2 月 19 日测试题解

题目难度一般，但是为什么四道题在洛谷上都有原题。

T1

简单排序，不写了

T2

题意

有 n 种药水，编号是 0 到 $n-1$，每种药水有一个价格，还会给你一些药水之间的配置关系，例如 $1+2 \to 3$ 表示你可以消耗一瓶 1 号药水与一瓶 2 药水来获得一瓶 3 号药水。现在问你配制出 0 号药水的最小花费是多少，并且有几种方法的花费最小。

$n \leq 1000$

思路

考场上我写的又是个错解，但是这数据又给我放过了。

考场上的思路是，我用一个 `vector` 存下药水之间的关系，然后一遍 DFS 再加上记忆化就做完了。时间复杂度 $O(n)$，样例一遍过，一切都是那么的完美。

事实上这个做法是假的，假的地方在于没有规定药水之间的配置关系不能存在环。你单纯地 DFS 很好，但是加上记忆化之后如果出现环的话就会把环直接给忽略了，所以这道题是不能用 DFS 来写的。

我们知道题意其实类似于一个最短路问题，于是我们应该从这方面下手。考虑一个类似 Dijkstra 的方法，用 `f[i]` 表示配制出第 i 种药水的最小花费，再用 `minCost[i]` 来表示是否找到了第 i 种药水的最小花费值。

然后我们做一遍 Dijkstra 就可以了，但是更新值的时候，我们在没有找出最小花费的点里面找最小值更新，这一点与原版的 Dijkstra 不同。这道题 $n \leq 1000$，而且貌似有的图是稠的，所以可以用不优化的 Dijkstra 来写。

最后，记得在洛谷上边交，原数据太弱了。

代码

```cpp
#include <cmath>
#include <iostream>
#include <vector>
#include <cstring>

using namespace std;
using i64 = long long;

static const int N = 1e3 + 50, INF = 0x3f3f3f3f;
int n;
int myPrice[N];
int f[N], way[N][N];
bool minCost[N];
i64 g[N];
int a, b, c;

int main() {
        ios::sync_with_stdio(false);
        cin.tie(0), cout.tie(0);

        memset(way, -1, sizeof(way));

        cin >> n;
        for (int i = 0; i < n; i++) {
                cin >> f[i];
                g[i] = 1;
        }
        while (cin >> a >> b >> c) {
                way[a][b] = c;
                way[b][a] = c;
        }

        for (int i = 0; i < n; i++) {
                int pivot, maxx = INF;
                for (int j = 0; j < n; j++) {
                        if (!minCost[j] && f[j] < maxx) {
                                pivot = j;
                                maxx = f[j];
                        }
                }
                minCost[pivot] = true;

                for (int j = 0; j < n; j++) {
                        if (minCost[j] && ~way[pivot][j]) {
                                int tgt = way[pivot][j];
```

```
                                if (f[pivot] + f[j] < f[tgt]) {
                                        f[tgt] = f[pivot] + f[j];
                                        g[tgt] = g[pivot] * g[j];
                                } else if (f[pivot] + f[j] == f[tgt]) {
                                        g[tgt] += g[pivot] * g[j];
                                }
                        }
                }
        }

        cout << f[0] << ' ' << g[0] << '\n';

        return 0;
}
```

T3

题意

给你 n 个区间的长度与两个整数 lo 和 hi，每个区间都有自己的权值。你要选出一些区间的并，使得没有区间之间存在互相包含关系，区间的长度 len 满足 $lo \le len \le hi$，并且总的区间权值和最大。

$n \le 1000$

思路

赛事一开始想的是贪心，认为应该尽量选择短的区间，因为这样能够使更多的区间没有相互的包含关系，但是贪心明显是伪的。

后来看到 $n \le 1000$，想了个 $O(n^2)$ 的 DP。

我们设 `f[i][j]` 表示起点不超过 i，终点不超过 j 所获得的最大值。于是我们有方程：

$$f_{i,j} = \begin{cases} \max(f_{i-1,j}, f_{i,j-1}) \\ \max(f_{i-1,j}, f_{i,j-1}, f_{i-1,j-1} + \sum_{k=i}^{j} w_k) \quad lo \le \sum_{k=i}^{j} len_k \le hi \end{cases}$$

于是答案就是 `f[n][n]`。

中间两个求和明显是可以用前缀和优化的，这样总体的时间复杂度为 $O(n^2)$。

代码

```cpp
#include <iostream>

using namespace std;
using i64 = long long;

static const int N = 1e3 + 50;
int n, lo, hi;
int l[N], m[N], ans;
i64 f[N][N], sum[N][2];


int main() {
	ios::sync_with_stdio(false);
	cin.tie(0), cout.tie(0);

	cin >> n >> lo >> hi;
	for (int i = 1; i <= n; i++) {
		cin >> l[i];
		sum[i][0] = sum[i - 1][0] + l[i];
	}
	for (int i = 1; i <= n; i++) {
		cin >> m[i];
		sum[i][1] = sum[i - 1][1] + m[i];
	}

	/*
	for (int i = 1; i <= n; i++) {
		cerr << sum[i][0] << ' ' << sum[i][1] << '\n';
	}
	*/

	for (int i = 1; i <= n; i++) {
		for (int j = i; j <= n; j++) {
			int tmp = sum[j][0] - sum[i - 1][0];
			f[i][j] = max(f[i - 1][j], f[i][j - 1]);
			if (tmp >= lo && tmp <= hi) {
				f[i][j] = max(f[i][j], f[i - 1][j - 1] + sum[j][
			}
			// cerr << i << ' ' << j << ' ' << f[i][j] << '\n';
		}
	}

	cout << f[n][n] << '\n';

	return 0;
```

```
}
```

T4

题意

给你一个 n 行 m 列的矩阵以及两个整数 $k1$，$k2$。你从矩阵的右上角开始走，走过第 i 个格子时在这个格子放一个数 i。一种走法是合法的，当且仅当每个格子都恰好被经过了一次。设第 i 格格子坐标为 (x_i, y_i)，定义一种合法走法的权值为下面这个式子：

$$\max_{1\le i\le \frac{n\times m}{2}}(k1\times|x_i-x_{i+\frac{n\times m}{2}}|+k2\times|y_i-y_{i+\frac{n\times m}{2}}|)$$

你要求这个权值的最小值。

$n\times m\le 50$，保证 $n\times m\equiv 0(\mathrm{mod}\ 2)$。

思路

看到最大值最小，首先想到的是二分，但是很抱歉这道题不满足单调性，于是只能用搜索。

$n\times m$ 看起来不大，但是其合法的状态数是呈指数级增长的，所以还是要剪下枝，不然会 T 飞。

一个显而易见的剪枝是最优性减值：当当前的答案已经超过目前的最优值时，直接剪枝。但这样还不够，我们需要一个可行性剪枝。

我们从判断状态的合法性下手：当一个点的上下都被搜索过而左右都没被搜索过或者左右都被搜索过而上下都没被搜索过时，这个状态一定不合法。

提供一个不是太严谨的证明，就比如说第一种，你无论是往左搜还是往右搜，最终都会出现一条断头路，你就卡着回不去了，但此时至少还有一个节点没搜到，就是那个一开始没选的那一边，于是这个状态非法。

于是开个数组记录一下就好了，初始化的时候可以把矩阵边上的点都标记一下，有巨大的优化效果。

我考试的时候写了爆搜，拿了 80 分。

代码

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>

using namespace std;
using i64 = long long;

static const int N = 55, INF = 0x3f3f3f3f;
int n, m, k1, k2, ans = INF;
int a[N][N], d[][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
pair<int, int> mp[N];

bool check(int x, int y) {
        if (a[x - 1][y] && a[x + 1][y] && !a[x][y - 1] && !a[x][y + 1]) {
                return false;
        } else if (!a[x - 1][y] && !a[x + 1][y] && a[x][y - 1] && a[x][y + 1]) {
                return false;
        } else {
                return true;
        }
}

void solve(int x, int y, int cur, int t = -INF) {
        a[x][y] = cur;
        mp[cur].first = x;
        mp[cur].second = y;
        if (cur == n * m) {
                ans = min(ans, t);
                a[x][y] = 0;
                return;
        }
        if (!check(x, y)) {
                // cout << "True" << '\n';
                a[x][y] = 0;
                return;
        }
        if (cur > n * m / 2) {
                int x1 = mp[cur].first, y1 = mp[cur].second;
                int x2 = mp[cur - n * m / 2].first, y2 = mp[cur - n * m / 2].sec
                t = max(t, k1 * abs(x1 - x2) + k2 * abs(y1 - y2));
                if (t > ans) {
                        a[x][y] = 0;
                        return;
                }
        }
```

```
        for (int i = 0; i < 4; i++) {
                int dx = x + d[i][0], dy = y + d[i][1];
                if (a[dx][dy]) continue;
                if (dx < 1 || dx > n || dy < 1 || dy > m) continue;
                solve(dx, dy, cur + 1, t);
        }

        a[x][y] = 0;
        return;
}

int main() {
        ios::sync_with_stdio(false);
        cin.tie(0), cout.tie(0);

        cin >> n >> m >> k1 >> k2;

        // memset(a, -1, sizeof(a));
        for (int i = 0; i <= n + 1; i++) {
                a[i][0] = 1;
                a[i][m + 1] = 1;
        }
        for (int i = 0; i <= m + 1; i++) {
                a[0][i] = 1;
                a[n + 1][i] = 1;
        }
        solve(1, 1, 1);

        cout << ans << '\n';
        return 0;
}
```