019 - Pick Two

競プロ典型90問

難易度: ★6

2026-04-19 | Reo Kambayashi

キーワード: 区間DP / 最適部分構造

atcoder.jp/contests/typical90/tasks/typical90_s

問題概要

設定

- 長さ $2N$ の正整数列 A
- 操作: **残っている列から隣接する2要素を選んで取り除く**
- 操作コスト: 取り除く2要素の差の絶対値
- この操作を N 回繰り返して全要素を取り除く

目標

コストの**総和の最小値**を求めよ

制約

- $1 \leq N \leq 200$
- $1 \leq A_i \leq 10^6$
- **例**: $A = (6, 2, 3, 9, 8, 6) \rightarrow$ 最小コスト = 2

発想の過程 (1/3) — 素朴なアプローチと壁

最初の思いつき

「隣接要素のペアを全列挙して貪欲に選ぶ」

- 毎回「最もコストの低い隣接ペア」を選ぶ Greedy
- → 局所最適が全体最適を保証しない（反例が作れる）

次のアプローチ: 操作順の全探索

- N 回の操作の順列: $N!$ 通り → **絶対に無理**

> 「隣接要素を取り除く」と隣が変わるため、操作順に依存して選択肢が変化する。
>
> 動的に変化する列の操作を「どう整理するか」が課題。

発想の過程 (2/3) — 着眼点の発見

操作の見方を変える

A_l と A_{r-1} を「最後に取り除くペア」として固定すると、間の要素 $A_{l+1}, \ldots, A_{r-2}$ の処理は独立した部分問題になる

> **核心的な観察**: どんな順序で取り除いても、
> 最終的に区間 $[l, r)$ を取り除く際の「ペアリング」は再帰的に決まる。
> → $dp[l][r]$ = 区間 $[l, r)$ を全取り除くコスト、という**区間DP**の形に帰着！

遷移の発見

区間 $[l, r)$ で A_l は必ず何かと組まれる。A_l のパートナーは:

- A_{r-1}: 両端をペアにして残りを再帰処理
- または区間を $[l, m)$ と $[m, r)$ に分割してそれぞれ独立に処理

気づきへのルート

問いを変える

「どの順番で取り除くか？」ではなく **「最後に取り除くのは誰と誰か？」**

- 順向きに考える（最初に何を取る）→ 選択肢が連鎖して $N!$ 通りに爆発
- **逆向きに考える**（最後に何が残るか）→ 最後のペアを固定すれば内側は独立、と見える

小さい例で手を動かす（$N = 2$、長さ4）

$A = (a,\ b,\ c,\ d)$ で「最後のペア」として取り除けるのは？

- (a, d) を最後に → 先に (b, c) を取り除く。内側は独立に処理できる
- (a, b) と (c, d) に分割 → それぞれ独立に処理できる

> $N = 2$ で確認できた「最後のペアを固定 → 内側は独立」という構造を、区間 $[l, r)$ に一般化すると**区間DP**になる

発想の過程 (3/3) — 正当性の直感

なぜこれで正しいか？

最適部分構造の成立: 区間 $[l, r)$ の最適解は、A_l のパートナーを固定したときの「残り部分の最適解」に依存する。パートナーの選び方が独立部分問題を生み出す。

> 「区間の両端を最後に取り除く」か「分割して独立処理」という2択が、
> 全ての可能な操作順をカバーしている → 最適解を必ず含む。

計算量の確認

- 区間数: $O(N^2)$（偶数長のみ）
- 各区間の遷移: $O(N)$（分割点ループ）
- → $O(N^3)$、$N \leq 200$ なら $\approx 8 \times 10^6$ → **余裕で間に合う**

着眼点・考察

問題の構造

「隣接要素を取り除く」操作は、取り除いた後の要素がつながる点がポイント

> **重要な観察**: どんな順序で取り除いても、最終的に取り除かれるペアは**区間 $[l, r]$ を再帰的に分解**したものと考えられる

区間DP の着想

- 区間 $[l, r]$（長さ偶数）を全部取り除くコストを $dp[l][r]$ と定義
- 最後に取り除かれるペアは (A_l, A_{r-1}) か、あるいは区間を2つに分割

> 最後に取り除かれるペアを固定すると、残りは独立した部分問題になる

区間DP とは（1/2） — 定義と直感

定義

区間 $[l, r]$ に対する最適値を、**より小さい区間の結果を組み合わせて**求める動的計画法。

適用条件

- 配列に対して「区間」単位で問題が分解できる
- 区間内の部分問題が**独立**（内側の処理が外側に依存しない）
- **最適部分構造**が成立する

直感的なイメージ

> 長い区間の最適解は「どこで分割するか」で決まる。短い区間の答えが既知なら、分割点を全探索するだけで求まる。だから**短い区間から順に埋める**。

区間DP とは（2/2） — 遷移と計算量

一般的な遷移式

$$dp[l][r] = \min_{l<m\le r}\left(dp[l][m] + dp[m][r] + \mathrm{cost}(l, m, r)\right)$$

区間 $[l, r]$ を分割点 m で左右に分け、両者の最適値と結合コストの和を最小化する。

計算順序

区間長の昇順（2, 3, 4, … の順）にループして埋める。

```
for len in 2..N:
    for l in 0..N-len:
        r = l + len
        for m in l+1..r:   // 分割点を全探索
            dp[l][r] = min(dp[l][r], dp[l][m] + dp[m][r] + cost)
```

計算量

区間数 $O(N^2)$ × 分割点ループ $O(N)$ → **合計 $O(N^3)$**

アルゴリズム選択

なぜ区間DP か？

- 区間 $[l, r]$ を取り除く問題は、より小さい区間に分解できる
- **最適部分構造**が成立
- $N \leq 200$ なので長さは最大 $400 \to O(N^3)$ で間に合う

DP の定義

$$dp[l][r] = \text{区間 } [l, r) \text{ を全て取り除く最小コスト}$$

（ただし区間の長さ $r - l$ は偶数のみ定義）

> **なぜ半開区間 $[l, r)$ か？** 区間長 $= r - l$、分割が $[l, m) + [m, r)$ でずれなし、右端要素は A_{r-1}（r は区間外）。

基底

$dp[l][l] = 0$（空区間）、　$dp[l][l+2] = |A_l - A_{l+1}|$

解法の詳細（1/2）

遷移式

区間 $[l, r)$ の取り除き方:

パターン1: A_l と A_{r-1} を最後に取り除くペアにする

$$dp[l][r] \leftarrow dp[l+1][r-1] + |A_l - A_{r-1}|$$

パターン2: 区間を $[l, m)$ と $[m, r)$ に分割

$$dp[l][r] \leftarrow dp[l][m] + dp[m][r] \quad (l < m < r,\ m - l\text{が偶数})$$

遷移の意味

- パターン1: A_l は必ず A_{r-1} と対になる
- パターン2: 左部分と右部分を独立に処理

解法の詳細（2/2）

例: $A = (6, 2, 3, 9, 8, 6)$（N=3）

$dp[l][r]$	候補値	最小
$dp[0][2]$	$\lvert 6 - 2 \rvert$	4
$dp[1][3]$	$\lvert 2 - 3 \rvert$	1
$dp[2][4]$	$\lvert 3 - 9 \rvert$	6
$dp[3][5]$	$\lvert 9 - 8 \rvert$	1
$dp[4][6]$	$\lvert 8 - 6 \rvert$	2
$dp[0][4]$	$3{+}1 = 4, \quad 4{+}6 = 10$	4
$dp[2][6]$	$3{+}1 = 4, \quad 6{+}2 = 8$	4
$dp[0][6]$	$0{+}2 = 2, \quad 4{+}4 = 8, \quad 4{+}2 = 6$	2

実装のポイント

```cpp
#include <bits/stdc++.h>
using namespace std;
const long long INF = 1e18;

int main() {
    int N; cin >> N;
    int sz = 2 * N;
    vector<long long> A(sz);
    for (auto& a : A) cin >> a;

    vector<vector<long long>> dp(sz+1, vector<long long>(sz+1, INF));

    // 基底: 長さ0の区間
    for (int i = 0; i <= sz; i++) dp[i][i] = 0;

    // 長さ2, 4, 6, ..., 2N の区間を順に処理
    for (int len = 2; len <= sz; len += 2) {
        for (int l = 0; l + len <= sz; l++) {
            int r = l + len;
            // パターン1: A[l] と A[r-1] を最後に取り除く
            if (dp[l+1][r-1] != INF)
                dp[l][r] = min(dp[l][r], dp[l+1][r-1] + abs(A[l] - A[r-1]));
            // パターン2: 分割
            for (int m = l+2; m < r; m += 2)
                if (dp[l][m] != INF && dp[m][r] != INF)
                    dp[l][r] = min(dp[l][r], dp[l][m] + dp[m][r]);
        }
    }
    cout << dp[0][sz] << endl;
}
```

計算量分析

時間計算量

- 区間の数: $O(N^2)$（長さ2, 4, ..., 2Nの偶数長区間）
- 各区間の遷移: $O(N)$（分割点 m のループ）
- **合計: $O(N^3)$**

$N \leq 200$ なので $200^3 = 8 \times 10^6 \rightarrow$ **十分間に合う**

空間計算量

- DPテーブル: $O(N^2)$

区間DPの典型パターン

問題	DP定義	遷移
石の分割	区間コスト	全分割点
Pick Two	区間削除コスト	両端ペア or 分割

類題（区間DP）

レベル	問題	ポイント
★ 入門	EDPC N - Slimes	隣接結合コスト最小化（教科書問題）
★ 入門	EDPC L - Deque	両端取得ゲーム × 区間DP
★ 入門	鉄則本 A21 - Block Game	ブロック除去（Pick Two 類似）
★★ 中級	AOJ 1611 - ダルマ落とし	隣接ペア除去の最大化
★★ 中級	TDPC - イウィ	文字列の区間除去
★★★ 上級	ARC 168 D	区間更新の最大化
★★★ 上級	AGC 062 B	列の分割・挿入コスト最小化

まとめ

着眼点

隣接ペア取り除き最適化
→ **区間DP** に帰着

DP定義

dp[l][r] = 区間 [l, r) を
全取り除く最小コスト

遷移（2パターン）

① 両端 A_l と A_{r-1} をペアに
② 区間を [l,m) + [m,r) に分割

注意点・計算量

偶数長区間のみ定義
計算量 **$O(N^3)$**