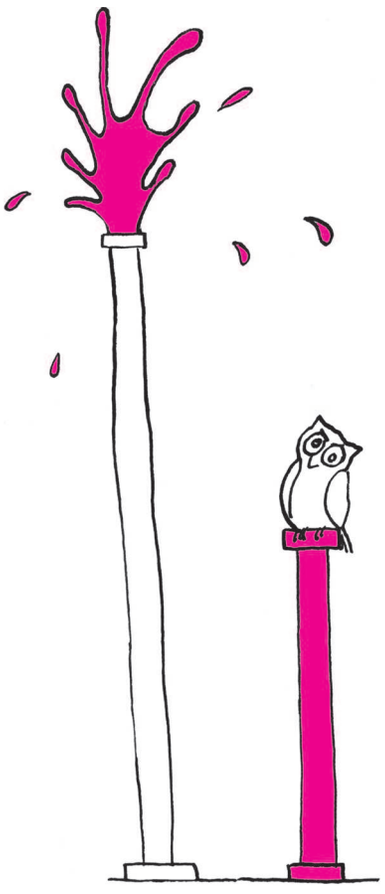


Informatyczny kącik olimpijski (82): Wiercenia

W tym kąciku omówimy zadanie *Wiercenia*, które pojawiło się na Potyczkach Algorytmicznych w roku 2009. Ekipa poszukująca złóż ropy wykonała dwa odwierty: w punkcie $A = 0$ natrafiono na ropę, zaś w punkcie $B = n + 1$ ropy nie znaleziono. Wiedząc, że złoże ropy zajmuje pewien spójny fragment odcinka AB, ekipa chce zlokalizować najdalszy punkt spośród punktów o współrzędnych $1, 2, \ldots, n$, w którym występuje ropa. Wykonanie odwiertu w punkcie i zajmuje czas $t[i]$; ekipa może wykonywać tylko jeden odwiert naraz. Należy wyznaczyć taki plan wierceń, aby czas potrzebny na ustalenie, dokąd sięga złoże ropy, był w pesymistycznym przypadku jak najkrótszy.

Na początek zauważmy, że jeśli wykonanie każdego z odwiertów zajmuje taki sam czas, to wtedy, stosując wyszukiwanie binarne, uzyskamy odpowiedź, wykonując optymalną liczbę $\lceil \log_2 n \rceil + 1$ odwiertów. Zróżnicowanie czasów istotnie komplikuje zadanie; spróbujemy je rozwiązać, korzystając z programowania dynamicznego.

Oznaczmy przez $d[a, b]$ optymalny czas lokalizacji granicy złoża na odcinku od punktu a do punktu b (włącznie), przy założeniu, że wiemy, iż w punkcie $a - 1$ jest ropa, a w punkcie $b + 1$ jej nie ma. Aby wyznaczyć granicę złoża na odcinku $[a, b]$, musimy zacząć od wykonania odwiertu w jednym z punktów i tego odcinka. Jeśli znajduje się tam ropa, to granicy złoża będziemy poszukiwać na odcinku $[i + 1, b]$, w przeciwnym przypadku granica jest na odcinku $[a, i - 1]$. Dostajemy zatem wzór rekurencyjny

$$(*) \qquad d[a, b] = \min_{a \leqslant i \leqslant b} \big(t[i] + \max(d[a, i - 1], d[i + 1, b])\big).$$

Zakładamy, że $d[a, b] = 0$ dla $a > b$. Rozwiązaniem jest wartość $d[1, n]$. Obliczenie jej wprost z powyższej rekurencji da nam algorytm o złożoności czasowej $O(n^3)$ i pamięciowej $O(n^2)$. Algorytm będzie miał n faz, w ℓ-tej fazie będziemy wypełniać komórki tablicy odpowiadające odcinkom o długości ℓ:

```
for ℓ := 0 to n − 1 do
  for a := 1 to n − ℓ do
    b := a + ℓ;  d[a, b] := ∞;
    for i := a to b do
      d[a, b] := min(d[a, b],  t[i] + max(d[a, i − 1], d[i + 1, b]));
```

Aby przyspieszyć powyższy algorytm, będziemy potrzebowali kilku obserwacji. Zauważmy, że jeśli przedłużymy jakiś odcinek, to czas szukania w nim granicy złoża jest zawsze nie mniejszy niż w oryginalnym odcinku. Zatem wraz ze wzrostem i we wzorze $(*)$ wartość $d[a, i - 1]$ nie zmniejsza się, a wartość $d[i + 1, b]$ nie rośnie. Istnieje więc taki indeks $s[a, b] \in [a - 1, b]$, że spełnione są nierówności

$$d[a, i - 1] \leqslant d[i + 1, b] \text{ dla } i \leqslant s[a, b] \quad \text{oraz} \quad d[a, i - 1] > d[i + 1, b] \text{ dla } i > s[a, b].$$

Możemy zatem przepisać wzór $(*)$ w równoważnej formie, pozbywając się liczenia maksimum:

$$(**) \qquad d[a, b] = \min\Big(\min_{a \leqslant i \leqslant s[a,b]} (t[i] + d[i + 1, b]),\ \min_{s[a,b] < i \leqslant b} (t[i] + d[a, i - 1])\Big).$$

Skorzystamy również z tego, że tablica s jest monotoniczna względem każdej współrzędnej, dzięki czemu wartość $s[a, b]$ jest ograniczona przez wartości dla krótszych odcinków:

$$s[a, b - 1] \leqslant s[a, b] \leqslant s[a + 1, b].$$

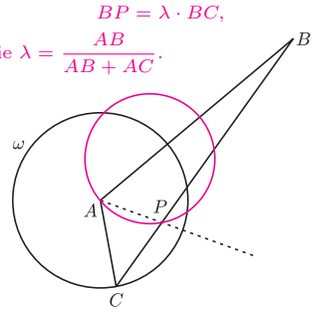
Istotnie, jeśli weźmiemy dowolne $i \leqslant s[a, b - 1]$ i znowu skorzystamy z tego, że przedłużenie odcinka nie zmniejsza czasu poszukiwania granicy złoża, to dostaniemy $d[a, i - 1] \leqslant d[i + 1, b - 1] \leqslant d[i + 1, b]$, zatem $i \leqslant s[a, b]$, co dowodzi lewej nierówności (prawej dowodzimy analogicznie). Ponieważ $s[a, a] = a$, to możemy dla wygody przyjąć $s[a, b] = b$ dla $a > b$. Zatem, aby wyznaczyć $s[a, b]$, wystarczy przejrzeć wartości od $s[a, b - 1]$ do $s[a + 1, b]$. Dzięki temu wyznaczenie ich dla wszystkich odcinków $s[a, a + \ell]$ o ustalonej długości ℓ zajmie sumarycznie czas liniowy:

$$\sum_{a=1}^{n-\ell} (s[a + 1, a + \ell] + 1 - s[a, a + \ell - 1]) = s[n - \ell + 1, n] - s[1, \ell] + n - \ell \leqslant 2n.$$

Rozwiązanie zadania M 1459.
Z twierdzenia o dwusiecznej wiemy, że $\frac{PC}{PB} = \frac{AC}{AB}$, a skąd

$$BP = \lambda \cdot BC,$$

gdzie $\lambda = \frac{AB}{AB + AC}$.



Zatem punkt P to obraz punktu C przy jednokładności o środku B i skali λ. Poszukiwany zbiór punktów P jest więc obrazem okręgu ω (bez dwóch punktów) przy tej jednokładności.

Rozwiązanie zadania F 882.
Maksymalny moment siły, jaki może uzyskać rowerzysta, opierając cały ciężar ciała na korbie ustawionej poziomo, wynosi $M_0 = mgr$, gdzie m to masa rowerzysty, r – długość korby. Moment siły, z jakim koło działa na podłoże, to $M_1 = \frac{28}{22} M_0$, a siła działająca na punkt styczności koła z podłożem

$$F_1 = \frac{M_1}{R},$$

gdzie R jest promieniem koła. Załóżmy, że koło toczy się bez poślizgu. Wtedy wartość siły tarcia statycznego F_T jest równa F_1. Siła równoległa do zbocza, działająca na rowerzystę z rowerem, jest równa $F_g = \frac{6}{5} mg \sin\alpha$, gdzie α to kąt nachylenia zbocza. Z warunku równowagi sił $F_T = F_g$ po uproszczeniu dostajemy warunek

$$\sin\alpha_{\max} = \frac{r}{R} \cdot \frac{28}{22} \cdot \frac{5}{6}.$$

Wstawiając dane, otrzymujemy maksymalną wartość kąta $\alpha_{\max} \approx 35°$. Dla większych kątów, nawet opierając cały ciężar swojego ciała na pedale, rowerzysta wraz z rowerem będzie się staczał do tyłu. W sytuacji granicznej, kiedy moment siły, z jaką rowerzysta naciska na pedał, przeniesiony przez przekładnie na podłoże, dokładnie równoważy składową ciężaru układu styczną do zbocza, rower spoczywa lub porusza się ruchem jednostajnym. W tej sytuacji warunek na brak poślizgu jest taki sam, jak dla klocka umieszczonego na równi pochyłej: $\mu \geqslant \operatorname{tg}\alpha$, czyli dla danych w zadaniu i $\alpha = \alpha_{\max}$ mamy

$$\mu \geqslant 0{,}71.$$

Jeżeli żądamy, aby koło toczyło się bez tarcia w całym zakresie kątów, to siła F_1 nie może przekroczyć maksymalnej siły tarcia statycznego równej

$$F_{T\max} = \mu \frac{6}{5} mg \cos\alpha.$$

Po uproszczeniu warunek $F_1 \leqslant F_{T\max}$ sprowadza się do

$$\mu \geqslant \frac{r}{R} \cdot \frac{28}{22} \cdot \frac{5}{6} \cdot \frac{1}{\cos\alpha}.$$

Oba otrzymane ograniczenia na współczynnik tarcia są rosnącymi funkcjami α i są tożsame dla kąta $\alpha = \alpha_{\max}$. Wystarczy zatem $\mu \geqslant 0{,}71$.

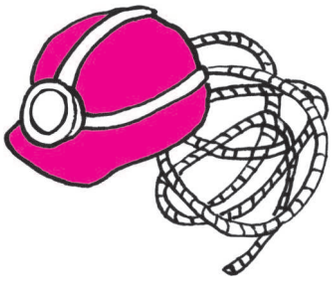
W końcu pokażemy, jak szybko wyliczać minima we wzorze $(**)$. Załóżmy, że mamy strukturę danych, która przechowuje pary (indeks, wartość) i udostępnia operacje jak na kolejce: wstawienie pary na koniec kolejki (*push*) oraz usunięcie pary z początku kolejki (*pop*). Dodatkowo operacja *first* będzie zwracać indeks pary z początku kolejki, a kluczowa operacja *min* będzie zwracała minimum ze wszystkich wartości w kolejce. Będziemy korzystać z $2n$ kolejek, które nazwiemy $A[i]$ oraz $B[i]$ dla $1 \leqslant i \leqslant n$.

Podczas wyliczania $d[a,b]$ w ℓ-tej fazie algorytmu będziemy zakładać, że wszystkie wartości z lewego minimum w $(**)$ znajdują się w kolejce $B[b]$ (w kolejności malejących indeksów i), a wszystkie wartości z prawego minimum w kolejce $A[a]$ (w kolejności rosnących indeksów). Kolejka $A[a]$ w fazie $\ell - 1$ była wykorzystywana podczas obliczeń dla komórki $d[a, b-1]$, zatem zawiera pary $(i, t[i] + d[a, i-1])$ dla indeksów $s[a,b-1] < i \leqslant b-1$. Ponieważ $s[a,b-1] \leqslant s[a,b]$, więc wystarczy usunąć z niej pary o indeksach nie większych niż $s[a,b]$ oraz dodać parę o indeksie b. Analogicznie uaktualniamy pary w kolejce $B[b]$. Pseudokod całego algorytmu jest następujący (zakładamy, że dla pustej kolejki pętla while nie wykonuje się, a operacja *min* zwraca ∞):

```
for ℓ := 0 to n − 1 do
  for a := 1 to n − ℓ do
    b := a + ℓ;
    for i := s[a, b − 1] to s[a + 1, b] do
      if d[a, i − 1] ⩽ d[i + 1, b] then
        s[a, b] := i;
    A[a].push(b, t[b] + d[a, b − 1]);
    while A[a].first ⩽ s[a, b] do A[a].pop;
    B[b].push(a, t[a] + d[a + 1, b]);
    while B[b].first > s[a, b] do B[b].pop;
    d[a, b] := min(A[a].min, B[b].min);
```

Kolejki możemy zaimplementować tak, aby wszystkie udostępniane operacje działały w zamortyzowanym czasie stałym. Zauważmy, że tuż przed wykonaniem operacji $push(i, v)$ możemy usunąć z końca kolejki wszystkie pary o wartościach nie mniejszych niż v, gdyż para (i, v) będzie lepszym kandydatem na minimum. Dzięki temu pary znajdujące się w kolejce będą zawsze posortowane rosnąco według wartości, zatem operacja *min* po prostu zwróci wartość pary z początku kolejki. Ostatecznie złożoność czasowa algorytmu wyniesie $O(n^2)$.

Tomasz IDZIASZEK



Grupowa eksploracja

*Dominik PAJĄK**

Wyobraźmy sobie sytuację, w której grupa speleologów chce wyeksplorować nieznaną jaskinię. Przy każdym rozgałęzieniu muszą podejmować decyzję, ilu z nich pójdzie każdym z nowych tuneli. Być może czasami będzie się opłacało zostawić część z nich przy rozgałęzieniu. Niektóre z tuneli będą się, być może, kończyły ślepo, a niektóre będą się dalej rozgałęziały. Speleolodzy mają telefony i mogą się ze sobą komunikować, więc gdy jeden z nich odkryje tunel z dużą liczbą rozgałęzień, może wezwać posiłki. Intuicyjnie rozsądne wydaje się wysyłanie większej liczby speleologów w te rejony jaskini, w których jest więcej tuneli. Chcielibyśmy jakoś sformalizować (i nieco uprościć) ten problem

*University of Cambridge