The first one :

分析

一道简单的模拟题，数据规模 $n \leq 20000$，直接用 sort 完全没问题。

Code

```
using namespace std;
const int N = 1e5 + 50;
int n, k;
int E[N], C[N];
struct person {
	int id, W;
	bool operator <(const person &tmp) const {
		if(W == tmp.W) return id < tmp.id;
		return W > tmp.W;
	}
}p[N];
int main()
{
	Read(n); Read(k);
	for(int i = 1; i <= 10; i ++) Read(E[i]);
	for(int i = 1; i <= n; i ++) Read(p[i].W), p[i].id = i;
	sort(p + 1, p + 1 + n);
	for(int i = 1; i <= n; i ++) {
		int tmp = (i - 1) % 10 + 1;
		p[i].W += E[tmp];
	}
	sort(p + 1, p + 1 + n);
	for(int i = 1; i <= k; i ++) print(p[i].id), putchar(' ');
	return 0;
}
```

The second one :

题意

给定几种药水的价格 和 某些药水之间的转化关系，求制得 0 号的最低花费和总计方案数。

分析

考场上的思维局限于建图，最后想不出正解用树形 dp 骗分潦草收场。

本题类似 dijkstra 的想法，用已知 (已经更新到最小值) 来更新未知 (没有更新到最小值)。
我们用数组 vis 记录当前数据是否已经是最小值，dis 记录当前获得该药剂的最小价格
不断更新每一种药剂的 dis (最小价格)。类比 dijkstra 的思想理解。
对于方案数 sum, 假定第 i 种药剂有 k 种合成方案，每种方案所需的两种药剂为 x_i，y_i，则
$sum_i = \sum_{i=1}^{k}(sum_{xi} * sum_{yi} * [dis_{xi} + dis_{yi} == dis_i])$, 这种转移可以类比与树形 dp。

Code

```
using namespace std;
typedef long long i64;
const int N = 1e3 + 50, Inf = 0x3f3f3f3f;
int n, g[N][N];
```

```
i64 sum[N], dis[N];
bool vis[N];
void solve()
{
        // 数据量较小， 用邻接表即可
        for(int i = 1; i <= n ; i++) {
                int u, t = Inf;
                for(int j = 1; j <= n; j ++) {
                        if(!vis[j] && dis[j] < t) {
                        // 枚举找到最小的价格，此时它一定是最小值
                                t = dis[j];
                                u = j;
                        }
                }
                if(t == Inf) break;// 找不到最小的， 说明已经无法在更新，程序结束
                vis[u] = true;
                for(int j = 1; j <= n; j ++) { // 枚举另一种所需合成药剂
                        if(!g[u][j] || !vis[j]) continue; // 如果不能合成 或 没有更新到最小值
                        if(dis[g[u][j]] > t + dis[j]) { // 如上文
                                dis[g[u][j]] = t + dis[j];
                                sum[g[u][j]] = 0;
                        }
                        if(dis[g[u][j]] == t + dis[j]) sum[g[u][j]] += sum[u] * sum[j];
                }
        }
        return;
}
int main()
{
        cin >>n;
        for(int i = 1; i <= n; i ++) cin >>dis[i], sum[i] = 1;
        int x, y, z;
   // 习惯用1为开始
        while(cin >>x >>y >>z) g[x + 1][y + 1] = g[y + 1][x + 1] = z + 1;
        solve();
        cout <<dis[1] <<" " <<sum[1];
        return 0;
}
```

The third one:

题意(按照我的做法的理解)

给定几条线段（左右端点为 x_i， y_i）， 每个线段有一个权值和长度， 可以选择任意条长度在 $[low, h_i]$ 之间的线段， 要求选定的任意两条线段不能完全重和（两条线段 i, j 被认为完全重合当且仅当 i 是 j 的子段 或 j 是 i 的子段）， 求能获得的最大权值之和。

分析

- 假定线段 i 的左端点为 x_i， 右端点为 y_i， 权值为 val_i， 长度为 len_i
- 因为我们选定的魔杖一定是一个连续的序列， 所以我们可以将一段魔杖看成一个线段，线段的左右端点即是最左端 和 最右端在序列中的位置， 权值即为这一段的魔力值。 我们可以选的其实就是 $low \leq len_i \leq h$ 的线段， 其他部分显然对答案无影响。
- 其次， 本题只要求最大值， 可以看出用 dp 的方式解决问题。

> - 先将线段按照左端点递增进行排序（本题貌似不需要）。

> - 我们令 $f[i][j]$ 表示递推到第 i 条线段，从 1 覆盖到 j（即 1 到 j 之间的部分已经被线段覆盖）的最大权值和。
> - 则有 $f[i][y_i] = \max\limits_{j=0}^{j<y_i}(f[i-1][j]) + val_i$
> - 道理很简单，枚举所有覆盖不到线段 i 右端点的情况，以保证不会将当前线段完全覆盖，此时当前线段可以被选择且不与其他已选定的线段冲突， 取最大值即可。
> - 注意到每次 dp 时只会使用上一次的结果， 故可以转化为一维。
>
> $f[y_i] = \max\limits_{j=0}^{j<y_i}(f[j]) + val_i$

- 讲的不好的话，请见谅。 具体细节见代码。

Code

```
using namespace std;
typedef long long i64;
const int N = 1e3 + 50;
i64 qz[N], qz2[N], dp[N], cnt;
// 好像洛谷上要开long long
int n, low, hi, L[N], M[N];
struct line {
        int x, y;
        long long val;
        bool operator <(const line &tmp) const {
                if(x == tmp.x) return y > tmp.y;
                return x < tmp.x;
        }
}a[N * N];
int main()
{
        Read(n); Read(low); Read(hi);
        for(int i = 1; i <= n; i ++) {
                Read(L[i]);
                qz[i] = qz[i - 1] + L[i]; // 计算前缀和， 方便后边计算线段
        }
        for(int i = 1; i <= n; i ++) {
                Read(M[i]);
                qz2[i] = qz2[i - 1] + M[i];
        }
        for(int i = 1; i <= n; i ++) {
                for(int j = i; j <= n; j ++) {
                        // 枚举所有可能的线段
                        int tmp = qz[j] - qz[i - 1];
                        if(tmp >= low && tmp <= hi) {
                                a[++cnt].x = i;
                                a[cnt].y = j;
                                // 记录线段的左右端点
                                a[cnt].val = qz2[j] - qz2[i - 1];
                        }
                }
        }
        // 应该可以不用排序，毕竟枚举时本来就按照左端点的升序储存
        sort(a + 1, a + 1 + cnt);

        for(int i = 1; i <= cnt; i ++) {
                for(int j = 0; j < a[i].y; j ++) {
                // 这里的枚举范围为[0, 右端点), 不能到右端点防止完全覆盖
                // 从0开始是因为可能出现只有一条线段的情况(我的10pts就是这里错了)
                        dp[a[i].y] = max(dp[a[i].y], dp[j] + a[i].val);
```

```
                }
        }
        i64 ans = 0;
        // 要取最大值，毕竟最终选定的权值和最大的线段组合不一定将[1, n]完全覆盖

        for(int i = 0; i <= n; i ++) {
                ans = max(ans, dp[i]);
        }
        print(ans);
        return 0;
}
```

The last one

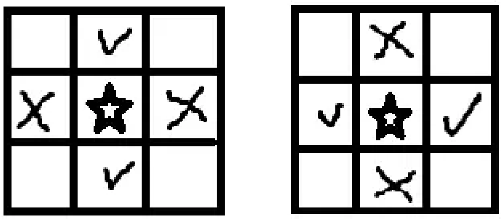
题意

给定一个 $n \times m$ 的矩阵， 小A从 (1, m) 出发， 可以向上下左右行走，
每个点只走一次， 且小A会对行走过的点按序编号，编号为 i 的点横坐标为 px_i，纵坐标为 py_i 。
最小化 $max(\max\limits_{i=0}^{n\times m/2-1}(k_1 * abs(px_i - px_{i+n\times m/2}) + k_2 * abs(py_i - py_{i+n\times m/2})))$ ， 输出最小值

分析

虽然， 我用 4 组特判含泪拿下 $40pts$。

本题正解是搜索， 加上可行性减枝 和 最优性剪枝就可以通过



左图来自hulean...

加入的可行性剪枝如图，应为题目要求药遍历完所有节点，故而以上两种情况不可能完成。看似这种剪枝有限，但如果考虑边界（边界不能遍历， 相当与上图中的叉）， 这时减掉的就很可观了

最优性剪枝：判断当前最大值是否大于答案

Code

不解释

```
using namespace std;
const int N = 550, Inf = 0x3f3f3f3f;
int n, m, k1, k2, Mod;
int px[3550], py[3550], ans = Inf;
bool vis[N][N];
int _next[4][2] = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};
void dfs(int x, int y, int id, int maxn)
{
```

```
        if(vis[x + 1][y] && vis[x - 1][y] && !vis[x][y + 1] && !vis[x][y - 1]) return;
        if(vis[x][y + 1] && vis[x][y - 1] && !vis[x + 1][y] && !vis[x - 1][y]) return;
        int res = 0;
        if(id <= n*m/2) px[id % Mod] = x, py[id % Mod] = y;
        else {
                int tmp = id % Mod;
                res = k1 * abs(px[tmp] - x) + k2 * abs(py[tmp] - y);
        }
        res = max(res, maxn);
        if(res >= ans) return;
        if(id == n * m) {
                ans = min(ans, res);
                return;
        }
        for(int i = 0; i <= 3; i ++) {
                int dx = x + _next[i][0], dy = y + _next[i][1];
                if(dx < 1 || dx > n || dy < 1 || dy > m || vis[dx][dy]) continue;
                vis[dx][dy] = 1;
                dfs(dx, dy, id + 1, res);
                vis[dx][dy] = 0;
        }
}
int main()
{
        cin >>n >>m >>k1 >>k2; Mod = n * m / 2;
        vis[1][1] = 1;
        for(int i = 0; i <= m + 1; i ++) { vis[0][i] = 1; vis[n + 1][i] = 1;}
        for(int i = 0; i <= n + 1; i ++) { vis[i][0] = 1; vis[i][m + 1] = 1;}
        dfs(1, 1, 1, 0);
        cout <<ans;
        return 0;
}
```