题解

小球下落

```cpp
#include <bits/stdc++.h>
using namespace std;
int x, y, n, times;
struct node {
    int x1, x2, y1;
} zaw[101];
int cmp(node a, node b) { return a.y1 > b.y1; } // 按高度从大到小排序

int main() {
     freopen("cball.in", "r", stdin);
    freopen("cball.out", "w", stdout);
    cin >> x >> y >> n;
    times += y;  // 下落到地面的基础时间
    for (int i = 1; i <= n; i++)
        cin >> zaw[i].y1 >> zaw[i].x1 >> zaw[i].x2;
    sort(zaw + 1, zaw + 1 + n, cmp);

    for (int i = 1; i <= n; i++) {
        if (y < zaw[i].y1) continue; // 只考虑在起点下方的障碍物
        if (x >= zaw[i].x1 && x <= zaw[i].x2) {
            times += 5;   // 延迟
            x = zaw[i].x2; // 小球移到障碍物右端
        }
    }
    cout << times;
    return 0;
}
```

数组选数

40 分做法

可以直接 $O(n^3)$ 暴力枚举每一个三元组 (i,j,k)，然后打擂台找到最小值。

```cpp
#include <bits/stdc++.h>
using namespace std;

const int maxn = 1e5 + 5;
int a[maxn], n;
long long ans = 1e18;  // 初始化为一个极大值
```

```cpp
int main() {
    // 读入
    cin >> n;
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
    }

    // 三重循环枚举所有可能的 (i, j, k)
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j) {
            for (int k = 1; k <= n; ++k) {
                // 保证 i, j, k 互不相同
                if (i != j && j != k && i != k) {
                    long long val = (long long)(a[i] + a[j]) * a[k];
                    ans = min(ans, val); // 更新最小值
                }
            }
        }
    }

    cout << ans << endl;
    return 0;
}
```

100 分做法

$n \leq 10^5$，需要我们找到一定的规律或者性质。想要使得三个数 $(a_i + a_j) \times a_k$ 最小。很多人容易想到对数组从小到大排序，最终 $a[1] \times (a[2] + a[3])$ 最小。

但是因为数组中的值可能有 0，也可能有负数，所以我们需要分类讨论：

```
1.全是正数：a1*(a2+a3) 最小*（次小+2次小）
2.全是负数：an*（a[n-1]+a[n-2] ）负数绝对值最小*（2个绝对次小的）
3.1负2正:a1*(a[n-1]+a[n]) 最小的负数*（2个最大的正数）
4.2负1正：(a1+a2)*a[n]（2个绝对值最大的负数相加）* 绝对值最大的正数
取4个情况最小的
```

当然，我们实现的时候，也可以简单对这些情况取 min，去避免分类讨论。

```cpp
#include <bits/stdc++.h>
using namespace std;

const int maxn = 1e5 + 5;
int a[maxn], n, ans;

int main(){
    // 输入数组长度
    cin >> n;
```

```
    // 输入数组元素
    for(int i = 1; i <= n; ++i) cin >> a[i];

    // 将数组升序排序，方便直接取极大值和极小值
    sort(a + 1, a + 1 + n);

    // 情况1：三个最小值
    ans = min(a[1] * (a[2] + a[3]),
              a[n] * (a[n - 1] + a[n - 2]));  // 情况2：三个最大值

    // 再比较情况3 和 情况4
    ans = min(ans, min(a[1] * (a[n - 1] + a[n]),    // 最小值 * (两个最大值的和)
                       (a[1] + a[2]) * a[n]));    // (两个最小值的和) * 最大值

    // 输出最终结果
    cout << ans;
    return 0;
}
```

序列归零

方法1

可以先了解 [NOIP2018 提高组] 铺设道路、[NOIP2013 提高组] 积木大赛 思路。
与以上两题不同的是，本题可能有正、负数；那么可以转化为正、负两个序列，分别按以上两题的思路求解；两个序列ans 求和即可。

```
#include <bits/stdc++.h>
using namespace std;
int z[500010],f[500010];
long long ans=0;
int main() {
    int n,x;
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        scanf("%d",&x);
        if(x>0) z[i]=x;
        if(x<0) f[i]=-x;
    }
    for(int i=1;i<=n;i++){
        if(z[i]-z[i-1]>0) ans=ans+z[i]-z[i-1];
        if(f[i]-f[i-1]>0) ans=ans+f[i]-f[i-1];
    }
    printf("%lld",ans);
    return 0;
}
```

方法2

这个问题其实是一个经典的“区间加减”最小操作次数问题，可以用**差分思想**来解。

差分序列思想：

定义 $a_0 = 0$（方便处理）。
我们观察一下操作的特点：

- 对于任意一个连续区间 ([l, r])做 +1 操作，相当于：

  a_l 增加 1，　a_{r+1} 没有变化

  也可以反过来理解为差分：只关心相邻元素的增减。
- 最终，我们需要“消掉”整个序列的变化，也就是：

  最小操作次数 $= \sum_{i=1}^{n+1} \max(0, d_i) = \sum_{i=1}^{n+1} \max(0, a_i - a_{i-1})$
  其中$a_0 = a_{n+1} = 0$

例1

```
4
1 1 1 2
```

数组：1, 1, 1, 2
差分（相对于前一项，包括首尾的 0）：

- i=1: a1 - 0 = 1
- i=2: 1 - 1 = 0
- i=3: 1 - 1 = 0
- i=4: 2 - 1 = 1
- i=5: 0 - 2 = -2

正的差分：1, 1
和 = 2 ✅

例 2

```
3
−1 1 −1
```

数组：-1, 1, -1
差分：

- i=1: -1 - 0 = -1
- i=2: 1 - (-1) = 2
- i=3: -1 - 1 = -2
- i=4: 0 - (-1) = 1

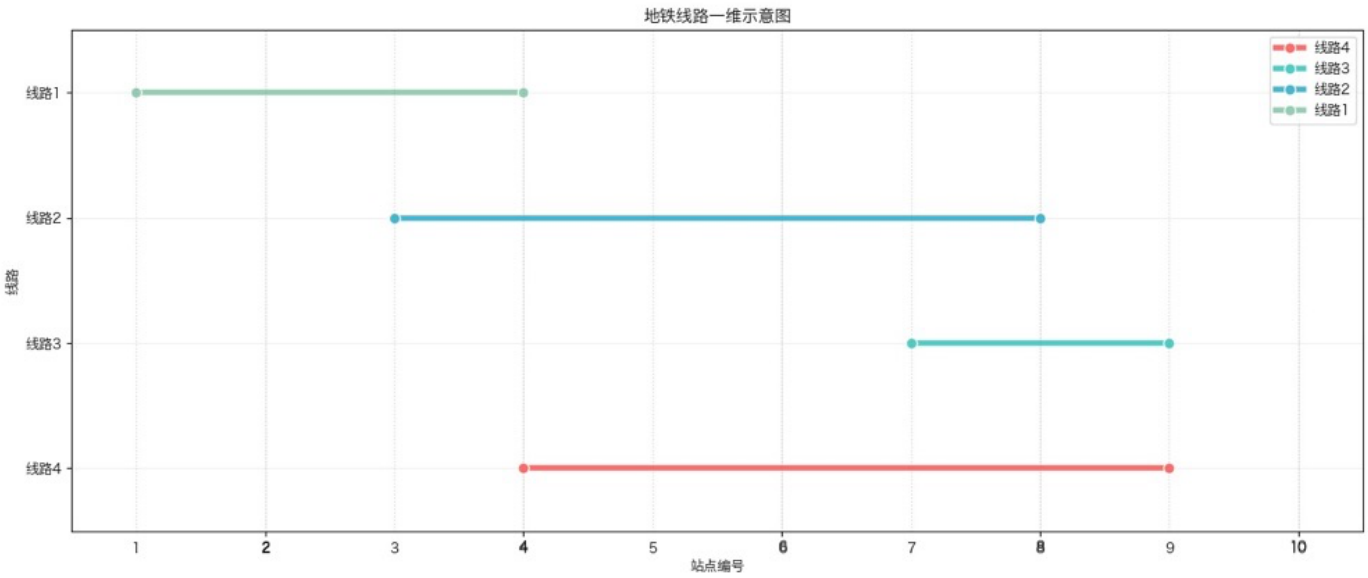
正的差分：2, 1
和 = 3 ✅

```
#include <bits/stdc++.h>
using namespace std;
```

```cpp
int a[500005];
int main() {
    /*
    a[0] 默认是 0（全局数组初始化为 0）
    a[n+1] 也视为 0（循环到 n+1 时，a[n+1] 未赋值，但全局数组是 0，所以没问题）
    循环从 i=1 到 n+1，每次比较 a[i] 和 a[i-1]，如果增加，就把增加量加到答案里
    最终输出这个和
    */
    int n;
    long long ans = 0;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
    }
    for (int i = 1; i <= n + 1; i++) {
        if (a[i] > a[i - 1]) {
            ans += a[i] - a[i - 1];
        }
    }
    cout << ans << endl;
    return 0;
}
```

地铁站

题解



注意到从 s 走到 t 和从 t 走到 s 是等价的。那么我们就假设所有的询问都满足 $s < t$。

我们每次肯定是坐到当前线路最靠右的地铁站，然后换乘。

这样我们就可以预处理 rm_i 表示从 i 出发不经过换乘能够到达的最靠右的站点（编号最大的地铁站）是什么。

- 初始时令 $rm_i = i$，然后对于每一条地铁站路线 $[l, r]$，都有 $rm_l = \max(rm_l, r)$，这样我们求一个前缀

max 就能得到正确的 r_i。

求出 rm_i 后，我们可以每次直接暴力从 i 跳到 rm_i 表示坐地铁线路坐到能够到达的最右边的站，显然这样的时间复杂度为 $O(n^2)$。

暴力跳：从 rm[s] 跳到s出发能走到的最右端点，再次从rm[rm[s]]继续暴力跳。这样最坏情况：每次跳一个单位导致超时。

```
#include <bits/stdc++.h>
using namespace std;
//40分
const int maxn = 5e5 + 5;
int n, m, q;
int rm[maxn]; // 记录每个站最远可达

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

    cin >> n >> m;
    for(int i = 1; i <= n; ++i) rm[i] = i;

    // 处理每条线路
    for(int i = 0; i < m; ++i){
        int l, r;
        cin >> l >> r;
        rm[l] = max(rm[l], r);
    }

    // 构建前缀最大覆盖
    for(int i = 2; i <= n; ++i){
        rm[i] = max(rm[i], rm[i-1]);
    }

    cin >> q;
    while(q--){
        int s, t;
        cin >> s >> t;
        if(s > t) swap(s, t);

        int ans = 0;
        int cur = s;         // 当前覆盖到的位置
        int far = rm[s]; // 当前线路能到的最远位置

        while(cur < t){
            if(far >= t){    // 可以直接到达终点
                break;
            }
            int next_far = far;
            // 扫描当前覆盖区间内能到的最远站
```

```
            for(int i = cur + 1; i <= far; ++i){
                next_far = max(next_far, rm[i]);
            }
            if(next_far == far){ // 无法前进
                ans = -1;
                break;
            }
            ++ans;       // 换乘一次
            cur = far;
            far = next_far;
        }
        cout << ans << '\n';
    }
    return 0;
}
```

倍增优化

我们可以考虑倍增，计算出 $f[i][j]$ 表示从 i 开始跳 2^j 个 rm_i 能够到达的站。

- 预处理 $f[i][0] = rm_i$
- 递推式子 $f[i][j] = f[f[i][j-1]][j-1]$

这样，对于每一组询问 s, t。我们只需要从大到小枚举 j，如果 $f[s][j] < t$ 就说明还没有到，令答案加上 2^j，并且令 $s = f[s][j]$。这样我们就能够在 $O(\log n)$ 的时间复杂度内求出一次询问的答案。

总时间复杂度为 $O((n+q)\log n)$。

```
#include <bits/stdc++.h>
using namespace std;

// 最大地铁站数量
const int maxn = 5e5 + 5;

// f[i][j] 表示从站 i 出发，通过最多 2^j 次换乘能到达的最远地铁站
int f[maxn][25], n, m, q;//2^20>10^6

int main() {
    // 读入/写入文件
    freopen("esubway.in", "r", stdin);
    freopen("esubway.out", "w", stdout);
        //加速
    cin.tie(0);
    // 读入地铁站数量 n 和地铁线路数量 m
    cin >> n >> m;

    // 初始化 f[i][0]
    for (int i = 1; i <= n; ++i)
        f[i][0] = i;  // 起点站自身最远能到自己rm[i]

    // 读入每条地铁线路，并更新 f[l][0] 为不换乘可到达的最右端
```

```cpp
    while (m--) {
        int l, r;
        cin >> l >> r;
        f[l][0] = max(f[l][0], r);  // 从站 l 不换乘能到的最远站
    }

    // 前缀最大值更新求出rm[i]
    // 经过前缀最大值处理后，f[i][0] 表示从 i 出发不换乘能到的最右边的站
    for (int i = 1; i <= n; ++i)
        f[i][0] = max(f[i - 1][0], f[i][0]);

    // 倍增预处理
    // f[i][j] 表示从站 i 出发，通过最多 2^j 次换乘能到达的最远站
    for (int j = 1; j <= 20; ++j) {
        for (int i = 1; i <= n; ++i) {
            f[i][j] = f[f[i][j - 1]][j - 1];//递推
        }
    }

    // 读入询问次数 Q
    cin >> q;
    while (q--) {
        int s, t;
        cin >> s >> t;

        // 保证 s < t，统一向右走处理
        if (s > t) swap(s, t);

        // 如果从 s 出发，即便换乘 2^20 次仍到不了 t，说明无法到达
        if (f[s][20] < t) {
            cout << "-1\n";
            continue;
        }

        int ans = 0;  // 换乘次数

        // 倍增法计算最少换乘次数，从大到小尝试 2^i 次跳跃

        for (int i = 20; i >= 0; --i) {
            if (f[s][i] < t) {   // 如果通过 2^i 次跳跃仍未到达 t
                ans += (1 << i); // 累加换乘次数
                s = f[s][i];     // 更新当前右端点位置
            }
        }

        cout << ans << '\n';
    }

    return 0;
}
```