

# ICPC模擬地区予選2021

## A: Harvest

原案: darsein

問題文: darsein

データセット: riantkb

解答: beet, darsein, hos, riantkb

解説: darsein

# 問題概要

- \ N種類の作物があり、それぞれ収穫期間が $[s_i, t_i]$ である
- \ K回収穫を行う。j回目の収穫は $h_j$ に行い、収穫期間内かつまだ収穫されていない作物をすべて収穫する
- \ K回の収穫後に収穫された作物の種類数を求めよ。
- \ 制約
  - \  $1 \leq N, K \leq 10^5, 1 \leq s_i \leq t_i \leq 10^9, 1 \leq h_j \leq 10^9$

# 解法

- \ 各植物  $i$  について、収穫時期開始日  $s_i$  以降で一番早い収穫日  $f_i$  を求める
  - \ 収穫予定日の列  $\mathbf{h} = [h_1, \dots, h_K]$  を  $s_i$  で二分探索:  $O(\log K)$
- \ 収穫日  $f_i$  が  $t_i$  以前であれば植物  $i$  は収穫される  
→ 答えを1増やす
- \ 全体で計算量は  $O(N \log K)$
- \ 二分探索をせず  $\mathbf{h}$  を全走査して  $f_i$  を求めると  $O(NK)$  で TLE

# 統計情報

\ AC / trying teams

\ 35 / 35 (100.00%)

\ First Acceptance

\ ゲスト: Rubikun (00:03)

\ 現役: SPJ (00:05)

# ICPC模擬地区予選2021

## B: Parse the Syntax Tree

原案: climpet

問題文: climpet

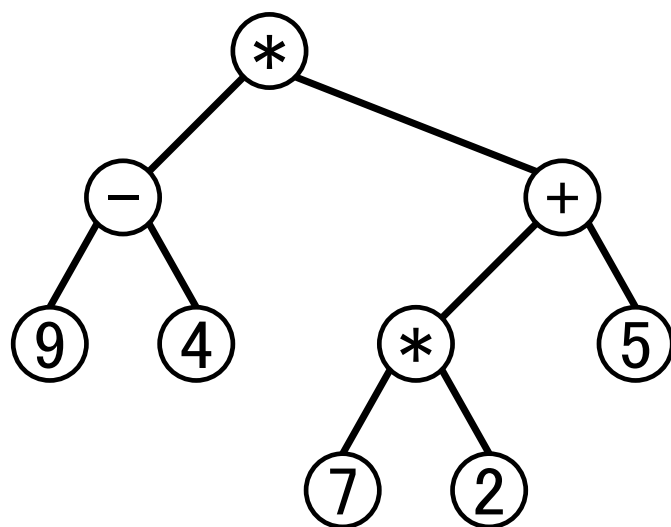
データセット: tsutaj

解答: beet, climpet, hos, riantkb, tsutaj

解説: climpet

# 問題概要

数式の構文木が与えられるので、計算せよ。



. . . \* . . . . .  
. - . . . . . + .  
9 . 4 . . \* . . 5  
. . . . 7 . 2 . .

## 解法1:再帰下降構文解析

$\text{calc}(t, l, r)$  = ( $t$  行目以下、 $l$  列目から  $r$  列目までの計算結果) とする。問題文中の BNF を参考に、次のように実装できる。

1.  $s_t[l..r]$  から、ピリオドでない文字を見つける。それが  $x$  列目にあるとする。
2. その文字が数字ならば、その値をそのまま返す。
3. その文字が演算子ならば、 $\text{calc}(t + 1, l, x - 1)$  と  $\text{calc}(t + 1, x + 1, r)$  にその演算を適用した値を返す。

## 解法2: 下から見る

- キューを一つ用意する。
- 下の行から順に、次のことを行う。
  - 文字を左から順に見る。その文字が数字ならば、キューにその値を追加する。その文字が演算子ならば、キューの先頭二要素を取り出し、それらに対する演算結果をキューに追加する。
- 最後にキューに一要素残るので、それが答え。



## 余談

同じく二次元の構文解析として、10 年ほど前の ICPC アジア地区予選で、[ASCII Expression](#) という問題が出題されています。もし解いたことがなければ練習しておきましょう。

# 統計情報

- AC / trying teams
  - 34 / 34
- First acceptance
  - The atama (8 分)

# ICPC2021 模擬地区予選 C 問題

## Permutation Magic

原案: climpet

問題文: tsutaj

データセット: climpet

解答: beet, climpet, hos, tsutaj

解説: tsutaj

2022 年 3 月 6 日

## Permutation Magic

- ▶ 長さとともに  $N$  である数列  $A = \{a_i\}$  と  $B = \{b_i\}$  が与えられる
  - ▶ 要素の値は全て 1 以上  $M$  以下
- ▶ 数列  $A$  は、長さ  $M$  の順列  $P = \{p_i\}$  を以下のように用いて  $A' = \{a'_i\}$  にできる
  - ▶  $a'_i = p_{a_i}$
  - ▶  $P$  は長さ  $M$  の任意の順列 (好きに決めてよい)
- ▶  $A'$  としてあり得る数列であって、以下を満たす数列を出力せよ
  - ▶  $B$  とのハミング距離が最も短い
  - ▶ 上記の条件を満たすもののの中で辞書順で最も小さい

# ハミング距離の最小化のみを行う場合

- ▶  $A_i = x$  だったものを  $A'_i = y$  にするとき、ハミング距離にどう影響するか？
  - ▶  $B_i \neq y$  のとき、ハミング距離が 1 かかる
  - ▶  $B_i = y$  のとき、要素が一致しているのでハミング距離に影響しない
- ▶ また、 $A$  から  $A'$  に変えるとき、順列  $P$  を使っている
  - ▶ 順列は要素が相異なるので、 $(a_u = a_v) \iff (a'_u = a'_v)$
- ▶ 以上より、 $A$  に順列を作用させたときのハミング距離への影響は以下のようにまとめられる

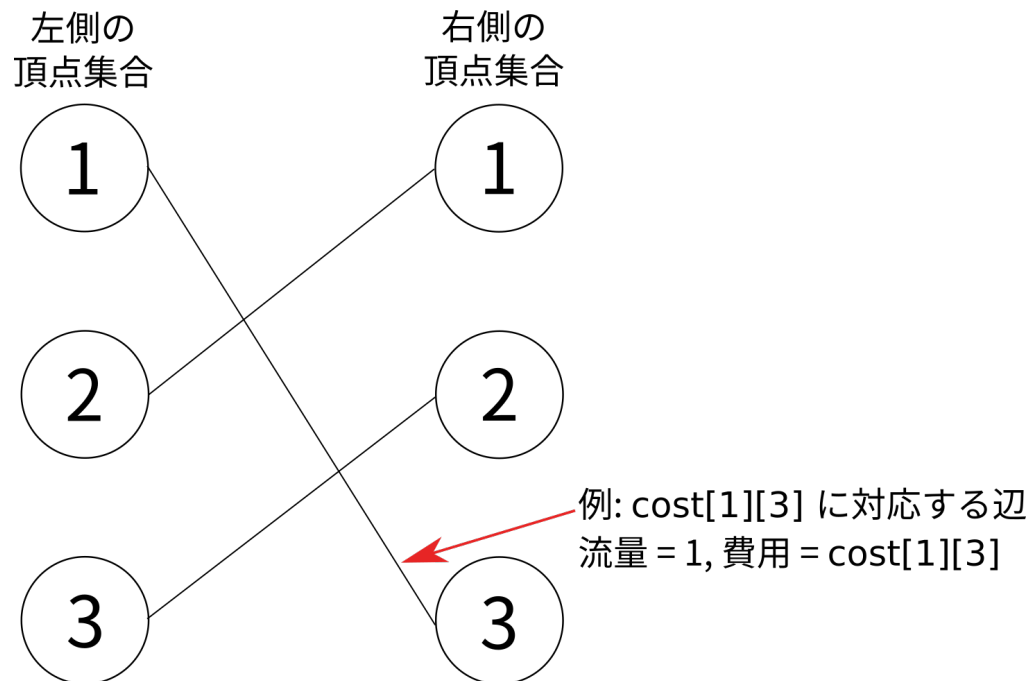
## ハミング距離への影響

$A$  で  $x$  であるものを  $A'$  で  $y$  に変化させたとき、ハミング距離への影響は以下のように求められる

$$\text{cost}[x][y] := \underbrace{|\{i \mid A_i = x \wedge B_i \neq y\}|}_{B \text{ と一致していないもの}} \quad (1)$$

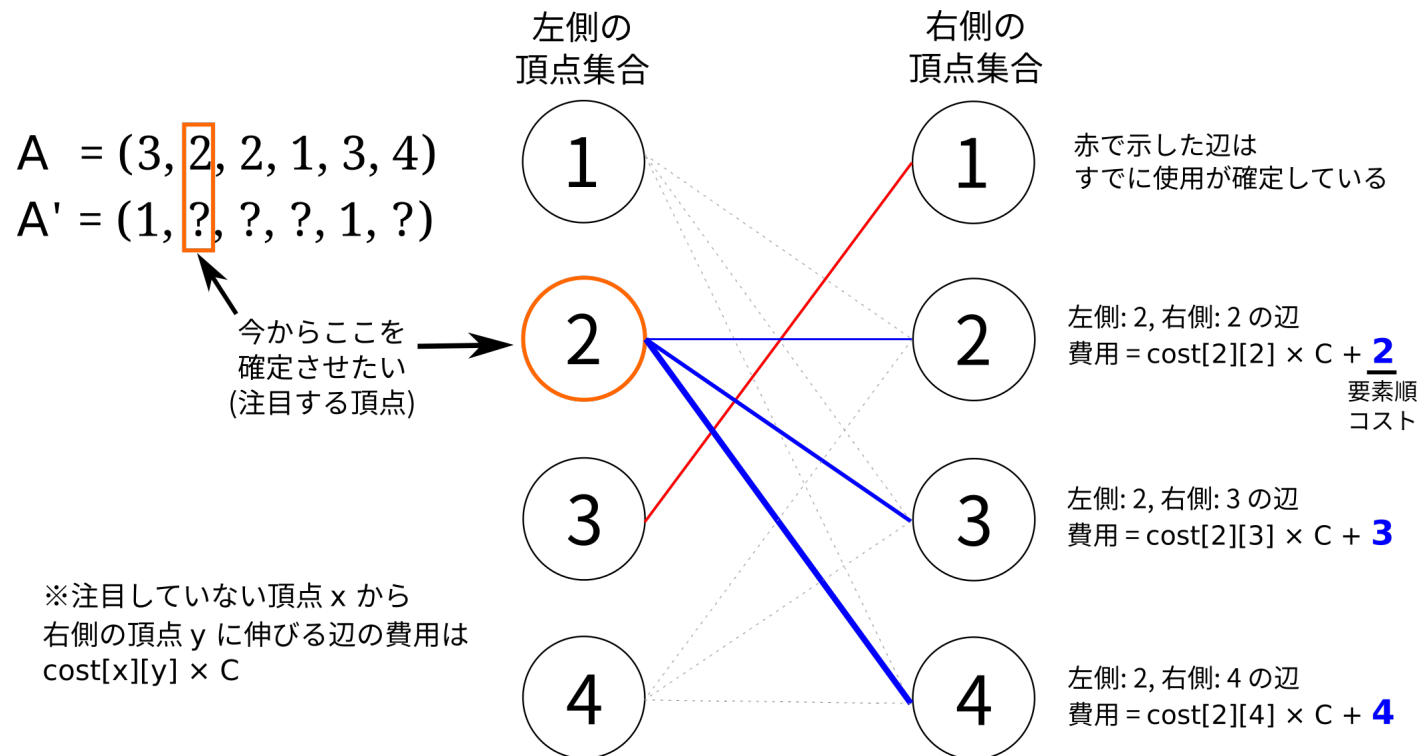
# ハミング距離の最小化のみを行う場合

- ▶ ある 1 種類の値についての距離への影響は前ページで求められたが、結局ハミング距離を最小化するにはどうすればよいか？
- ▶ “順列を作用させる” というのを、グラフのようにとらえてみよう
  - ▶  $\text{cost}[x][y]$  は、下図における左側の頂点集合の  $x$  番目と、右側の頂点集合の  $y$  番目を結ぶ辺のコスト
  - ▶ 順列を考えているので、各頂点の次数はちょうど 1 であるべき
- ▶ これは **二部グラフの最小費用流** である！
- ▶ このグラフを作ってフローを流せばハミング距離の最小化ができる



# 辞書順最小化も行う場合

- ▶  $A'$  で出現が早いもののほど値を貪欲に小さくしていきたい
- ▶ 以下のように高々  $M$  回フローを流して、1 種類ずつ決定すれば良い
  - ▶ はじめは順列  $P$  の要素が全て未確定
  - ▶  $A$  の要素を前から順に見ていく
    - ▶  $p_{a_i}$  の値が確定していないなら、下図のようにフローを流して  $p_{a_i}$  を確定



- ▶  $C$  は十分大きな定数
- ▶ 注目する要素を、できるだけ小さい要素に割り当てるようにするための“要素順コスト”に注意

- ▶ 前ページでは 1 種類ずつ決定していたが、複数種類をまとめて決定することもできる
  - ▶  $K$  種類をまとめて決定するとき、前ページの“要素順コスト”を  $M$  進数で入れていく
    - ▶ 1 種類についての要素順コストを 0 から  $M - 1$  までで表現するとき、 $K$  種類の場合は 0 から  $M^K - 1$  までで表現可能
    - ▶  $A$  での出現順が先のものほど優先度が高いので、大きい位に割り当てる
  - ▶ これを行うことで  $K = \log_M X$  種類ずつ決定することができる
    - ▶  $X$  は要素順コストの最大値
    - ▶  $\text{cost}[x][y]$  に掛けるための十分大きい定数  $C$  もこれに応じて大きくする
    - ▶ やりすぎるとオーバーフローするので注意



## ▶ Writer 解

- ▶ beet (C++・276 行・6289 bytes)
- ▶ climpet (C++・99 行・1874 bytes)
- ▶ hos (C++・199 行・6066 bytes)
- ▶ tsutaj (C++・230 行・7830 bytes)

## ▶ 統計

- ▶ AC / tried: 24 / 54 (44.4 %)
- ▶ First AC
  - ▶ Saitama University: seica is gone (24 min 34 sec)

# ICPC模擬地区予選2021

## D: MFP: Most Fluctuated Player

原案: climpet

問題文: darsein

データセット: climpet

解答: beet, climpet, hos

解説: climpet

# 問題概要

- $N$  人でクイズ大会を行う。
- $i$  番目のクイズでは参加者  $a_i$  が  $p_i$  点を得る。 $p_i$  は正とは限らない。
- 各参加者は、順位が変動するたびに、(順位の変動幅の絶対値)枚のコインを獲得する。これは他の参加者の正解による順位変動を含む。
- $Q$  問のクイズ終了後の、各参加者の獲得したコインの枚数を求めよ。

## 制約

- $1 \leq N, Q \leq 10^5$
- $|p_i| \leq 10^9$

# 順位の求め方

- まずは、各参加者の現在の順位を求められるようになる必要がある。
- 予め  $Q$  問のクイズを順に処理し、各参加者の得点がどのように変化するかを求めておく。
- これらの得点を座標圧縮しておく。以下、参加者の得点は  $1$  以上  $Q + 1$  以下であるとみなす。
- 得点が  $s$  の人の順位は、得点  $s$  以下の参加者の人数から求められる。これは BIT を用いると効率よく管理できる。

# コインの枚数の処理

- 各参加者がコインを獲得するタイミングには、以下の二つがある。
  - (a) その参加者自身が問題に正解したとき。
  - (b) 他の参加者が問題に正解したとき。
- (a) によるコインの獲得枚数は、順位から容易に計算できる。
- (b) では、ある参加者の得点が  $s$  から  $t$  に変動したとき、得点が  $\min(s, t)$  以上  $\max(s, t) - 1$  以下の参加者に、1 枚ずつコインが配られる。
- 複数人が一気にコインを獲得する可能性があるため、愚直に処理することはできない。そこで、(b) によるコインの獲得については、その人が次に問題を正解するとき  
に遅延処理する。

# コインの枚数の遅延処理

- $C$  を整数配列とする。
- ある参加者の (座標圧縮後の) 得点が  $s$  から  $t$  に変動したとき、 $C[\min(s, t)], \dots, C[\max(s, t) - 1]$  に 1 を足す。
- 配列  $D$  を用意する。参加者  $i$  が前回正解して得点  $s$  になった場合、そのときの  $C[s]$  を  $D[i]$  として保持する。
- 得点  $s$  の参加者  $i$  が、前回の順位変動後 (b) で得たコインの枚数は、 $C[s] - D[i]$  として求められる。これはその参加者が次に問題に正解したときに遅延処理する。
- 最後に、全参加者が 0 点問題に正解したとみなして、更新処理を行う。
- 配列  $C$  は、BIT を用いることで効率的に実現できる。  
(順位を求めるための BIT とは別に用意する。)

## まとめ・統計情報

- 計算量は全体で  $O(N + Q \log Q)$  時間となる。
- AC / trying teams
  - 22 / 22
- First acceptance
  - mirei\_minami (43 分)

# JAG ICPC模擬地区予選2021

## E: Underground's SUNDAY

---

原案: shora\_kujira16

問題文: riantkb

データセット: Darsein

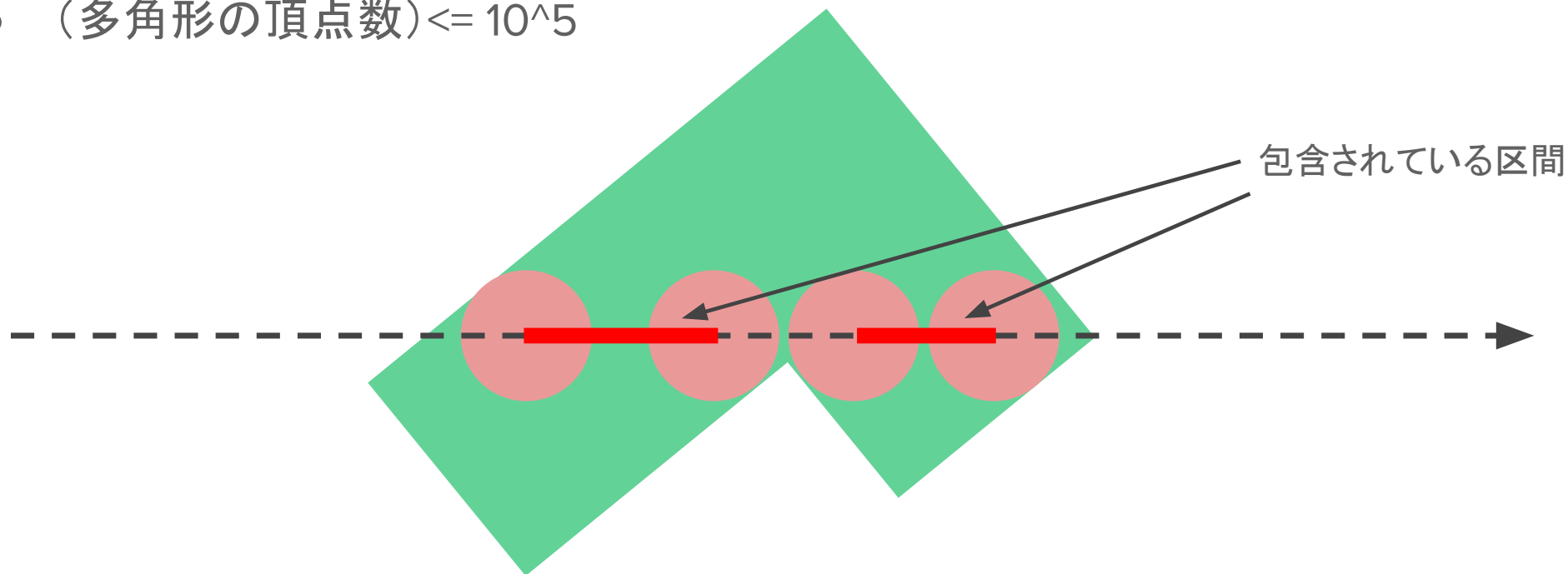
解答: climpet, hos, riantkb

解説: riantkb



# 問題概要

- 二次元平面上に凸とは限らない多角形がある
  - サンプルには凸なものしか入ってなかったようです ...
- x 軸上を移動する半径 R の円が多角形に完全に包含される総時間を求めよ
- (多角形の頂点数)  $\leq 10^5$

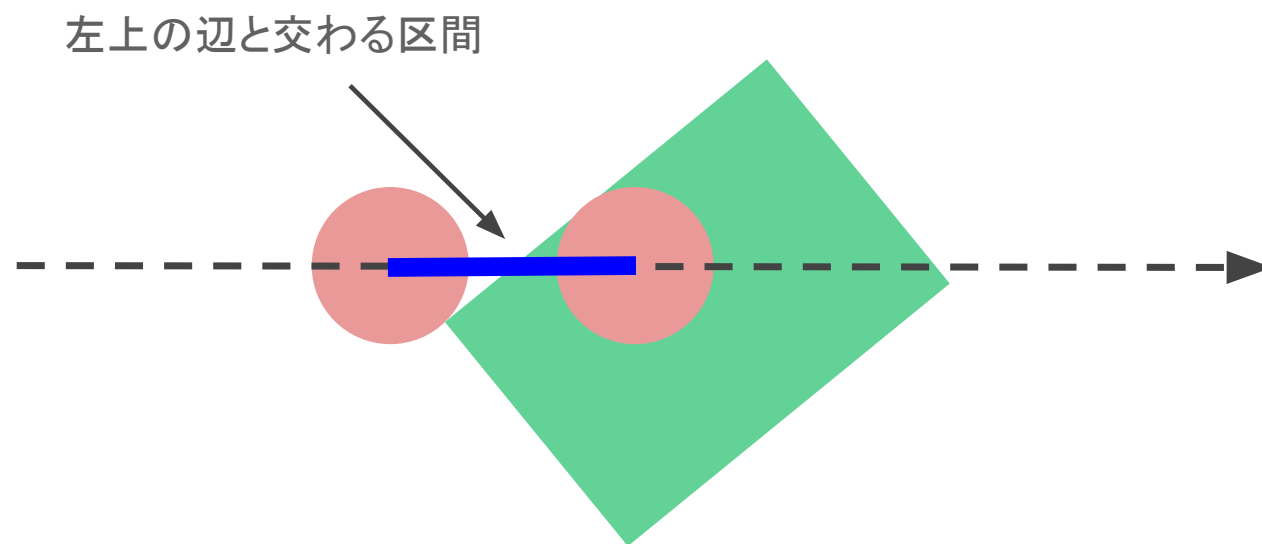


## 考察

- 円が多角形に包含されているかどうかが入れ替わるタイミングでは、必ず多角形と円は(辺または頂点で)接している
- つまり、円と多角形が接するタイミング全てに対し以下の 2 つが判定できれば良い
  - 他の辺が円と交わっていない
  - 円の中心が多角形の内部にある
- しかし、どちらも愚直に判定すると多角形の頂点数を  $N$  として  $O(N)$  にかかるため、全体で  $O(N^2)$  になってしまう

## 考察

- 多角形の全ての辺と円が交わっていないかを高速に判定したい
- これは、多角形の一辺(線分)と円の交差条件(いつからいつまで交差しているか)を求めておくことで、「いま何本の辺と交わっているか」を管理できるので判定できる



# 考察

- 円の中心が多角形の内部にあるかを高速に判定したい
- これは、 $x$  軸と各辺がいつ交わるかを求めておくことで、多角形の辺と奇数回交わったならば多角形の内部であると言える
  - $x$  軸にピッタリ重なる辺や端点が  $x$  軸上にある辺の処理が面倒になるため、実際には直線  $y = 0.5$  と各辺の交点を管理すると楽
- 別の方法として、各頂点を反時計回りにした上で、  
 $y$  座標が小さい方へ進む辺を横切った場合内部に入った、  
 $y$  座標が大きい方へ進む辺を横切った場合外部へ出た、とすることもできる

# 解法

- 各辺について、以下のタイミングを頑張って求めておく
  - 辺と円が交わり始めるタイミング
  - 辺と円が交わり終わるタイミング
  - 円の中心が辺上に乗るタイミング
- それらを早い順に見て、何本の辺と交わっているかと円の中心が多角形の内部にあるかどうかを管理する
- 1本の辺とも交わってないかつ中心が多角形の内部にある場合に、次のタイミングまでの時間を答えに足す
- 計算量は  $O(N \log N)$  となる

# 解法

- 実装によっては  $x$  軸に平行な辺、 $y$  軸に平行な辺、端点が  $x$  軸上にある辺などで壊れるので注意が必要
  - 惜しい提出が WA になっているのはおそらく大体この辺のせい

## ジャッジ解

- climpet (C++): 156 lines, 3.3 kB
- hos (C++): 156 lines, 5.0 kB
- riantkb (C++): 151 lines, 4.1 kB

## 統計情報

- Acceptances / Submissions
  - 4 / 30 (13.33 %)
- AC teams / Trying teams
  - 4 / 10 (40.00 %)
- First Acceptance
  - SPJ (220 min)



# JAG ICPC模擬地区予選2021

## F: qarentheziz zepuence

---

原案: riantkb

問題文: riantkb

データセット: Darsein

解答: beet, hos, riantkb

解説: riantkb

## 問題概要

- 括弧列  $S$  に対し、以下のクエリを  $Q$  個処理せよ
- クエリ1:  $S$  の  $l$  文字目から  $r$  文字目までについて、 $'($  と  $)'$  を flip する
- クエリ2:  $S$  の  $l$  文字目から  $r$  文字目までからなる文字列に対し、  
以下の操作を好きなだけ行ってバランスの取れた  
括弧列にする時の操作の最小回数
  - 操作1:  $S$  の先頭に  $'($  を足す
  - 操作2:  $S$  の末尾に  $)'$  を足す
  - 操作3:  $S$  の隣り合う 2 つの文字を swap する
- $|S|, Q \leq 150,000$

# 考察

- まず、単一の文字列に対し最小の操作回数を求めることを考える
  - 操作1、2については両方行う必要はなく、開き括弧と閉じ括弧の個数が一致していない時に必要数分のみ最初に行うとして問題ない  
(以下ではすでにそれを行ったものとする)
- 同じ文字同士を swap させる意味はなく、また “()” → “)(” とするメリットもないため、“)(” → “()” のみ考えれば良い
- このとき、累積和を考えると swap した箇所が 2 だけ大きくなることがわかる

(	)	)	)	(	(
1	0	-1	-2	-1	0
(	)	)	(	)	(
1	0	-1	0	-1	0

## 考察

- バランスの取れた括弧列である必要十分条件が累積和で負の値が存在しない (かつ全体の和が 0) であることを踏まえると、操作回数の下界は累積和を  $s_i$  と置くと  $\sum_{s_i} \max(0, \lceil \frac{-s_i}{2} \rceil)$  であるとわかる
- かつ、上記は達成可能である
  - バランスの取れた括弧列でないときに必ず負の位置を +2 するという swap が可能であるということが言えればよいが、例えば  $s_i$  が最小となる  $i$  がそれに該当する

# 考察

- 今回の問題を考える
- flip クエリを処理しつつ「ある区間の累積和である値以下のものの総和」がわかれば良い
  - 切り上げをしなければならないので、場合によってはある値以下の奇数である値の個数、などもわかる必要がある
- これは平方分割で実現できる

# 解法

- 以下では、長さ  $B$  のバケット  $A$  個に分割するとする
- 例えば平方分割の各バケットに以下の値を持たせると実現できる
  - 各要素が '(' (+1) か ')' (-1) か
  - そのバケットが flip しているかどうか
  - バケットの左端から累積和を取ったとき、 $-B \sim B$  の値がそれぞれ何回登場するか
    - これを  $\text{freq}$  と呼ぶことにする
  - $\text{freq}_i$  の累積和
  - $i * \text{freq}_i$  の累積和
  - $i$  が奇数のところのみの  $\text{freq}_i$  の累積和

# 解法

- 変更クエリは以下のように処理する
- そのバケットが完全に包含される場合は、flip しているかどうかのフラグのみ反転させる
- バケットの一部が含まれている場合、その区間の各要素をそれぞれ flip し、その後累積和等を計算し直す
- 計算量は  $O(A + B)$

# 解法

- 出力クエリは以下のように処理する
- 左から処理する。バケットの一部のみ含まれる場合は愚直に計算し、完全に包含される場合はそのバケットに入ってきたタイミングでの「深さ」について  $((-\text{深さ}) \text{ 以下の値について総和} + \text{奇数の数}) / 2$  を計算すればよい
- 計算量は  $O(A + B)$



# 解法

- よって、 $A = B = O(\sqrt{N})$ 、とすることで、計算量は  $O(N + Q\sqrt{N})$  となる
  - 定数倍がキツかったり、 $\log$  がつくと基本的には通らなかったりすると思いますが、愚直  $\Theta(NQ)$  が通らないように調整した結果です、すみません。
  - 複数個のジャッジ解が DOMjudge 上で 0.6 ~ 0.7 sec 程度で通ることを確認しています。

## ジャッジ解

- beet (C++): 164 lines, 3.3 kB
- hos (C++): 246 lines, 5.7 kB
- riantkb (C++): 217 lines, 5.4 kB

## 統計情報

- Acceptances / Submissions
  - 4 / 16 (25.00 %)
- AC teams / Trying teams
  - 4 / 7 (57.14 %)
- First Acceptance
  - UT a.k.a Is (155 min)

# ICPC2021 模擬地区予選 G 問題

## Pizza Delivery

原案: smiken

問題文: tsutaj

データセット: riantkb

解答: beet, hos, riantkb, tsutaj

解説: tsutaj

2022 年 3 月 6 日

## Pizza Delivery

- ▶  $N$  人の客が同時にピザの注文を行った
  - ▶  $t_i$ : ピザ屋から客  $i$  の家までの所要時間
    - ▶ 客  $i$  の家からピザ屋まで帰るときも同様
  - ▶  $a_i$ : 客  $i$  の怒りっぽさ
- ▶ これらの注文を一人の配達員が捌いていく
- ▶ 客は配達時間が遅くなるほど・先に配達された別の客の数が多いほど、ストレスがたまる
  - ▶ 問題文の  $s_i = a_i \times (h_i + p_i)$  に相当する部分
  - ▶  $p_i$ : 客  $i$  に配達する前に配達された客の人数
- ▶  $N$  人の客のストレス量の合計の最小値はいくつになるか？

# 想定誤解法

- ▶  $N$  人の客を適切に順序付けて、その順番通りに配達していく問題
- ▶ “ $dp[S] :=$  注文を届け終えた客の集合が  $S$  であるときの、ストレス量の合計の最小値” という bitDP が考えられる
  - ▶ しかし  $O(2^N)$  かかるので間に合わない
- ▶ bitDP では客の順序を全て考慮することができるが、本当に全部の順序を考えなければならないのか？
  - ▶ もう少し楽をできないか考えてみよう

## $N = 2$ の場合

- ▶ 簡単のため、客が 2 人しかいない場合をまず考える
  - ▶ 以降、客のストレス量の合計を  $X$  で表す
  - ▶  $X$  の下付き文字は客の順序を表す
- ▶ 客 1  $\rightarrow$  客 2 の順で配達を行う場合
  - ▶ 客 2 への配達時間が  $2t_1 + t_2$  であることに注意
  - ▶ 客 2 より前に客 1 への配達が完了している (その分のストレスも加算)

$$X_{1 \rightarrow 2} = a_1 t_1 + a_2 (2t_1 + t_2 + 1) \quad (1)$$

- ▶ 客 2  $\rightarrow$  客 1 の順で配達を行う場合
  - ▶ 客 1 への配達時間が  $2t_2 + t_1$  であることに注意
  - ▶ 客 1 より前に客 2 への配達が完了している (その分のストレスも加算)

$$X_{2 \rightarrow 1} = a_2 t_2 + a_1 (2t_2 + t_1 + 1) \quad (2)$$

## $N = 2$ の場合

- ▶ どちらの方がストレス量が少ないか知りたいので、大小関係を見よう
- ▶ 引き算を試みる
  - ▶ 次の式の結果が負なら  $X_{1 \rightarrow 2}$  のほうが、正なら  $X_{2 \rightarrow 1}$  のほうがストレス量が少ない
  - ▶ 結果が 0 ならストレス量は等しい (以降、“0 以上” と “負” に分類することにする)

$$\begin{aligned} X_{1 \rightarrow 2} - X_{2 \rightarrow 1} &= \{a_1 t_1 + a_2 (2t_1 + t_2 + 1)\} - \{a_2 t_2 + a_1 (2t_2 + t_1 + 1)\} \\ &= 2a_2 t_1 + a_2 - 2a_1 t_2 - a_1 \\ &= a_2 (2t_1 + 1) - a_1 (2t_2 + 1) \end{aligned}$$

- ▶ ストレス量の大小を知りたいので、上式の符号に興味がある
  - ▶  $a_2 (2t_1 + 1) - a_1 (2t_2 + 1)$  が 0 以上になるかならないか
- ▶ 符号に応じて、適切な順序を選び取ればよい



# $N = 2$ の場合

- ▶ 前ページより、2 人の客の順序は以下の比較関数によって決定できる

## 比較関数

- ▶ 2 人の客  $u, v$  に対する比較関数  $C(u, v) := a_v (2t_u + 1) - a_u (2t_v + 1)$
- ▶  $C(u, v) < 0$  のとき、 $u \rightarrow v$  という順序で配達すると最適
  - ▶ このとき、移項して  $\frac{a_v}{2t_v+1} < \frac{a_u}{2t_u+1}$  が成り立つ  $\dots$  (★)
- ▶  $C(u, v) \geq 0$  のとき、 $v \rightarrow u$  という順序で配達すると最適

## $N \geq 3$ の場合 (一般の場合)

- ▶ 実は 3 人以上いる場合でも同様に順序を決めて良い
  - ▶ 客  $u, v$  について最適な順序が  $u \rightarrow v$  で、客  $v, w$  について最適な順序が  $v \rightarrow w$  であるとする
  - ▶ このとき、客  $u, w$  について  $u \rightarrow w$  が最適な順序になる
    - ▶ 前ページ (★) 式を使って導かれます
  - ▶ これを全ての客に適用すると、 $\frac{a_i}{2t_i+1}$  の降順に並べると最適になる！
- ▶ 例えば C++ の場合、比較関数によるソートは以下のように書けます
  - ▶ 小数演算が入らないよう、掛け算で処理するのをおすすめします

```
// vector<long long int> T(N), A(N) が上で定義されている

vector<int> order(N);
// 0 から N-1 まで連番でいれる
iota(order.begin(), order.end(), 0);
// 比較関数でソート
sort(order.begin(), order.end(), [&](int i, int j) {
    return A[i] * (2 * T[j] + 1) > A[j] * (2 * T[i] + 1);
});
```

## ▶ Writer 解

- ▶ beet (C++・97 行・999 bytes)
- ▶ hos (C++・61 行・1463 bytes)
- ▶ riantkb (Python・25 行・428 bytes)
- ▶ tsutaj (C++・31 行・705 bytes)

## ▶ 統計

- ▶ AC / tried: 29 / 36 (80.6 %)
- ▶ First AC
  - ▶ The University of Tokyo: DELIAIR (17 min 53 sec)



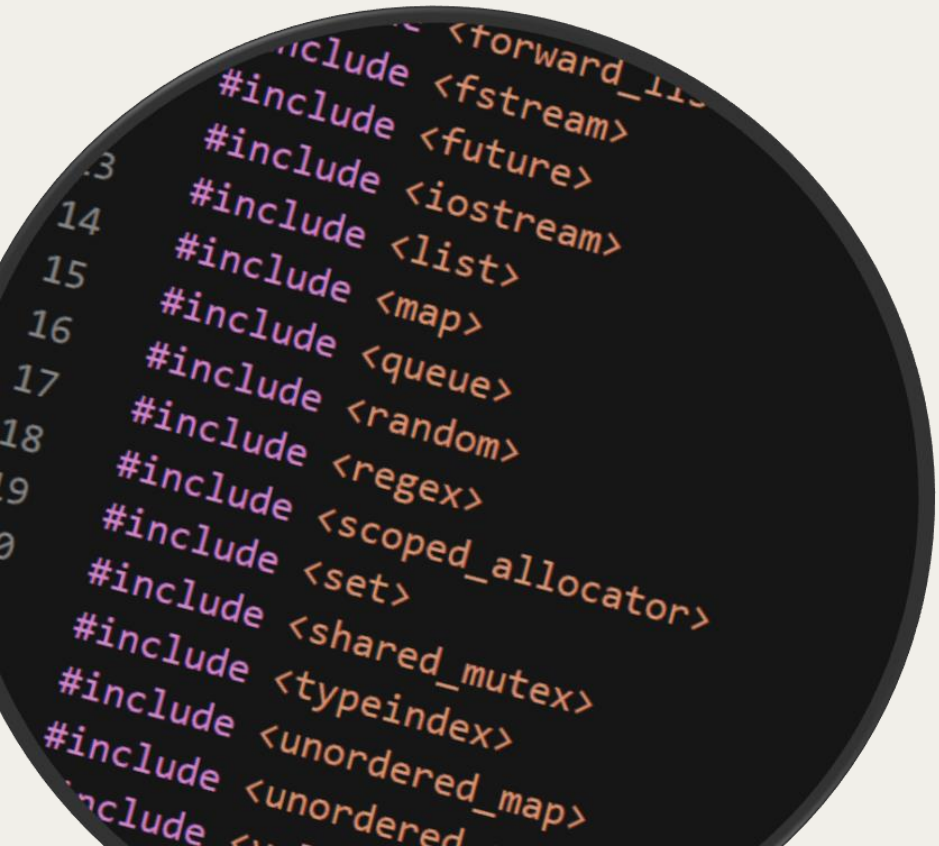
# Include

原案・データセット：TumoiYorozu

問題文：darsein

解答：TumoiYorozu, beet, hos

解説：TumoiYorozu



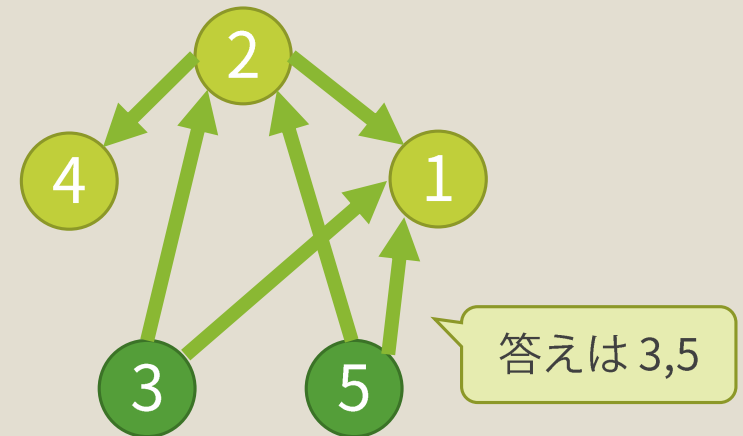
```
#include <forward_list>
#include <fstream>
#include <future>
#include <iostream>
#include <list>
#include <map>
#include <queue>
#include <random>
#include <regex>
#include <scoped_allocator>
#include <set>
#include <shared_mutex>
#include <typeindex>
#include <unordered_map>
#include <unordered_set>
```

# 問題概要

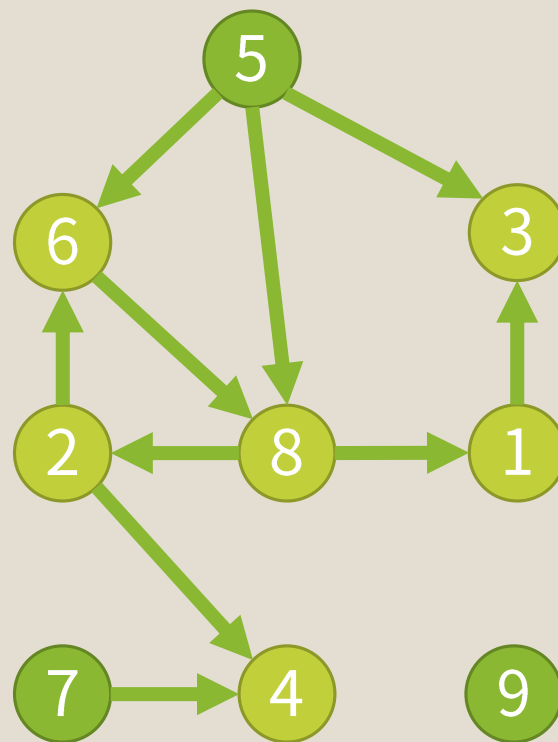
- Include (取り込み) したいソースコードが  $N$  個ある
- ファイル  $a_i$  はファイル  $b_i$  を include している
- 全てのソースコードをincludeするには、最低で何個includeするべきか

## 制約

- ファイル数  $N \leq 30000$
- include 関係数  $M \leq 5 \times 10^5$
- 解が複数ある場合は、ファイル番号の総和が最小になるように答える

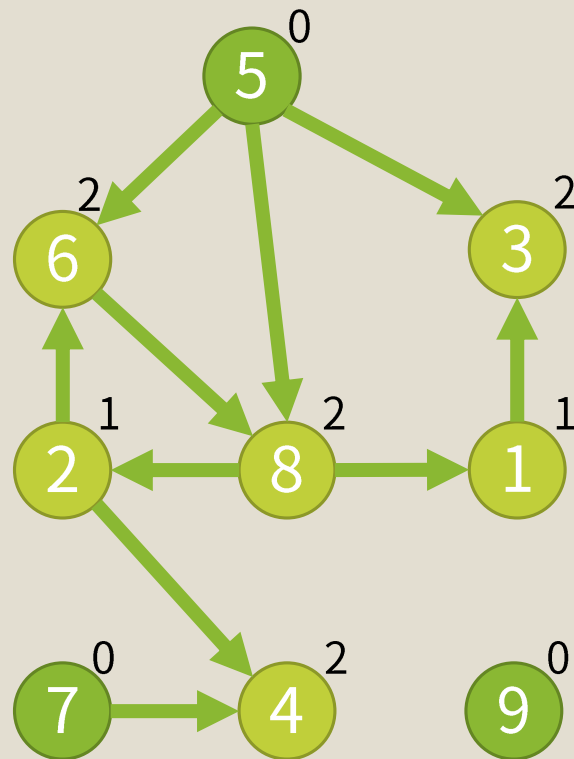


# サンプル #3



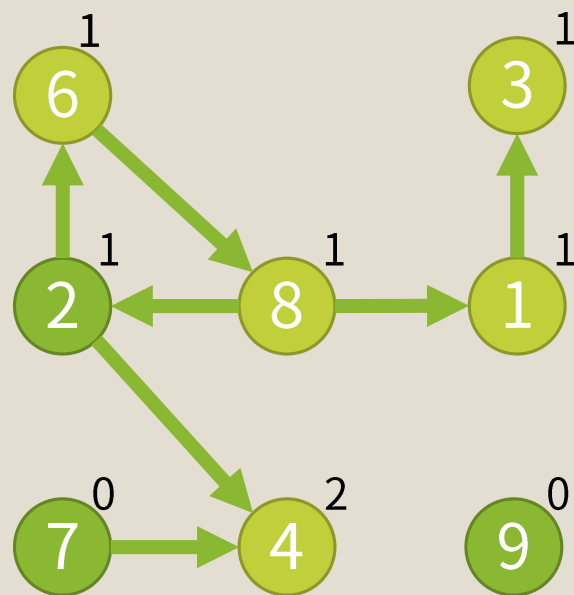
答えは 5,7,9

# 嘘解法



入次数が0の頂点を選択していく！

# 嘘解法

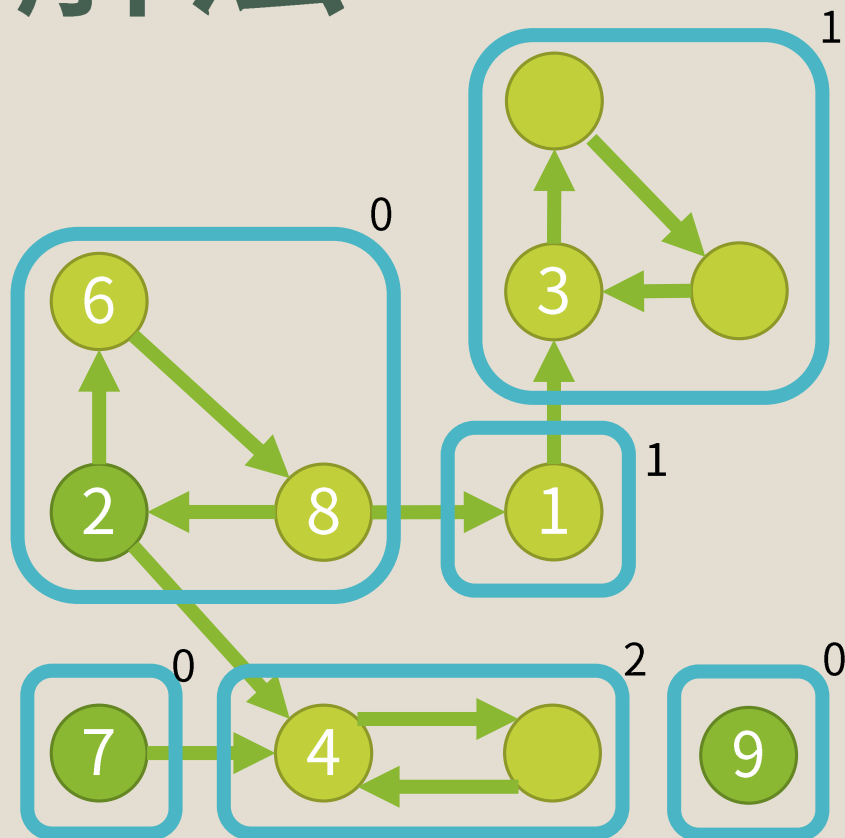


入次数が1の頂点を  
選択しない！

サンプルは弱いので通るが、  
左のようなグラフの場合、  
入次数が1である「2」の頂点を  
選択しないといけない



# 解法



- 強連結成分分解 (SCC) をする
  - SCC とは、有向グラフで互いに行き来できる頂点同士を同じグループにまとめる操作
  - DFS を2回行うとできる。計算量は  $O(N + M)$
  - AtCoder Library (ACL) にも収録されている
- 強連結成分への入次数が0の成分で、その中で一番番号が小さい点を選択
- 同じグループ内へ行く辺を無視するのを忘れないこと (忘れてもサンプルは AC する)  
~~WAした人は反省~~ 1発ACした人はえらい！
- 全体の計算量は、答えをソートするための計算量による。  $O(N \log N + M)$  または  $O(N + M)$

# 余談

GCC9.3 の C++17 で <bits/stdc++.h> を include すると、以下の88ヘッダがincludeされる

cassert	csetjmp	wchar	ctgmth	functional	list	set	valarray	future	thread	charconv
cctype	csignal	cwctype	cuchar	iomanip	locale	sstream	vector	initializer_list	tuple	filesystem
cerrno	cstdarg	ccomplex	algorithm	ios	map	stack	array	mutex	typeindex	optional
cfloat	cstddef	cfenv	bitset	iosfwd	memory	stdexcept	atomic	random	type_traits	memory_resource
ciso646	cstdio	cinttypes	complex	iostream	new	streambuf	chrono	ratio	unordered_map	string_view
climits	cstdlib	cstdalign	deque	istream	numeric	string	codecvt	regex	unordered_set	variant
locale	cstring	cstdbool	exception	iterator	ostream	typeinfo	condition_variable	scoped_allocator	shared_mutex	bit
cmath	ctime	cstdint	fstream	limits	queue	utility	forward_list	system_error	any	version

Clang10.0.0(C++17)では、以下の31ヘッダをincludeすれば上の88ヘッダがincludeされる

fstream	iostream	codecvt	queue	variant	map	scoped_allocator	charconv	cfenv	locale	cstdbool
ctgmth	regex	shared_mutex	unordered_set	valarray	list	any	cinttypes	cfloat	csetjmp	
random	future	condition_variable	unordered_map	set	forward_list	typeindex	cassert	ciso646	csignal	

※左のヘッダほど多くのヘッダをincludeする。斜体の8ヘッダは他のヘッダをincludeしない。

詳しい内容を [Qiita の記事](https://qiita.com/TumoiYorozu/private/e645ccf7c35e56b0eca2) で書いたので、興味がある方はご覧ください。

<https://qiita.com/TumoiYorozu/private/e645ccf7c35e56b0eca2>

# ジャッジ解

- TumoiYorozu(C++) : 105行
- TumoiYorozu(C++) : 55行 + ACL
- beet (C++) : 108行
- Hos (C++) : 128行

# 統計情報

- AC / Trying Teams
  - 34/34
- WA
  - 19
- First Acceptance
  - SPJ (22 minutes)

# ICPC模擬地区予選2021

## I: $(N+1)$ -legged race

原案: climpet

問題文: climpet

データセット: tsutaj

解答: beet, climpet, hos

解説: climpet

# 問題概要

- $S$  人の生徒がいる。各生徒は運動能力  $A_i$  と身長  $H_i$  を持つ。
- この中から走者として  $N$  人を選び、一列に並べる。このチームの強さは (運動能力の総和) - (隣り合う二人の身長差の絶対値の総和) となる。
- チームの強さを最大化せよ。

## 制約

- $2 \leq S \leq 10^5$
- $2 \leq N \leq \min(S, 200)$



# 生徒の並び順

- 走者  $N$  人を決めたとき、この  $N$  人の並べ方については、明らかに身長順が最適である。
- このとき、チームの強さは、  
(運動能力の総和) + (身長の最小値) - (身長の最大値)  
となる。

## 解法1 (by hos)

- 生徒を身長昇順に並べる。
- $dp_j[k] = (j \text{ 番目までの生徒から } k \text{ 人を選ぶときの、(運動能力の総和) - (身長 of 最小値) の最大値) \text{ とする。}$
- $j$  番目の生徒について、次のように値を更新する。
  - $ans = \max(ans, dp_{j-1}[N-1] + A_j - H_j)$
  - $dp_j[1] = \max(dp_{j-1}[1], A_j + H_j)$
  - $dp_j[k] = \max(dp_{j-1}[k], dp_{j-1}[k-1] + A_j) \text{ for each } 2 \leq k < N$
- 計算量は  $O(S(N + \log S))$  時間。



## 解法2 (by beet)

- 最適解において、運動能力が一番低い走者を  $w$  とする。
- 運動能力が  $w$  より高い生徒を身長順に並べてできる列を  $B$  とする。
  - 運動能力が同じ生徒については、適当に順位をつけておく。
- 最適解は、 $B$  中の**連続する**  $N$  人 ( $w$  を含む) を並べたものとなる。
  - 証明:  $w$  を含む連続しない  $N$  人を選ぶのが最適であると仮定する。このとき、スキップされた生徒の一人を  $x$  とする。 $w$  の代わりに  $x$  を選ぶことにすると、 $A_x > A_w$  という前提から運動能力の総和は大きくなる一方、身長によるペナルティは同じか小さくなる。したがって、チームの強さが増加するが、これは最適解が  $w$  を含むという仮定に反する。

## 解法2 (by beet)

- 列  $B$  を平衡二分探索木を用いて管理する。
- 生徒を身長の高順に処理する。
  - その生徒を  $B$  に追加する。
  - その生徒の前後 (最大)  $N - 1$  人ずつを取り出す。
  - これら (最大)  $2N - 1$  人から、連続する  $N$  人の選び方をすべて試す。
- 連続する  $2N - 1$  人を取り出す部分がボトルネックとなるが、これは一回あたり  $O(N + \log S)$  時間で実現できる。
  - 実装上は、イテレータを単純にインクリメント・デクリメントするだけでよい。ただし定数倍はやや遅いことがある。
- 計算量は解法 1 と同じく  $O(S (N + \log S))$  時間。

# 統計情報

- AC / trying teams
  - 32 / 33
- First acceptance
  - UT a.k.a ls (15 分)

# JAG ICPC模擬地区予選2021

## J: Isomorphic?

---

原案: not

問題文: climpet

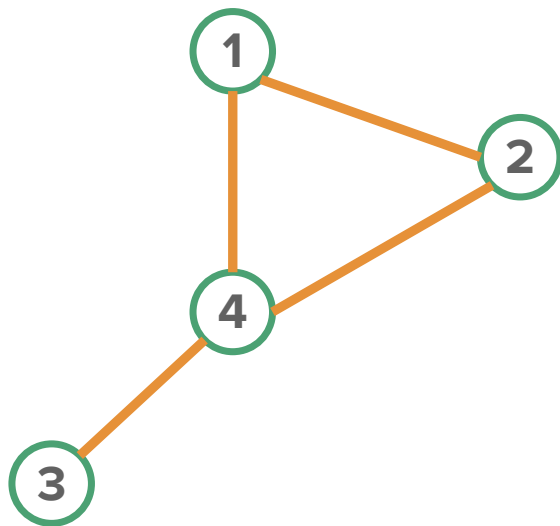
データセット: riantkb

解答: beet, climpet, hos, riantkb

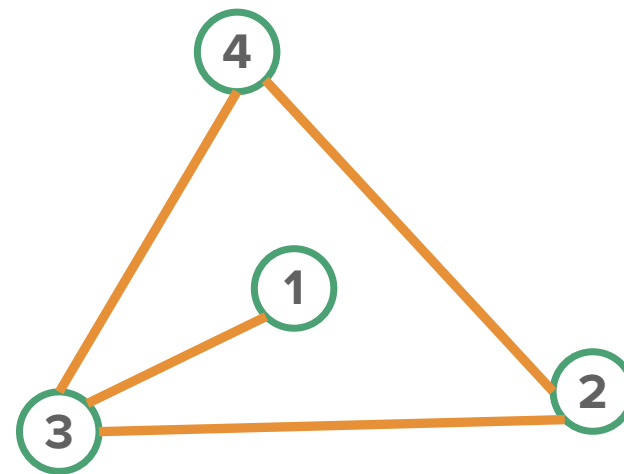
解説: riantkb

## 問題概要

- $N$  頂点  $N$  辺の単純連結無向グラフが 2 つある
- この 2 つのグラフが同型かどうか判定せよ

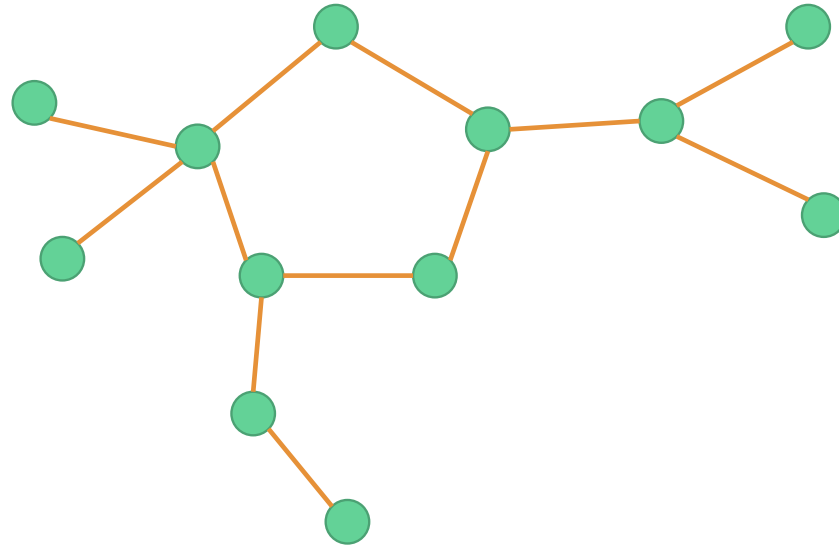


この 2 つのグラフは同型



# はじめに

- $N$  頂点  $N$  辺の連結無向グラフは、ちょうど一つサイクルがありそこから木が複数生えているような形をしている
  - このようなグラフを **Unicyclic graph** と呼んだり、日本語で **なもりグラフ** と呼ぶことがある

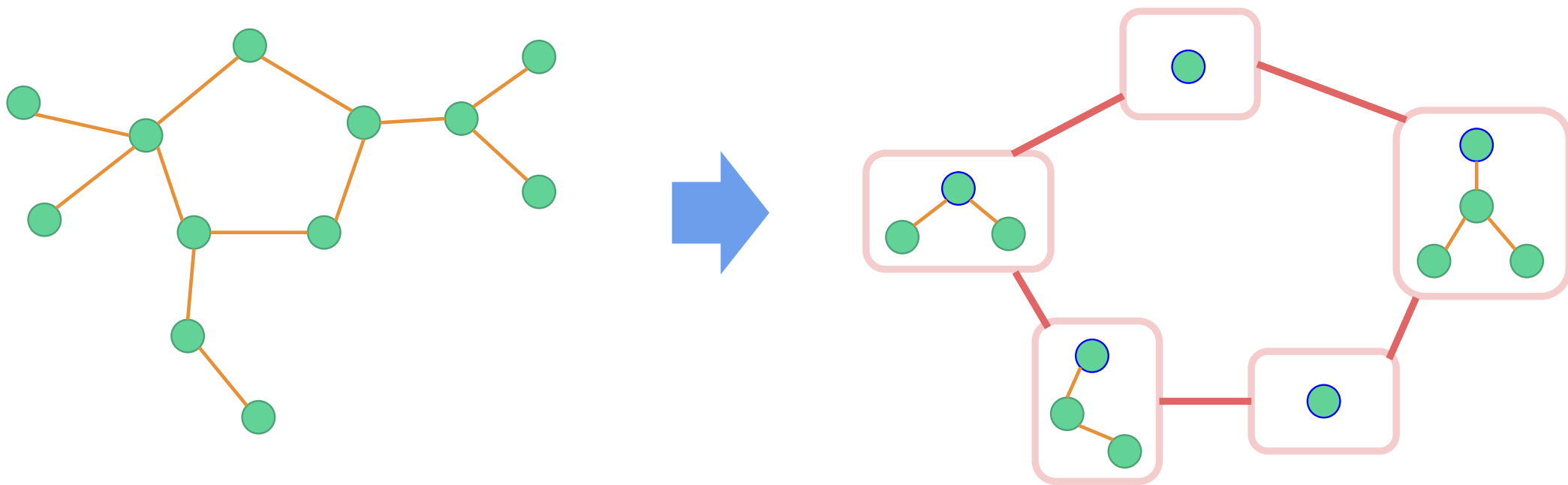


# はじめに

- 2つの根付き木が同型かどうか、という問題にはいくつか解決方法があり、例えば AHU アルゴリズムや根付き木に対するハッシュを設計する方法がある
  - ここでは詳しく解説しないので、詳しく知りたい方は例えば以下の記事をご覧ください
- AHU アルゴリズムについて
  - <https://chocobaby-aporo.hatenablog.com/entry/2017/12/05/233027>
- 根付き木に対するハッシュ設計について
  - <https://rng-58.blogspot.com/2017/02/hashing-and-probability-of-collision.html>

## 考察

- 与えられるグラフはサイクルに木が生えた形をしているため、それぞれの木をハッシュ等に変換しておくことで、それらの列が同じかどうかで元のグラフが同型であるかどうかを判定できると考えられる





## 考察

- サイクルを切り開いて列として考えると、それぞれのグラフから得られたハッシュ列  $A, B$  に対する以下の問題の答えが本問題の答えと同じであると考えられる
  - 要素列  $A, B$  が与えられる。 $A$  に以下の操作を好きな順序で好きなだけ行い、 $A$  を  $B$  に一致させられるか？
    - $A$  全体を rotate する ( $a_1, a_2, \dots, a_N \rightarrow a_2, a_3, \dots, a_N, a_1$ )
    - $A$  全体を reverse する ( $a_1, a_2, \dots, a_N \rightarrow a_N, a_{N-1}, \dots, a_1$ )

## 考察

- 前述した問題は例えば A, B, B という順で連結した列に対し z-algorithm を適用したり、Rolling Hash などを用いることで解くことができる
  - 反転操作を考慮するため、A または B を反転したものについて再度判定する必要があることに注意

# 解法

- 与えられたグラフをサイクルとそれに繋がっている木、という形に整理し、それぞれの木についてハッシュ等を求める
- それらをサイクルの順に見ることでハッシュ列を得て、2つのハッシュ列がサイクルで見ても一致しているかを z-algorithm や Rolling Hash などと判定する
- 計算量は  $O(N)$  または  $O(N \log N)$  などとなる

## ジャッジ解

- beet (C++): 162 lines, 3.2 kB
- climpet (C++): 221 lines, 4.2 kB
- hos (C++): 170 lines, 3.9 kB
- riantkb (C++): 174 lines, 4.7 kB

# 統計情報

- Acceptances / Submissions
  - 25 / 113 (22.12 %)
- AC teams / Trying teams
  - 25 / 29 (86.21 %)
- First Acceptance
  - SPJ (30 min)

# ICPC模擬地区予選2021

## K: Zombie Land 2

原案: sumiken

問題文: riantkb

データセット: darsein

解答: darsein, hos, riantkb

解説: darsein

# 問題概要

- \ 2次元平面上に1人のゾンビとN人の人がいる
- \ ゾンビ、および人はそれぞれ初期位置  $(x_i, y_i)$  から速度  $(v_{xi}, v_{yi})$  で動き続ける
- \ 人はゾンビの半径  $D$  以内に近づいてしまうと、ゾンビになってしまう
  - \ 以降このゾンビの半径 $D$ 以内に近づいても人はゾンビになる
- \ 各人について、ゾンビになるタイミングを求めよ。
  - \ ゾンビにならない場合  $-1$  を出力
- \ 制約
  - \  $1 \leq N \leq 10^3, 0 \leq D \leq 10^4, -10^4 \leq x_i, y_i, v_{xi}, v_{yi} \leq 10^4$

# 解法

\ 2つのパートに分けて考える

1. (ゾンビと人を区別せず) 全ペアについて距離が $D$ 以内になる時間の区間を求める
2. 各人がゾンビになる最速時間を上記区間に基づいて求める



# 解法: 全ペア区間列挙

- 人  $(x_a, y_a, v_{xa}, v_{ya})$  と人  $(x_b, y_b, v_{xb}, v_{yb})$  の時間  $t$  における距離  $D_t$  について、

$$D_t^2 = ((x_a + v_{xat}) - (x_b + v_{xbt}))^2 + ((y_a + v_{yat}) - (y_b + v_{ybt}))^2 \text{ が成り立つ}$$

- 式を変形すると、

$$D_t^2 = (V_x^2 + V_y^2)t^2 - 2(XV_x + YV_y)t + X^2 + Y^2$$

ここで、 $X = x_a - x_b$ ,  $Y = y_a - y_b$ ,  $V_x = v_{xa} - v_{xb}$ ,  $V_y = v_{ya} - v_{yb}$

- よって、距離  $D$  以内になる時間  $t$  は

$$(V_x^2 + V_y^2)t^2 - 2(XV_x + YV_y)t + X^2 + Y^2 \leq D^2 \text{ を満たす}$$

# 解法: 全ペア区間列挙

\  $a = V_x^2 + V_y^2$ ,  $b = -2(XV_x + YV_y)$ ,  $c = X^2 + Y^2 - D^2$  とおくと、 $at^2 + bt + c \leq 0$  の形でかけるため、二次方程式を解けばこれを満たす  $t$  の区間がわかる

\  $a = 0$  のときは  $bt + c \leq 0$

\  $b = 0$  のとき  $c \leq 0$  で  $[-\infty, \infty]$ 、 $c > 0$  で距離  $D$  以内になる区間なし

\  $b < 0$  なら  $[-c/b, \infty]$ 、 $b > 0$  なら  $[-\infty, -c/b]$

\  $a \neq 0$  のときは  $[-b - \sqrt{b^2 - 4ac} / 2a, -b + \sqrt{b^2 - 4ac} / 2a]$

\ ただし、 $b^2 - 4ac < 0$  のときは交点がない

→ ( $a > 0$  より下に凸なので) 距離  $D$  以内になる区間なし

\ 上記の場合分け・計算は  $O(1)$  で可能 → 全ペアについて計算して全体で  $O(N^2)$

# 解法: 最速ゾンビ化時間

＼ ゾンビ集合を $Z$ 、人集合を $P$ 、時間  $t = 0$  とおく

＼  $t$  までにまだゾンビではない人  $p \in P$  について、すでにゾンビになっている人たち  $Z$  と距離 $D$ 以内になる時間のうち、 $t$  以降で最速の時間  $t_p$  をそれぞれ求める

＼ 人  $p$  とゾンビ  $z$  が距離 $D$ になる時間を  $[a, b]$  とすると、 $t$  以降で最速の時間  $t_{pz}$  は  $t \leq b$  のときのみ定義され、 $t_{pz} = \max(a, t)$

＼  $t_p = \min_{z \in Z} \{t_{pz}\}$

＼  $P$  のうち、最速でゾンビになる人をゾンビにして、時間を進める

＼  $t = \min_{p \in P} \{t_p\}$  とおき、 $P = P \setminus \{p\}$ ,  $Z = Z \cup \{p\}$  と更新する

# 解法: 最速ゾンビ化時間

- ＼ このアルゴリズムを愚直に実装すると、集合Pのサイズが1つ増えるたび、 $P \times Z$ について  $t_{pz}$  を求めるのに  $O(N^2)$ 、サイズが増える回数が高々N回なので全体で  $O(N^3)$
- ＼ 実際には集合Pのサイズが1つ増えるときに更新される  $t_{pz}$  は、増えた人  $p$  に関するもののみなので、高々N個の更新のみでよい。この更新の工夫により、全体で  $O(N^2)$
- ＼  $O(N^2)$  ダイクストラの更新と同様

# 解法: まとめ

- ＼ 二次方程式を解いて全ペア区間列挙する:  $O(N^2)$
- ＼ ダイクストラ法の要領でその時点以降最速でゾンビ化する人を順次求めていく:  $O(N^2)$
- ＼ 全体でも  $O(N^2)$
- ＼ 区間計算は実際には前処理にする必要はなく、 $t_{pz}$ の更新時に求めても全体の計算量は変わらない

# 統計情報

\ AC / trying teams

\ 20 / 24 (83.33%)

\ First Acceptance

\ The atama (00:38)