JOI 2024/2025 春季トレーニング Day3 会議 (Conference) 解説

解説: ???

Step 0

問題概要

0 問題概要

- 長さ N の文字列 S があります
- S は A, B, C, ? からなります

- ? に A, B, C を入れて文字が変わる回数を少なくしたいです
- AABBACA なら AA | BB | A | C | A で 4 回

- Q クエリ解きたいです

- 各クエリでは ? に A, B, C をそれぞれ X, Y, Z 個割り当てる場合に上の問題を解いてください

0 制約

- N ≦ 300 000
- Q ≦ 200 000
- X + Y + Z は S の？の個数

0 小課題

小課題番号	N	S	Q	Z	配点
1	50	? は 13 個以下			4 点
2	500				7 点
3	5000		10		13 点
4	5000				18 点
5			10		12 点
6		C がない		Z = 0	8 点
7				Z = 0	13 点
8					25 点

0 今後について

- 解説では文字が変わる回数のことを**コスト**と言います
- AABBC なら 2 で、ACCBA なら 3 みたいな感じ
- 答えはコストの最小値です

- ネタバレ：
- とても大変な証明をすると問題が 2 パターンの形になって、
  とても大変な考察(と実装)をすると高速化できて解けます

Subtask 1

N ≦ 50, S の ? は 13 個以下

1 全探索

- ? が 13 個以下
- つまり ? への入れ方は高々 3^13 = 1594323 通り

- ? への入れ方を決めたらコストの計算は O(N)
- 普通に for 文を回せば OK
- 前もって全入れ方におけるコストを計算しておく

- ? の個数を M として O(N 3^M + Q) などで解ける

- 流石に各クエリで全部試す O(N 3^M Q) は通らないかも

1 高速に解く

- 差分更新のイメージで？に入れる時にその前後だけ考えると O(M 3^M + Q) になる
- AB??C なら 1 つめの？に入れたとき重要なのは B? と ?? のところで、2 つめなら ?? と ?C のところだけといった感じ
- 遅くても間に合いそうなときはシンプルな方を優先するとバグったときのタイムロスが少なくなります

Subtask 2

$N \leqq 500$

2 考察

- 小課題 1 の解法だと 3^500 ≒ 3.6 * 10^238 ぐらいの計算が必要

- 何をどう考えても厳しい

- 差分更新の話を思い出す
- 差分更新は結局決めたところの近傍だけ大事でした

- このアイデアを使えないか

2 DP

- 先頭から決めてくことを考える
- 新たに決めたときコストの変化は自分とその 1 つ前のところだけ
- AABB から AABBC にしたとき大事なのは BC のところ
- 状態として大事なのは？に入れた A, B, C の個数と末尾
- DP を考える

2 DP

- DP[i][j][k][l][m] := 先頭から i 文字目まで見て？に A, B, C を j, k, l 個入れて、末尾が m (= A, B, C のどれか)となるとき先頭から i 文字目までについてのコスト

- 初期化は DP[0][0][0][0][A,B,C] = 0 で他は +inf
- 遷移は min の形で書ける
- 1 遷移 O(1)

- 状態数 O(N^4) とかで遷移は O(N^4)
- TLE や MLE になりそう

2 高速化

- DP[i][j][k][l][m] := 先頭から i 文字目まで見て？に A, B, C を j, k, l 個入れて、末尾が m (= A, B, C のどれか)となるとき先頭から i 文字目までについてのコスト

- まず次元を減らす l は i, j, k から分かるので減らせる
- というわけで O(N^3) で解けました

- 出してみると不正解と返ってくる
- ジャッジが壊れていそう

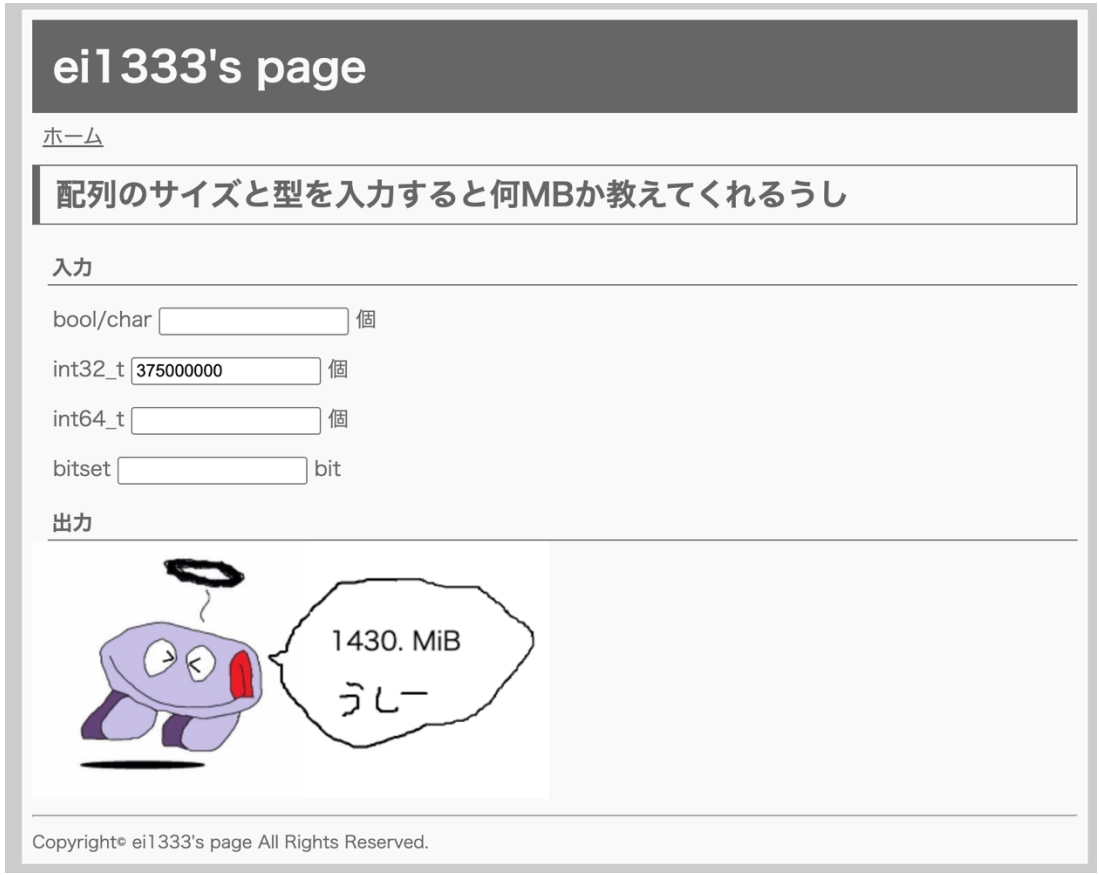
2 省メモリ化

- 自分のコードを読んでみる

- 500^3 * 3 の int 配列は重たい
- 1.5GB ぐらい
- 実際は多次元なのでもっと重たい

- ① inplace にする
- ② short 配列を使う

- (引用: ei1333's page)



2 省メモリ化

- ① inplace にする
- 昔の JOI 過去問を AtCoder で解くと ML: 64MB になってるがその頃のを解くのに使われたテクニック
- DP[i] が DP[i - 1] だけから反映されるとき 2 つの配列 A, B を用意して (A, B) = (DP[0], DP[1]), (DP[2], DP[1]), ... と更新していく
- 時間は変わらないがメモリは 1 次元削減

2 省メモリ化

- ② short 配列を使う

- [-2^15, 2^16 - 1] = [-32768, 32767] まで使える数値型
- int の半分のデータ量

- ① でもそうだが配列をグローバルに宣言したり vector ではなく int dp[501][501][3] みたいに宣言するとより良い

2 参考

- CMS のテスト機能を使えばメモリを使うか確認できる
- プラクティスでこの仕様があることを確認しましたか？

Subtask 3

$N \leqq 5\,000, Q \leqq 10$

3 その前に

- 後ろの方の小課題の方が簡単に感じることもある
- 今回は小課題 6 がそれにあたる
- 一旦小課題 6 の話をしよう

Subtask 6

S に C がない, Z = 0

6 制約の意味

- S に C がない → S は A, B, ? のみ
- Z = 0 → ? には A, B だけ割り当てる
- つまり文字が A, B の 2 種類に減った
- 文字が 1 種類ならどうなるか → 全部 A だからコスト 0 でした

6 重要なアイデア

- ? への入れ方は A を入れる個数で一意

- ここで重要なアイデア「**文字列を分割**」を使う
- S = A???B?A?ABBA とかで考えてみる

6 文字列を分割

- ここで重要なアイデア「**文字列を分割**」を使う
- S = A???B?A?ABBA とかで考えてみる

- このときコストは A???B + B?A + A?A + AB + BB + BA についてのコストの総和になる
- 文字と文字の間に着目して分けた感じ

- 各文字列は (A か B) + (? が何個か) + (A か B) の形
- このような分割は一意になる
- また長さの総和は O(N) になる(各文字の寄与を考えれば OK)

6 各文字列ごとに考える

- 各文字列は (A か B) + (? が何個か) + (A か B) の形
- ? に入れる A と B の個数を決めたときコストはどうなるか

- ① 両端が同じケース
- A???A で考えてみよう

- A が 0 個、B が 3 個 → ABBBA コストは 2
- A が 1 個、B が 2 個 → AABBA や ABABA や ABBAA がある
- コストは 2 か 4 なので最小コストは 2
- A が 2 個、B が 1 個 → AABAA とすると最小コストは 2
- A が 3 個、B が 0 個 → AAAAA コストは 0

6 下界を考える

- ① 両端が同じケース
- A???A で考えてみよう
- 全部 A ならコスト 0 でそうでないならコスト 2 でした

- **予想**「全部 A なら 0 で B があるなら 2」を立ててみる
- **証明**をする
- 全部 A なら 0 は明らか

- 下界を考える B があるとする
- 左から見てくことを考えると 左 A 右 B と 左 B 右 A の 2 回は起こる つまり 2 以上

6 下界を達成する

- ① 両端が同じケース
- A???A で考えてみよう
- **予想**「全部 A なら 0 で B があるなら 2」
- **証明**

- B があるなら下界は 2 以上だった あとは 2 を作れれば OK
- ? の左側に B を詰めて ? の右側に A を詰めれば達成可能
- 同じ文字を 1 つに集めるイメージ
- 例えば A が 1 個で B が 2 個なら AABBA とすればできる

- よって証明できた

6 予想を立てる

- ② 両端が違うケース
- A???B で考えてみよう

- A が 0 個、B が 3 個 → ABBBB コストは 1
- A が 1 個、B が 2 個 → AABBB や ABABB や ABBAB がある
- コストは 1 か 3 なので最小コストは 1
- A が 2 個、B が 1 個 → AAABB とすると最小コストは 1
- A が 3 個、B が 0 個 → AAAAB コストは 1

- **予想**「常に 1」を立ててみる
- **証明**をする

6 証明する

- ② 両端が違うケース
- A???B で考えてみよう
- **予想**「常に 1」
- **証明**

- 下界を考える B があるとする
- 左から見ていくと 左 A 右 B の 1 回は起こる つまり 1 以上
- 逆に左に A を入れて右に B を入れれば達成可能

- よって証明できた

6 ここまでのまとめ

- 各文字列について

- ① 両端が同じケース
- 全部端の文字なら 0 で、そうでないなら 2

- ② 両端が違うケース
- 常に 1

- 本質的な気付き「**どの文字を入れるか入れないかだけが重要**」
- 1 個以上なら何個でも同じということ

6 貪欲戦略

- つまり A を何個か入れて残りに B を入れるなら

- 両端が同じ文字列についてはその間を端の文字で全部埋めたい
- A???A なら AAAAA にしたいということ

- よって以下の戦略が最適
- A + (? が何個か) + A の形について短い順に貪欲に A を入れる
- B + (? が何個か) + B の形について短い順に貪欲に B を入れる
- あとは適当に入れる

6 コストの計算

- よって以下の戦略が最適
- A + (? が何個か) + A の形について短い順に貪欲に A を入れる
- B + (? が何個か) + B の形について短い順に貪欲に B を入れる
- あとは適当に入れる

- コストは [(A + (? が何個か) + A の形で B が入っている個数) + (B + (? が何個か) + B の形で A が入っている個数)] * 2 + (A + (? が何個か) + B の形の個数) * 1 と計算できる

- (A + (? が何個か) + B の形の個数) の計算は簡単

6 コストの計算

- (A + (? が何個か) + A の形で B が入っている個数) を数える

- (A + (? が何個か) + A の形で A だけが入っている個数) を数えられれば良い
- これは ? の個数を並べて sort した数列を考えれば数列の先頭から足していってある数を超えるのはいつですか という形

- 例えば A??A, A???A, A?A, A?A で A を 5 個入れるなら
- (1, 1, 2, 3) の先頭から足していって 5 を超えるのはいつですか

- これは累積和などをとって二分探索すれば良い

6 答え

- よってクエリあたり O(logN) などで解ける
- 全体では O(N + QlogN)

6 別方針

- 別方針
- あらかじめ ans[i] := ? に A を i 個入れる時の答え を用意する

- ans[0] は簡単 全部に B を入れる
- ans[i] から ans[i + 1] を作りたい
- これはさっきの貪欲戦略にしたがって B を A に入れ替えることをすると O(N) で ans[i] が列挙できる

- 全体では O(N + Q)

Subtask 3

$N \leqq 5\,000, Q \leqq 10$

3 文字列の分割

- 小課題 6 のように分割すると以下のような問題になる

- (A, B, C のどれか l) + [? が何個か] + (A, B, C のどれか r) のような形の文字列が何個か与えられる (C[???]B みたいな感じ)
- ? に A, B, C を決められた個数入れる
- それぞれにおけるコストを足し合わせた値を最小化したい

- 各文字列について [?] のところに入れた文字で、l, r のどちらでもない文字の種類数を**ペナルティ**とする (A[BBC]A なら 2)
- A, B のときはコストは両端とペナルティだけに依存していた

3 文字列の分割

- 各文字列について [?] のところに入れた文字で、l, r のどちらでもない文字の種類数を**ペナルティ**とする (A[BBC]A なら 2)

- **予想**「A, B, C でもコストは両端とペナルティだけに依存する」
- なんと成り立つ
- **証明**をする

3 証明

- **予想**「A, B, C でもコストは両端とペナルティだけに依存する」
- **証明**

- ① 両端が同じケース (l = r)
- ペナルティが 0, 1, 2 のとき下界として 0, 2, 3 が取れる
- これは達成できる
- 先ほどと同様に同じ文字を 1 つに集めれば OK
- A???A に B を 2 個と C を 1 個なら ABBCA にするイメージ

3 証明

- **予想**「A, B, C でもコストは両端とペナルティだけに依存する」
- **証明**

- ② 両端が異なるケース (l ≠ r でない)
- ペナルティが 0, 1 のとき下界として 1, 2 が取れる
- これも達成できる
- 先ほどと同様に同じ文字を 1 つに集めれば OK
- A???B に B を 2 個と C を 1 個なら ACBBB にするイメージ

- よって証明できた

3 言い換え

- **性質**「A, B, C でもコストは両端とペナルティだけに依存する」
- これを用いて問題文を書き直す

- (A, B, C のどれか l) + [? が何個か] + (A, B, C のどれか r) のような形の文字列が何個か与えられる (C[???]B みたいな感じ)
- ? に A, B, C を決められた個数入れる
- それぞれにおけるコストを足し合わせた値を最小化したい

- 各文字列について [?] のところに入れた文字で、l, r のどちらでもない文字の種類数を**ペナルティ**とする (A[BBC]A なら 2)

3 言い換え

- 各文字列について [?] のところに入れた文字で、l, r のどちらでもない文字の種類数を**ペナルティ**とする (A[BBC]A なら 2)

- 両端が同じとき　ペナルティが 0, 1, 2 ならコストは 0, 2, 3
- 両端が異なるとき ペナルティが 0, 1　ならコストは 1, 2

- 今回は N ≦ 5 000 で Q ≦ 10 で解けばいい
- クエリあたり O(N^2) とか O(N^2logN) とかで解きたい

3 用語の準備

- 小課題 7 では貪欲な戦略を考えた
- 今回も同様にできないか？

- 実はある
- まず色々用語の準備をしよう

- A[?]A の形における？の個数の合計を p とする
- B[?]B, C[?]C, A[?]B, B[?]C, C[?]A についても同様に q, r, s, t, u と定める

3 用語の準備

- A[?]A の形における？の個数の合計を p とする
- B[?]B, C[?]C, A[?]B, B[?]C, C[?]A についても同様に q, r, s, t, u と定める

- S = A???AB??B??C??B なら
- A[???]A + A[]B + B[??]B + B[??]C + C[??]B となって
- (p, q, r, s, t, u) = (3, 2, 0, 0, 3, 0) ということ

- p, q, r, s, t, u は明らかに 0 以上
- 証明で使うことがあるのでここで明記

3 Large の定義

- A[?]A の形における？の個数の合計を p とする
- B[?]B, C[?]C, A[?]B, B[?]C, C[?]A についても同様に q, r, s, t, u と定める

- 各文字がたくさんあるか全然ないかを定義したい
- A, B, C について **Large** と **Small** という概念を定義する

- **Large:** 個数が自分の文字が端となる文字列における？の個数の総和より多い
- 例えば X > p + s + u なら A を Large と呼ぶ
- B, C については Y > q + s + t, Z > r + t + u

3 Small の定義

- **Large:** 個数が自分の文字が端となる文字列における？の個数の総和より多い
- 例えば X ＞ p ＋ s ＋ u なら A を Large と呼ぶ
- B, C については Y ＞ q ＋ s ＋ t, Z ＞ r ＋ t ＋ u

- **Small:** 個数が自分の文字が両端となる文字列における？の個数の総和より少ない
- 例えば X ＜ p なら A を Small と呼ぶ
- B, C については Y ＜ q, Z ＜ r

3 用語のイメージ

- **Large:** 文字がたくさんあってペナルティが増えるような文字列にはみ出る
- **Small:** 文字があまりなくて自分の文字を両端とする文字列に他の文字が入ってくる

- ? の個数の議論より X + Y + Z = p + q + r + s + t + u が成立
- Large や Small の個数で場合分けをしていく

3 怒涛の証明

- ここから場合分けしつつ問題の性質を証明します
- なんと ?? 個もあります
- 基本的なアイデアは「**このように操作しても損しないので、最適解はこれが成り立っているとして良い**」というもの
- 以降示した性質は成り立っているとします

3 怒涛の証明の前に

- ここから場合分けしつつ問題の性質を証明します
- なんと ?? 個もあります
- 基本的なアイデアは「**このように操作しても損しないので、最適解はこれが成り立っているとして良い**」というもの
- 多分もう話し疲れているので一旦休憩

3 証明 01

- **性質 01:**
- V = 3 となる文字列の数 d は 1 以下

- **証明 01:**
- d ≧ 2 とする V = 3 の文字列を 2 つ取る

- ① A [A * p1, B * q1, C * r1] A と A [A * p2, B * q2, C * r2] A のように両端の文字が同じとき
- 前者の B と後者の C を min(q1, r2) 回 swap すると d を 1 か 2 減らしてコストも 1 減らせる

3 証明 01

- **性質 01:**
- V = 3 となる文字列の数 d は 1 以下

- **証明 01:**
- ② A [A * p1, B * q1, C * r1] A と B [A * p2, B * q2, C * r2] B のように両端の文字が違うとき
- 前者の B と後者の A を min(q1, p2) 回 swap すると同様に示せる

3 場合分け

- Large の個数で場合分けをする

- ① Large が 3 つあるとき
- $X > p + s + u, Y > q + s + t, Z > r + t + u$ ということ
- これは $X + Y + Z = p + q + r + s + t + u$ に矛盾

- ② Large が 2 つあるとき
- A と B が Large としていい
- $X > p + s + u, Y > q + s + t$ より $Z < r - s$
- 特に $Z < r$ すなわち C は Small

3 場合分け

- ② Large が 2 つあるとき
- A と B が Large としていい
- X ＞ p ＋ s ＋ u, Y ＞ q ＋ s ＋ t より Z ＜ r - s
- 特に Z ＜ r すなわち C は Small

- このとき以下の性質が成り立つ
- **性質 02:** A[?]A の中に C は入らない
- 他にも 〇〇の中に×× は入る/ 入らないといった形があります

3 証明 02

- **性質 02:**
- A[?]A の中に C は入らない

- **証明 02:**
- Z < r より C[?]C の中に A か B があって、これと A[?]A の中の C を swap する方針です

3 まとめ

- ② Large が 2 つあるとき
- A と B が Large とする このとき C は Small となる

- **性質 02:** A[?]A の中に C は入らない

- 他の性質と共に考えると解の構造、つまり A や B や C がどこに入るかなどがかなり限定されます

3 まとめ

- ② Large が 2 つあるとき
- A と B が Large とする このとき C は Small となる

- 結論としては A[?]B に A, B を何個入れるかが重要になります
- すると次の **Subproblem1** の形に帰着できる

3 まとめ

- **Subproblem1**
- 数列 c が与えられる (C[?]C の ? の個数に対応)
- Z 回 c の要素を 1 つ選んで減らす (C[?]C の ? に C を入れること)
- この後の c について**罰金**を以下のように定める 最小化せよ

- $c_i \geqq 1$ なる i の個数を W とする (C 以外が入る C[?]C の数)
- c の部分和で a が作れるなら 2 * W
- 作れないなら 2 * W + 1 (ペナルティが 2 となる分を足す)

3 Subproblem1 の解法

- **Subproblem1** を考える
- 部分和問題の亜種の形に帰着できます
- 部分和問題は DP[i番目までを見た][総和がj] となる DP で O(N^2) とできます

- O(N^2) などで解くことができます

3 場合分けに戻ってくる

- ③ Large が 1 つあるとき
- A が Large としていい
- $X > p + s + u$ より $Y + Z < q + r + t$

- [#1] $Y > q + t$ かつ $Z > r + t$ のとき
- $Y + Z$ の条件に矛盾

- [#2] $Y > q + t$ のとき
- $Z < r$ である
- ② と同様の性質が成立するので **Subproblem1** に帰着できる
- B が Large でないため $Y \leqq q + s + t$ となる

3 証明

- ③ Large が 1 つあるとき

- [#2] Y > q + t のとき
- q + t < Y ≦ q + s + t より B[?]B, B[?]C に B を入れた後の残りは全部 A[?]B に入れれば良い
- よって A[?]B に入れる A の個数は一意なので **Subproblem1** が少し簡単に書ける
- やることは変わらないが範囲の端の計算が簡潔

- [#3] Z > r + t のとき
- [#2] と同様に解ける

3 証明

- ③ Large が 1 つあるとき
- [#4] Y ≦ q + t かつ Z ≦ r + t のとき

- このとき以下の性質が成り立つ
- **性質 ??:** B[?]B, C[?]C の中にそれぞれ C, B は入らない
- 他にも ○○の中に××は入る/ 入らないといった形があります

3 証明 ??

- **性質 ??:**
- B[?]B, C[?]C の中にそれぞれ C, B は入らない

- **証明 ??:**
- B[?]B, C[?]C の中にそれぞれ C, B は入らないを証明する
- これもやはり損をしない swap に注目するイメージです

3 まとめ

- ③ Large が 1 つあるとき
- [#4] Y ≦ q + t かつ Z ≦ r + t のとき

- このとき以下の性質が成り立つ
- **性質 ??:** B[?]B, C[?]C の中にそれぞれ C, B は入らない
- 他にも 〇〇の中に××は入る/ 入らないといった形があります

3 まとめ

- ③ Large が 1 つあるとき
- [#4] $Y \leqq q + t$ かつ $Z \leqq r + t$ のとき

- 先に B, C を入れて後で A を残りに入れることを考える
- すると次の **Subproblem2** の形に帰着できる

3 まとめ

- **Subproblem2**

- 3 つの書店 D, E, F がある それぞれ本を販売している
- 金貨 Y 枚と銀貨 Z 枚を持っている (B の個数と C の個数)
- 以下のように本を買う
- 書店 D で本 i を金貨 d_i 枚で買う (B[?]B の？に B のみを入れる)
- 書店 E で本 i を銀貨 e_i 枚で買う (C[?]C の？に C のみを入れる)
- 書店 F で本 i を金貨と銀貨合わせて f_i 枚で買う (B[?]C の？に B か C を入れる)

- 買い方に対して**罰金**を以下のように定める 最小化せよ

3 まとめ

- **Subproblem2**

- 買い方に対して**罰金**を以下のように定める 最小化せよ
- ((書店 D で買えなかった本の数) + (書店 E で買えなかった本の数)) * 2 + (書店 F で買えなかった本の数) * 1

- つまり A を？に入れることによるコストの増加分を最小化したい

3 まとめ

- **Subproblem2**

- 変数 W を W = 0 で初期化する
- D, E で本を買うたびに +2 して F で買うたびに +1 する
- 操作後の W を最大化せよ

- という形になる
- 買えなかった分に関する式の最小化を
- 買えた分に関する式の最大化に言い換えるイメージ

- こちらの方が見やすいので以降こちらを採用

3 Subproblem2 の解法

- **Subproblem2** を考える

- 書店 D と書店 E で買う本の数を全探索すると書店 F でどれだけ買えるかは二分探索できる
- よって O(N^2logN) などで解けました

3 場合分けに戻ってくる

- ④ Large がないとき

- ここにも場合分けと性質の列挙があります
- Small の個数で場合分けしていきます

- Small の個数が 1 つなら Subproblem1 に帰着できて
- Small がないならペナルティを全て 0 にできることが示せます

3 結論

- よってこの問題は **Subproblem1** と **Subproblem2** に帰着できた

- **Subproblem1** はクエリあたり O(N^2) で解けた
- **Subproblem2** はクエリあたり O(N^2logN) で解けた

- よって全体で O(QN^2logN) で解けました

Subtask 4

$N \leqq 5\,000$

4 小課題 3 との違い

- N は小さいままだが Q は大きくなった
- 小課題 3 の解法をクエリに対応するようにしたい

4 高速化

- **Subproblem1:** 部分和問題の DP を c が大きい順に埋めつつクエリを処理していくイメージです
- **Subproblem2:** 書店 F で買う個数の偶奇を固定して考えると貪欲アルゴリズムに落ち着きます Segment Tree を使います

- 全体では O(N^2logN + Q) になります

- セグ木の二分探索で log がもう 1 つつくとダメかも

Subtask 5

$Q \leqq 10$

5 小課題 3 との違い

- Q は小さいままだが N は大きくなった
- 小課題 3 の解法を高速化したい

5 高速化

- **Subproblem1:** 部分和問題の DP を更新するとき同じ値を一緒に更新するとなんと計算量が改善します
- **Subproblem2:** 書店 D, E で買う個数の合計を固定して考えると書店 F の買い方は貪欲アルゴリズムに落ち着きます

- よって全体では $O(QN\sqrt{N})$ になります

Subtask 6

S に C がない, Z = 0

6 ふりかえり

- 小課題 3 のところでやったのでここではおさらい

- 以下の戦略が最適
- A + (? が何個か) + A の形について短い順に貪欲に A を入れる
- B + (? が何個か) + B の形について短い順に貪欲に B を入れる
- あとは適当に入れる

- A[?]A の ? の個数を並べた列と B[?]B の個数を並べた列を sort して累積和 + 二分探索するとクエリあたり O(logN) で解ける
- 全体で O(N + QlogN)

6 ふりかえり

- 別方針
- あらかじめ ans[i] := ? に A を i 個入れる時の答え を用意する

- ans[0] は簡単 全部に B を入れる
- ans[i] から ans[i + 1] を作りたい
- さっきの戦略を基に考えると O(N) で ans[i] が列挙できる

- 全体では O(N + Q)

Subtask 7

$z = 0$

7 Z = 0 の意味

- 場合分けや証明、帰着させた後の問題が少し楽になります
- **Subproblem1:** C を入れることがないので部分和問題そのものになります
- **Subproblem2:** 少しいい形になります B[?]B と B[?]C に B を入れて残りに A を入れる形 (もしくは A と B が逆の形) になりこれは価値が 1 と 2 のナップサック問題のようになります
- 価値が 1 の個数の偶奇を決めれば貪欲になります
- 全体で $O(N\sqrt{N} + Q)$ などになります

Subtask 8

追加制約なし

8 小課題 4 と小課題 5 の復習

- 小課題 4 では解法をクエリに対応できるようにした
- 小課題 5 ではクエリあたりを高速にした

- これの両方ができれば小課題 8 が解けそう
- 本当にできるのか

8 アイデアの merge

- 小課題 4 と 5 を合体させるイメージになる
- 全体では $O(N\sqrt{N} + QlogN)$ になる
- 実装はとても大変

Step 9

得点分布 & 余興

9 得点分布

- 理想: 100 点 27 人
- 現実: ???