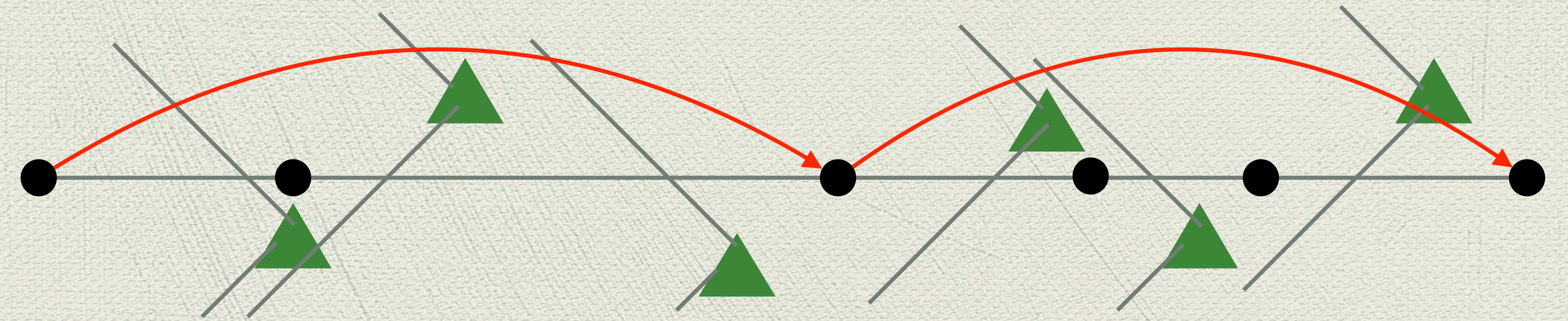
F: Dangerous Delivery
- 危険な輸送 -

原案：井上

解答：井上

問題概要

- 一直線に並んだ都市1から都市Nへ移動したい
- 都市はM人の敵に見張られている
- 1日の移動には、出発地点の都市を見張っている敵の数×移動距離だけリスクが生じる
- D日までの移動で都市1から都市Nへ移動するリスクの和を最小化せよ



想定解法

- d日目に都市iを見張っている敵の数w[d][i]を求める

- DPによってd日目に都市iにいるときの最小リスクを求める
- dp[d+1][i] := min{dp[d][j] + w[d][j]*abs(p[i]-p[j])}

想定TLE解法

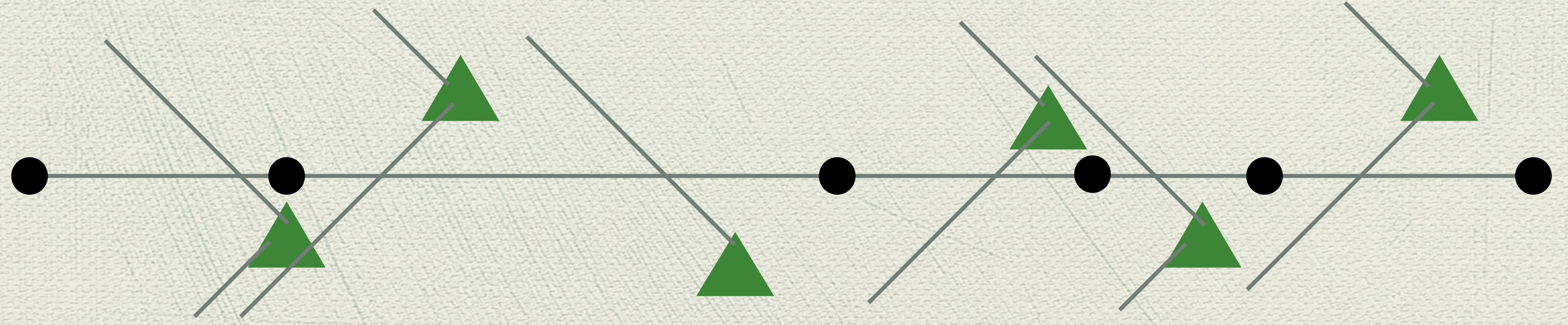
- d日目に都市iを見張っている敵の数w[d][i]を求める
  - それぞれの敵が見張っているか普通に判定: O(DNM)
- DPによってd日目に都市iにいるときの最小リスクを求める
- dp[d+1][i] := min{dp[d][j] + w[d][j]*abs(p[i]-p[j])}
  - 愚直にdpを更新: $O(DN^2)$

想定TLE解法

- d日目に都市iを見張っている敵の数w[d][i]を求める
  - 全ての敵について見張っているか判定: O(DNM)
- DPによってd日目に都市iにいるときの最小リスクを求める
- dp[d+1][i] := min{dp[d][j] + w[d][j]*abs(p[i]-p[j])}
  - 愚直にdpを更新: $O(DN^2)$

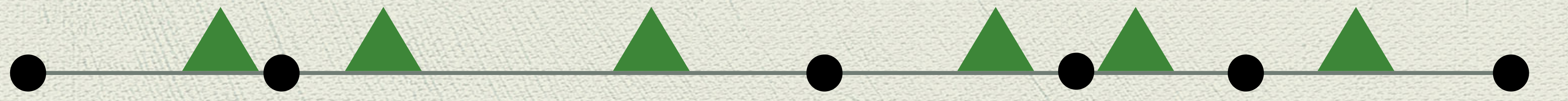
敵の処理

- 位置(x,y)にいる敵が監視できる範囲は(x-|y|,0)より左
    - x軸上に移動させて考えても良い



敵の処理

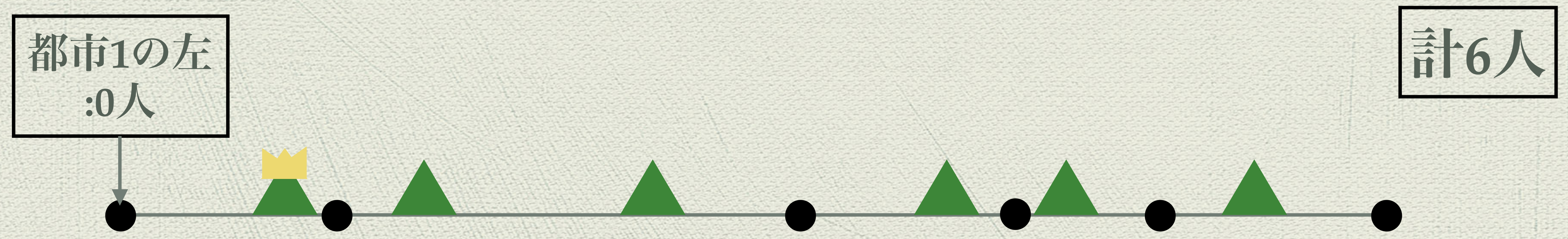
- 位置(x,y)にいる敵が監視できる範囲は(x-|y|,0)より左
    - x軸上に移動させて考えても良い



敵の処理

- 敵の位置をx軸に集め、ソートする
- 都市の右にいる敵の数=都市を監視する敵の数
- 左の都市から順に調べることで線形に計算できる

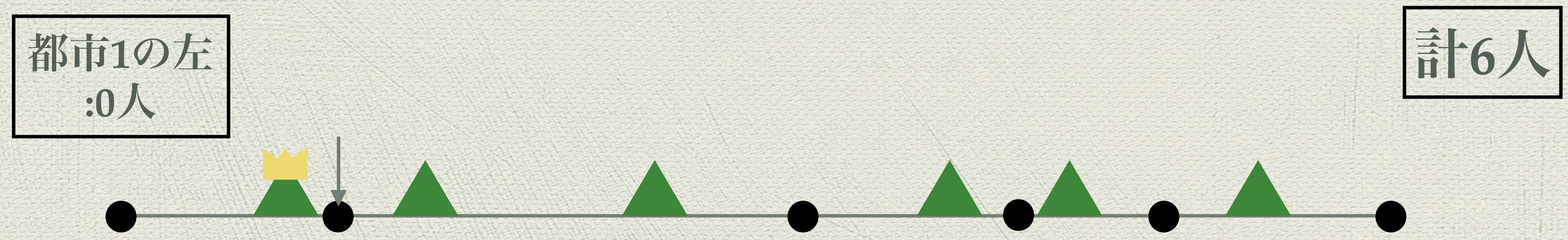
  = O(D(N+M))



敵の処理

- 敵の位置をx軸に集め、ソートする
- 都市の右にいる敵の数=都市を監視する敵の数
- 左の都市から順に調べることで線形に計算できる

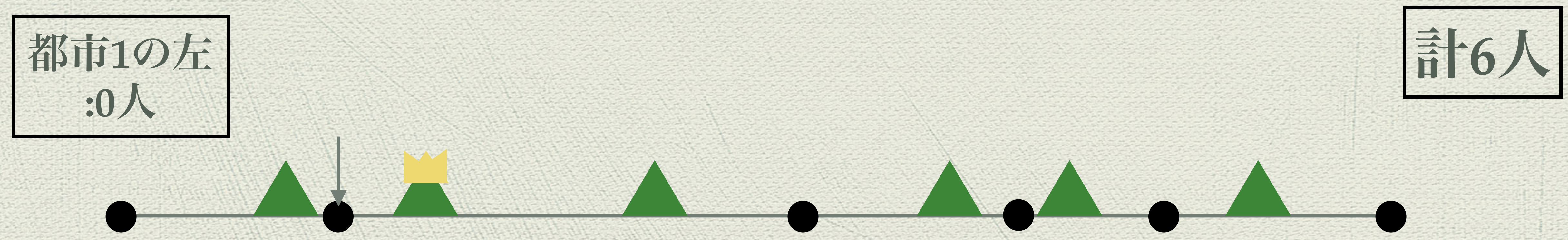
= O(D(N+M))



敵の処理

- 敵の位置をx軸に集め、ソートする
- 都市の右にいる敵の数=都市を監視する敵の数
- 左の都市から順に調べることで線形に計算できる

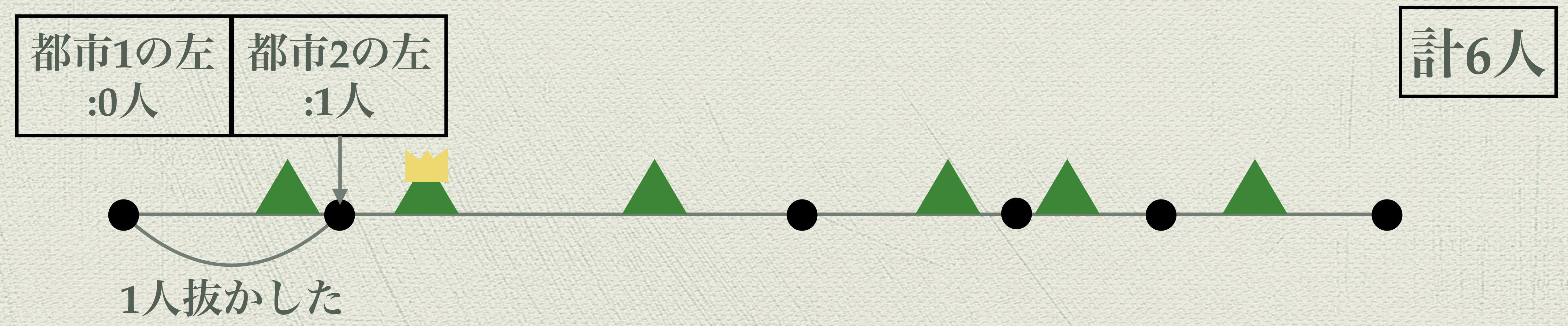
  $= O(D(N+M))$



敵の処理

- 敵の位置をx軸に集め、ソートする
- 都市の右にいる敵の数=都市を監視する敵の数
- 左の都市から順に調べることで線形に計算できる

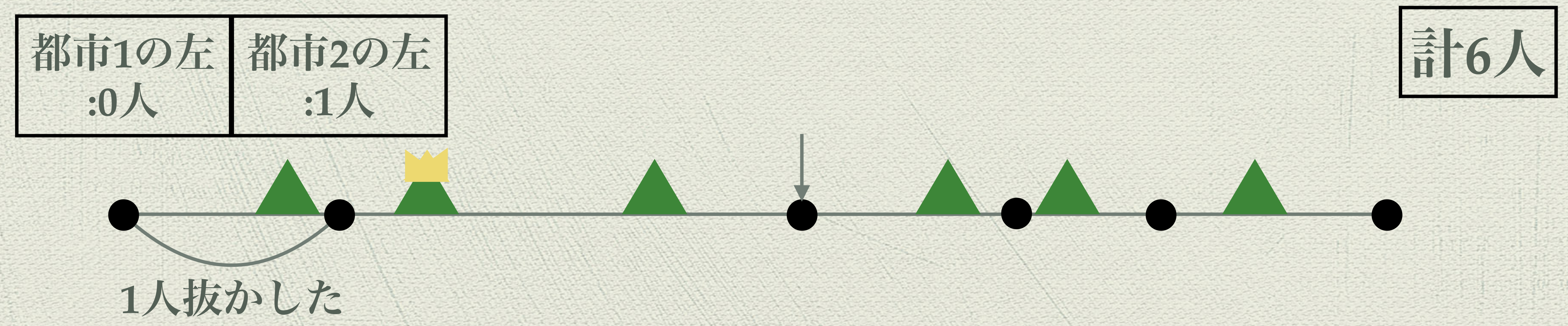
  $= O(D(N+M))$



敵の処理

- 敵の位置をx軸に集め、ソートする
- 都市の右にいる敵の数=都市を監視する敵の数
- 左の都市から順に調べることで線形に計算できる

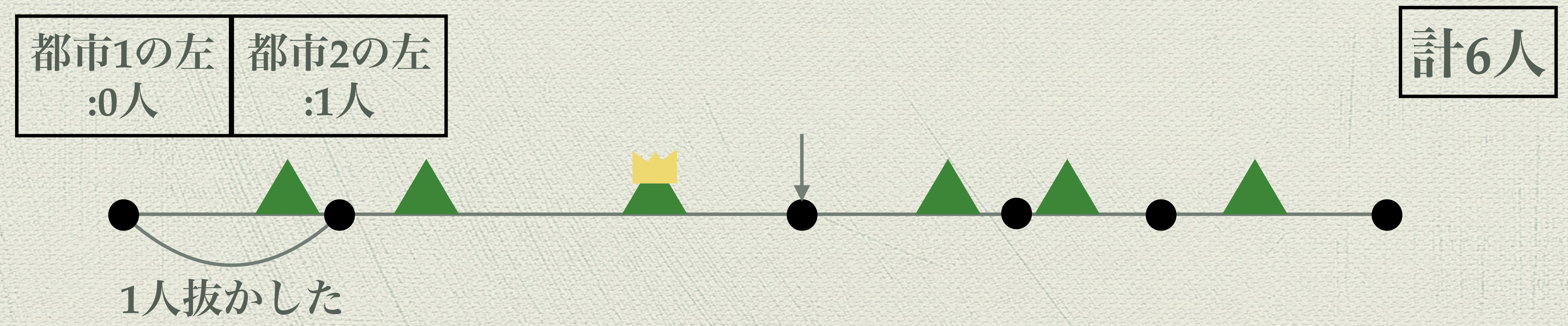
  = O(D(N+M))



敵の処理

- 敵の位置をx軸に集め、ソートする
- 都市の右にいる敵の数=都市を監視する敵の数
- 左の都市から順に調べることで線形に計算できる

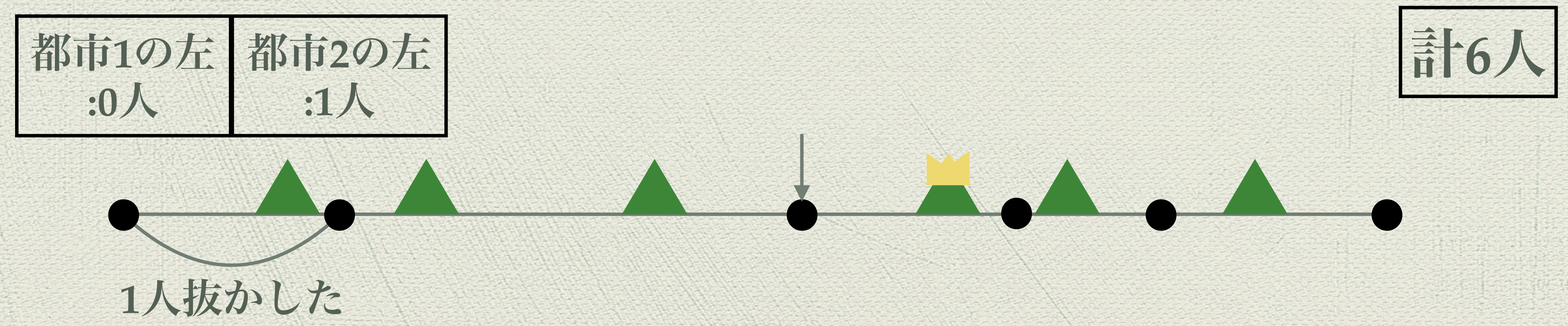
= O(D(N+M))



敵の処理

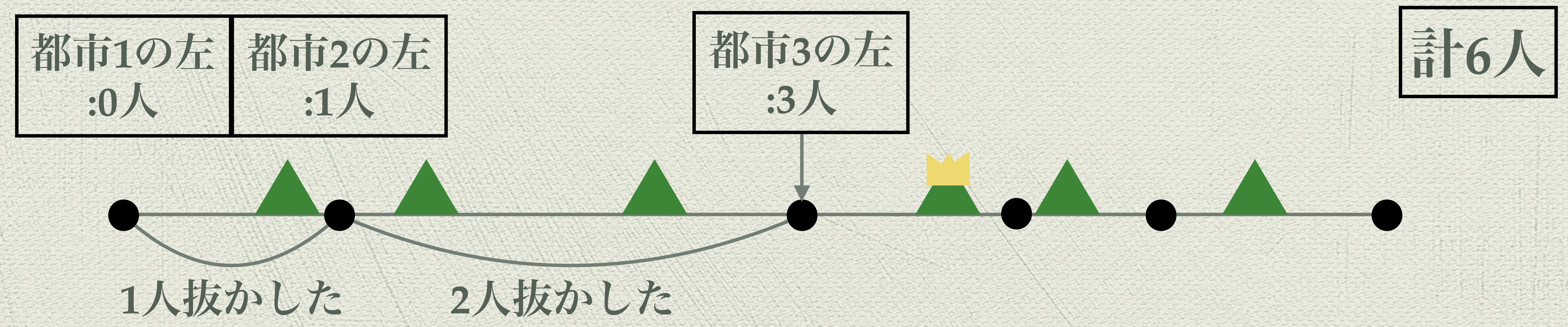
- 敵の位置をx軸に集め、ソートする
- 都市の右にいる敵の数=都市を監視する敵の数
- 左の都市から順に調べることで線形に計算できる

  = O(D(N+M))



敵の処理

- 敵の位置をx軸に集めソートする = O(M logM)
- 都市の右にいる敵の数=都市を監視する敵の数
- 左の都市・左の敵から順に調べることで線形で計算できる = O(D(N+M))



想定TLE解法

- d日目に都市iを見張っている敵の数w[d][i]を求める
  - 全ての敵について見張っているか判定: O(DNM)

- DPによってd日目に都市iにいるときの最小リスクを求める
- dp[d+1][i] := min{dp[d][j] + w[d][j]*abs(p[i]-p[j])}
  - 愚直にdpを更新: O(DN^2)

DPの高速化

- DPの式に注目し、変形を行う
  - dp[d+1][i] := min{dp[d][j] + w[d][j]*abs(p[i]-p[j])}

    後戻りは意味がない
  - dp[d+1][i] := min$_{j<i}${dp[d][j] + w[d][j]*abs(p[i]-p[j])}

    p[i]>p[j]
  - dp[d+1][i] := min$_{j<i}${dp[d][j] + w[d][j]*(p[i]-p[j])}

    展開・変形
  - dp[d+1][i] := min$_{j<i}${w[d][j]*p[i] + dp[d][j] - w[d][j]*p[j]}
- 赤：iの関数　青：iに関して定数

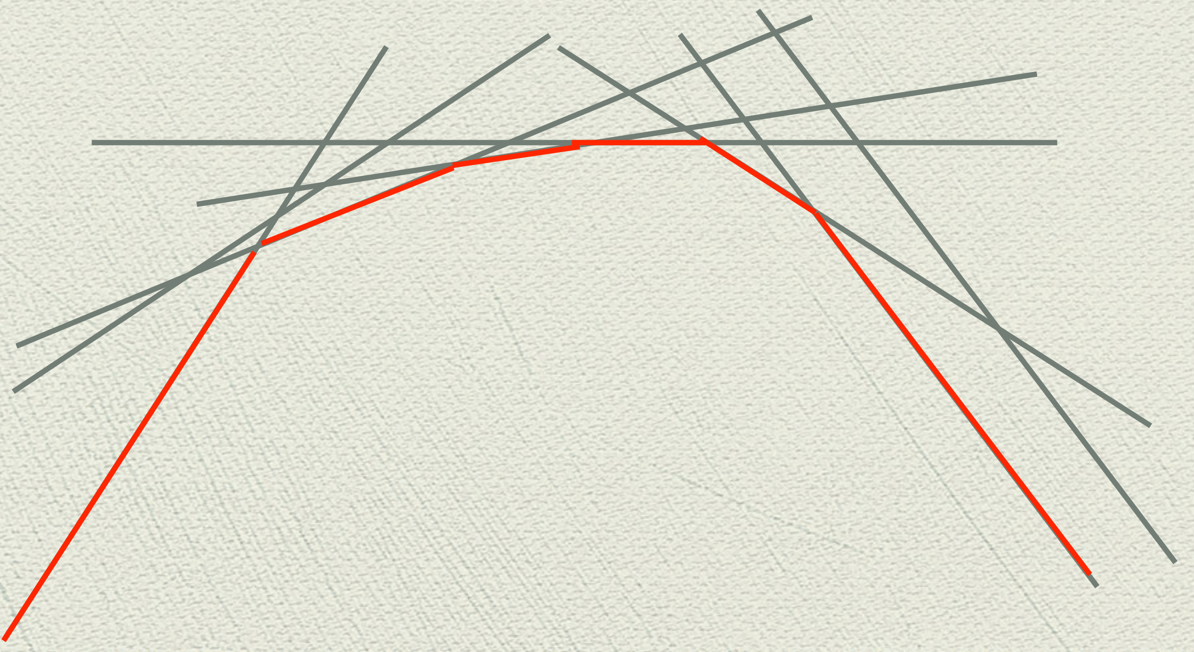
DPの高速化

- dp[d+1][i] := $\min_{j<i}$ {w[d][j]*p[i] + dp[d][j] - w[d][j]*p[j]}

- dp[d+1][x] := min{a * x + b}の形をしている
- すなわち、傾きw[d][j],切片dp[d][j]-w[d][j]*p[j]の直線i-1個からなる直線集合のうち、x=p[i]で最小値をとるものがわかればよい

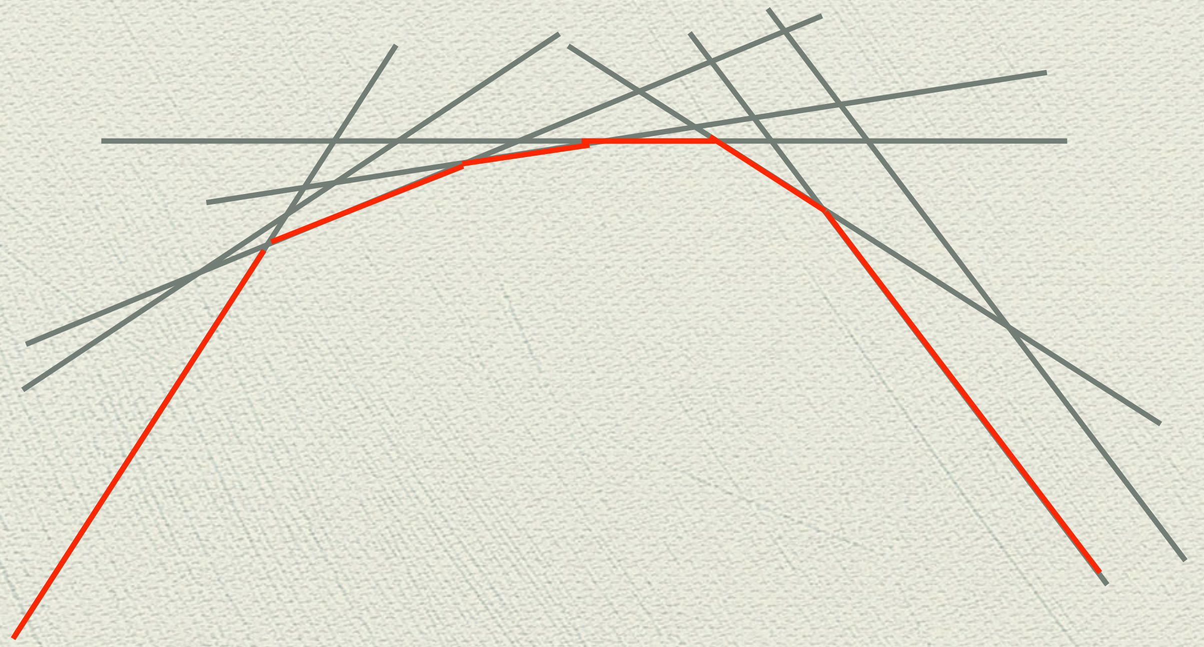
Convex Hull Trick

- dp[d+1][x] := $\min_{j<i}\{a * x + b\}$の形をしている
- 直線i-1個からなる
- 最小値を取る直線のみを考え、その直線の1次式にp[i]を代入することでO(1)で最小値が求まる



Convex Hull Trick

- 通常の場合、新たに直線を追加するときに「最小値となりうるか」を動的に管理するためO(logN)必要
- 今回は傾きw[d][j]が単調非増加なことを利用することで、(ならし)O(1)で管理できる
  - w[d][j]≧w[d][j+1]: 監視されている人数は右側の都市ほど少ない



Convex Hull Trick

- アバウトな流れ

for(i from 1 to N){

1. 直線集合から、x=p[i]について最小となる直線f(x)を計算(ならしO(1))
2. dpを更新： dp[d+1][i] = f(p[i])
3. 必要のない直線を削除し(ならしO(1))、傾きw[d][i]、切片dp[d][i]-w[d][i]*p[i]の直線を追加

}

Convex Hull Trick

- Convex Hull Trickにより、dp[d][i]を1≦i≦Nの区間で更新するのがO(N)でできるようになった
- よって、DPが全体でO(DN)

- Convex Hull Trickのより詳細な説明は下記参照
  - 蟻本2版 p.304 「K-Anonymous Sequence」
  - sune2さんのブログ: http://d.hatena.ne.jp/sune2/20140310/1394440369

writer解

- 井上(C++) 87行
- 井上(C++) 75行 (蟻本パクリ)

提出状況

- First Acceptance
    - on-site: 正答者なし
    - on-line: Komaki (01:59)
- 正答率 1/4 (25.0%)