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

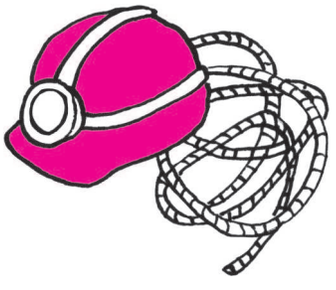
W końcu pokażemy, jak szybko wyliczać minima we wzorze $(**)$. Załóżmy, że mamy strukturę danych, która przechowuje pary (indeks, wartość) i udostępnia operacje jak na kolejce: wstawienie pary na koniec kolejki (*push*) oraz usunięcie pary z początku kolejki (*pop*). Dodatkowo operacja *first* będzie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

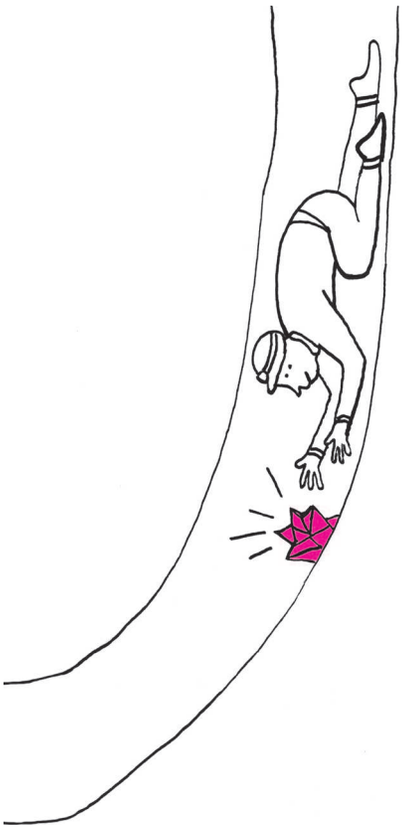


Grupowa eksploracja

*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



i przeanalizować takie naturalne podejście. Uproszczenia będą dwa: jaskinia będzie przedstawiona jako drzewo, czyli spójny graf bez cykli, a zespół speleologów będzie dość liczny...

Problem definiujemy następująco: mamy danych k agentów, początkowo umieszczonych w korzeniu r *nieznanego* drzewa

$$T = (V, E)$$

o *znanej* liczbie wierzchołków

$$n = |V|.$$

Drzewo jest nieznane, to znaczy, że nasza początkowa wiedza to tylko liczba krawędzi wychodzących z korzenia. W pierwszym kroku nasz algorytm musi wybrać, ilu agentów przetrawersuje krawędzie wychodzące z r, a ilu zostanie w r do następnej rundy. Jeżeli agentowi przydzielimy trawersowanie krawędzi z r do pewnego sąsiada v, to przejście krawędzi zajmie agentowi całą rundę i pojawi się on w wierzchołku v na początku następnej rundy. W następnej rundzie będzie on mógł przetrawersować dowolną krawędź wychodzącą z v. Wszyscy agenci, którzy w danej rundzie przechodzą krawędzie, robią to równocześnie.

Przez D będziemy oznaczać odległość od korzenia r do najdalszego liścia l w drzewie T. Odległość jest liczona jako liczba krawędzi na ścieżce łączącej r z l. Oczywiście, nie jesteśmy w stanie wyeksplorować w czasie mniejszym niż D, ponieważ przejście ścieżki z r do l zajmuje D rund.

Gdy wierzchołek jest odwiedzony przez pewnego agenta po raz pierwszy, to otrzymujemy informację o jego stopniu (liczbie wychodzących krawędzi). Ponadto dostajemy też informację o tym, która krawędź prowadzi do korzenia. Po pewnej liczbie kroków eksploracji już coś wiemy o wyglądzie drzewa T. Oznaczmy przez $T^{(t)}$ poddrzewo drzewa T, składające się z wierzchołków, które zostały poznane do chwili t. Na przykład $T^{(1)}$ to korzeń razem z dziećmi korzenia, ponieważ już na początku pierwszej rundy wiemy, ile dzieci ma korzeń. Drzewo $T^{(t)}$ zatem zawiera pewne wierzchołki, które nie są jeszcze odwiedzone, ale są już znane. Dla $v \in V$ oznaczmy przez $T^{(t)}(v)$ poddrzewo drzewa $T^{(t)}$ zawierające tylko te wierzchołki, dla których v jest przodkiem (ucinamy krawędź łączącą v z jego rodzicem i dostajemy $T^{(t)}(v)$). Przez $L(T^{(t)}(v))$ oznaczamy liczbę wierzchołków znanych, ale nieodwiedzonych w drzewie $T^{(t)}(v)$. Czasami na wierzchołki znane, ale nieodwiedzone będziemy mówili „nieukończone ścieżki", ponieważ każdy taki wierzchołek jest korzeniem, być może dużego, nieznanego drzewa.

Dostajemy do dyspozycji pokaźnych rozmiarów zespół agentów

$$k = \lceil Dn^c \rceil,$$

dla pewnej stałej $c > 1$, ale chcemy eksplorować szybko, czyli w czasie jak najbliższym D.

Propozycja algorytmu. Na początku wszyscy agenci w korzeniu są nieaktywni. W każdej rundzie aktywujemy x agentów w korzeniu (x wyznaczymy później). Wszyscy aktywni agenci w każdej rundzie idą w dół drzewa. Agentów dzielimy proporcjonalnie do liczby nieukończonych ścieżek w poddrzewach. Drobnym niuansem są zaokrąglenia; w pseudokodzie zapiszemy wszystko dokładnie:

Algorithm $\mathcal{A}$ (T, r, x) **at time step** s:
  Aktywuj x agentów w korzeniu r.
  for each odwiedzony $v \in V(T^{(s)})$ **do**: { ustalamy, gdzie posłać agentów z v }
    Niech $\mathcal{A}_v^{(s)}$ – zbiór agentów w v na początku rundy s.
    Oznaczmy przez $v_1, v_2, \ldots, v_d$ wszystkie dzieci v.
    Niech $i^* := \arg\max_i \{L(T^{(s)}(v_i))\}$. { dziecko z największą liczbą „nieukończonych ścieżek" }
    Podziel $\mathcal{A}_v^{(s)}$ na rozłączne zbiory $\mathcal{A}_{v_1}, \mathcal{A}_{v_2}, \ldots, \mathcal{A}_{v_d}$, takie że:
    (i) $|\mathcal{A}_{v_i}| = \left\lfloor \dfrac{|\mathcal{A}_v^{(s)}| \cdot L(T^{(s)}(v_i))}{L(T^{(s)}(v))} \right\rfloor$, dla każdego $i \in \{1, 2, \ldots, d\} \setminus \{i^*\}$,
    (ii) $|\mathcal{A}_{v_{i^*}}| = |\mathcal{A}_v^{(s)}| - \sum_{i \in \{1,2,\ldots,d\} \setminus \{i^*\}} |\mathcal{A}_{v_i}|$.
    for each $i \in \{1, 2, \ldots, d\}$ **do for each** agent $g \in \mathcal{A}_{v_i}$ **do move**$^{(s)}$ g to vertex v_i.
end algorithm $\mathcal{A}$.

Analiza. Algorytm $\mathcal{A}$ nie robi niczego odkrywczego, po prostu wysyła więcej agentów tam, gdzie jest więcej ścieżek do odkrycia. Ale jego analiza nie jest oczywista. Algorytm, odkrywając kolejne wierzchołki drzewa, na bieżąco zmienia wartości liczby nieukończonych ścieżek w poddrzewach (funkcja L), co ma wpływ na proporcje podziału liczby agentów. Wydaje się, że jest to kluczowa własność algorytmu, więc w naszej analizie musimy ją zbadać. Oczywiście, musimy też wyznaczyć odpowiednią wartość parametru x. Na razie ustalmy tylko, że $x > 2n$.

Weźmy dowolny liść f z drzewa T i ścieżkę długości $D_f \leqslant D$ łączącą f z korzeniem r. Oznaczmy tę ścieżkę przez $\mathcal{F} = (f_0, f_1, f_2, \ldots, f_{D_f})$, gdzie $f_0 = r$ i $f_{D_f} = f$. Chcemy się przyjrzeć, jak liczne grupy agentów będą wędrowały po ścieżce $\mathcal{F}$. Liczba agentów wysyłana wzdłuż ścieżki zależy od liczby nieukończonych ścieżek w odpowiednich poddrzewach. Jeżeli oznaczymy $\lambda_j^{(i)} = L(T^{(i)}(f_j))$, to i-tym w kroku z wierzchołka f_j wyślemy do f_{j+1} mniej więcej (z dokładnością do zaokrągleń) $\dfrac{\lambda_{j+1}^{(i)}}{\lambda_j^{(i)}}$-tą część agentów znajdujących się w f_j na początku rundy i. Zapomnijmy na chwilę o zaokrągleniach i pójdźmy dalej tym tropem. Do liścia f dotrze $\alpha_i = x\dfrac{\lambda_1^{(i)}}{\lambda_0^{(i)}}\dfrac{\lambda_2^{(i+1)}}{\lambda_1^{(i+1)}}\cdots\dfrac{\lambda_{D_f}^{(i+D_f-1)}}{\lambda_{D_f-1}^{(i+D_f-1)}}$ agentów wysłanych w rundzie i z korzenia. Zauważmy, jak wyglądają pierwsze dwie wartości:

$$\alpha_1 = x\frac{\lambda_1^{(1)}}{\lambda_0^{(1)}}\frac{\lambda_2^{(2)}}{\lambda_1^{(2)}}\frac{\lambda_3^{(3)}}{\lambda_2^{(3)}}\cdots\frac{\lambda_{D_f}^{(D_f)}}{\lambda_{D_f-1}^{(D_f)}},$$
$$\alpha_2 = x\frac{\lambda_1^{(2)}}{\lambda_0^{(2)}}\frac{\lambda_2^{(3)}}{\lambda_1^{(3)}}\frac{\lambda_3^{(4)}}{\lambda_2^{(4)}}\cdots\frac{\lambda_{D_f}^{(D_f+1)}}{\lambda_{D_f-1}^{(D_f+1)}}.$$

Jeżeli wymnożymy $\alpha_1 \cdot \alpha_2$, to dość dużo się uprości. Wymnóżmy zatem a kolejnych wartości α_i. Biorąc pod uwagę, że z definicji funkcji L mamy $1 \leqslant \lambda_i^{(j)} \leqslant n$, to dostaniemy $\prod_{i=1}^{a}\alpha_i \geqslant \dfrac{x^a}{n^{a+D}}$. Czyli jeżeli ustalimy $a = \left\lceil \dfrac{D\log_2 n}{\log_2(x/(2n))} \right\rceil$, to dostaniemy $\prod_{i=1}^{a}\alpha_i \geqslant 2^a$, czyli dla pewnego $j^* \in \{1, 2, \ldots, a\}$ mamy $\alpha_{j^*} \geqslant 2$. Gdyby nie było zaokrągleń i każde dziecko dostawałoby należny przydział agentów, to α_{j^*} agentów spośród grupy, która wyrusza w rundzie j^*, dotarłoby do liścia f. W rzeczywistości liczby agentów są czasem zaokrąglane w dół, zgodnie z regułą (i) z algorytmu, ale można pokazać, że te zaokrąglenia zbyt dużo nie psują.

Przyjrzyjmy się liczbie agentów, podążających ścieżką $\mathcal{F}$, spośród grupy, która wyrusza w rundzie j^*. Zauważmy, że w przypadku, gdy $\lambda_{i+1}^{(j^*+i)}/\lambda_i^{(j^*+i)} > 1/2$, to wtedy drzewo $T^{(j^*+i)}(f_{i+1})$ zawiera co najmniej połowę wszystkich otwartych ścieżek drzewa $T^{(j^*+i)}(f_i)$. W takim przypadku wierzchołek f_{i+1} będzie tym wyróżnionym dzieckiem wierzchołka f_i i dostanie swój należny przydział agentów (reguła (ii)). Zauważmy, że α_{j^*} jest iloczynem x i pewnej liczby czynników nie większych od 1. Skoro $\alpha_{j*} \geqslant 2$, to co najwyżej $\log_2 x - 1$ z tych czynników może być nie większych niż $1/2$. Agenci, którzy wyruszyli z korzenia w rundzie j^* i podążali ścieżką $\mathcal{F}$, zostali „zaokrągleni w dół" co najwyżej $\log_2 x - 1$ razy. Stąd da się już łatwo wykazać, że z grupy, która wyruszyła w rundzie j^*, co najmniej jeden agent dotrze do liścia f. Jeżeli wypuścimy z korzenia co najmniej $a = \left\lceil \dfrac{D\log_2 n}{\log_2(x/(2n))} \right\rceil$ grup agentów, to jedna z nich wyeksploruje dowolnie wybrany liść f, czyli a grup wyeksploruje całe drzewo. Całkowity czas eksploracji to $a + D - 1 \leqslant D \cdot \left(1 + \dfrac{\log_2 n}{\log_2(x/(2n))}\right)$, ponieważ tyle rund potrzeba, żeby a-ta grupa dotarła do liści. Łączna liczba użytych agentów to co najwyżej $x \cdot D \cdot \left(1 + \dfrac{\log_2 n}{\log_2(x/(2n))}\right)$. Ustalając $x = n^c/b$ dla odpowiednio dobranej stałej b, dostajemy:

Twierdzenie. *Dla dowolnego ustalonego $c > 1$ i znanego n problem eksploracji drzew może być rozwiązany w czasie $D \cdot \left(1 + \dfrac{1}{c-1} + o(1)\right)$ przy użyciu $\lceil Dn^c \rceil$ agentów.*

Notacja $o(1)$ oznacza funkcję, która dąży do 0 dla n dążącego do nieskończoności. Zauważmy, że znajomość n jest potrzebna algorytmowi jedynie do podziału zespołu wszystkich k agentów na grupy po x. Da się skonstruować algorytm, który eksploruje w czasie $O(D)$, bez znajomości n, używając Dn^c agentów.

Uwagi końcowe

- Można wykazać, że nie da się eksplorować szybciej niż w czasie $D \cdot \left(1 + \frac{1}{c} - o(1)\right)$, czyli jesteśmy już dość blisko optymalnego algorytmu.
- Algorytm można uogólnić na dwa sposoby. Można przerobić na wersję rozproszoną, gdzie agenci podejmują suwerenne decyzje, bazując na swojej wiedzy, i mogą się komunikować z innymi agentami będącymi w tym samym wierzchołku. Można też w ten sposób eksplorować dowolne grafy. Co ciekawe, oba uogólnienia nie dodają zbyt dużo do czasu eksploracji, czyli cała trudność problemu leżała w „centralnie sterowanej" eksploracji drzew.
- Czytelników, którzy zastanawiają się, czy to podejście da się zastosować dla mniejszej grupy agentów, gorąco zachęcam do próbowania.