T1

维护跳 k 次能达到的最大编号，记为 b_k

$b_k = \max(a_i + i) \text{ where } i \le b_{k-1}$

简单模拟即可

T2

本题难度不止T2，但是是**原题**，希望大家从此注意**补题**，不是老师讲多少做多少

注意到产量较小，我们考虑从产量入手解决

设 $\text{dp}[i][j][k]$ 为前 i 条产线，生产了 j 个汉堡，k 个薯条能够生产的**最大**饮料个数

我们枚举前 $i-1$ 条产线生产的汉堡和薯条个数，记为 u, v，则我们第 i 条产线花费了 $(j-u) \times A + (k-v) \times B$ 的时间生产汉堡和薯条，其余时间生产饮料即可

注意初值和转移时判定某个产量是否合法即可

$\text{Bonus}: \min(\{a, b, c, \cdots\})$ 可以判多个数的 $\min$

T3

注意到 $a_{i,j}$ 范围很小（小于 nm），因此我们可以考虑 $O(n^3)$ 的算法

对于朴素的二维前缀和而言，我们可以在其基础上加一维值域，从而确认我们当前维护的是**哪一个数**的数量

对每个位置，我们简单枚举每种值，查看它在**板子**所在的子矩阵的个数是否和整个矩阵的个数相等，若相等，则说明该数没有露出来，反之亦然

```
void solve() {
    int n, m, k, h, w;
    cin >> n >> m >> k >> h >> w;
    vector mp(n + 1, vector(m + 1, vector(k + 1, 0)));

    for (int i = 1; i <= n; i++) for (int j = 1; j <= m; j++) {
        int u; cin >> u;
        mp[i][j][u]++;
        for (int kk = 1; kk <= k; kk++)
            mp[i][j][kk] += mp[i - 1][j][kk] + mp[i][j - 1][kk] - mp[i - 1][j - 1][kk];
    }

    for (int i = 1; i <= n - h + 1; i++) for (int j = 1; j <= m - w + 1; j++) {
        int cnt = 0;
        for (int kk = 1; kk <= k; kk++) {
            cnt += (mp[n][m][kk] - (mp[i + h - 1][j + w - 1][kk] - mp[i + h - 1][j - 1][kk] -
                                    mp[i - 1][j + w - 1][kk] + mp[i - 1][j - 1][kk])) > 0;
        }
        cout << cnt << " \n"[j == m - w + 1];
    }
```

```
}
```

T4

同样的，**补题，补题，补题**（这题还是有很多同学试图在补，值得鼓励）

首先，我们很显然可以用 LCA 得到从 u 点到 v 点会经过哪些点

如果说我们严格按照经过的城市（即不考虑某个城市是否在路上被经过过了）顺序行驶，就是个 LCA 板题

故我们只需要考虑维护访问序列即可

然而，在我们行驶路上，可能会遇到一些之后被访问的城市，这些城市我们不需要再次访问

单纯的打 tag 的方法，由于 tag 的维护是 $O(n)$ 的，因此会超时

注意到我们每个点只有在**第一次经过时**会被打上 tag ，我们可以将所有的被打上 tag 的点，和他祖先中，最近的没有被打 tag 的点**合并**（之所以是祖先，因为我们火车是沿着这条路径行驶的），此处用**并查集**维护即可

细节看代码

```
namespace LCA{

vector<int> e[maxn];
int f[__lg(maxn) + 1][maxn], dep[maxn], n, lim;
vector<int> vis, fa;

void dfs(int u, int fa) {
    f[0][u] = fa; dep[u] = dep[fa] + 1;
    for (auto v: e[u])
        if (v != fa) dfs(v, u);
}

// init前需要输入树边
void init(int sz, int rt) {
    n = sz; lim = __lg(n);
    dep[rt] = 1; dfs(rt, 0);
    for (int i = 1; i <= lim; i++) for (int u = 1; u <= n; u++) f[i][u] = f[i - 
1][f[i - 1][u]];
    vis.resize(n + 1, 0); fa = vis;
    iota(all(fa), 0);
}

int lca(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    for (int i = lim; i >= 0; i--) if (dep[u] - (1 << i) >= dep[v]) u = f[i][u];
    if (u == v) return u;

    for (int i = lim; i >= 0; i--) {
        if (!f[i][u]) continue;
        if (f[i][u] != f[i][v]) u = f[i][u], v = f[i][v];
    }
    return f[0][u];
```

```
};

int find(int u) {
    return (u == fa[u]) ? u : fa[u] = find(fa[u]);
};

void merge(int u, int v) {
    u = find(u); v = find(v);
    fa[v] = u;
};

void run(int u, int fa) {
    vis[u] = 1;
    u = find(u), fa = find(fa);
    while (u != fa) {
        merge(f[0][u], u);
        u = find(u);
        vis[u] = 1;
    }
}

int get_dis(int u, int v) {
    return abs(dep[u] - dep[v]);
}

void clear() {
    for (int i = 1; i <= n; i++) e[i].clear();
    for (int i = 1; i <= lim; i++) for (int u = 1; u <= n; u++) f[i][u] = 0;
}

}

void solve() {
    int n, m, a; cin >> n >> m >> a;
    for (int i = 1; i < n; i++) {
        int u, v; cin >> u >> v;
        LCA::e[u].push_back(v); LCA::e[v].push_back(u);
    }

    LCA::init(n, 1);
    vector pos(m, 0); for (auto& v: pos) cin >> v;

    ll ans = 0;

    LCA::vis[a] = 1;
    for (auto v: pos) if (!LCA::vis[v]) {
        auto L = LCA::lca(a, v);
        ans += LCA::get_dis(a, L) + LCA::get_dis(v, L);
        LCA::run(a, L); LCA::run(v, L);
        a = v;
    }

    cout << ans << endl;

}
```