T4 - Constructive 题解

题目简述

题目大意:
给定 N 种二维向量 $v_i = (a_i, b_i)$ 及其代价 c_i，每种向量可无限次使用。求构造出目标向量 $u = (x, y)$ 的最小总代价。

- 向量数量 $N \leq 90$。
- 目标坐标 $x, y \leq 10^9$。
- 向量分量 $a_i, b_i \leq 10$。
- 向量代价 $c_i \leq 10^9$。

The Key:
这是一个 **二维无界背包问题**，或者等价于 **二维网格上的最短路问题。**

子任务

Subtask 1: $N = 2, v_1 = (0, 1), v_2 = (1, 0)$

对于这种情况来说，我们只能横着走或者竖着走，而且步长为 1，那么答案显然。

时间复杂度：$\mathcal{O}(1)$

Subtask 2 & 3: $x, y \leq 10^3$

由于坐标很小，我们可以考虑最短路解法。

对于平面上的点 $(i, j), 0 \leq i \leq x, 0 \leq j \leq y$，我们只需要向 $(i + a_k, j + b_k)$ 连一条长度为 c_k 的边，由于边权非负，那么这是个经典的最短路算法，我们可以用 `Dijkstra` 求解。

时间复杂度：$\mathcal{O}(N \times x \times y + x \times y \times \log(xy))$

Subtask 4: $x = y, x, y \leq 10^6, a_i = b_i$

那么，所有的向量都在 $f(x) = x$ 这条直线上，问题就退化为了一维的背包。用 a_i 凑出 x，代价为 c_i。

设 $dp[w]$ 为凑出 w 的最小代价，那么就有转移：$dp[w] = \min(dp[w - a_i] + c_i)$。

时间复杂度：$\mathcal{O}(N \times x)$

Subtask5: $x = y, a_i = b_i$

对于这个，虽然还是在一条直线上，但是坐标比较大，我们可以考虑对刚刚的背包优化，我们可以尝试同余的方式，先找出性价比比较高的 $a_{best}, v_{best}, c_{best}$，只对小范围进行预处理，然后枚举 DP 范围内的余数 r，如果 $x - r$ 能被 a_{best} 整除的话，就直接：

$$Ans = dp[r] + \frac{x - r}{a_{best}} \times c_{best}$$

取最小的值即可。

时间复杂度：$\mathcal{O}(N \times L)$

正解

解法 1:

这个解法是**官方题解**中给出的。

当 $x, y \leq 10^9$ 的时候，我们无法对所有的坐标进行枚举，我们这里做一个**引理**

> **几何中点引理**
> 如果一组短向量的和为 (x, y)，那么我们通过调整顺序，一定可以把他们分成两半，使得每一半的和都在 $(\frac{x}{2}, \frac{y}{2})$ 的常数的范围内。

我们定义 $f(x, y)$ 为凑出 (x, y) 的最小代价，那么我们可以把问题规模减半，(x, y) 是大概接近 $(\frac{x}{2}, \frac{y}{2})$ 的向量相加而成。

转移方程：

$$f(x, y) = \min_{\delta_x, \delta_y} \{ f(\lfloor \frac{x}{2} \rfloor + \delta_x, \lfloor \frac{y}{2} \rfloor + \delta_y) + f(\lceil \frac{x}{2} \rceil - \delta_x, \lceil \frac{y}{2} \rceil - \delta_y) \}$$

其中 δ 是偏移量，范围约为 $[-10, 10]$。

我们可以用记忆化搜索，直接用 `unordered - map` 或数组存就行（我没有卡 `umap`。

时间复杂度：$O(\delta^4 \cdot \log(\max(x, y)))$

```cpp
//官方 std
#include <bits/stdc++.h>
#define uwu return 0;

using namespace std;

#define fs first
#define sc second

const long long INF = 4e18;

const int MAX_LENGTH = 10;

long long one_vec[MAX_LENGTH + 1][MAX_LENGTH + 1];

unordered_map <long long, long long> dp;

const int MAX_X = 1'000'000'001;

long long DP(long long x, long long y){
    long long pos = x * MAX_X + y;
    if(dp.count(pos))
        return dp[pos];

    long long tmp = INF;
    if(x ≤ MAX_LENGTH && y ≤ MAX_LENGTH)
        tmp = one_vec[x][y];

    for (long long dx = -MAX_LENGTH; dx ≤ MAX_LENGTH; dx++){
        for (long long dy = -MAX_LENGTH; dy ≤ MAX_LENGTH; dy++){
            long long x1 = x / 2 + dx, y1 = y / 2 + dy;
            long long x2 = x - x1, y2 = y - y1;
            if(min({x1, x2, y1, y2}) < 0)
                continue;
            if(x1 + y1 == 0 || x2 + y2 == 0)
                continue;
            tmp = min(tmp, DP(x1, y1) + DP(x2, y2));
        }
    }
```

```
40      dp[pos] = tmp;
41      return dp[pos];
42  }
43
44  int main(){
45      int N;
46      long long x, y;
47      cin >> N >> x >> y;
48
49      for (int i = 0; i ≤ MAX_LENGTH; i++){
50          for (int j = 0; j ≤ MAX_LENGTH; j++){
51              one_vec[i][j] = INF;
52          }
53      }
54
55      for (int a, b, c; N--;){
56          cin >> a >> b >> c;
57          one_vec[a][b] = c;
58      }
59
60      long long ans = DP(x, y);
61
62      if(ans ≥ INF)
63          cout << "-1\n";
64      else
65          cout << ans << '\n';
66      uwu;
67  }
```

解法 2:

这个解法是我昨天 2026 − 2 − 7 号在写这个题解的时候受部分分启发想出来的。

> **Trick**:
>
> 我们想一下，最终的 (x, y) 一定是先通过两个向量，或者说是基底的大量使用，在加上原点附近小范围微调达到的。即:
>
> $$(x, y) = \underbrace{(dx, dy)}_{\text{微调部分}} + \underbrace{k_i v_i + k_j v_j}_{\text{基底部分}}$$

上面这个在说一件什么事呢？就是说，我们在做一件事的时候，总是很快的去接近结果，然后在很靠近结果的时候再微调。形象的理解就是赛车的时候，我们会在过弯之前快速接近弯道，等到接近弯道再减速漂移。

我们只需要预处理一个 $K \times K$ 的区间（本题的 $K \in [30, 200]$ 都可以通过），计算原点到达每个点 (dx, dy) 的最小代价。那直接就用 `Dijkstra` 预处理一下（此处的思路与 subtask 2 & 3 类似）

然后我们枚举所有可能的向量对 (v_i, v_j) 作为基底，对于每个基底，都枚举没一个预处理过的微调点 (dx, dy)，通过解二元一次方程组的方式，判断剩下的距离 $(x - dx, y - dy)$ 是否能被 v_i, v_j 用非负整数线性表达出来即可。

时间复杂度：$\mathcal{O}(N \times K \log k + N^2 \times K^2)$

```cpp
#include<iostream>
#include<queue>
#include<cstring>
#include<algorithm>
using namespace std;

typedef long long ll;
const ll INF = 4e18;

struct vec{
    int a, b;
    ll c;
}v[105];

struct node{
    int x, y;
    ll d;
    bool operator > (const node& o)const{
        return d > o.d;
    }
};

int n;
ll tx, ty, res = INF;
ll dp[205][205];

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
```

```
    cin >> n >> tx >> ty;
    for(int i = 1;i ≤ n;i ++){
        cin >> v[i].a >> v[i].b >> v[i].c;
    }
    for(int i = 0;i ≤ 30;i ++){
        for(int j = 0;j ≤ 30;j ++){
            dp[i][j] = INF;
        }
    }
    priority_queue<node, vector<node>, greater<node>> pq;
    dp[0][0] = 0;
    pq.push({0, 0, 0});
    while(!pq.empty()){
        node cur = pq.top(); pq.pop();
        if(cur.d > dp[cur.x][cur.y]) continue;
        for(int i = 1;i ≤ n;i ++){
            int nx = cur.x + v[i].a, ny = cur.y + v[i].b;
            if(nx ≤ 30 && ny ≤ 30 && dp[nx][ny] > cur.d +
v[i].c){
                dp[nx][ny] = cur.d + v[i].c;
                pq.push({nx, ny, dp[nx][ny]});
            }
        }
    }
    for(int i = 1;i ≤ n;i ++){
        for(int j = i;j ≤ n;j ++){
            ll det = 1LL * v[i].a * v[j].b - 1LL * v[j].a *
v[i].b;
            for(int dx = 0;dx ≤ 30;dx ++){
                for(int dy = 0;dy ≤ 30;dy ++){
                    if(dp[dx][dy] == INF) continue;
                    ll rx = tx - dx, ry = ty - dy;
                    if(rx < 0 || ry < 0) continue;
                    if(det ≠ 0){
                        ll na = rx * v[j].b - ry * v[j].a;
                        ll nb = ry * v[i].a - rx * v[i].b;
                        ll d = det;
                        if(d < 0) d = - d, na = - na, nb = - nb;
                        if(na ≥ 0 && nb ≥ 0 && na % d == 0 &&
nb % d == 0){
                            res = min(res, dp[dx][dy] + (na / d)
* v[i].c + (nb / d) * v[j].c);
                        }
```

```
                    }
                    else{
                        if(1LL * v[i].a * ry == 1LL * v[i].b *
rx){
                            ll k = - 1;
                            if(v[i].a ≠ 0 && rx % v[i].a == 0) k
= rx / v[i].a;
                            else if(v[i].a == 0 && v[i].b ≠ 0 &&
ry % v[i].b == 0) k = ry / v[i].b;
                            else if(v[i].a == 0 && v[i].b == 0 &&
rx == 0 && ry == 0) k = 0;
                            if(k ≥ 0) res = min(res, dp[dx][dy]
+ k * v[i].c);
                        }
                    }
                }
            }
        }
    }
    if(res ≥ INF) cout ≪ - 1 ≪ '\n';
    else cout ≪ res ≪ '\n';
    return 0;
}
```