Ezoteryczne Kartki
Algorytmy randomizowane I

Jakub Bachurski

wersja 1.0.4.2

1 Oszustwo

Od samego początku przyzwyczajani jesteśmy do rozwiązywania problemów deterministycznie – uruchamiając program wiele razy, oczekuje się, że zadziała dokładnie tak samo. Jednakże! Tak być nie musi. Czasami można wykorzystać potęgę prawdopodobieństwa na naszą korzyść. Zrzucając brudną robotę na kogo innego (w tym wypadku matematykę) często znacznie upraszczamy implementację i dochodzimy do zaskakująco prostych rozwiązań dorównujących wzorcowym. Możliwe, że takie rozwiązanie będzie lepsze od wzorcowego lub nawet nim będzie.

W rozwiązaniach randomizowanych będziemy korzystać z założeń (np. że wynik znajdziemy stosunkowo szybko) oraz z pseudolosowości – będziemy losować pewne wyniki częściowe.

2 Fundamenty

2.1 Losowanie

- Losując przedmiot z grupy w której $\frac{1}{p}$ przedmiotów spełnia pewną własność, wylosujemy taki przedmiot z prawdopodobieństwem $\frac{1}{p}$.
- Powtarzając losowanie dopóki nie napotkamy takiego przedmiotu, uda nam się to po *około* p próbach.

Na przykład, losując liczby całkowite do n po $O(\log n)$ próbach trafimy na liczbę pierwszą (bo gęstość liczb pierwszych jest rzędu $O(\frac{1}{\log n})$).

n	$\log n$	Średnia liczba prób
10^2	3.991	4.605
10^4	8.187	9.210
10^6	12.729	13.815
10^8	17.401	18.420
10^{10}	21.893	23.025

2.2 Wzór Stirlinga

Silnie są nieprzyjemne pod tym względem, że trudno je do czegoś przyrównać, ze względu na ich trudną obliczeniowo naturę. Na szczęście istnieje wzór, który z całkiem dobrą precyzją aproksymuje silnię – *wzór Stirlinga*.

$$n! \approx \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$$

n	$n!$	$\frac{n!}{\text{Stirling}} - 100\%$
6	720	1.397%
10	3628800	0.836%
42	$10^{51.147}$	0.198%
100	$10^{157.97}$	0.083%

3 Paradoks dnia urodzin

Niezwykle przydatnym i zaskakującym bytem jest *paradoks dnia urodzin*. Jego nazwa pochodzi od następującego problemu, w którym się pojawia:

Powiedzmy, że w pokoju jest k osób. Ile musi co najmniej wynosić k, aby prawdopodobieństwo, że są *dwie osoby o takim samym dniu urodzin* (tzn. tego samego dnia roku), wynosiło co najmniej 50%?

Rozumowanie przebiega następująco. Będziemy rozpatrywać kolejno każdą osobę i sprawdzać prawdopodobieństwo, z jakim ma ona inny dzień urodzin niż wszystkie pozostałe, które rozpatrzyliśmy przedtem. Możliwych dni jest $n = 365$[1]. Oznaczmy poszukiwane prawdopodobieństwo przez $\mathcal{P}$.

$$\mathcal{P} = \frac{n}{n} \cdot \frac{n-1}{n} \cdot \frac{n-2}{n} \cdot \ldots \cdot \frac{n-k+1}{n} = \frac{n!}{(n-k)!} n^{-k}$$

Dalsze wyprowadzenia korzystają z mniej fajnej matmy, więc odsyłam np. [tutaj]. Co ważne, prawdopodobieństwo dla dostatecznie małych k można zaokrąglić jako:

$$\mathcal{P} \approx \frac{k^2}{2n}$$

Możemy z tego wywnioskować, że odpowiedzią na nasze pytanie jest $k \approx \sqrt{n}$. Sprawdźmy doświadczalnie, ile średnio prób potrzebujemy, by dwa razy wylosować ten sam element ze zbioru n-elementowego:

n	$\sqrt{n}$	Średnia liczba prób
42	6.480	8.752
365	19.104	24.643
10^5	316.227	394.687
10^7	3162.277	3931.156

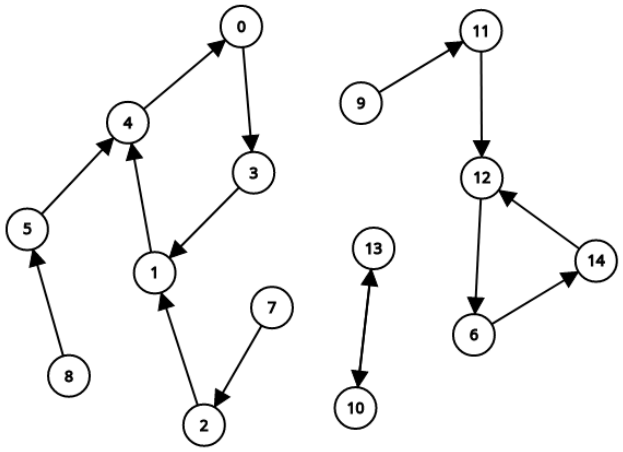
[1] W tej lepszej, alternatywnej rzeczywistości nie ma lat przestępnych.

4 Fakciki

- W losowej permutacji długości n liczba unikatowych wartości w ciągu maksimów prefiksowych jest $O(\log n)$. Dowód dla czytelnika.[2]

- Gdy losujemy n punktów równomiernie z pewnego koła, średnia wielkość ich otoczki wypukłej jest $O(n^{\frac{1}{3}})$. Natomiast gdy są one losowane z pewnego wielokąta wypukłego o k wierzchołkach, wielkość otoczki jest $O(k \log n)$.[3]

- Graf funkcyjny to taki, że z każdego wierzchołka wychodzi dokładnie jedna skierowana krawędź. Taki graf wygląda mniej więcej tak[4]:

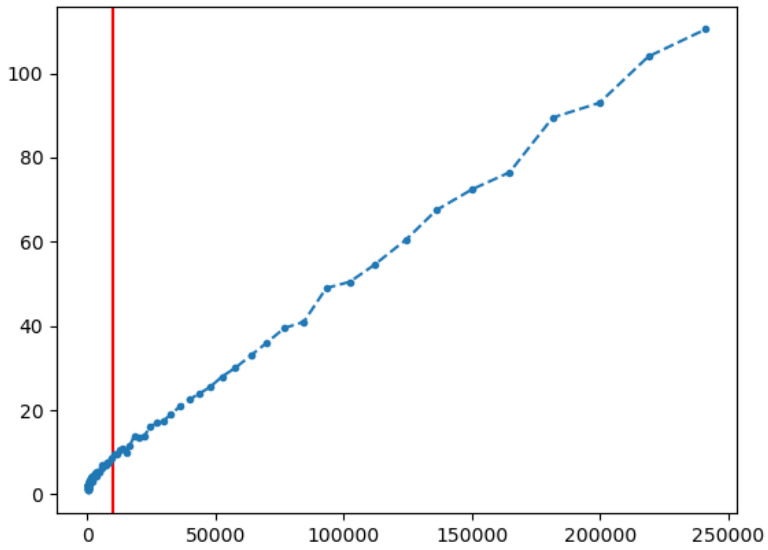


  Rozważmy pewien wierzchołek tego grafu. Zaczynając od niego i chodząc po kolejnych krawędziach po około $O(\sqrt{n})$ przejściach trafimy na wcześniej odwiedzony już wierzchołek. Wynika to z paradoksu dnia urodzin – możemy traktować przejście krawędzią jako losowanie następnego wierzchołka. Nazwa „graf funkcyjny" pochodzi od tego, że można go zinterpretować jako funkcję $f : \mathbb{Z}_n \to \mathbb{Z}_n$ ($\mathbb{Z}_n$ oznacza tu zbiór $\{0, 1, 2, ..., n-1\}$), gdzie krawędź wychodząca z $v \in V$ to $f(v)$.

- Średnia długość najdłuższej ścieżki w losowym DAG-u jest mała.

[2] `https://codeforces.com/blog/entry/62602?#comment-465550`
[3] `https://arxiv.org/abs/1111.5340`
[4] Wizualizacja stworzona za pomocą `https://csacademy.com/app/graph_editor/`

Do ostatniego fakciku mam wykresik, bo nie mogłem znaleźć źródła:



Oś x to liczba krawędzi, oś y to średnia długość najdłuższej ścieżki. Czerwona kreska przedstawia testowaną wartość liczby wierzchołków: $n = 10^4$. Na podstawie wykresu wnioskujemy (pochopnie), że gdy $x = O(n)$, $y = O(\text{mało})$.

Gdy coś jest losowe, zawsze przetestuj jego własności!

Celem nie jest koniecznie zapamiętanie tych wszystkich własności, a umiejętność znalezienia ich samodzielnie. Wykres nie jest przypadkowy – dobra wizualizacja danych może pomóc w zrozumieniu zależności.

5 Gwarancje

Ważną częścią rozwiązywania zadań na algorytm randomizowany jest analiza gwarancji. Jeżeli mamy dane, że pewne ścieżki w grafie są długie, poszukiwany podzbiór jest wielkości połowy zbioru, et cetera, to warto przyjrzeć się np. jaka jest szansa, że wylosujemy krawędź na poszukiwanej ścieżce, czy wylosujemy element(-y) należące do poszukiwanego podzbioru.

6 Implementacja

6.1 Losowe liczby

Jeżeli wcześniej losowaliście liczby, to najpewniej używaliście do tego `rand()`. Jest to okropny pomysł – `rand` ma na niektórych systemach bardzo mały za-

kres, gorszą wydajność od alternatyw i krótki okres. Lepszym pomysłem jest wykorzystać bibliotekę `<random>`, dodaną w `C++11`[5]. Przykład wykorzystania:

```
// ziarno dla generatora - liczba
int seed = 42;
// czas (w sekundach)
int seed1 = time(0);
// czas (w małych jednostkach)
int seed2 = chrono::high_resolution_clock::now()
            .time_since_epoch().count();

// Tworzymy generator
mt19937 gen(seed); // lub mt19937_64 dla 64-bitowych

cout << gen() << endl; // liczba od 0 do 2^32 - 1 (2^64 - 1)

// losowa liczba w zakresie [lo, hi], losowana równomiernie
// jako parametr szablonu można podać inny typ
long randint(long lo, long hi)
{
    // Dystrybucja wykorzystuje generator by wylosować liczbę
    return uniform_int_distribution<long>{lo, hi}(gen);
}

cout << randint(42, 1337) << endl;
```

Nazwa generatora pochodzi od wykorzystywanej implementacji – Mersenne Twister, z okresem $2^{19937}-1$. A jeśli ktoś nie wierzy, że `rand` jest słaby, odsyłam do bloga: `https://codeforces.com/blog/entry/61587`.

6.2 Tasowanie

Shuffle (po rodzimemu tasowanie) to operacja losowej zmiany kolejności elementów ciągu. W `C++11` zapewniono nam funkcję która przyjmuje parę iteratorów oraz generator (taki jak `mt19937`) – `shuffle`.

```
vector<int> a = {1, 2, 3, 4, 5};
shuffle(a.begin(), a.end(), gen);
```

Tasowanie przydaje się gdy chcemy wybrać np. k losowych elementów ze zbioru bez powtórzeń (tasujemy i bierzemy pierwsze k indeksów) albo pozbyć się nielosowości danych (żeby nasz algorytm nie napotkał przypadków pesymistycznych – najprostszy quicksort dzięki tasowaniu ma właściwie gwarantowane $O(n \log n)$).

[5] Zatem do opcji kompilacji trzeba dodać `std=c++11`.

7 Zadanka

7.1 [Finał XXVI OI] Długie podróże

Treść Mamy dany graf nieskierowany o n wierzchołkach i m krawędziach, oraz q zapytań o długość najkrótszej ścieżki między danymi wierzchołkami u_i i v_i. Zagwarantowano nam, że wynik jest równy co najmniej $\frac{n}{10}$.

Rozwiązanie Przyjrzyjmy się jednemu z zapytań. Jak rozwiązać je w porządnym czasie? Najprościej byłoby policzyć BFSa z wierzchołka u_i i sprawdzić odległość do v_i. Jednak jest to beznadziejny pomysł – jest to wybór okropnie specyficzny, a wredni autorzy testów z pewnością spowodują, że nic więcej nie zrobimy z tymi informacjami. Zamiast tego *wylosujmy* sobie wierzchołek s, z którego puścimy BFSa.

Niech odległość wierzchołka w od s wynosi $d_s(w)$, a długość najkrótszej ścieżki między u i v wynosi $d(u, v)$. Spróbujmy teraz wykorzystać dane nam informacje. Najsensowniej jest sprawdzić, czy $d_s(u_i) + d_s(v_i)$ nie jest przypadkiem poprawnym wynikiem. Z jakim prawdopodobieństwem się to wydarzy? Zauważmy, że

$$d_s(u_i) + d_s(v_i) = d(u_i, v_i)$$

jest równoważne założeniu, że s leży na pewnej najkrótszej ścieżce między u_i i v_i. Ponieważ takich wierzchołków jest co najmniej $\frac{n}{10}$, to s należy do tych wierzchołków z prawdopodobieństwem $\frac{1}{10}$. Zatem losując s (wierzchołek pośredni) T razy mamy prawdopodobieństwo na poprawność wyniku $1 - \left(\frac{9}{10}\right)^T$.

Losując wierzchołek s puszczamy z niego BFS w $O(n + m)$, i za jego pomocą sprawdzamy, czy nie znaleźliśmy lepszego wyniku dla każdego z q zapytań. Robiąc to $T = 200$ razy mamy prawdopodobieństwo poprawności każdego z wyników rzędu 10^{-10}, co jest jak najbardziej wystarczające i mieści się w limitach zadania. Sumaryczna złożoność to $O(T(n + m + q))$.

7.2 [Sesja próbna Finału PA 2018] Wieżowce

Z zadaniem pierwszy raz spotkałem się w Blogewooshu #6: `https://codeforces.com/blog/entry/62602`.

Treść Zadanie interaktywne. Wyrocznia zna wysokości h_i każdego z n wieżowców ponumerowanych od 1 do n, które wyrażone są liczbami naturalnymi (zachodzi $1 \leqslant h_i \leqslant H$). Możemy zadawać jej pytania postaci „czy $h_i \geqslant x$", gdzie x możemy wybrać w każdym zapytaniu. Naszym zadaniem jest znaleźć maksymalną wysokość wieżowca.

Rozwiązanie Skorzystamy z faktu o maksimach prefiksowych losowego ciągu: przetasujmy ciąg indeksów $1, 2, ..., n$. Teraz nasz ciąg możemy potraktować jako losowy. W tej losowej kolejności będziemy odwiedzać kolejne wieżowce, utrzymując dotychczasową maksymalną wysokość r. Jeżeli $h_i \geqslant r + 1$, to znaczy, że

ten wieżowiec jest wyższy od wszystkich dotychczas napotkanych. Wiadomo, że wydarzy się tak $O(\log n)$ razy. Za każdym razem możemy wyszukać binarnie, jaka jest faktyczna wysokość nowego najwyższego wieżowca w $O(\log H)$. Zatem mamy rozwiązanie w $O(n + \log n \log H)$.

7.3 [NEERC15] J – Jump

Zapożyczone od Marka Sommera

Treść Wyrocznia zna ciąg binarny długości n ($2|n$). Możemy zadawać jej pytania o własnoręcznie stworzone ciągi binarne, otrzymując jedną z trzech odpowiedzi:

- n, jeżeli ciągi są takie same
- $\frac{n}{2}$, jeżeli ciągi zgadzają się na połowie pozycji
- 0, w przeciwnym przypadku

Celem jest znalezienie ukrytego ciągu.

Rozwiązanie Nie wiadomo co zrobić, więc losujemy ciągi i zastanawiamy się z jakim prawdopodobieństwem dostaniemy każdą z odpowiedzi. Oczywistym jest, że rozwiązanie znajdziemy z 2^{-n}, 0 nas nie interesuje, ciekawiej jest z $\frac{n}{2}$. Wykorzystajmy wzór Stirlinga by zaokrąglić $\binom{n}{\frac{n}{2}} = \binom{2m}{m}$:

$$\binom{2m}{m} = \frac{(2m)!}{m!^2} = \frac{(2m)^{2m}\sqrt{4\pi m}}{m^{2m}2\pi m} = \frac{2^n}{\sqrt{\pi \frac{n}{2}}}$$

Zatem z prawdopodobieństwem $\sqrt{\frac{2}{\pi n}}$ dostaniemy odpowiedź $\frac{n}{2}$. Dobrze, to poświęćmy $O(\sqrt{n})$ zapytań na znalezienie naszego pomocnego ciągu S, który ma poszukiwaną odpowiedź. Zauważmy teraz, że jeżeli zbudujemy S' w której znaki na pozycjach i oraz j są odwrócone, i otrzymamy odpowiedź 0, to znaczy, że oba znaki na pozycjach i oraz j albo się zgadzają, albo się nie zgadzają – zawsze oba równocześnie zachowają się tak samo. Nazwijmy takie pozycje współstanowymi.

Powtórzmy to dla większej liczby par – np. dla wszystkich par 0 oraz i. Oznaczmy zbiór indeksów współstanowych z 0 jako X. Wiadomo, że w rozwiązaniu wszystkie pozycje X się nie zgadzają (porównując do S), a pozostałe się zgadzają, albo na odwrót. Wystarczy zatem sprawdzić 2 możliwości - ciąg Z_1, powstały przez odwrócenie ciągu S na pozycjach X, oraz Z_2, który jest odwróconym ciągiem Z_1 (odwróconym, to znaczy o przeciwnych znakach). Sumarycznie zadamy około $\sqrt{\pi \frac{n}{2}} + n + 2$ zapytania.

7.4 *Zadanka do pokminienia*

- [PA 2018, Runda 2A] Heros
- Zapożyczone od Stasia Strzeleckiego:
  - [Codeforces #507B Div 1.] Subway Pursuit
  - [Codeforces #192C Div 1.] Graph Reconstruction
  - [Codeforces #213D Div 1.] GHD
- Zapożyczone od Marka Sommera (`http://mareksom.w.staszic.waw.pl/kolko/2018/kartki/03.10.2018-teoria.pdf`):
  - [PA 2013, Runda 6B] Filary
  - [Kartka] Diamenty
  - [Kartka] Ścieżka o największej sumie
- [Mikołaj Bulge & Ja] Kwadratowy Generator Kongruencyjny (Losujemy liczby generatorem $x' = (x^2 + a) \bmod n$. Parametry a i n. Ile unikatowych wartości otrzymamy dla ziarna x_0? n jest rzędu 10^{14})