AMPPZ 2015

Uniwersytet Wrocławski

25 października 2015

D: Studniówka

Zadanie (Autor: Karol Pokorski)

Dane są dwie tablice $A = (a_1, a_2, \ldots, a_n)$ i $B = (b_1, b_2, \ldots, b_m)$. Znaleźć liczbę par elementów (a_i, b_j), że $|a_i - b_j| \leqslant k$.

Statystyki

AC	47
WA	75
TLE	63
RE	14
$\sum$	199

Najszybsze zaakceptowane zgłoszenie: Uniwersytet Warszawski 8 (*Prochowska, Łuszczyk, Gutowski*) [3'36"]

D: Studniówka

- Sortujemy obie tablice.
- Dla kolejnych elementów a_i z posortowanej tablicy A utrzymujemy pozycje:
  - najmniejszego elementu tablicy B większego lub równego $a_i - k$,
  - największego elementu tablicy B mniejszego lub równego $a_i + k$.
- Wszystkie elementy wewnątrz tego przedziału tworzą szukaną parę z elementem a_i.
- Po przesunięciu się do kolejnego elementu a_i przesuwamy wskaźniki w tablicy B (tylko w prawo) – sumarycznie sprawdzeń i przesunięć będzie liniowo wiele.

Można też użyć dwóch wyszukiwań binarnych w tablicy B.

F: Flamaster

Zadanie (Autor: Karol Pokorski)

Dane jest nawiasowanie złożone z trzech typów nawiasów. Dostawić jak najmniej nawiasów na początku lub na końcu, aby stało się poprawne.

Statystyki

AC	50
WA	9
TLE	22
RE	28
$\sum$	109

Najszybsze zaakceptowane zgłoszenie: Uniwersytet Warszawski 6 (*Pszeniczny, Farbiś, Czajka*) [9'30"]

F: Flamaster

- Przeglądamy nawiasy z wejścia od lewej do prawej.
- Jeśli nawias otwierający – wrzucamy na stos.
- Jeśli nawias zamykający:
  - nawias na szczycie stosu mu odpowiada – zdejmujemy ze stosu.
  - nawias na szczycie stosu mu nie odpowiada – odpowiedź `NIE`.
  - stos jest pusty – trzeba dopisać odpowiadający nawias otwierający na początku nawiasowania.
- Jeśli po przejrzeniu wejścia pozostały na stosie jakieś nawiasy – trzeba dopisać ich odpowiedniki na końcu nawiasowania.

B: Bogacze

Zadanie (Autor: Jakub Tarnawski)

Dany jest układ m nierówności postaci $\pm x_i \pm x_j > 0$. Czy istnieje rzeczywiste rozwiązanie $(x_1, x_2, \ldots, x_n)$ całego układu?

Statystyki

AC	41
WA	2
TLE	6
RE	44
$\sum$	93

Najszybsze zaakceptowane zgłoszenie: Uniwersytet Warszawski 2 (*Zawalski, Paluszek, Hołubowicz*) [7'40"]

B: Bogacze

- Tworzymy graf, w którym wierzchołkami są zmienne (x_i) oraz liczby do nich przeciwne ($-x_i$).
- Nierówność $\pm x_i \pm x_j > 0$ przekształcamy na dwa sposoby: $\pm x_i > \mp x_j$ oraz $\pm x_j > \mp x_i$ i tworzymy krawędzie od wartości mniejszych do większych.

Twierdzenie

Układ jest sprzeczny wtedy i tylko wtedy, gdy utworzony graf zawiera cykl.

Dowód $\Rightarrow$

Jeśli graf ma cykl to otrzymujemy dla dowolnej zmiennej x_i na cyklu: $x_i > x_i$.

Dowód $\Leftarrow$ (indukcja po liczbie wierzchołków grafu)

- Graf, który nie zawiera cyklu, ma wierzchołek x, którego stopień wejściowy jest równy 0.
- Struktura grafu zapewnia, że w takim razie wierzchołek $-x$ ma stopień wyjściowy równy 0.
- Po usunięciu wierzchołków x oraz $-x$ z grafu, nadal nie ma w nim cyklu, więc ma on pewne rozwiązanie (założenie indukcyjne).
- Pozostaje przypisać wartość zmiennej x na podstawie przypisanych wartości dla następników x i poprzedników $-x$. Można to zrobić trywialnie, bez stworzenia sprzeczności.

A: Album liczb

Zadanie (Autor: Karol Pokorski)

Zaprojektuj strukturę reprezentującą multizbiór liczb naturalnych (z operacjami dodawania i usuwania) umożliwiającą obliczenie średniej minimów wszystkich różnych podzbiorów multizbioru.

Dwa podzbiory uważamy za równe, jeśli każda liczba występuje w obu w takiej samej liczbie egzemplarzy.

Statystyki

AC	18
WA	1
TLE	19
RE	54
$\sum$	92

Najszybsze zaakceptowane zgłoszenie: Uniwersytet Warszawski 1 (*Sokołowski, Smulewicz, Nadara*) [14'30"]

A: Album liczb (rozwiązanie 1)

- Multizbiór reprezentujemy jako słownik (`map`), w którym kluczami są elementy zbioru, zaś wartościami – ich krotności w strukturze.
- Jeśli najmniejszy element x występuje k razy, to $\frac{k}{k+1}$ wszystkich podzbiorów ma minimum równe x.
- Pozostałe podzbiory rozważamy, jakby x już nie istniało (tj. rozpatrujemy najmniejszy element z pozostałych), pamiętając o domnożeniu czynnika $\frac{1}{k+1}$.

Sprytne „oszustwo”

Aby zachować dokładność bezwzględną wyniku 10^{-6} wystarczy zawsze przeanalizować $\log_2 \frac{10^9}{10^{-6}} \approx 50$ najmniejszych elementów.

Uwzględniliśmy niepotrzebnie pusty zbiór – wystarczy domnożyć wynik przez $\frac{\Pi+1}{\Pi}$ (gdzie Π było iloczynem krotności wszystkich liczb).

A: Album liczb (rozwiązanie 2)

- Skanujemy wejście i zapamiętujemy wszystkie różne liczby, które wystąpią na wejściu.
- Multizbiór reprezentujemy jako drzewo przedziałowe. W liściach pamiętamy krotność każdej liczby w strukturze. Liście drzewa pamiętamy tylko dla liczb, które wystąpiły na wejściu.
- Pamiętamy też mnożnik w liściach równy $\frac{k}{k+1}$ (podobnie jak w pierwszym rozwiązaniu).
- Wynik dla rodzica to wynik dla lewego syna plus wartość prawego syna przemnożona przez pozostały mnożnik (1 minus mnożnik lewego syna). Bardzo podobnie aktualizuje się mnożnik dla węzła.
- Analogicznie jak w rozwiązaniu pierwszym należy „zapomnieć" o zbiorze pustym (przemnożyć wynik korzenia przez odwrotność mnożnika korzenia).

L: Listonosz

Zadanie (Autor: Karol Pokorski)

Dane jest drzewo. Chcemy odwiedzić wszystkie wierzchołki drzewa, aby suma czasów odwiedzenia wierzchołków była minimalna.

Przebycie każdej krawędzi drzewa zwiększa aktualny czas o 1. Czas odwiedzenia wierzchołka to pierwszy moment, w którym wierzchołek jest odwiedzony.

Statystyki

AC	15
WA	6
TLE	19
RE	41
$\sum$	81

Najszybsze zaakceptowane zgłoszenie: Google (*Mitrichev, Kornakov, Modelski*) [88'49"]

L: Listonosz

Kluczowa obserwacja

Dla ustalonego wierzchołka startowego (korzenia drzewa) dowolne obejście algorytmem DFS daje taki sam (optymalny dla tego wyboru) sumaryczny czas odwiedzenia drzewa. Innymi słowy: kolejność odwiedzania synów nie ma znaczenia.

Dowód

- Ustalmy korzeń drzewa i oznaczmy D_v – głębokość wierzchołka v.
- Porządkujemy wierzchołki w kolejności rosnącego czasu pierwszego odwiedzenia.
- Czas odwiedzenia i-tego z kolei wierzchołka używając algorytmu DFS to: $2(i-1) - D_v$ (byłoby dokładnie $2(i-1)$ gdyby chcieć wrócić do korzenia).
- Dowolna strategia odwiedzania wierzchołków (dla być może innego uporządkowania wierzchołków) daje co najmniej taki sam czas odwiedzenia i-tego z kolei wierzchołka (dla każdego i).

L: Listonosz

Przesuwając korzeń do jednego z synów wynik zmienia się o n minus dwukrotność rozmiaru poddrzewa syna.

Ostatecznie algorytm oblicza najpierw liniowo (używając algorytmu DFS) wynik dla dowolnego wierzchołka startowego, a następnie (ponownie używając DFSa) potrafi obliczyć wynik, gdyby przesuwać wierzchołek startowy.

Zadanie (Autor: Karol Pokorski)

Każdy środek jest opisany krotką $(a, k, t) \in \{0, 1, 2, \ldots, 255\}^3$. Za wyeliminowanie każdego z n podanych insektów wygrywamy p, jeśli produkujemy odpowiadający mu środek. Za możliwość użycia użycia danego atraktora, karmy lub trucizny płacimy odpowiednio c_r, c_g i c_b. Ile najwięcej można zyskać?

Statystyki

AC	10
WA	4
TLE	1
RE	3
$\sum$	18

Najszybsze zaakceptowane zgłoszenie: Uniwersytet Wrocławski 1 (*Gołębiewski, Dulęba, Dudek*) [101'30"]

I: Insekty

- Tworzymy sieć przepływową:
  1. Wszystkie możliwe atraktory, karmy i trucizny ($3 \cdot 256$ wierzchołków) łączymy ze źródłem krawędziami o przepustowościach odpowiednio c_r, c_g i c_b.
  2. Insekty łączymy z ujściem krawędziami o przepustowościach p.
  3. Każdego insekta łączymy z odpowiadającym mu atraktorem, karmą i trucizną krawędziami o przepustowościach ∞.

Zadanie sprowadza się do przecięcia niektórych krawędzi sieci, w taki sposób, aby nie istniała ścieżka ze źródła do ujścia.

- Problem minimalnego przekroju rozwiązujemy dowolnym (sensownym) algorytmem przepływowym.

Ostateczny wynik to $p \cdot n$ pomniejszone o wartość przepływu.

K: Klocki

Zadanie (Autor: Karol Pokorski)

Ile jest permutacji n budynków wysokościach $\{1, 2, \ldots, n\}$, w których z lewej strony widać a, zaś z prawej b budynków?

Statystyki

AC	4
WA	1
TLE	18
RE	19
$\sum$	42

Najszybsze zaakceptowane zgłoszenie: Uniwersytet Warszawski 6 (*Pszeniczny, Farbiś, Czajka*) [149'09"]

K: Klocki

Programowanie dynamiczne

$DP[i][j]$ oznacza liczbę sposobów uzyskania widoku j budynków z lewej strony przy użyciu i najwyższych budynków (o wysokościach $\{n-i+1, n-i+2, \ldots, n\}$).

Albo najniższy budynek (o wysokości $n-i$) ustawiamy na początku (zwiększając liczbę widocznych z lewej budynków) albo nie:

$$DP[i][j] = DP[i-1][j-1] + (i-1) \cdot DP[i-1][j]$$

Wynik dla zapytania (n, a, b) to:

$$\sum_{p=0}^{n} \binom{p}{a-1} DP[p][a-1] \cdot DP[n-p-1][b-1]$$

Ale trzeba tę sumę liczyć szybciej.

K: Klocki

Aby uzyskać wynik dla a budynków z lewej i b budynków z prawej wystarczy obliczyć

$$\binom{a+b-2}{a-1} \cdot DP[n-1][a+b-2]$$

Permutację z wyrzuconym najwyższym budynkiem i widocznymi $a+b-2$ budynkami z lewej, można przekształcić na żądaną sytuację na $\binom{a+b-2}{a-1}$ sposobów.

Symbole Newtona oraz wartości tablicy DP można obliczyć tylko raz, zanim zabierzemy się do odpowiadania na zapytania, dzięki czemu potrafimy (po kosztownym preprocessingu) odpowiadać na zapytania w czasie stałym.

Ciekawostka

Wartości $DP[i][j]$ są liczbami Stirlinga pierwszego rodzaju.

C: Celne strzały

Zadanie (Autor: Karol Pokorski)

Ile minimalnie bitów należy zamienić w liczbie n, aby stała się podzielna przez m? Na ile sposobów można to zrobić? Jakie jest najmniejsze rozwiązanie?

Statystyki

AC	1
WA	23
TLE	6
RE	0
$\sum$	30

Najszybsze zaakceptowane zgłoszenie: Uniwersyt Wrocławski 4 (*Siejba, Głuch, Kostka*) [239'49"]

C: Celne strzały

- Jeśli m jest duże – kandydatów do sprawdzenia jest mało, można sprawdzić wszystkich. Dalej zakładamy, że m jest małe.
- Dzielimy zbiór bitów liczby na dwie (mniej więcej równe) części po $\leqslant 25$ bitów.
- Dla każdej połówki obliczamy odpowiedzi na pytania z zadania przy założeniu, że chodzi o uzyskanie ustalonej reszty z dzielenia p w lewej połówce, zaś $(m - p) \bmod m$ w prawej.
- Dla ustalonej reszty z dzielenia i ustalonej połowy bitów liczby możemy przejrzeć wszystkich kandydatów na tę połowę bitów (dających resztę z dzielenia p lub $(m - p) \bmod m$ odpowiednio).

Nie jest to wcale za wolne: jeśli analizowanych reszt z dzielenia jest m, to kandydatów wewnątrz reszty z dzielenia jest $\leqslant \frac{2^{25}}{m} + 1$.

C: Celne strzały

Limit pamięci

Jak poradzić sobie z niskim limitem pamięci?

- Wystarczy następująca struktura pętli:
  - Zewnętrzna pętla ustalająca wymaganą resztę w lewej połowie liczby,
  - Wewnątrz pętla testująca liczby w lewej połowie bitów, do której dążymy – celem jest obliczyć odpowiedzi na trzy pytania postawione w zadaniu dla lewej części liczby (przy założeniu tam reszty z dzielenia p).
  - Po wykonaniu wewnętrznej pętli, kolejna wewnętrzna, która dla prawej części liczby robi podobną rzecz jak poprzednia (tylko, że dla reszty $(m - p) \bmod m$).
- Do skombinowania wyników lewej i prawej części wystarczą już tylko operacje arytmetyczne beż żadnych iteracji.
- Ostatecznie rozwiązanie może działać w pamięci stałej.

C: Celne strzały

Trzeba umieć jeszcze wyznaczyć ile bitów należy zmienić aby przekształcić liczbę n na wybranego kandydata x.

Wystarczy policzyć liczbę zapalonych bitów (wagę Hamminga) liczby $n \oplus x$.

- Nie można robić tego zbyt wolno – w szczególności użycie $\log n$ operacji spowoduje przekroczenie limitu czasu.
- Rozwiązanie do tego problemu (jedno z wielu): stablicować wyniki dla liczb 16-bitowych – wtedy ustalenie liczby zapalonych bitów liczby 64-bitowej wymaga kilku operacji bitowych i czterech odczytów tablicy.

J: Jaskinie

Zadanie (Autor: Damian Straszak)

Dana sieć komnat. W dokładnie jednej z komnat znajduje się skarb. Możemy poruszać się korytarzami lub wykonywać teleport do losowej komnaty. Znaleźć optymalną strategię eksploracji komnat - tak aby zminimalizować oczekiwany czas na odnalezienie skarbu.

Statystyki

AC	1
WA	9
TLE	1
RE	1
$\sum$	12

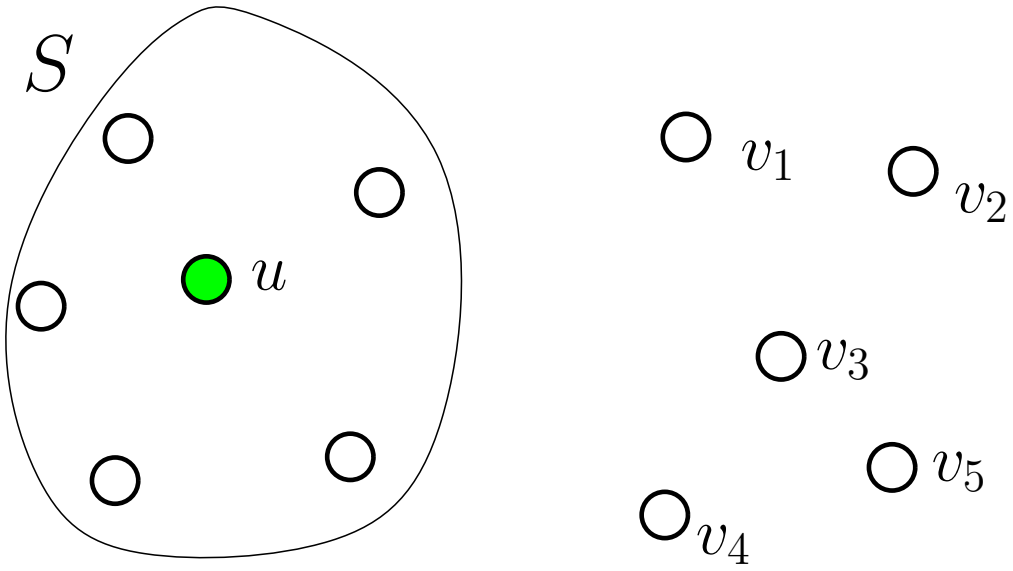
Najszybsze zaakceptowane zgłoszenie: Google (*Mitrichev, Kornakov, Modelski*) [174'01"]

J: Jaskinie

Idea: programowanie dynamiczne

$E[S][u]$ = optymalny średni czas odnalezienia skarbu startując z u przy założeniu, że skarbu nie ma w zbiorze wierzchołków S.

- S - dotychczas odwiedzone wierzchołki,
- $u \in S$ aktualne położenie,
- $v_1, v_2, ..., v_k$ - nieodwiedzone wierzchołki.



J: Jaskinie

Obliczamy $E[S][u]$ w kolejności $|S| = n, n-1, ..., 1$

- S - dotychczas odwiedzone, $u \in S$ aktualne położenie,
- $v_1, v_2, ..., v_k$ - nieodwiedzone wierzchołki.

Idea

Istnieje dokładnie $k+1$ różnych strategii! Albo idziemy zwyczajnie do jednego z wierzchołków v_i albo wykonujemy teleport.

Rekurencja

$$E[S][u] = \min \begin{cases} \text{dist}(u, v_1) + (1-p_1)E[S \cup \{v_1\}][v_1] \\ \text{dist}(u, v_2) + (1-p_2)E[S \cup \{v_2\}][v_2] \\ \vdots \\ \text{dist}(u, v_k) + (1-p_k)E[S \cup \{v_k\}][v_k] \\ t + \frac{1}{n}\sum_{w \in S} E[S][w] + \frac{1}{n}\sum_{i=1}^{k}(1-p_i)E[S \cup \{v_1\}][v_1] \end{cases}$$

Rekurencja

Dla wszystkich wierzchołków $u \in S$ mamy:

$$E[S][u] = \min(\alpha_u, T)$$

$$T = \frac{1}{n} \sum_{w \in S} E[S][u] + \beta$$

Gdzie α_u, β to pewne wartości znane na tym etapie obliczeń.

- Wystarczy obliczyć T!
- Sposób 1: Wyszukiwanie binarne.
- Sposób 2: Gdzie leży T w ciągu $\alpha_{u_1} \leqslant \alpha_{u_2} \leqslant ... \leqslant \alpha_{u_{|S|}}$?
- Złożoność: $O(2^n \cdot n^2)$.

G: Gra planszowa

Zadanie (Autor: Paweł Gawrychowski)

Danych jest n kolorowych punktów na płaszczyźnie. Szukamy pary najbardziej odległych różnokolorowych punktów.

Statystyki

AC	3
WA	26
TLE	2
RE	1
$\sum$	32

Najszybsze zaakceptowane zgłoszenie: Google (*Mitrichev, Kornakov, Modelski*) [131'52"]

G: Gra planszowa

Przypomnienie: para najbardziej odległych punktów

- Wystarczy rozważać punkty, które są wierzchołkami otoczki wypukłej.
- Parą punktów antypodalnych dla nachylenia α nazywamy te punkty otoczki, które leżą najniżej/najwyżej po obróceniu całej otoczki o kąt α.
- Para najbardziej odległych punktów jest parą punktów antypodalnych dla pewnego nachylenia.
- Wszystkie pary punktów antypodalnych można wygenerować przechodząc dwoma palcami po otoczce.

G: Gra planszowa

Powyższe rozumowanie można dość łatwo uogólnić do przypadku, w którym punkty mają dwa różne kolory (i szukamy pary najbardziej odległych różnokolorowych punktów).

- Wystarczy rozważać tylko te czerwone punkty, które są wierzchołkami otoczki wypukłej wszystkich czerwonych punktów. Podobnie z niebieskimi punktami.
- Parą punktów antypodalnych dla nachylenia α nazywamy punkt na czerwonej otoczce, który leży najniżej, i punkt na niebieskiej otoczce, który leży najwyżej po obróceniu obydwu otoczek o kąt α.
- Para najbardziej odległych różnokolorowych punktów jest parą punktów antypodalnych dla pewnego nachylenia.
- Wszytkie pary punktów antypodalnych można wygenerować przechodząc po obu otoczkach.

G: Gra planszowa

Dla większej liczby kolorów stosujemy dziel-i-zwyciężaj.

1. Podziel zbiór kolorów na dwie grupy o podobnych licznościach.
2. Rozwiąż rekurencyjnie problem dla wszystkich punktów o kolorach z pierwszej grupy.
3. Rozwiąż rekurencyjnie problem dla wszystkich punktów o kolorach z drugiej grupy.
4. Potraktuj wszystkie kolory z pierwszej grupy jako czerwony, a wszystkie kolory z drugiej grupy jako niebieski. Zastosuj metodę z poprzedniego slajdu.

Daje to rozwiązanie działające w czasie $\mathcal{O}(n \log^2 n)$. Jeśli w wywołaniu rekurencyjnym będziemy zwracać otoczkę wypukłą wszystkich punktów, czas zmniejszy się do $\mathcal{O}(n \log n)$.

H: Harcerze

Zadanie (Autor: Paweł Gawrychowski)

Umieść podane liczby w wierzchołkach drzewa binarnego tak, że podana liczba a_i jest w i-tym wierzchołku w kolejności inorder, a maksymalna suma liczb na ścieżce z korzenia do liścia jest jak najmniejsza.

Statystyki

AC	1
WA	51
TLE	21
RE	1
$\sum$	0

Najszybsze zaakceptowane zgłoszenie: Google (*Mitrichev, Kornakov, Modelski*) [175'15"]

H: Harcerze

Standardowe programowanie dynamiczne

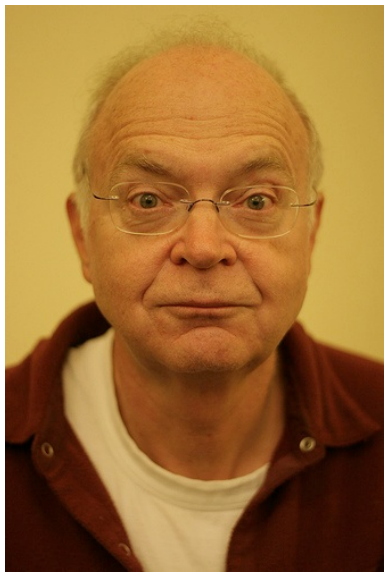
Niech $t[i][j]$ oznacza najmniejszą maksymalną wagę dla drzewa, w którego wierzchołkach umieszczamy $a_i, a_{i+1}, \ldots, a_{j-1}$. Wtedy:

$$t[i][j] = \min_{k=i,i+1,\ldots,j-1} a[k] + \max(t[i][k], t[k][j]).$$

Ale $\mathcal{O}(n^3)$ nie brzmi jak rozwiązanie wzorcowe.

H: Harcerze

Powyższa rekurencja jest bardzo podobna do tej używanej przy wyznaczaniu optymalnego alfabetycznego drzewa binarnych przeszukiwań, w którym można zbić złożoność do $\mathcal{O}(n^2)$ używając tak zwanej optymalizacji Knutha. Może działa i tutaj?



Niestety nie.

H: Harcerze

Trywialna obserwacja

$$t[i][j] \leqslant t[i-1][j] \qquad t[i][j] \leqslant t[i][j+1]$$

Wyobraźmy sobie, że wyznaczamy kolejne $t[i][j]$ w kolejności od najmniejszego do największego. Załóżmy, że poprawnie wyznaczyliśmy już wszystkie pary (i, j), dla których $t[i][j] \leqslant X$.

Kluczowa obserwacja

Dla każdego k wyliczmy:

$$\begin{aligned} \mathrm{left}[k] &= \min\{i : t[i][k] \leqslant X\} \\ \mathrm{right}[k] &= \max\{j : t[k+1][j] \leqslant X\} \end{aligned}$$

wtedy podproblemy (i, j), dla których można umieścić $a[k]$ w korzeniu płacąc $a[k] + X$, to dokładnie $i \geqslant \mathrm{left}[k]$, $j < \mathrm{right}[k]$. Każde takie (i, j) nazywamy kandydatem o koszcie $a[k] + X$.

H: Harcerze

Algorytm:

- Wyciągnij kandydata (i, j) o najmniejszym koszcie.
- Jeśli (i, j) nie zostało jeszcze przetworzone, uznaj koszt związany z kandydatem za $t[i][j]$.
- Uaktualnij wszystkie $\mathrm{left}[k]$ oraz $\mathrm{right}[k]$ tworząc nowych kandydatów.

Jak dokładnie tworzymy nowych kandydatów? Gdy tylko zauważymy, że $\mathrm{left}[k]$ lub $\mathrm{right}[k]$ zmieniło się o jeden, tworzymy jednego nowego kandydata. Dzięki temu każde k spowoduje utworzenie tylko $\mathcal{O}(n)$ kandydatów.

Po ustaleniu $t[i][j]$ należy sprawdzić też $t[i+1][j]$ oraz $t[i][j-1]$!

Po wyliczeniu $t[i][j]$ próbujemy zmieniać $\mathrm{left}[i-1]$ oraz $\mathrm{right}[j]$ o jeden tak długo jak to tylko możliwe. Zamortyzowany czas potrzebny na wyliczenie jednego $t[i][j]$ to $\mathcal{O}(\log n)$.

E: Efektowne wykresy

Zadanie (Autor: Paweł Gawrychowski)

Dana jest permutacja $1, 2, \ldots, n$. Chcemy znaleźć k parami rozłącznych rosnących podciągów tej permutacji, które mają jak największą sumę długości.

Statystyki

AC	0
WA	8
TLE	2
RE	2
$\sum$	12

Najszybsze zaakceptowane zgłoszenie:

—

$k = 1$, czyli najdłuższy podciąg rosnący

Przetwarzamy kolejne elementy permutacji $p(1), p(2), \ldots, p(n)$ utrzymując tablicę $\mathrm{t}[1] < \mathrm{t}[2] < \ldots < \mathrm{t}[\ell]$, gdzie $\mathrm{t}[j]$ to najmniejszy element kończący (dowolny) podciąg rosnący długości dokładnie j już rozważonego prefiksu $p(1), p(2), \ldots, p(i)$.

Okazuje się, że dla większych k wystarczy utrzymywać k tablic:

$$\mathrm{t}_1[1] < \mathrm{t}_1[2] < \mathrm{t}_1[3] < \ldots < \mathrm{t}_1[\ell_1]$$
$$\mathrm{t}_2[1] < \mathrm{t}_2[2] < \mathrm{t}_2[3] < \ldots < \mathrm{t}_2[\ell_2]$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$\mathrm{t}_k[1] < \mathrm{t}_k[2] < \mathrm{t}_k[3] < \ldots < \mathrm{t}_k[\ell_k]$$

E: Efektowne wykresy

Aby przetworzyć kolejny element permutacji $p(i)$ wstawiamy go do t_1. Procedura wstawiania elementu x do t_j jest rekurencyjna:

- jeśli x jest większy niż $\mathrm{t}_j[\ell_j]$, dopisz go na końcu tablicy t_j
- w przeciwnym wypadku znajdź najmniejsze p takie, że $x < \mathrm{t}_j[p]$
- rekurencyjnie wstaw $\mathrm{t}_j[p]$ do t_{j+1} i ustaw $\mathrm{t}_j[p] := x$

Przetworzenie każdego $p(i)$ wymaga k wyszukiwań binarnych, czyli cały algorytm działa w czasie $\mathcal{O}(nk \log n)$.

Wypisujemy $\ell_1 + \ell_2 + \ldots + \ell_k$.

E: Efektowne wykresy

Przydatna obserwacja

Kolumny $\mathrm{t}_1[i], \mathrm{t}_2[i], \ldots$ są rosnące.

Wyobraźmy sobie, że utrzymujemy n tablic, i całkiem przypadkiem wejściowa permutacja to:

$$\mathrm{t}_n[1], \ldots, \mathrm{t}_n[\ell_k], \mathrm{t}_{k-1}[1], \ldots, \mathrm{t}_{n-1}[\ell_{n-1}], \ldots, \mathrm{t}_1[1], \ldots, \mathrm{t}_1[\ell_1].$$

Wtedy oczywiście bez problemu skonstruujemy k rozłącznych podciągów o sumarycznej długości $\ell_1 + \ell_2 + \ldots + \ell_k$. Równie łatwo możemy też pokroić całą permutację na podciągi malejące tak, że długość j-tego malejącego podciągu to wysokości j-tej kolumny. Z takiego podciągu możemy wygrać co najwyżej k elementów, więc wynik nie może być lepszy niż $\ell_1 + \ell_2 + \ldots + \ell_k$.

To tylko intuicja!

E: Efektowne wykresy

Szczegółowe omówienie tego zadania (i innych) już wkrótce na:

`http://fajnezadania.wordpress.com`