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

解説: 蜂矢 倫久 (Mitsubachi)

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
- S の先頭と末尾は A
- 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

- スライドもとても大変で **181** ページもあります

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

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 6 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 17 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 17 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
- 今日の駒場の写真を流す

3 休憩タイム

- この場所から南に行くとあります
- 馬神と書いてある

3 休憩タイム

- きれいだ
- 証明をしよう

3 証明 01

- **性質 01:**
- ペナルティが 2 となる文字列の数 d は 1 以下

- **証明 01:**
- d ≧ 2 とする ペナルティが 2 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:**
- ペナルティが 2 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, B[?]B の中に C は入らない
- **性質 03:** A[?]B の中に C は入らない
- **性質 04:** A[?]A, B[?]B の中はそれぞれ A, B のみ入る
- **性質 05:** B[?]C, C[?]A の中にそれぞれ A, B は入らない
- **性質 06:** B[?]C, C[?]A の中に C は入らない

3 記法の定義

- 証明の中の記法として A[BC]A のようなものを使います
- これは両端が A で？に入れた端でない文字は B と C の 2 種類の文字列ですよという意味 (A は入れて良い)
- 他には A[C]B なら両端が A と B で？に入れた端でない文字は C の 1 種類の文字列ですよという意味 (A, B は入れて良い)
- 同じものを 1 つにしたよと思えば良い
- **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」を抑えると自然な記法

3 証明 02

- **性質 02:**
- A[?]A, B[?]B の中に C は入らない

- **証明 02:**
- 背理法を使う ある A[?]A の中に C が入っているとしよう
- Z < r より C[?]C の中の A, B と swap することで A[?]A の中の C を全て C[?]C の中に持ってくることが可能

- A[?]A は A[BC]A か A[C]A の形

3 証明 02

- **性質 02:**
- A[?]A, B[?]B の中に C は入らない

- **証明 02:**
- ① A[BC]A の場合
- **性質 01**「ペナルティが 2 となる文字列の数 d は 1 以下」より C[?]C の中に A, B があるものは CAA...AC か CBB...BC の形
- A[BC]A と C[A]C で swap → 最悪でも A[B]A と C[AB]C でコストは増えない
- A[BC]A と C[B]C で swap → A[B]A と C[B]C でコストは 1 減少

3 証明 02

- **性質 02:**
- A[?]A, B[?]B の中に C は入らない

- **証明 02:**
- ② A[C]A の場合
- A[C]A と C[A]C で swap → A[]A と C[A]C でコストは 2 減少
- A[C]A と C[B]C で swap → 最悪でも A[B]A と C[B]C でコストは増えない
- A[C]A と C[AB]C で swap → 最悪でも A[B]A と C[AB]C でコストは増えない

3 証明 02

- **性質 02:**
- A[?]A, B[?]B の中に C は入らない

- **証明 02:**
- よってコストを増やさないまま A[?]A の中の C を C[?]C に移すことができた

- B[?]B の中に C は入らないはさっきの証明の A を B に変えるだけ

3 証明 03

- **性質 03:**
- A[?]B の中に C は入らない

- **証明 03:**
- A[?]B の中に C は入らないも同様にできる
- むしろ A[?]B の中に A, B を入れる分にはペナルティが増えないので損をすることがない

3 証明 04

- **性質 04:**
- A[?]A, B[?]B の中はそれぞれ A, B のみ入る

- **証明 04:**
- A[?]A の中は A のみ入るが示せれば A と B を変えるだけで良い
- まず**性質 02**「A[?]A の中に C は入らない」より B があるなら A[B]A の形
- 同様に B[?]B の中に A があるなら B[A]B の形
- A[B]A と B[A]B が両方あるならこの 2 つで swap すれば良い
- よって B[A]B はない

3 証明 04

- **性質 04:**
- A[?]A, B[?]B の中はそれぞれ A, B のみ入る

- **証明 04:**
- A[B]A があると仮定
- 同様に考えると A[?]B, B[?]C の中に A があればコストが増えない swap が可能
- A[?]B, B[?]C の中に A はない

- つまり A は A[?]A, A[?]C, C[?]C のどれかに入る

3 証明 04

- **性質 04:**
- A[?]A, B[?]B の中はそれぞれ A, B のみ入る

- **証明 04:**
- A[B]A があると仮定
- A は A[?]A, A[?]C, C[?]C のどれかに入る
- A が Large より $X > p + s + u \geqq p + u$ である つまり A[?]A の？の個数と A[?]C の？の個数は埋められる

- よって A[?]A の中にある B の個数以上 C[?]C の中に A がある

3 証明 04

- **性質 04:**
- A[?]A, B[?]B の中はそれぞれ A, B のみ入る

- **証明 04:**
- A[B]A があると仮定
- A[?]A の中にある B の個数以上 C[?]C の中に A がある

- A[?]A の中にある B を全て C[?]C の中の A と swap すれば良い
- A[?]A によるコストの寄与の差分は -2 以下で、C[?]C によるコストの寄与は +1 以下で抑えられる
- C[A]C → C[AB]C となることは 1 個で抑えられるため

3 証明 04

- **性質 04:**
- A[?]A, B[?]B の中はそれぞれ A, B のみ入る

- **証明 04:**
- A[?]A の中は A のみ入るが示された
- A と B を変えるだけで B[?]B の中は B のみ入るが示せる

- よって示されました

3 証明 05

- **性質 05:**
- B[?]C, C[?]A の中にそれぞれ A, B は入らない

- **証明 05:**
- B[A]C があるとする
- C[B]C, A[C]B, C[B]A があるならこれと swap すればいい

- もしないなら Y ≦ q + t となり Y が Large に矛盾
- よって B[A]C はない 同様に C[B]A もない
- 以上より示されました

3 証明 06

- **性質 06:**
- B[?]C, C[?]A の中に C は入らない

- **証明 06:**
- B が Large と A[?]A と A[?]C に B がない(**性質 02, 05** より)

- よって B[?]C の中にある C の分 C[?]C の中に B がある
- これを swap すれば良い
- C[?]A についても同様

- 以上より示されました

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, B[?]B の中に C は入らない
- **性質 03:** A[?]B の中に C は入らない
- **性質 04:** A[?]A, B[?]B の中はそれぞれ A, B のみ入る
- **性質 05:** B[?]C, C[?]A の中にそれぞれ A, B は入らない
- **性質 06:** B[?]C, C[?]A の中に C は入らない

- つまり A[?]A, C[?]A の中は A が入り B[?]B, B[?]C の中は B が入る また C は C[?]C の中のみに入る

3 最適解の構造

- 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

- A[?]A, A[?]C は A のみ入る
- B[?]B, B[?]C は B のみ入る
- C は C[?]C の中のみに入る

- 決まってないのは A[?]B に A, B を何個入れるか

- A[?]B の A の個数を固定してみる C[?]C に A を a 個入れるとする
- すると次の **Subproblem1** の形に帰着できる

3 Subproblem1

- **Subproblem1**

- 数列 c が与えられる (C[?]C の ? の個数に対応)
- c の正の要素を 1 つ選んで減らす操作を Z 回行う (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)
- $a \leqq \mathrm{sum}(c_i)$ - Z は成り立っています

3 Subproblem1 の解法

- **Subproblem1** を考える

- まず W は最小値としていい
- W の最小値として W’ が達成できるなら罰金は 2 * W’ + 1 以下にできる
- 逆に W ≧ W’ + 1 なら罰金は 2 * W’ + 2 以上になり不適

- W の最小化は常に最小の c_i を 1 減らせば良い
- 操作の残り回数的にもう $c_i = 0$ となる i を増やせなくなったとする

3 Subproblem1 の解法

- **Subproblem1** を考える

- W の最小化は常に最小の c_i を 1 減らせば良い
- 操作の残り回数的にもう $c_i = 0$ となる i を増やせなくなったとする

- c = (3, 4, 6, 7) で Z = 8 なら 7 回の操作後には
- c = (0, 0, 6, 7) で後 1 回減らせるタイミングに来たという感じ
- a (部分和で作りたい数) が 5 なら 6 を減らせばいいし、6 なら 7 を減らせばいい
- どっちを減らせばいいんだ

3 Subproblem1 の解法

- **Subproblem1** を考える

- 実は解ける
- もう $c_i = 0$ となる i を増やせなくなったタイミングにおける残り操作回数を z とする
- すると z 回操作した後の部分和で a が作れることの必要十分条件はそのタイミングでの部分和で [a, a + z] のどれかが作れること
- 十分性は a + k が作れるならその部分和に採用されたのについて k 回操作してそうでないのから z - k 回操作すれば示せる
- 必要性は z 回後に a が作れるなら a を作るために選ばれた要素の z 回前の総和を考えると示せる

3 Subproblem1 の解法

- **Subproblem1** を考える

- z 回操作した後の部分和で a が作れることの必要十分条件はそのタイミングでの部分和で [a, a + z] のどれかが作れること

- c = (3, 4, 6, 7) で Z = 8 なら 7 回の操作後には
- c = (0, 0, 6, 7) で後 z = 1 回減らせるタイミングに来たという感じ
- この c の部分和は 6, 7, 13 が取れる
- a = 5: [5, 5 + 1] = [5, 6] で 6 が作れるので OK
- a = 6: [6, 6 + 1] = [6, 7] で 6 と 7 が作れるので OK

3 Subproblem1 の解法

- **Subproblem1** を考える

- z 回操作した後の部分和で a が作れることの必要十分条件はそのタイミングでの部分和で [a, a + z] のどれかが作れること

- 十分性は a + k が作れるならその部分和に採用されたのについて k 回操作してそうでないのから z - k 回操作すれば示せる
- 必要性は z 回後に a が作れるなら a を作るために選ばれた要素の z 回前の総和を考えると示せる

3 Subproblem1 の解法

- **Subproblem1** を考える

- 部分和問題は DP[i][j] := i 番目まで見て総和が j を作れるかとする DP で O(N^2) で解ける

- a の値は A[?]B の？の個数を固定すると一意に定まるのだった
- だが A[?]B の？の個数は色々ありうる

- しかし、A[?]B の？の個数を 1 変化すると a は 1 変化する
- よって a のとりうる値は範囲となる

3 Subproblem1 の解法

- **Subproblem1** を考える

- a のとりうる値は範囲となる
- そして知りたいものは a がある範囲を動く (= A[?]B に使う A の個数を動かす) 中での **Subproblem1** の答え (= C[?]C の部分のコストへの寄与) の最小値
- また Z の値は a によらず一緒 → 同じ部分和問題の DP！

- よって **Subproblem1** の [a, a + z] を [min(a), max(a) + z] に置き換えたものを解けば良い
- 同様に O(N^2) で解ける

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 07:** B[?]B, C[?]C の中にそれぞれ C, B は入らない
- **性質 08:** A[?]B, C[?]A の中にそれぞれ C, B は入らない
- **性質 09:** A[?]A の中に B, C は入らない
- **性質 10:** A[?]B, C[?]A の中にそれぞれ B, C は入らない

3 証明 07

- **性質 07:**
- B[?]B, C[?]C の中にそれぞれ C, B は入らない

- **証明 07:**
- B[?]B の中に C が入らないを証明したい

- Z ≦ r + t より B[?]B の中にある C の個数分の A か B が C[?]C と B[?]C の中にある
- **証明 02** と同様の議論を考えると B[?]B の中にある C を全部 C[?]C や B[?]C の中の A と B で swap してコストは増えないので損しない

3 証明 07

- **性質 07:**
- B[?]B, C[?]C の中にそれぞれ C, B は入らない

- **証明 07:**
- B[?]B の中に C が入らないを証明した
- B と C を入れ替えれば C[?]C の中に B が入らないも証明できる

- よって示されました

3 証明 08

- **性質 08:**
- A[?]B, C[?]A の中にそれぞれ C, B は入らない

- **証明 08:**
- A[?]B の中に C は入らないを証明する

- **証明 07** と同様の議論をする Y ≦ q + t より A[?]B の中にある C の個数分の A か B が B[?]B と B[?]C の中にある
- swap してもコストは増えないため損しない

- よって示されました

3 証明 08

- **性質 08:**
- A[?]B, C[?]A の中にそれぞれ C, B は入らない

- **証明 08:**
- A[?]B の中に C は入らないを証明する
- B, C を入れ替えれば C[?]A の中に B は入らないも証明できる

- よって示されました

3 証明 09

- **性質 09:**
- A[?]A の中に B, C は入らない

- **証明 09:**
- **証明 07** と同様の議論をする

- A[?]A の B を B[?]B や B[?]C の A か C と swap するイメージ

3 証明 10

- **性質 10:**
- A[?]B, C[?]A の中にそれぞれ B, C は入らない

- **証明 10:**
- **証明 07** と同様の議論をする

- A[?]B の B を B[?]B の A と swap するイメージ

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 07:** B[?]B, C[?]C の中にそれぞれ C, B は入らない
- **性質 08:** A[?]B, C[?]A の中にそれぞれ C, B は入らない
- **性質 09:** A[?]A の中に B, C は入らない
- **性質 10:** A[?]B, C[?]A の中にそれぞれ B, C は入らない

- つまり A[?]A, A[?]B, C[?]A の中は A が入り B は B[?]B, B[?]C の中のみに入る また C は C[?]C, B[?]C の中のみに入る

3 最適解の構造

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

- A[?]A, A[?]B, A[?]C は A のみ入る
- B は B[?]B, B[?]C の中のみに入る
- C は C[?]C, B[?]C の中のみに入る

- 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 Subproblem2

- **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 Subproblem2

- **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 Subproblem2

- **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 O(N^2) 通り全探索する
- 書店 F でどれだけ買えるかを知りたい
- D と E で使った分から F でどれだけ使えるか分かる
- あとはどれだけ買えるかを二分探索する O(logN)

- 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

- [#1] Small が 3 つあるとき
- $X + Y + Z < p + q + r$ より個数の総和の条件に矛盾

- [#2] Small が 2 つあるとき
- B, C が Small とする $B < q$, $C < r$ より $A > p + s + t + u$
- $A > p + s + t + u \geqq p + s + u$ となり矛盾

3 場合分けに戻ってくる

- ④ Large がないとき
- [#3] Small が 1 つあるとき
- C が Small とする $X \geqq p, Y \geqq q, Z < r$ である

- A, B は Large ではないので総和の条件を考えると
- $p + u \leqq X \leqq p + s + u, q + t \leqq Y \leqq q + s + t, Z < r$ となる
- また $p + q + s + t + u \leqq X + Y$

3 場合分けに戻ってくる

- ④ Large がないとき
- [#3] Small が 1 つあるとき

- このとき以下の性質が成り立つ
- **性質 11:** A[?]A, B[?]B, A[?]B の中に C は入らない
- **性質 12:** A[?]A の中に B は入らない
- **性質 13:** B[?]B の中に A は入らない
- **性質 14:** C[?]A の中に B は入らない
- **性質 15:** B[?]C の中に A は入らない
- **性質 16:** B[?]C の中に C は入らない
- **性質 17:** C[?]A の中に C は入らない

3 証明 11

- **性質 11:**
- A[?]A, B[?]B, A[?]B の中に C は入らない

- **証明 11:**
- Z < r よりそれらに入っている C の個数の分 C[?]C の中に A, B が入っている
- これらで swap すれば良い

3 証明 12

- **性質 12:**
- A[?]A の中に B は入らない

- **証明 12:**
- A[?]A の中に B が入っているとする
- A[?]B や B[?]B の中には B のみ入っている
- そうでないなら**性質 11** より A が入っているのでこれらを swap すれば良い

- これと p + u ≦ X より A[?]A の中に入っている B の個数の分の A が C[?]C か B[?]C にある

3 証明 12

- **性質 12:**
- A[?]A の中に B は入らない

- **証明 12:**
- A[?]A の中に B が入っているとする
- これと p + u ≦ X より A[?]A の中に入っている B の個数の分の A が C[?]C か B[?]C にある

- これらを swap すれば良い
- C[A]C → C[AB]C となることがあるがこれによるコストの上昇は +1 以下で A[B]A → A[]A でコストは 2 減るので正当

3 証明 13

- **性質 13:**
- B[?]B の中に A は入らない

- **証明 13:**
- A と B を入れ替えることで **証明 12** と同様の形になる

3 証明 14

- **性質 14:**
- C[?]A の中に B は入らない

- **証明 14:**
- C[?]A の中に B が入っているとする

- A[?]B, B[?]C に A はないとしていい あるなら swap 可能
- p + u ≦ X より C[?]A の中の B の個数分の A が C[?]C に入っている

- これらを swap すれば良い

3 証明 15

- **性質 15:**
- B[?]C の中に A は入らない

- **証明 15:**
- A と B を入れ替えることで **証明 14** と同様の形になる

3 証明 16

- **性質 16:**
- B[?]C の中に C は入らない

- **証明 16:**
- B[?]C の中に C があるとする
- C[?]C の中に B があるなら swap していいのでないとする
- C[?]C の中には A か C しかない
- C[?]A の中に C があるなら C が Small (= C ＜ r) より C[?]C の中に A が入っていてこれと C[?]C の中の A を swap すれば良い
- よって C[?]A の中は C のみ

3 証明 16

- **性質 16:**
- B[?]C の中に C は入らない

- **証明 16:**
- これとここまでの性質より B は B[?]B, A[?]B, B[?]C にしかない

- 結局 C[?]C の中の A と A[?]B の中の B は両方 B[?]C の中の C の個数分取れる
- この A, B, C を A → B → C のサイクルに沿って swap すれば結局 C[?]C の中の A の個数が減ってその分 C が増える
- よってコストは増えないので示された

3 証明 17

- **性質 17:**
- C[?]A の中に C は入らない

- **証明 17:**
- A と B を入れ替えることで **証明 16** と同様の形になる

3 まとめ

- ④ Large がないとき
- [#3] Small が 1 つあるとき

- **性質 11:** A[?]A, B[?]B, A[?]B の中に C は入らない
- **性質 12:** A[?]A の中に B は入らない
- **性質 13:** B[?]B の中に A は入らない
- **性質 14:** C[?]A の中に B は入らない
- **性質 15:** B[?]C の中に A は入らない
- **性質 16:** B[?]C の中に C は入らない
- **性質 17:** C[?]A の中に C は入らない

3 最適解の構造

- ④ Large がないとき
- [#3] Small が 1 つあるとき

- A[?]A, A[?]C は A のみ入る
- B[?]B, B[?]C は B のみ入る
- C は C[?]C の中のみに入る

- 決まってないのは A[?]B に A, B を何個入れるか

- これは ② と同様に **Subproblem1** の形に帰着できる

3 最適解の構造

- ④ Large がないとき
- [#4] Small がないとき

- まず A[?]A, B[?]B, C[?]C に A, B, C を全部入れる
- うまく A[?]B, B[?]C, C[?]A に入れると A[]B, B[]C, C[]A のみにできる
- これで明らかな下界を達成してる
- これは？の個数について帰納法で証明できる
- 帰納法をせずとも余裕のある方から貪欲に入れてけば良い

- よってこのケースは簡単

3 まとめ

- 以上より場合分けが全部できました
- 大変な場合分けでした

3 まとめ

- 以上より場合分けが全部できました

- ① Large が 3 つあるとき: 矛盾
- ② Large が 2 つあるとき: **Subproblem1** に帰着
- ③ Large が 1 つあるとき: **Subproblem2** に帰着
- ④ Large がないとき:
- [#1] Small が 3 つあるとき: 矛盾
- [#2] Small が 2 つあるとき: 矛盾
- [#3] Small が 1 つあるとき: **Subproblem1** に帰着
- [#4] Small がないとき: 簡潔に解ける

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

- イメージとして **Subproblem1** は 1 つ Small があって、他の文字が入ってくる部分のコストを小さくする
- **Subproblem2** は 1 つ Large があって、それが他にはみ出る部分のコストを小さくする

Subtask 4

$N \leqq 5\,000$

4 今後の流れ

- この問題は **Subproblem1** と **Subproblem2** に帰着できる
- 各問題を高速化したりクエリに対応するようにするのが目標
  - **Subproblem1**

  - 数列 c が与えられる (C[?]C の？の個数に対応)
  - c の正の要素を 1 つ選んで減らす操作を Z 回行う (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)
  - $a \leqq \mathrm{sum}(c_i) - Z$ は成り立っています

4 今後の流れ

- この問題は **Subproblem1** と **Subproblem2** に帰着できる
- 各問題を高速化したりクエリに対応するようにするのが目標
  - **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)
  - 変数 W を W = 0 で初期化し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)
  -
  - 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

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
- **Subproblem1** と **Subproblem2** でクエリにすると何が変わるかを考えよう
- まずどの形の文字列における？の個数が重要かを考える

4 Subproblem1 をクエリに対応

- **Subproblem1** でクエリにすると何が変わるかを考えよう
- まずどの形の文字列における？の個数が重要かを考える
- さっきは C[?]C における？の個数の列を使った
- X, Y, Z の値によっては A[?]A, B[?]B におけるものが重要になりうる
- いずれの場合でもどれか 1 つの列が重要で、列の中身は端の文字が同じなら同じ　つまり 3 通りのどれかが大事

4 Subproblem1 をクエリに対応

- **Subproblem1** でクエリにすると何が変わるかを考えよう

  - **Subproblem1**

  - 数列 c が与えられる (C[?]C の？の個数に対応)
  - c の正の要素を 1 つ選んで減らす操作を Z 回行う (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)
  - $a \leqq \mathrm{sum}(c_i) - Z$ は成り立っています

4 Subproblem1 をクエリに対応

- **Subproblem1** でクエリにすると何が変わるかを考えよう
- どの列を使うかでクエリを分ける C[?]C の列を使うとする
- すると同じ列 c について以下のような形のクエリを処理することになる
- c の大きい方から w 個取り出しその部分和で [l, r] を作ることができるか判定せよ
- クエリでは (w, l, r) が与えられます

4 Subproblem1 をクエリに対応

- **Subproblem1** でクエリにすると何が変わるかを考えよう

- c の大きい方から w 個取り出しその部分和で [l, r] を作ることができるか判定せよ
- クエリでは (w, l, r) が与えられます
- 正確にはクエリにより変わる z から w を計算する必要がある

- これは w が小さい順に見ていけば部分和問題の DP テーブルを進めつつクエリに答えられる

4 Subproblem1 をクエリに対応

- **Subproblem1** でクエリにすると何が変わるかを考えよう

- c の大きい方から w 個取り出しその部分和で [l, r] を作ることができるか判定せよ
- これは w が小さい順に見ていけば部分和問題の DP テーブルを進めつつクエリに答えられる

- DP[0] を用意 → w = 0 のクエリを処理 →
- DP[1] を用意 → w = 1 のクエリを処理 → ... というイメージ
- 全部で O(N^2 + QlogQ) 丁寧に sort すると O(N^2 + Q)

4 Subproblem2 をクエリに対応

- **Subproblem2** でクエリにすると何が変わるかを考えよう

  - **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)
  - 変数 W を W = 0 で初期化し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)
  -
  - 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

4 Subproblem2 をクエリに対応

- **Subproblem2** でクエリにすると何が変わるかを考えよう
- 先ほどと同様に文字列の？に関する個数の列のうち、どれが欲しいかはクエリによって変わりうる
- しかし列の中身か変わらないので、どの列を使うかでクエリを場合分けしておくとよい
- B[?]B, B[?]C, C[?]C の？に関する個数の列が重要とする

4 Subproblem2 をクエリに対応

- **Subproblem2** でクエリにすると何が変わるかを考えよう

- クエリでは (y, z) が与えられる (金貨と銀貨の枚数)
- 書店の情報はクエリによらず同一

- あらかじめ全ての (y, z) について列挙することを考える
- まず d, e, f (本の値段) は sort してあるとしていい
- d は先頭から足していって y 以下となる最後のところで打ち切っていい それ以降の金貨で買える本は買えないため
- e についても同様

4 Subproblem2 をクエリに対応

- **Subproblem2** でクエリにすると何が変わるかを考えよう

- あらかじめ全ての (y, z) について列挙することを考える

- このとき金貨と銀貨を同一視して良い
- 書店 D と E (それぞれ金貨、銀貨だけで買うやつ) から買う戦略を考えると示せる

- つまり d = (2, 3, 3), e = (1, 4, 10), f = (1, 5), (y, z) = (4, 5) なら
- d = (2), e = (1, 4) と打ち切っていい
- また d と e は (1, 2, 4) とまとめて、通貨は 9 枚あるとしていい

4 Subproblem2 をクエリに対応

- **Subproblem2** でクエリにすると何が変わるかを考えよう

- このとき金貨と銀貨を同一視して良い
- 書店 D と E (それぞれ金貨、銀貨だけで買うやつ) から買うことを考えると示せる

- D と E で買うと W は +2 で F では +1
- よってこの問題は価値が 1 か 2 かのナップサック問題となる

- 価値が 1 か 2 という設定を使いたい

4 Subproblem2 をクエリに対応

- **Subproblem2** でクエリにすると何が変わるかを考えよう

- この問題は価値が 1 か 2 かのナップサック問題となる

- 価値が 1 なら → 安い方から貪欲
- 2 が増えるとどうなるか？ → 価値を 1 種類にしたい
- 価値 1 のものを 2 つ買えば価値 2 とみなせる

- 価値が 1 のものを買う個数の偶奇を固定し、価値が 1 のものを安い順に 2 つずつ merge して価値が 2 のものを作る

4 Subproblem2 をクエリに対応

- **Subproblem2** でクエリにすると何が変わるかを考えよう

- 価値が 1 のものを買う個数の偶奇を固定し、価値が 1 のものを安い順に 2 つずつ merge して価値が 2 のものを作る

- d = (2, 3, 3), e = (1, 4, 10), f = (1, 5), (y, z) = (4, 5) なら
- d = (2), e = (1, 4) と打ち切る
- また d と e は (1, 2, 4) とまとめて、通貨は 9 枚あるとしていい

- f から奇数買う → (1, 2, 4) から 9 - 1 = 8 枚で買う
- f から偶数買う → (1, 2, 4, 6 (= 1 + 5)) から 9 枚で買う

4 Subproblem2 をクエリに対応

- **Subproblem2** でクエリにすると何が変わるかを考えよう

- 価値が 1 のものを買う個数の偶奇を固定し、価値が 1 のものを安い順に 2 つずつ merge して価値が 2 のものを作る

- f から奇数買う → (1, 2, 4) から 9 - 1 = 8 枚で買う
- f から偶数買う → (1, 2, 4, 6 (= 1 + 5)) から 9 枚で買う

- どれだけ買えるかは二分探索！
- 数列を小さい順に足していってある値を超えるところを調べる

4 Subproblem2 をクエリに対応

- **Subproblem2** でクエリにすると何が変わるかを考えよう

- これを全ての (y, z) について探索したい
- f から偶数買う場合を計算できれば同じことを 2 回すれば良い

- y を固定して z を増やしていく
- y が固定　　→ d を打ち切るところは同じ
- z が増えていく → e を打ち切るところが後ろになっていく

- つまり「前から足していってある値を超えるのはいつか」の二分探索の対象となる数列が増えていく

4 Subproblem2 をクエリに対応

- **Subproblem2** でクエリにすると何が変わるかを考えよう

- これを全ての (y, z) について探索したい
- 以下を処理したい
- ① 数列にある値を追加
- ② 小さい順に足していってある値を超えるのはいつか計算

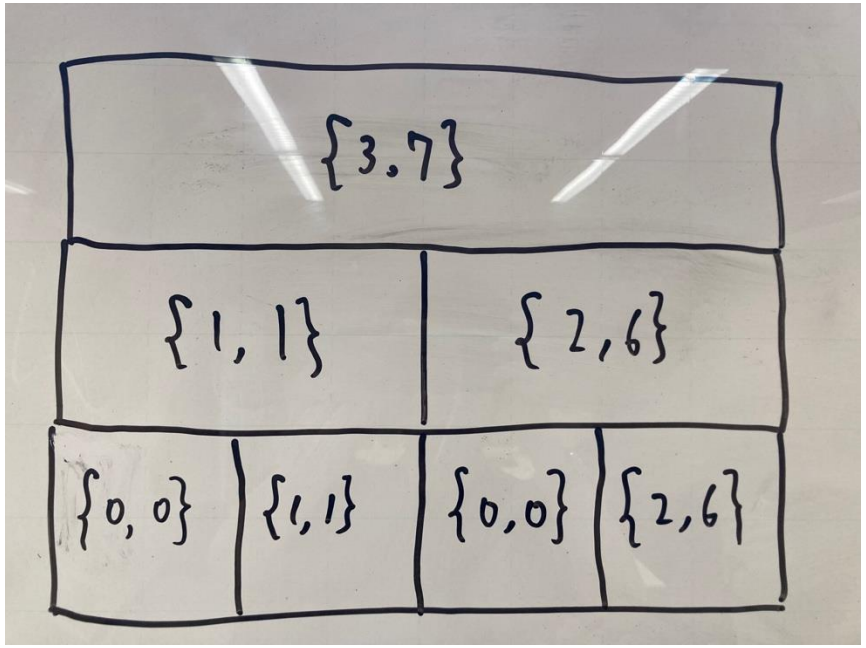
- ただ値は小さい
- どの値が数列に何回登場したかを管理できないか
- 「**個数と総和を管理する Segment Tree**」を使う
- Segment Tree のノードに {個数, 総和} の pair を置く

4 Subproblem2 をクエリに対応

- **Subproblem2** でクエリにすると何が変わるかを考えよう

- 「**個数と総和を管理する Segment Tree**」を使う
- Segment Tree のノードに区間の {個数, 総和} の pair を置く

- (1, 3, 3) なら右図のようになる



- ① 値の追加
- これはいつもと同じ
- O(logN)

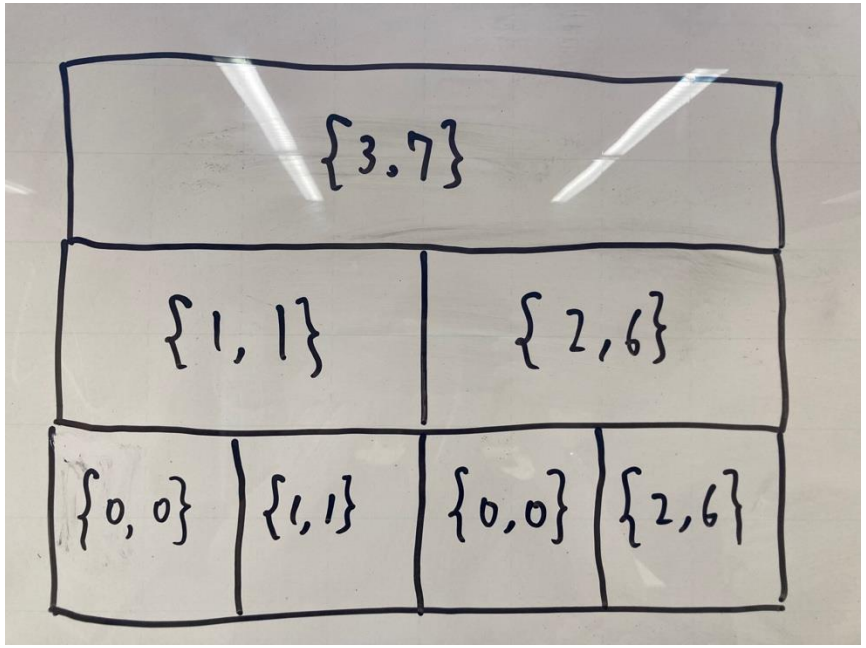
4 Subproblem2 をクエリに対応

- **Subproblem2** でクエリにすると何が変わるかを考えよう

- ②小さい順に足していってある値を超えるのはいつか計算

- これを「どの値まで足して良いか」
- に変換する

- (1, 3, 3) と 5 なら 2 まで足していい
- 1 から M まで全て足せるような
- M を求めたい



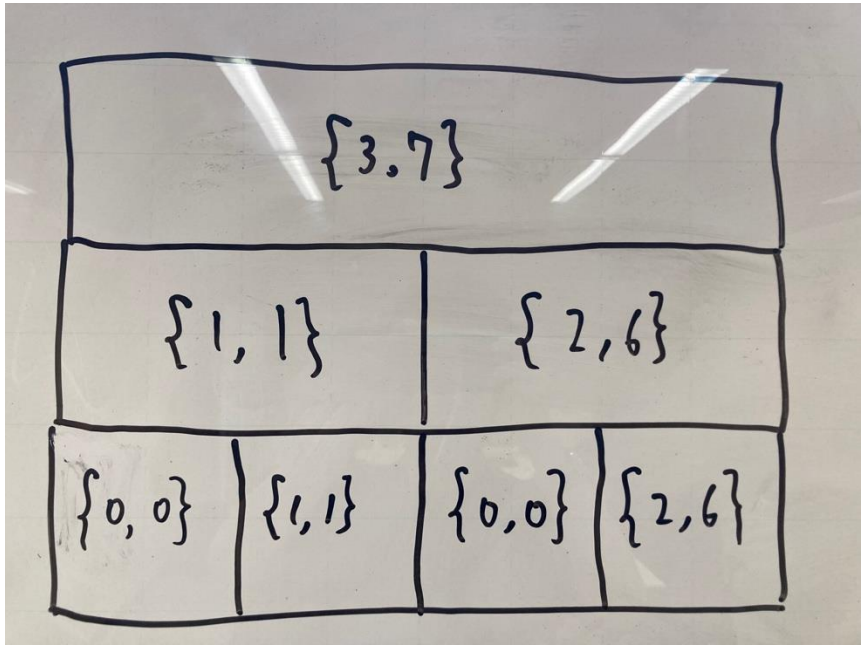
4 Subproblem2 をクエリに対応

- **Subproblem2** でクエリにすると何が変わるかを考えよう

- ②小さい順に足していってある値を超えるのはいつか計算

- 1 から M まで全て足せるような
- M を求めたい

- Segment Tree 上の二分探索！
- M で二分探索する → O(log^2N)



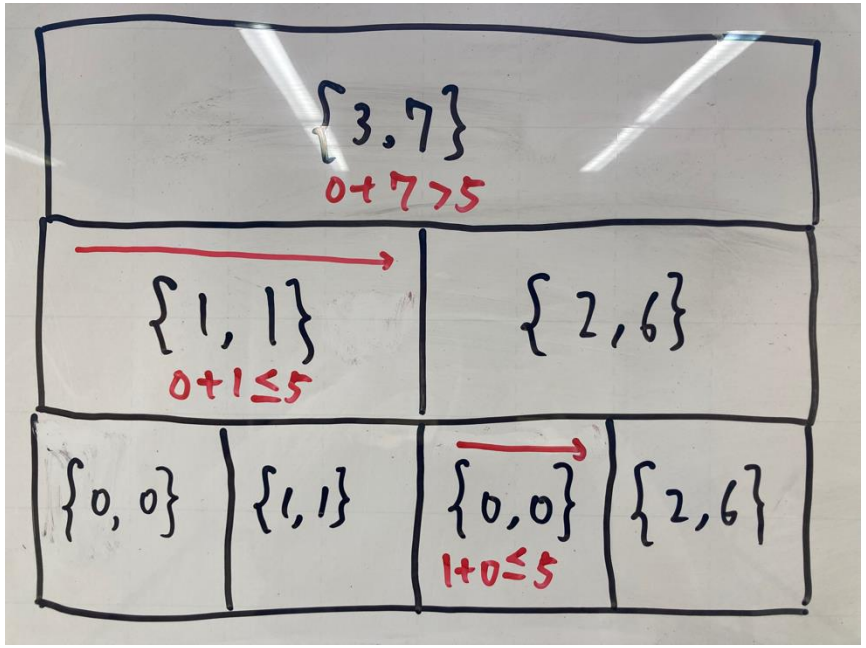
4 Subproblem2 をクエリに対応

- **Subproblem2** でクエリにすると何が変わるかを考えよう

- Segment Tree 上の二分探索
- なんと O(logN) にもできる

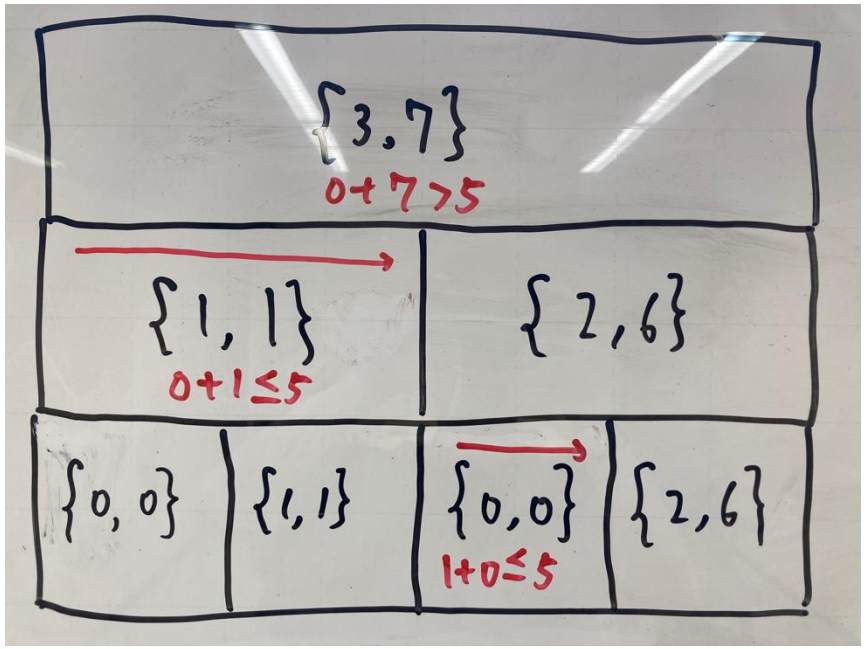
- 図のイメージで答えを上の bit から
- 決めつつ下のノードへ向かう

- インターネットに色々あります



4 Subproblem2 をクエリに対応

- **Subproblem2** でクエリにすると何が変わるかを考えよう
- y を固定したときの (y, z) について O(NlogN) で計算できた
- y を動かすと O(N^2logN) で全ての
- (y, z) について計算できる
- よって O(N^2logN + Q) で解けた



4 まとめ

- **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
- **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なる Segment Tree を使う

- **Subproblem1:** O(N^2 + Q) や O(N^2 + QlogQ)
- **Subproblem2:** O(N^2logN + Q) や O(N^2log^2N + Q)

- よって全体では O(N^2logN + Q)
- セグ木の二分探索で log がもう 1 つつく O(N^2log^2N + Q) はかなり TLE になりそう

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
- **Subproblem1** と **Subproblem2** で高速化できそうなところを探そう

5 Subproblem1 の高速化

- **Subproblem1** で高速化できるところを考えよう

  - **Subproblem1**

  - 数列 c が与えられる (C[?]C の？の個数に対応)
  - c の正の要素を 1 つ選んで減らす操作を Z 回行う (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)
  - $a \leqq \mathrm{sum}(c_i) - Z$ は成り立っています

5 Subproblem1 の高速化

- **Subproblem1** で高速化できるところを考えよう

- 部分和問題は DP[i][j] := i 番目まで見て総和が j を作れるかとする DP で O(N^2) で解けるのでした

- なんとか高速化できないか
- 小手先の改善を考える

- 同じ値をまとめて遷移しよう
- DP[i][j] := i までの値を処理して総和が j を作れるかとする

5 Subproblem1 の高速化

- **Subproblem1** で高速化できるところを考えよう

- DP[i][j] := i までの値を処理して総和が j を作れるかとする
- DP[i] から DP[i + 1] を作る

- i + 1 なる値がないならそのまま
- n 個あるとする
- DP[i + 1][j] が True となる条件は DP[i][j], DP[i][j - (i + 1)], DP[i][j - 2(i + 1)], ..., DP[i][j - n(i + 1)] のどれかが True

- これは i + 1 個飛ばしの累積和で高速化できる

5 Subproblem1 の高速化

- **Subproblem1** で高速化できるところを考えよう

- よって O((値の種類数) * N) になった
- 値の種類数は O(N) なので定数倍しか早くならない

5 Subproblem1 の高速化

- **Subproblem1** で高速化できるところを考えよう

- よって O((値の種類数) * N) になった
- 値の種類数は O(N) なので定数倍しか早くならない

- ということはなく計算量が O(N√N) に改善している
- 値の総和は O(N) なら値の種類数は O(√N) になっている

- 非自明に見えるが、実は 1 + 2 + ... + √N が N/2 ぐらいなので成立している

5 Subproblem2 の高速化

- **Subproblem2** で高速化できるところを考えよう

  - **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)
  - 変数 W を W = 0 で初期化し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)
  -
  - 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

5 Subproblem2 の高速化

- **Subproblem2** で高速化できるところを考えよう

- 書店 D と書店 E で買う本の数を O(N^2) 通り全探索する
- 書店 F でどれだけ買えるかを知りたい
- D と E で使った分から F でどれだけ使えるか分かる
- あとはどれだけ買えるかを二分探索する O(logN)

- よって O(N^2logN) などで解けていた

- D と E で買う本の合計 M を全探索できないか

5 Subproblem2 の高速化

- **Subproblem2** で高速化できるところを考えよう

- D と E で買う本の合計 M を全探索できないか
- D と E で買ってない中で安い方を買うことを M 回繰り返す
  貪欲が最適
- D と E で合計 M 冊買うのに必要な金額は d, e を merge して
  sort しておけば二分探索で O(logN)
- 残りで F で買える分も二分探索で O(logN)

- つまり O(NlogN) で解ける

5 まとめ

- **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必要な金額の計算は F と同様の貪欲アルゴリズムになります

- **Subproblem1:** $O(N\sqrt{N})$
- **Subproblem2:** $O(N\log N)$

- よって全体では $O(QN\sqrt{N})$

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:**
- ? に 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
- 部分和である区間のどれかが作れますかというもの

- O(N√N + Q) などで解けます

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

- **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

- 小課題 4 でやったように価値が 1 のもの 2 つをまとめる

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

- **Subproblem2:**
- 小課題 4 でやったように価値が 1 のもの 2 つをまとめる
- というか Z = 0 なので小課題 4 の解法と同じ
- 小課題 4 では y を固定して z を増やしたが z を (0 に)固定して y を増やすイメージ

- O(NlogN + Q) などで解けます
- 今回は Segment Tree 上の二分探索で log が余計に 1 つつけて O(Nlog^2N + Q) にしても間に合うと思います

7 まとめ

- **Subproblem1:** 普通の部分和問題の形になります
- **Subproblem2:** 小課題 4 の前準備と同じことをすればいいです
  Z = 0 なので計算量から N が 1 つ落ちます
- log が 1 つ余分についても余裕だと思います

- **Subproblem1:** O(N√N + Q)
- **Subproblem2:** O(NlogN + Q) や O(Nlog^2N + Q)

- よって全体では O(N√N + Q)

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 Subproblem1 でアイデアを合体

- **Subproblem1** で高速化とクエリ対応の両方をしよう

  - **Subproblem1**

  - 数列 c が与えられる (C[?]C の？の個数に対応)
  - c の正の要素を 1 つ選んで減らす操作を Z 回行う (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)
  - $a \leqq \mathrm{sum}(c_i) - Z$ は成り立っています

8 Subproblem1 でアイデアを合体

- **Subproblem1** で高速化とクエリ対応の両方をしよう

- 高速化のアイデアは部分和問題の DP を i までの値を処理して総和が j を作れるかとするものだった
- クエリ対応のアイデアは DP テーブルの途中の状態でクエリに答えて、DP テーブルを更新してを繰り返すものだった

- この 2 つを合体しよう！

8 Subproblem1 でアイデアを合体

- **Subproblem1** で高速化とクエリ対応の両方をしよう

- クエリごとにどの DP テーブルを使うかを決める
- c_i を大きい方から M まで全部足して Z 以下となる最小の M が DP テーブルの 1 つめのキーとなる

- この M を求めるのは簡単
- 小課題 4 で使った Segment Tree を使ってもいいし c の中身は変わらないので累積和 + 二分探索でも良い

8 Subproblem1 でアイデアを合体

- **Subproblem1** で高速化とクエリ対応の両方をしよう

- あとは DP[i][j] := i 以上の c_i の部分和で j を作れるか
- から i - 1 を追加で何個か追加してある範囲ができるかになる
- もはや文章では意味不明

- c = (1, 3, 3, 4) で Z = 5 とかなら 1, 3 まで消せる
- DP[4] (4 以上の要素のみでできる部分和の情報) と 3 を使うか使わないかからある範囲が作れるかを知りたい

8 Subproblem1 でアイデアを合体

- **Subproblem1** で高速化とクエリ対応の両方をしよう

- c = (1, 3, 3, 4) で Z = 5 とかなら 1, 3 まで消せる
- DP[4] (4 以上の要素のみでできる部分和の情報) と 3 を何個使うか (と z) からある範囲が作れるかを知りたい
- とりあえずある数 a が作れるか (範囲の長さ 1) とすると DP[4][[a, a + z]] と DP[4][[a - 3, a - 3 + z]] のどれかが True を確認
- 範囲にしても DP[4][[min(a), max(a) + z]] と DP[4][[min(a) - 3, max(a) - 3 + z]] のどれかが True を確認すれば良い

8 Subproblem1 でアイデアを合体

- **Subproblem1** で高速化とクエリ対応の両方をしよう

- c = (1, 3, 3, 4) で Z = 5 とかなら 1, 3 まで消せる
- DP[4] (4 以上の要素のみでできる部分和の情報) と 3 を何個使うか (と z) からある範囲が作れるかを知りたい
- DP[4][[min(a), max(a) + z]] と DP[4][[min(a) - 3, max(a) - 3 + z]] のどれかが True を確認すれば良い

- ただ確認する範囲の数がこのままでは O(N) になって TLE
- ぴえん

8 Subproblem1 でアイデアを合体

- **Subproblem1** で高速化とクエリ対応の両方をしよう

- DP’[i][j] := DP[i][j - k(i - 1)] が True となる最小の k とする
- そのような k がなければ +inf にする

- DP[4][[min(a), max(a) + z]] と DP[4][[min(a) - 3, max(a) - 3 + z]] のどれかが True を確認するというのは
- DP’[4][[min(a), max(a) + z]] の min ≦ k になる

8 Subproblem1 でアイデアを合体

- **Subproblem1** で高速化とクエリ対応の両方をしよう

- DP'[i][j] := DP[i][j - k(i - 1)] が True となる最小の k とする
- そのような k がなければ +inf にする

- DP' の計算は DP 配列をにらめば j = 0, 1, ..., の順に O(N) で列挙できる
- 計算すべき i は O(√N) 種類なので全体で O(N√N)
- 区間の min を取るのは Segment Tree でクエリあたり O(logN)

8 Subproblem1 でアイデアを合体

- **Subproblem1** で高速化とクエリ対応の両方をしよう

- DP’ の計算は DP 配列をにらめば i あたり O(N)
- 計算すべき i は O(√N) 種類なので全体で O(N√N)
- 区間の min を取るのは Segment Tree でクエリあたり O(logN)

- 全体では O(N√N + QlogN)

8 Subproblem2 でアイデアを合体

- **Subproblem2** で高速化とクエリ対応の両方をしよう

  - **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)
  - 変数 W を W = 0 で初期化し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)
  -
  - 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

8 Subproblem2 でアイデアを合体

- **Subproblem2** で高速化とクエリ対応の両方をしよう

- 高速化のアイデアは D と E で買う本の合計を全探索するものだった
- クエリ対応のアイデアは F で買う本の偶奇を場合分けして 2 つの本をまとめるものだった

- この 2 つを合体しよう！

8 Subproblem2 でアイデアを合体

- **Subproblem2** で高速化とクエリ対応の両方をしよう

- まずクエリについて d や e を金貨と銀貨で買える分で打ち切る
- これ以降はどうせ買えないのでないとしていい

- F で買う偶奇を固定すると結局 d の前何項と e の前何項と f を merge して sort して先頭から足していってある値を超えるのがいつか調べるものになる

8 Subproblem2 でアイデアを合体

- **Subproblem2** で高速化とクエリ対応の両方をしよう

- d の前何項と e の前何項と f を merge して sort して先頭から足していってある値を超えるのがいつか調べるものになる

- 小課題 4 と同じように値の分布を管理してどの値までは全部足せるかを調べたい
- d, e, f それぞれについては簡単に計算できてこれらを足し合わせれば良い
- d = (3, 6), e = (2, 2, 3, 3), f = (1, 8) で 2 までならそれぞれ {0 個, 総和 0}, {2 個, 総和 4}, {1 個, 総和 1} という感じ

8 Subproblem2 でアイデアを合体

- **Subproblem2** で高速化とクエリ対応の両方をしよう

- d, e, f それぞれについては簡単に計算できてこれらを足し合わせれば良い
- d = (3, 6), e = (2, 2, 3, 3), f = (1, 8) で 2 までならそれぞれ {0 個, 総和 0}, {2 個, 総和 4}, {1 個, 総和 1} という感じ
- 足し合わせれば {3 個, 総和 5} になる

- これの総和がある値以下になる最大の M が分かれば M 以下の個数がわかる
- M + 1 がいくつ足せるかは M 以下の総和から分かる

8 Subproblem2 でアイデアを合体

- **Subproblem2** で高速化とクエリ対応の両方をしよう

- d, e の打ち切りを考慮する必要があるが、これは d, e の累積和をとっておけば簡単にできる

- d = (3, 6) で 2 までなら {0 個, 総和 0} などの情報を用意するのは lower_bound を使えば O(logN)
- この操作は O(logN) 回あるのでクエリあたり O(log^2N)

- 前もって累積和を d, e, f について取得すれば操作が O(1)
- クエリあたり O(logN) になる

8 Subproblem2 でアイデアを合体

- **Subproblem2** で高速化とクエリ対応の両方をしよう

- この操作は O(logN) 回あるのでクエリあたり O(log^2N)
- 前準備でクエリあたり O(logN) になる

- F で買う個数の偶奇を固定したものがこれで解けた
- 2 回これをすればいい

- 全体で O((N + QlogN)logN) や O((N + Q)logN) で解ける

8 まとめ

- **Subproblem1:** O(N√N + Q)
- **Subproblem2:** O((N + QlogN)logN) や O((N + Q)logN)

- よって全体では O(N√N + QlogN) や O(N√N + Qlog^2N)
- おそらく **Subproblem2** の前準備をサボって O(N√N + Qlog^2N) にしても間に合うのではないか

- ところで実装はいかがでしょうか

8 まとめ

- 実装はとても大変そう
- ここに辿り着いた人はみんな辛いので遠くに行きたい気持ちになると思う
- 九州の写真を流そう

8 九州

- きれいだ
- 甑島大橋
- 鹿児島の西にある

8 九州

- きれいだ 現実に戻る
- 関門大橋
- 地下を徒歩で歩ける

8 まとめ

- 実装はとても大変だと思う
- 実装力も大事！
- 大変な問題を捨てる決断力も大事！

Step 9

得点分布

9 得点分布

- 理想:

点数	1	2	3	4	5	6	7	8	人数	累計
100	o	o	o	o	o	o	o	o	27	27

- 選抜という意味では差がつかないな

9 得点分布

- 現実:

点数	1	2	3	4	5	6	7	8	人数	累計
19	o	o	-	-	-	o	-	-	12	12
12	o	-	-	-	-	o	-	-	2	14
11	o	o	-	-	-	-	-	-	2	16
8	-	-	-	-	-	o	-	-	2	18
4	o	-	-	-	-	-	-	-	4	22
0	-	-	-	-	-	-	-	-	5	27