Na podstawie tak przeprowadzonych wstępnych obliczeń możemy już sobie poradzić z obsługą niepełnych fragmencików. Na początku dla każdego fragmencika w ciągu a wyznaczamy jego typ (liczbę całkowitą od 0 do $\frac{1}{2}\sqrt{n}-1$) oraz początkowy element. Przy obsłudze zapytania identyfikujemy jeden lub dwa niepełne fragmenciki, składające się na segment S z zapytania, następnie na podstawie ich typów i tego, jakie ich kawałki są zawarte w S, za pomocą pojedynczych odwołań do wyżej skonstruowanej tablicy wyznaczamy szukane minima. Ponieważ były one obliczone przy założeniu, że początkowy element fragmencika jest równy 0, to musimy je przeskalować o faktyczne początkowe elementy rozważanych fragmencików. Za pomocą kilku prostych wzorów możemy z opisu zapytania w czasie stałym wyłuskać potrzebne nam parametry dotyczące skrajnych fragmencików i ich kawałków, które są zawarte w segmencie, zatem cały ten krok może zostać wykonany dla każdego zapytania RMQ w złożoności czasowej $O(1)$. Łączny czas wszystkich wykonanych po drodze wstępnych obliczeń jest liniowy względem n, co pokazuje, że otrzymaliśmy algorytm, działający tak szybko, jak chcieliśmy. Jest to zarazem najszybszy algorytm na jaki można było praktycznie liczyć (nie sposób sobie wyobrazić istnienia algorytmu o asymptotycznie mniejszej niż liniowa złożoności czasowej wstępnych obliczeń), możemy zatem uznać, że znaleźliśmy najlepsze możliwe rozwiązanie zarówno problemu RMQ, jak i LCA.

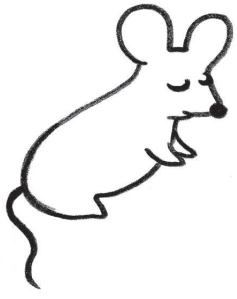
Empiryczne badanie uogólnionej wersji gry Hamleta

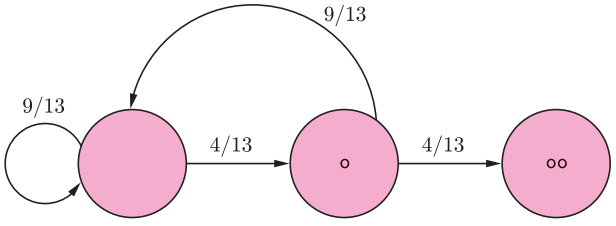
motywy Logo

Andrzej WALAT

W październikowym numerze *Delty* rozważaliśmy następującą grę. Losujemy litery ze słynnej kwestii Hamleta: `to be or not to be` (być albo nie być) tak długo, aż otrzymamy dwuliterowe słowo `to`; za każde wylosowanie litery płacimy złotówkę, ale na końcu, po uzyskaniu słowa `to` otrzymujemy nagrodę n złotych. Ustaliliśmy, że aby gra była opłacalna, wartość nagrody n powinna być większa niż średnia liczba losowań, jakie trzeba wykonać, by otrzymać słowo `to`. Obliczyliśmy, że teoretyczna wartość tej średniej to $\overline{x} = \frac{13}{3} \cdot \frac{13}{4} = \frac{169}{12} \approx 14{,}08$.

Tym razem zajmiemy się innymi wariantami gry Hamleta. Na początek przyjmijmy, że naszym docelowym słowem jest `oo` i obliczmy średni czas oczekiwania na to słowo (tj. średnią liczbę losowań, jakie trzeba wykonać, by otrzymać `oo`). W tym przypadku gra Hamleta jest równoważna losowej wędrówce pionka po planszy przedstawionej na rysunku 1 od pola początkowego x do pola końcowego `oo`,





Rys. 1

a odpowiedni układ równań liniowych ma następującą postać:

$$\begin{cases} \overline{x} = 1 + \frac{4}{13}\overline{o} + \frac{9}{13}\overline{x} \\ \overline{o} = 1 + \frac{4}{13} \cdot 0 + \frac{9}{13}\overline{x} \end{cases}$$

Po jego rozwiązaniu otrzymujemy średni czas oczekiwania na dwuliterowe słowo `oo`:

$$\overline{x} = \frac{221}{16} = \frac{13}{4} + \left(\frac{13}{4}\right)^2.$$

Czy w grze Hamleta średni czas oczekiwania na trzyliterowe słowo `ooo` jest równy

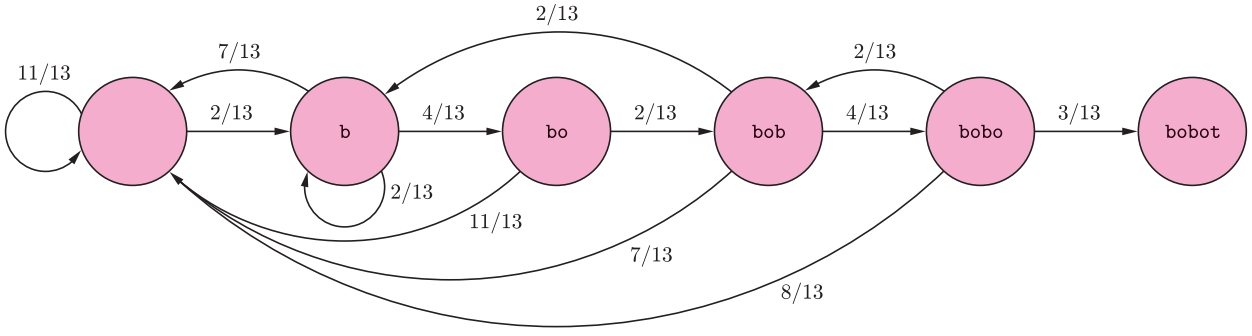
$$\frac{13}{4} + \left(\frac{13}{4}\right)^2 + \left(\frac{13}{4}\right)^3?$$

Fakt, że średni czas oczekiwania na dwuliterowe słowo `oo` okazał się krótszy niż odpowiedni średni czas oczekiwania na słowo `to`, chyba nikogo nie zdziwił. Te dwa słowa różnią się tylko pierwszą literą i litera `o` występuje w haśle `tobeornottobe` częściej niż litera `t`. Intuicja podpowiada nam, że na słowa utworzone z liter występujących w haśle z większą częstością będziemy średnio czekali krócej niż na słowa utworzone z liter występujących rzadziej. Czy wobec tego średni czas oczekiwania na `ooo` jest krótszy niż odpowiedni średni czas oczekiwania na słowo `too`?

Żeby odpowiedzieć na to pytanie, wystarczy ułożyć i rozwiązać dwa układy równań liniowych z trzema niewiadomymi. Ale potem pojawią się kolejne pytania i układanie oraz rozwiązywanie odpowiednich układów liniowych w końcu nam się znudzi. Dobrze byłoby mieć jakieś rozwiązanie ogólne, które dla dowolnego hasła oraz docelowego słowa daje średnią liczbę losowań, jakie trzeba wykonać, by wybierając losowo litery z hasła, utworzyć dane słowo docelowe. W tym artykule przedstawię empiryczne rozwiązanie tego problemu, a w kolejnym, grudniowym numerze *Delty* – rozwiązanie teoretyczne. W obu przypadkach posłużę się jako narzędziem językiem Logo.

Rozwiązanie empiryczne

W ogólnym przypadku dane jest dowolne hasło i cel – słowo, które chcemy wygenerować, losując litery z hasła. Ten proces wymaga wielu losowań, nie mniej niż długość słowa docelowego, ale zwykle znacznie więcej. Wyróżniamy tyle różnych stanów procesu, ile jest różnych prefiksów danego słowa docelowego. Na przykład, jeśli dane jest hasło Hamleta `tobeornottobe` oraz cel `bobot`, to wyróżniamy 6 różnych stanów: słowo puste, jednoliterowe słowo `b` oraz `bo`, `bob`, `bobo`, `bobot`. Rysunek 2 przedstawia graf przejść pomiędzy stanami procesu.



Rys. 2

Naszym zadaniem jest zdefiniowanie funkcji `lilo`, która dla danego hasła i celu wyznacza losową liczbę kroków (losowań), po których zaczynając od stanu początkowego, osiągamy cel. Przedtem jednak rozwiążemy problem bardziej ogólny. Zdefiniujemy funkcję `lk`, która dla danego dowolnego stanu procesu, danego hasła oraz słowa docelowego wyznacza losową liczbę kroków od danego (aktualnego) stanu do celu. Wynik tej funkcji obliczamy w następujący sposób: sprawdzamy, czy dany (aktualny) stan jest docelowy, jeśli tak, to wynikiem funkcji jest liczba 0 i koniec, w przeciwnym przypadku trzeba wykonać jeden krok do jakiegoś wyznaczonego losowo następnego stanu – żeby obliczyć wynik, do liczby 1 musimy dodać liczbę kroków od następnego stanu do celu.

W Logo można to zakodować w następujący sposób.

```
oto lk :stan :hasło :cel
jeśli :stan = :cel [wynik 0]
wynik 1 + lk następny :stan :hasło :cel
już
```

A jak wyznaczyć następny stan procesu? Wybieramy losowo literę z hasła i dopisujemy ją na koniec aktualnego stanu. Wynikiem jest najdłuższy sufiks otrzymanego w ten sposób słowa, który jest prefiksem danego słowa docelowego.

```
oto następny :stan
wynik suprefiks nak los :hasło :stan :cel
już
```

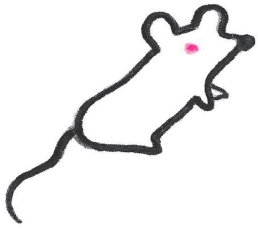
Brakuje jeszcze definicji funkcji suprefiks, która znajduje najdłuższy sufiks pierwszego danego słowa będący prefiksem drugiego danego słowa.

```
oto suprefiks :s1 :s2
jeśli prefiks? :s1 :s2 [wynik :s1]
wy suprefiks bp :s1 :s2
już
```

```
oto prefiks? :s1 :s2
jeśli puste? :s1 [wy "prawda]
jeśli (pierw :s1) <> pierw :s2 [wy "fałsz]
wy prefiks? bp :s1 bp :s2
już
```

Obliczenie liczby kroków od stanu początkowego (czyli od słowa pustego) do celu jest tylko szczególnym przypadkiem zadania obliczenia liczby kroków od dowolnego stanu procesu do końca. Zapowiedzianą wyżej funkcję `lilo` można zdefiniować w następujący sposób.

```
oto lilo :hasło :cel
wynik lk " :hasło :cel
już
```



Możemy teraz eksperymentalnie sprawdzić, ile razy trzeba losować litery z hasła Hamleta, by wygenerować nasze przykładowe słowo `bobot`. Jednocześnie przetestujemy nasze rozwiązania programistyczne.

```
? powtórz 10 [wpisz lilo "tobeornottobe "bobot wpisz "|, |]
5678, 417, 1508, 727, 650, 2446, 1400, 1087, 2365, 2177,
```

Najszybciej otrzymaliśmy słowo `bobot` po 417 losowaniach, chociaż teoretycznie mogłoby się nam to udać już po pięciu. Największa liczba losowań to 5678. Trudno dokładnie ocenić odpowiednią średnią wartość liczby losowań na podstawie 10 wyników. Zdefiniujemy jeszcze jedną funkcję `ślilo`, której wynikiem jest średnia z n wartości funkcji `lilo` dla danego hasła i celu.

```
oto ślilo :n :hasło :cel
niech "suma 0
powtórz :n [przyp "suma :suma + lilo :hasło :cel]
wynik :suma / :n
już
```

Obliczmy średnią ze 100 wartości `lilo`. Po napisaniu polecenia:

```
pisz ślilo 100 "tobeornottobe "bobot
```

komputer wypisał `1798.37`, ale po wywołaniu tego samego polecenia jeszcze raz komputer wypisał komunikat o błędzie:

```
Błąd w wierszu 2 procedury prefiks?: Zbyt wiele wywołań procedur.
Prawdopodobnie nieskończona rekurencja.
```

Nasze procedury, chociaż są poprawne, mają ograniczony zakres stosowalności. Zadanie wyznaczania średniego czasu oczekiwania na słowo `bobot` leży

Rozwiązanie zadania F 704.
Początkowo tłoki znajdowały się w stanie równowagi, zatem były spełnione równania

$$2p_0 = p_0 + mg/S$$

oraz

$$3p_0 = p_0 + 2mg/S,$$

gdzie S jest polem przekroju poprzecznego naczynia. Po dociążeniu górnego tłoka, z prawa Boyle'a–Mariotte'a mamy, że:

$$3p_0 dS = l(Q + 2p_0 S),$$

$$2p_0 dS = (d - l)(Q + p_0 S),$$

gdzie l to nowa wysokość dolnego tłoka, a Q to siła dociskająca górny tłok. Z powyższych równań otrzymujemy:

$$l = d(3 - \sqrt{6}) \approx 2{,}75 \text{ cm}.$$

na granicy tego zakresu. Można napisać inną iteracyjną wersję procedury `lilo`, by ten zakres znacznie poszerzyć. Pozostawiam to jako otwarte zadanie dla Ambitnego Czytelnika. Tymczasem posłużymy się funkcją `ślilo`, by odpowiedzieć na kilka pytań dotyczących gry Hamleta (i nie tylko).

1. Na które słowo: `too` czy `ooo` trzeba średnio dłużej czekać?
Żeby to sprawdzić, napiszmy:

```
pisz ślilo 1000 "tobeornottobe "too
44.858
```

a następnie:

```
pisz ślilo 1000 "tobeornottobe "ooo
48.201
```

Wynik tego pojedynku jest korzystny dla `too`. Średni czas oczekiwania na `too` w tysiącu próbach okazał się krótszy niż w przypadku `ooo`. Wątpiącemu Czytelnikowi, który ma trudność z pogodzeniem tego faktu z intuicją, proponuję narysowanie dwóch grafów podobnych, jak na rysunkach 1 i 2, reprezentujących procesy losowania liter z hasła Hamleta, aż do uzyskania odpowiedniego słowa too oraz `ooo`, a następnie wyobrażenie sobie, jak mogą przebiegać te dwa procesy.

2. Czy średni czas oczekiwania na słowa: `too`, `oto` oraz `oot` jest taki sam?
Tym razem napiszemy po kolei następujące dwa polecenia:

```
pisz ślilo 1000 "tobeornottobe "oot
44.709
```

```
pisz ślilo 1000 "tobeornottobe "oto
49.081
```

Średni czas oczekiwania na słowo `oot` okazał się zbliżony do otrzymanego przed chwilą średniego czasu oczekiwania na `too`. Średni czas oczekiwania na `oto` okazał się wyraźnie dłuższy. Czy te eksperymentalne wyniki dają podstawę do twierdzenia, że odpowiednie teoretyczne średnie czasy oczekiwań na `too` oraz `oot` są równe i mniejsze niż teoretyczny średni czas oczekiwania na `oto`? Tym i innymi podobnymi zagadnieniami zajmiemy się w następnym numerze. Ambitny Czytelnik może potraktować je jako zadanie domowe.

Na koniec zajmiemy się jeszcze jednym problemem, który z pozoru nie ma związku z grą Hamleta.

3. Ile razy średnio trzeba rzucić kostką, żeby otrzymać trzy szóstki po kolei?
Teoretycznie: $6 + 6^2 + 6^3 = 258$.

Posłużymy się funkcją `ślilo`, by skonfrontować ten teoretyczny wynik z doświadczeniem. W przypadku, gdy hasłem jest sześcioliterowe słowo utworzone z kolejnych cyfr `123456`, a słowo docelowe to sekwencja trzech szóstek `666`, gra Hamleta jest równoważna z rzucaniem kostką aż do uzyskania trzech szóstek po kolei. Użyjemy funkcji `ślilo`, by obliczyć 10 razy średni czas oczekiwania na trzy szóstki w serii 1000 prób.

```
powtórz 10 [wpisz ślilo 1000 123456 666 wpisz "|, |]
253.904, 269.942, 257.992, 256.343, 263.492, 258.231, 252.877, 257.92,
263.571, 260.772,
```

Otrzymane empiryczne średnie są dość zgodne ze średnią teoretyczną, ale jak zwykle tylko do pewnego stopnia. Największa otrzymana eksperymentalnie średnia różni się od dokładnej wartości teoretycznej o około 5%. Rzucanie kostką aż do uzyskania określonej sekwencji wyników nie jest jedynym przykładem procesu losowego, który sprowadza się do gry Hamleta. To jeden z powodów, by się nią zainteresować. W kolejnym odcinku wrócimy do tej gry i będziemy ją rozważali z perspektywy teoretycznej.