        for (int j = 1; j <= m; ++j)
        {
            bo[i][j] = s[j] == '0';
            if (bo[i][j])
            {
                pos[i][j][0] = ++nV;
                pos[i][j][1] = ++nV;
                pos[i][j][2] = ++nV;

                for (int k = 0; k < 4; ++k)
                    if (i + j & 1) //根据黑白染色和源点或汇点连四条边算 A 贡献
                        addEdge(src, pos[i][j][0], 1, A * k);
                    else
                        addEdge(pos[i][j][0], des, 1, A * k);
                for (int k = 1; k <= 2; ++k) //x 和拆出的点连边算 B 贡献
                {
                    insEdge(pos[i][j][0],pos[i][j][k],1, 0    , i + j & 1);
                    insEdge(pos[i][j][0],pos[i][j][k],1, B - A, i + j & 1);
                }
            }
```

```
        }
    }
    for (inti = 1; i<= n; ++i)
        for (int j = 1; j <= m; ++j)
        {
            if (!bo[i][j])
                continue;

            if (i != n &&bo[i + 1][j]) //和相邻点连边
                insEdge(pos[i][j][1], pos[i + 1][j][1], 1, 0, i + j & 1);
            if (j != m &&bo[i][j + 1])
                insEdge(pos[i][j][2], pos[i][j + 1][2], 1, 0, i + j & 1);
        }
    MCMF();
    return 0;
}
```