Notatki z AiSD. Nr X. 25 maja 2015

Wybór k-tego elementu

IIUWr. II rok informatyki. *Opracował: Krzysztof Loryś*

1 Definicja problemu

Dane: $T[1..n]$ - ciąg elementów należących do zbioru liniowo uporządkowanego
k - liczba z przedziału $\langle 1, n\rangle$.
Wynik: k-ty co do wielkości element ciągu T

Założenie (nie zmniejszające ogólności): wszystkie elementy w T są różne.

Model obliczeń: na elementach ciągu T dokonujemy jedynie porównań.

2 Szczególne przypadki

- $k = 1$.
  Konieczna i wystarczająca liczba porównań $= n - 1$.
- $k = 2$.

  Twierdzenie 1 *W tym przypadku potrzeba i wystarcza $n - 2 + \lceil \log n \rceil$ porównań.*

  Dowód(szkic)
  $\Rightarrow$
  Konstruujemy algorytm, działający w $\lceil \log n \rceil$ rundach: w 1-szej rundzie porównywane są pary elementów $\langle T[2k-1], T[2k]\rangle$ (dla $k = 1, \lfloor \frac{n}{2} \rfloor$). "Zwycięzcy" tych porównań (oraz element $T[n]$ - w przypadku nieparzystego n) przechodzą do następnej rundy. W kolejnych rundach postępujemy analogicznie.
  Oczywiście ostatnia runda wyznaczy element największy w T. Do tej pory algorytm wykona $n - 1$ porównań. Można to łatwo udowodnić, jeśli na działanie tego algorytmu popatrzeć jako na drzewo binarne o n liściach: liście tego drzewa etykietujemy elementami $T[i]$, a każdy wierzchołek wewnętrzny - większą spośród etykiet jego synów. Tak więc wierzchołki wewnętrzne odpowiadają porównaniom wykonanym przez algorytm.
  Znalezienie 2-ego co do wielkości elementu sprowadza się teraz do znalezienia największego elementu pośród tych, które były porównywane z elementem największym. Ponieważ elementów tych jest $\lceil \log n \rceil$, wystarczy teraz $\lceil \log n \rceil - 1$ porównań.
  $\Leftarrow$
  Rozważmy dowolny algorytm $\mathcal{A}$ wyznaczający 2-gi co wielkości element. Zauważmy, że $\mathcal{A}$ wyznacza jednocześnie element największy. Niech bowiem $X = T[j]$ będzie rozwiązaniem podanym przez $\mathcal{A}$. Elementem największym jest ten, z którym X "przegrał" porównanie. Element taki musi istnieć (w przeciwnym razie $\mathcal{A}$ podałby $T[j]$ jako rozwiązanie także dla takich danych, w których $T[j]$ zwiększylibyśmy dowolnie, a pozostałe elementy pozostawilibyśmy bez zmian) i to dokładnie jeden (w przeciwnym razie X nie byłby drugim elementem). Tak więc $\mathcal{A}$ musi wykonać $n - 1$ porównań, by wyznaczyć element największy. Porównania te nie wnoszą żadnej informacji o wzajemnej relacji pomiędzy elementami, które jedynie z nim przegrały porównanie (a X jest największym z tych elementów). Tak więc nasz dowód sprowadza się do pokazania, że w najgorszym przypadku element największy bierze udział w co najmniej $\lceil \log n \rceil$ porównaniach wykonywanych przez $\mathcal{A}$. To będzie treścią zadania na ćwiczenia (rozwiązanie możesz znaleźć w ([7], s.212).

3 Przypadek ogólny

3.1 Algorytm deterministyczny

IDEA Stosujemy metodę Dziel i Zwyciężaj. Rozdzielamy T na dwa podzbiory: U i $V = T \setminus U$, takie że wszystkie elementy U są mniejsze od wszystkich elementów V. Teraz porównanie k z mocą U pozwala określić, w którym ze zbiorów znajduje się szukany element. W ten sposób redukujemy problem do problemu szukania elementu w zbiorze mniejszym.

Procedure *SELECTION(k,T)*

1. **if** $|T|$ małe **then** $sort(T)$ & **return** $(T[k])$
2. $p \leftarrow$ jakiś element z T
3. $U \leftarrow$ elementy T mniejsze od p
4. **if** $k \leq |U|$ **then return** ($SELECTION(k, U)$)
   else return $(SELECTION(k - |U|, T \setminus U))$

UWAGA: W powyższej procedurze zbiory T i U wygodnie jest pamiętać w tablicach.

Aby algorytm był efektywny, musimy zagwarantować, że zbiory U i $T \setminus U$ są istotnie mniej liczne od zbioru T. W tym celu zbiór T dzielimy na 5-cioelementowe grupy. W każdej grupie wybieramy medianę (tj. środkowy element) a następnie rekurencyjnie wybieramy medianę tych median.

function *med(T)*

1. Podziel T na rozłączne podzbiory 5-cioelementowe C_j $(j = 1, \ldots, \lceil |T|/5 \rceil)$
   {jeśli $|T|$ nie dzieli się przez 5, to $C_{\lceil |T|/5 \rceil}$ zawiera mniej niż 5 elementów }
2. **for** $i \leftarrow 1$ **to** $\lceil |T|/5 \rceil$ **do** $s_i \leftarrow adhocmed(C_i)$
3. $S \leftarrow \{s_i \mid i = 1, \ldots, \lceil |T|/5 \rceil\}$
4. **return** $(SELECTION(\lceil \frac{|S|}{2} \rceil, S))$

Twierdzenie 2 *Jeśli w procedurze SELECTION wybór elementu p dokonywany jest funkcją Pseudomed, to procedura SELECTION wyznacza k−ty element ciągu T w czasie $O(n)$.*

Pomijamy dowód poprawności procedury *Selection*. W oszacowaniu jej kosztu kluczowym punktem jest następujący lemat:

Jeśli element p został wybrany funkcją *Pseudomed*, to każdy ze zbiorów U i $T \setminus U$ zawiera nie mniej niż $\frac{3}{10}n - 4$ elementy.

DOWÓD: Istnieje co najmniej (a przy założeniu, że wszystkie elementy w T są różne - dokładnie) $\frac{1}{2}\lceil n/5 \rceil$ grup, których mediana jest nie mniejsza od p. W każdej z tych grup, poza tą, która zawiera największą z median s_i znajdują się co najmniej 3 elementy nie mniejsze od p (takimi są na pewno elementy większe od mediany w danej grupie). Tak więc elementów nie mniejszych od p jest co najmniej $3(\frac{1}{2}\lceil n/5 \rceil - 1)$. W tym wliczony jest p. Ponieważ zakładamy, że wszystkie elementy są różne, więc elementów większych od p jest co najmniej $\frac{3}{10}n - 4$.

W podobny sposób pokazujemy, że co najmniej tyle samo jest elementów mniejszych od p. □

KOSZT Procedury *Selection*

Niech T będzie funkcją ograniczającą z góry ten koszt. Bez zmniejszenia ogólności możemy założyć, że T jest monotoniczną funkcją niemalejącą. Ponieważ koszt wszystkich operacji poza rekurencyjnymi wywołaniami *Selection* można ograniczyć funkcją liniową otrzymujemy następującą nierówność rekurencyjną:

$$T(n) \leq T(\lceil n/5 \rceil) + T(7n/10 + 4) + O(n) \qquad \text{dla odpowiednio dużych } n.$$

Można łatwo sprawdzić, że $T(n)$ jest $O(n)$.

UWAGI:

- Stałą ukrytą w Twierdzeniu 2 pod "dużym O" można oszacować z góry przez 5.43.
- Obecnie najszybszy asymptotycznie algorytm wykonuje $2.9442n + o(n)$ porównań ([3]).
- Dolna granica - każdy algorytm deterministyczny wykonuje co najmniej $2n$ porównań.

3.2 Algorytmy zrandomizowane

3.2.1 Algorytm Hoare'a

W procedurze *Selection* element p wybieramy w sposób losowy. W ten sposób otrzymujemy algorytm o małej średniej liczbie porównań (choć oczywiście nadal liniowej) ([5],[2]).

Zauważ podobieństwo tego algorytmu do zrandomizowanego *Quicksortu*. Podobnie jak w tamtym algorytmie element rozdzielający można wybierać inaczej, np. jako medianę spośród losowej próbki kilku elementów zbioru T ([4]).

3.2.2 Algorytm *LazySelect*

IDEA Ze zbioru S wybieramy losowo próbkę R. Próbka powinna być niezbyt liczna, by można było szybko ją posortować. Znajdujemy w R dwa elementy L i H, takie że z dużym prawdopodobieństwem podzbiór $P \subseteq S$, elementów większych od L i mniejszych od H, jest nieduży oraz szukany element należy do P (możemy go wówczas łatwo znaleźć po posortowaniu P).

Algorytm *LazySelect*

1. Wybierz losowo, niezależnie, z powtórzeniami próbkę R złożoną z $n^{\frac{3}{4}}$ elementów zbioru S.
2. Posortuj R w czasie $O(n^{\frac{3}{4}} \log n)$.
3. $x \leftarrow kn^{-\frac{1}{4}}$; $L \leftarrow R[l]$; $H \leftarrow R[h]$
   gdzie $l = \max\{\lfloor x - \sqrt{n} \rfloor, 1\}$ oraz $h = \min\{\lfloor x + \sqrt{n} \rfloor, n^{\frac{3}{4}}\}$
4. Porównując L i H ze wszystkimi elementami z S oblicz:
   - $r_S(L) = \#\{y \in S \mid y < L\}$
   - $P \leftarrow \{y \in S \mid L \leq k \leq H\}$
5. Sprawdź czy:
   - k-ty element zbioru S znajduje się w P, tj. czy $r_S(L) < k \leq r_S(L) + |P|$,
   - $|P| \leq 4n^{\frac{3}{4}} + 2$.

   Jeśli nie, to powtórz kroki 1-4.
6. Posortuj P.
   return $(P_{(k-r_S(L)+1)})$.

Twierdzenie 3 *Z prawdopodobieństwem* $= 1 - O\left(\frac{1}{\sqrt[4]{n}}\right)$ *LazySelect potrzebuje tylko jednej iteracji kroków* 1-4 *do znalezienia k-tego elementu zbioru S. Tak więc z prawdopodobieństwem* $1 - O\left(\frac{1}{\sqrt[4]{n}}\right)$ *LazySelect zatrzymuje się po wykonaniu 2n+o(n) porównań.*

Dowód tego twierdzenia nie jest trudny, ale wykracza poza zakres wykładu. Można go znaleźć w książce [8] (str. 47-50). Zainteresowani będą mogli go poznać na wykładzie Algorytmy Zrandomizowane.
Teraz ograniczymy się do podania szkicu:

1. pokazujemy, że $kn^{-\frac{1}{4}}$ jest wartością oczekiwaną liczby elementów w R nie większych od szukanego,
2. pokazujemy, że wariancja tej liczby jest mniejsza od $\sqrt{n}$,
3. korzystamy z poniższej Nierówności Czebyszewa.

Twierdzenie 4 (nierówność Czebyszewa) *Jeśli X jest zmienną losową (o wartościach rzeczywistych) o wartości oczekiwanej μ_X i odchyleniu standardowym σ_X. Wówczas*

$$\forall_{t \in \mathcal{R}_+} \quad \mathbf{P}\Big[|X - \mu_X| \geq t\sigma_X\Big] \leq \frac{1}{t^2}$$

Literatura

[1] M.Blum, R.W.Floyd, V.Pratt, R.L.Rivest, R.E.Tarjan, Time bounds for selection, *Journal of Computer and System Sciences*, 7(1973), 448–461.

[2] T.Cormen, C.Leiserson, R.L.Rivest, *Introduction to Algorithms*, The MIT Press, 1990.

[3] D.Dor, U.Zwick, Median selection requires $O(2+\epsilon)n$ comparisions, Proceedings of the 37th FOCS, 1996, 125-134.

[4] R.W.Floyd, R.L.Rivest, Expected time bounds for selection, *Communication of the ACM*, 18(1975), 165–172.

[5] C.A.R.Hoare, Algorithm 63 (partition) and 65 (find), *Communication of the ACM*, 4(1961), 321–322.

[6] R.M.Karp, Probabilistic recurrence relations, w: *Proceedings of the 23rd STOC*, 1991, 190–197.

[7] D.E.Knuth, *The Art of Computer Programming,* vol. 3, Addison-Weslay, 1973.

[8] R.Motwani, P.Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.

3.4.1. Algorytm

Podstawowa idea algorytmu opiera się na *próbkowaniu*, które omówiliśmy w podrozdziale 1.2. Celem jest znalezienie dwóch elementów, które są blisko siebie w posortowanym zbiorze S, a mediana znajduje się między nimi. W szczególności poszukujemy dwóch elementów $d, u \in S$ takich, że

(1) $d \leqslant m \leqslant u$ (mediana znajduje się między d a u);

(2) dla $C = \{s \in S: d \leqslant s \leqslant u\}$, $|C| = o(n/\log n)$ (łączna liczba elementów między d a u jest mała).

Próbkowanie to szybka i efektywna metoda znalezienia dwóch takich elementów.

Twierdzimy, że po zidentyfikowaniu takich dwóch elementów znalezienie mediany w czasie liniowym staje się łatwe. Algorytm oblicza (w czasie liniowym) liczbę ℓ_d elementów z S, które są mniejsze od d, następnie sortuje (w czasie subliniowym, tzn. $o(n)$) zbiór C. Zauważmy, że ponieważ $|C| = o(n/\log n)$, zbiór C można posortować w czasie $o(n)$, używając dowolnego standardowego algorytmu sortowania, który wymaga czasu $O(m \log m)$ do posortowania m elementów. Elementem na pozycji $\lfloor n/2 \rfloor - \ell_d + 1$ w posortowanym zbiorze C jest m, ponieważ mamy dokładnie $\lfloor n/2 \rfloor$ elementów w S, które są mniejsze od tej wartości ($\lfloor n/2 - \ell_d \rfloor$ w zbiorze C i ℓ_d w $S - C$).

Aby znaleźć elementy d i u, próbkujemy z powtórzeniami multizbiór R zawierający $\lfloor n^{3/4} \rfloor$ elementów z S. Przypomnijmy, że próbkowanie z powtórzeniami oznacza, że każdy element w R jest wybrany losowo w sposób jednostajny z S, niezależnie od poprzednich wyborów. Zatem ten sam element z S może pojawić się więcej niż raz w multizbiorze R. Próbkowanie bez powtórzeń daje odrobinę lepsze oszacowania, ale zarówno w analizie, jak i implementacji jest znacznie trudniejsze. Warto dodać, że zakładamy, iż możemy wybrać losowo element z S w stałym czasie.

Ponieważ R jest losową próbką zbioru S, to spodziewamy się, że mediana m zbioru S znajdzie się blisko mediany R. Dlatego wybieramy d i u w R tak, by były w otoczeniu mediany R.

Chcemy, aby algorytm działał *z dużym prawdopodobieństwem*, czyli z prawdopodobieństwem co najmniej $1 - O(1/n^c)$ dla pewnej stałej $c > 0$. Aby zagwarantować, że zbiór C będzie zawierał medianę m z dużym prawdopodobieństwem, ustalamy, że d i u są, odpowiednio, elementem na pozycji $\lfloor n^{3/4}/2 - \sqrt{n} \rfloor$

oraz $\lfloor n^{3/4}/2 + \sqrt{n} \rfloor$ w posortowanym zbiorze R. W ten sposób zbiór C zawiera wszystkie elementy z S znajdujące się wśród $2\sqrt{n}$ próbek z otoczenia mediany R. W analizie wyjaśni się, że rozmiar R oraz d i u zostały tak dobrane, by spełnić dwa warunki: (a) zbiór C jest wystarczająco duży, by zawierał m z dużym prawdopodobieństwem i (b) zbiór C jest na tyle mały, by można było go posortować w czasie subliniowym z dużym prawdopodobieństwem. Formalny opis tej procedury prezentujemy w algorytmie 3.1. W dalszej części, dla wygody będziemy traktować $\sqrt{n}$ oraz $n^{3/4}$ jako liczby całkowite.

Zrandomizowany algorytm wyznaczania mediany

Dane wejściowe: zbiór S złożony z n elementów z dobrze uporządkowanej przestrzeni.

Wynik: mediana zbioru S oznaczona przez m.

1. Wybierz (multi)zbiór R złożony z $\lceil n^{3/4} \rceil$ elementów z S wybranych niezależnie i w sposób jednostajny z powtórzeniami.
2. Sortuj zbiór R.
3. Niech d będzie $(\lfloor \frac{1}{2} n^{3/4} - \sqrt{n} \rfloor)$-tym najmniejszym elementem w posortowanym zbiorze R.
4. Niech u będzie $(\lfloor \frac{1}{2} n^{3/4} + \sqrt{n} \rfloor)$-tym najmniejszym elementem w posortowanym zbiorze R.
5. Porównując każdy element w S z u i d, wyznacz zbiór $C = \{x \in S : d \leqslant x \leqslant u\}$ oraz liczby $\ell_d = |\{x \in S : x < d\}|$ i $\ell_u = |\{x \in S : x > u\}|$.
6. Jeśli $\ell_d > n/2$ lub $\ell_u > n/2$, to PORAŻKA.
7. Jeśli $|C| \leqslant 4n^{3/4}$, to sortuj zbiór C, a w przeciwnym wypadku PORAŻKA.
8. Wypisz element na pozycji $\lfloor n/2 \rfloor - \ell_d + 1$ w posortowanym zbiorze C.

Algorytm 3.1. Zrandomizowany algorytm wyznaczania mediany

3.4.2. Analiza algorytmu

Pamiętając o wcześniejszej dyskusji, najpierw udowodnimy, że niezależnie od losowego kroku wykonanego w trakcie procedury algorytm (a) zawsze się zatrzymuje w czasie liniowym i (b) daje poprawną odpowiedź lub komunikat PORAŻKA.

Twierdzenie 3.9. *Zrandomizowany algorytm wyznaczania mediany zatrzymuje się w liniowym czasie i, jeśli nie zwraca komunikatu PORAŻKA, to podaje poprawną medianę wejściowego zbioru S.*

Dowód. Poprawność wynika stąd, że algorytm tylko wtedy mógłby dać niepoprawną odpowiedź, gdyby nie było mediany w C. Ale wówczas albo $\ell_d > n/2$ albo $\ell_u > n/2$, a krok 6. algorytmu gwarantuje, że w takich przypadkach algorytm zwraca komunikat PORAŻKA. Podobnie, dopóki C jest wystarczająco

mały, to całkowity czas działania jest liniowy względem rozmiaru S. Krok 7. zatem gwarantuje, że algorytm nie będzie działał dłużej niż w czasie liniowym; gdyby sortowanie miało trwać za długo, to algorytm nie sortuje, tylko zwraca komunikat PORAŻKA. □

Pozostała, interesująca część analizy poza twierdzeniem 3.9 polega na oszacowaniu prawdopodobieństwa tego, że algorytm zwróci komunikat PORAŻKA. Oszacujemy to prawdopodobieństwo, identyfikując trzy „złe" zdarzenia i dowodząc, że jeśli żadne z nich nie zachodzi, to algorytm znajduje medianę. W kilku kolejnych lematach oszacujemy prawdopodobieństwo każdego z tych zdarzeń i pokażemy, że ich suma jest tylko rzędu $O(n^{-1/4})$.

Rozważmy zatem trzy następujące zdarzenia:

$\mathcal{E}_1$: $Y_1 = |\{r \in R \mid r \leqslant m\}| < \frac{1}{2}n^{3/4} - \sqrt{n}$,

$\mathcal{E}_2$: $Y_2 = |\{r \in R \mid r \geqslant m\}| < \frac{1}{2}n^{3/4} - \sqrt{n}$,

$\mathcal{E}_3$: $|C| > 4n^{3/4}$.

Lemat 3.10. *Zrandomizowany algorytm wyznaczania mediany nie działa wtedy i tylko wtedy, gdy zachodzi co najmniej jedno ze zdarzeń $\mathcal{E}_1$, $\mathcal{E}_2$ lub $\mathcal{E}_3$.*

Dowód. Porażka w kroku 7. algorytmu jest równoważna zdarzeniu $\mathcal{E}_3$. Porażka w kroku 6. algorytmu ma miejsce wtedy i tylko wtedy, gdy $\ell_d > n/2$ lub $\ell_u > n/2$. Lecz aby $\ell_d > n/2$, to element na pozycji $(\frac{1}{2}n^{3/4} - \sqrt{n})$ w posortowanym zbiorze R musi być większy niż m, a to jest równoważne zdarzeniu $\mathcal{E}_1$. Podobnie $\ell_u > n/2$ jest równoważne zdarzeniu $\mathcal{E}_2$. □

Lemat 3.11.

$$\Pr(\mathcal{E}_1) \leqslant \frac{1}{4}n^{-1/4}.$$

Dowód. Zdefiniujmy zmienną losową X_i następująco

$$X_i = \begin{cases} 1 & \text{jeśli } i\text{-ta próba jest mniejsza od mediany lub jej równa} \\ 0 & \text{w przeciwnym wypadku.} \end{cases}$$

Zmienne X_i są niezależne, gdyż próbkowanie odbyło się z powtórzeniami. Ponieważ mamy $(n-1)/2+1$ elementów w S, które są mniejsze od mediany lub jej równe medianie, to prawdopodobieństwo, że losowo wybrany element z S jest mniejszy od mediany lub jej równy można wyrazić następująco

$$\Pr(X_i = 1) = \frac{(n-1)/2+1}{n} = \frac{1}{2} + \frac{1}{2n}.$$

Zdarzenie $\mathcal{E}_1$ jest równoważne nierówności

$$Y_1 = \sum_{i=1}^{n^{3/4}} X_i < \frac{1}{2}n^{3/4} - \sqrt{n}.$$

Ponieważ Y_1 jest sumą prób Bernoulliego, to ma rozkład dwumianowy z parametrami $n^{3/4}$ i $1/2+1/2n$. Stąd, korzystając z wyniku z punktu 3.2.1, otrzymujemy

$$\begin{aligned}\mathbf{Var}[Y_1] &= n^{3/4}\left(\frac{1}{2}+\frac{1}{2n}\right)\left(\frac{1}{2}-\frac{1}{2n}\right)\\ &= \frac{1}{4}n^{3/4}-\frac{1}{4n^{5/4}}\\ &< \frac{1}{4}n^{3/4}.\end{aligned}$$

Stosując nierówność Czebyszewa, mamy

$$\begin{aligned}\Pr(\mathcal{E}_1) &= \Pr\left(Y_1 < \frac{1}{2}n^{3/4}-\sqrt{n}\right)\\ &\leqslant \Pr(|Y_1-\mathbf{E}[Y_1]| > \sqrt{n})\\ &\leqslant \frac{\mathbf{Var}[Y_1]}{n}\\ &< \frac{\frac{1}{4}n^{3/4}}{n} = \frac{1}{4}n^{-1/4}.\end{aligned}$$

□

W podobny sposób otrzymujemy takie samo oszacowanie prawdopodobieństwa zdarzenia $\mathcal{E}_2$. Teraz oszacujemy prawdopodobieństwo trzeciego złego zdarzenia, $\mathcal{E}_3$.

Lemat 3.12.

$$\Pr(\mathcal{E}_3) \leqslant \frac{1}{2}n^{-1/4}.$$

Dowód. Jeśli zachodzi $\mathcal{E}_3$, czyli $|C| > 4n^{3/4}$, to zachodzi jedno z dwóch następujących zdarzeń:

$\mathcal{E}_{3,1}$: co najmniej $2n^{3/4}$ elementów z C jest większych od mediany,

$\mathcal{E}_{3,2}$: co najmniej $2n^{3/4}$ elementów z C jest mniejszych od mediany.

Oszacujmy prawdopodobieństwo zajścia pierwszego zdarzenia, a drugie, przez symetrię będzie takie samo. Jeśli mamy co najmniej $2n^{3/4}$ elementów w C powyżej mediany, to numer pozycji elementu u w posortowanym zbiorze S jest równy co najmniej $\frac{1}{2}n+2n^{3/4}$, a zatem zbiór R ma co najmniej $\frac{1}{2}n^{3/4}-\sqrt{n}$ próbek wśród $\frac{1}{2}n-2n^{3/4}$ największych elementów w S.

Niech X będzie liczbą próbek wśród $\frac{1}{2}n-2n^{3/4}$ największych elementów w S oraz niech $X=\sum_{i=1}^{n^{3/4}} X_i$, gdzie

$$X_i = \begin{cases}1 & \text{jeśli } i\text{-ta próba jest wśród } \frac{1}{2}n-2n^{3/4} \text{ największych elementów w } S\\ 0 & \text{w przeciwnym wypadku.}\end{cases}$$

Ponownie, X jest zmienną losową o rozkładzie dwumianowym i

$$\mathbf{E}[X] = \frac{1}{2}n^{3/4} - 2\sqrt{n}$$

oraz

$$\mathbf{Var}[X] = n^{3/4}\left(\frac{1}{2} - 2n^{-1/4}\right)\left(\frac{1}{2} + 2n^{-1/4}\right) = \frac{1}{4}n^{3/4} - 4n^{1/4} < \frac{1}{4}n^{3/4}.$$

Stosując nierówność Czebyszewa, otrzymujemy

$$\Pr(\mathcal{E}_{3,1}) = \Pr(X \geqslant \frac{1}{2}n^{3/4} - \sqrt{n}) \tag{3.3}$$

$$\leqslant \Pr(|X - \mathbf{E}[X]| \geqslant \sqrt{n}) \leqslant \frac{\mathbf{Var}[X]}{n} < \frac{\frac{1}{4}n^{3/4}}{n} = \frac{1}{4}n^{-1/4}. \tag{3.4}$$

Podobnie

$$\Pr(\mathcal{E}_{3,2}) \leqslant \frac{1}{4}n^{-1/4}$$

oraz

$$\Pr(\mathcal{E}_3) \leqslant \Pr(\mathcal{E}_{3,1}) + \Pr(\mathcal{E}_{3,2}) \leqslant \frac{1}{2}n^{-1/4}.$$

Łącząc wyprowadzone oszacowania, otrzymujemy, że prawdopodobieństw tego, iż algorytm zwróci komunikat PORAŻKA, jest ograniczone przez

$$\Pr(\mathcal{E}_1) + \Pr(\mathcal{E}_2) + \Pr(\mathcal{E}_3) \leqslant n^{-1/4}.$$

Prowadzi to do następującego twierdzenia.

Twierdzenie 3.13. *Prawdopodobieństwo, że zrandomizowany algorytm znajd wania mediany nie znajdzie rozwiązania, jest nie większe niż* $n^{-1/4}$.

Powtarzając algorytm 3.1 tak długo, aż znajdzie medianę, możemy otrzym algorytm iteracyjny, który zawsze znajduje poprawne rozwiązanie, ale jego cz działania jest losowy. Próbki wybierane w kolejnych iteracjach algorytmu są n zależne, czyli sukces w każdej iteracji jest niezależny od innych iteracji i st liczba iteracji do pierwszego sukcesu jest zmienną losową o rozkładzie ge metrycznym. Jako zadanie proponujemy Czytelnikowi udowodnić, że ta wer algorytmu (który działa do chwilii znalezienia odpowiedzi) ma również linio oczekiwany czas działania.

Algorytmy zrandomizowane, które mogą dać niepoprawną odpowiedź, na wamy algorytmami *Monte Carlo*. Czas działania algorytmu Monte Carlo zwy

nie zależy od losowych kroków, które on wykonuje. Na przykład, jak pokazaliśmy w twierdzeniu 3.9, zrandomizowany algorytm znajdowania mediany zawsze kończy się w czasie liniowym, niezależnie od wyborów losowych.

Algorytm zrandomizowany, który zawsze zwraca poprawną odpowiedź, nazywamy algorytmem *Las Vegas*. Widzieliśmy, że zrandomizowany algorytm Monte Carlo znajdowania mediany można przekształcić w algorytm Las Vegas, powtarzając go do chwili, aż odniesie sukces. Podkreślmy jeszcze raz: przekształcenie go w algorytm Las Vegas oznacza, że czas działania jest zmienną losową. Jednakże jej wartość oczekiwana pozostaje liniowa.

3.5. Zadania

Zadanie 3.1. Niech X będzie liczbą całkowitą wybraną losowo w sposób jednostajny z przedziału $[1, n]$. Znajdź $\mathbf{Var}[X]$.

Zadanie 3.2. Niech X będzie liczbą całkowitą wybraną losowo w sposób jednostajny z przedziału $[-k, k]$. Znajdź $\mathbf{Var}[X]$.

Zadanie 3.3. Rzucamy standardową kostką 100 razy. Niech X będzie sumą liczb, które pojawiły się w tych 100 rzutach. Skorzystaj z nierówności Czebyszewa, by oszacować $\Pr(|X - 350| \geqslant 50)$.

Zadanie 3.4. Udowodnij, że dla dowolnej liczby rzeczywistej c i dyskretnej zmiennej losowej X, $\mathbf{Var}[cX] = c^2\mathbf{Var}[X]$.

Zadanie 3.5. Dla danych dwóch zmiennych losowych X i Y, z liniowości wartości oczekiwanej mamy $\mathbf{E}[X - Y] = \mathbf{E}[X] - \mathbf{E}[Y]$. Udowodnij, że jeśli X i Y są niezależne, to $\mathbf{Var}[X - Y] = \mathbf{Var}[X] + \mathbf{Var}[Y]$.

Zadanie 3.6. Rzucamy monetą, dla której prawdopodobieństwo pojawienia się orła, niezależnie w każdym rzucie, wynosi p. Jaka jest wariancja liczby rzutów do chwili otrzymania k-tego orła?

Zadanie 3.7. W prostym modelu giełdy przyjmuje się, że każdego dnia akcja w cenie q wzrośnie o czynnik $r > 1$ do qr z prawdopodobieństwem p i spadnie do q/r z prawdopodobieństwem $1 - p$. Zakładając, że zaczynamy z akcją w cenie 1, znajdź wzór na wartość oczekiwaną i wariancję ceny akcji po d dniach.

Zadanie 3.8. Rozpatrzmy algorytm, który na wejściu ma ciąg n bitów. Jeśli bity na wejściu są wybrane niezależnie, w sposób jednostajny, to wartość oczekiwana czasu działania wynosi $O(n^2)$. Co możemy wywnioskować z nierówności Markowa na temat czasu działania tego algorytmu w najgorszym przypadku?