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

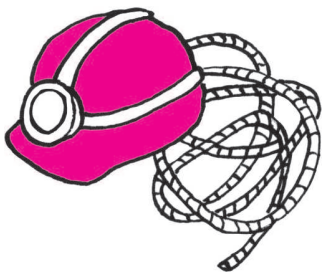
W końcu pokażemy, jak szybko wyliczać minima we wzorze $(**)$. Załóżmy, że mamy strukturę danych, która przechowuje pary (indeks, wartość) i udostępnia operacje jak na kolejce: wstawienie pary na koniec kolejki (*push*) oraz usunięcie pary z początku kolejki (*pop*). Dodatkowo operacja *first* będzie 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