Algorytmy i programowanie zajęcia 19 i 20

Marcin Żurowski

18 maja 2020

1 Zadania - procedury

Termin wykonania poniższych zadań (23:59 24 maja 2020).

1. Wyznaczyć zależność rekurencyjną określającą liczbę spójnych obszarów, na które dzieli płaszczyznę n prostych, z których żadne dwie nie są równoległe i żadne trzy nie przecinają się w jednym punkcie.

2. Znaleźć zależność rekurencyjną określającą liczbę różnych sposobów wejścia po schodach zbudowanych z n stopni, jeśli w każdym kroku można pokonać jedne lub dwa stopnie.
   Rozwiązanie:

   - należy znaleźć liczbę rozwiązań dla 1 stopnia, czyli $a_1 = 1$ pierwszy krok to krok o jeden stopień (nie mamy innego wyboru)
   - dla dwóch stopni $a_2 = 2$ mamy dwa scenariusze:
     - pierwszy krok robimy o jeden stopień, i pozostaje nam zrobić drugi krok o jeden stopień i przeszliśmy całe schody
     - pierwszy krok dwa stopnie i przeszliśmy całe schody
   - dla trzech stopni rozważamy znów dwa scenariusze:
     - pierwszy krok idziemy o jeden stopień, i pozostaje nam przypadek z dwoma stopniami
     - pierwszy krok idziemy o dwa stopnie i pozostaje nam przypadek z jednym stopniem
   - ...

3. Podać definicję funkcji wyznaczającej najmniejszą wartość tablicy `A`, przy czym chcemy, aby funkcja sprawdzała dowolny podciąg $[d, g]$ kolejnych elementów tablicy, gdzie $1 \leqslant d \leqslant g \leqslant n$.
   Uwaga! Funkcję należy tak napisać, aby obsługiwała tablice indeksowane od 0, natomiast dla potrzeb zadania z **adjule** należy ją wywołać w następujący sposób:
   `min(A, n, d - 1, g - 1);`
   adjule zadanie: APR_038_MIN_PRZEDZIAL

4. W tablicy `A` zawierającej liczby znajdują się dwa uporządkowane (niemalejąco) ciągi na miejscach od indeksu p do indeksu q i od indeksu $q+1$ do indeksu r, gdzie $p \leqslant q < r$. Zapisać definicję procedury, która scali te dwa ciągi w jeden niemalejący ciąg i umieści go w tablicy `A` na miejscach od p do r.
adjule zadanie: APR_039_SCAL_PRZEDZIAL

5. Zadana jest tablica `A[p..r]` zawierająca liczby. Napisać definicję procedury, która dziel tę tablicę (poprzez przestawienie jej elementów) na dwie tablice `A[p..q - 1]` i `A[q + 1..r]` w ten sposób, że każdy element z pierwszej podtablicy jest nie większy niż element `A[q]`, który z kolei jest mniejszy od każdego elementu z drugiej podtablicy. Obliczenie indeksu q ma stanowić część tej procedury podziału.
adjule zadanie: APR_040_PODZIEL

2 Złożoność obliczeniowa

2.1 Zadanie

Niech będą dane dwie liczby $a \in \mathbb{R}$ i $n \in \mathbb{N}$. Oblicz a^n.

2.2 Odpowiedź

Zadanie możemy rozwiązać na kilka sposobów oto przykładowe algorytmy:

1.
```
POTEGA1(a, n)
    pot = 1
    for i = 1 to n
       pot = pot * a
    return pot
```

2.
```
POTEGA2(a, n)
    i = n
    pot = 1
    pod = a
    while i =/ 0
       if i MOD 2 =/ 0
          pot = pot * pod
       pod = pod * pod
       i = i DIV 2
    return pot
```

3.
```
POTEGA3(a, n)
    if n = 0
       return 1
    else
```

```
return a * POTEGA(a, n - 1)
```

Który z podanych algorytmów jest najlepszy? (w sensie zużycia najmniej czasu)

Dla takich samych danych wejściowych możemy sprawdzić ile wykonuje się *operacji znaczących.*

Operacją znaczącą możemy nazwać dowolną operację którą naszym zdanie zużywa najwięcej czasu procesora (pozostałe operacje zużywają mniej czasu). Dla powyższych przykładów taką operacją może być mnożenie.

Niech T będzie funkcją określającą liczbę operacji znaczących.

ad 1 dla algorytmu `POTEGA1` funkcja czasu $T_1(n) = n$ ponieważ mnożenie występuje tylko w pętli `for` która ma dokładnie n iteracji. Prosze zauważyć, że wartość funkcji T_1 jest zależna od wartości n.

ad 2 algorytm `POTEGA2` wykonuje się szybciej niż algorytm `POTEGA1`. W algorytmie `POTEGA2` wykorzystuje się następującą zasadę: aby policzyć a^b należy znać wartość $a^{\lfloor \frac{n}{2} \rfloor}$ i wykonać jedną operację (kiedy $\lfloor \frac{n}{2} \rfloor$ jest parzyste), lub dwie operacje (kiedy $\lfloor \frac{n}{2} \rfloor$ jest nieparzyste). Warto zauważyć że wykładnik takiej potęgi przy każdej takiej operacji zmniejsza się o połowę, więc dla pesymistycznego przypadku takich iteracji jest $2 \log_2 n$, stąd $T_2(n) = 2 \log_2 n$. (ponieważ w informatyce często używa się logarytmu o podstawie 2, więc przyjął się następujący skrót: $\log_2 n = \lg n$)

ad 3 dla algorytmu `POTEGA3` liczbę operacji znaczących możemy obliczyć w następujący sposób:

$$\begin{aligned} T_3(n) &= a \cdot T_3(n-1) \\ &= a \cdot a \cdot T_3(n-2) \\ &\dots \\ &= a \cdot a \cdot \ldots \cdot a \cdot T_3(2) \\ &= a \cdot a \cdot \ldots \cdot a \cdot a \cdot T_3(1) \end{aligned}$$

Ponieważ $T_3(1) = 0$, więc algorytm wykonał dokładnie n mnożeń, stąd $T_3(n) = a * n$.

Wydaje się, że algorytm `POTEGA2` wykonuje się najszybciej. Aby ten fakt sprawdzić potrzebujemy pewnego narzędzia. Najpierw rozważmy następujący ciąg pewnych znanych ciągów:

$$1, \lg n, \ldots, \sqrt[4]{n}, \sqrt[3]{n}, \sqrt{n}, n, n \lg n, n\sqrt{n}, n^2, n^3, n^4, \ldots, 2^n, n!, n^n \qquad (1)$$

każdy z tych ciągów jest O od wszystkich ciągów na prawo od niego. Dowód powyższego faktu możemy znaleźć w [3]. Definicja notacji O jest następująca:

$$O(g(n)) = \{f(n) : \exists_{c>0} \exists_{n_0 \in \mathbb{N}} \forall_{n \geqslant n_0} 0 \leqslant f(n) \leqslant c \cdot g(n)\} \qquad (2)$$

Aby pokazać, że algorytm `POTEGA2` jest najszybszy z pośród trzech przedstawionych algorytmów pokażemy, że $T_2(n) = 2 \lg n = O(\lg n)$.

Aby wykazać powyższy fakt musimy zgodnie z (2) okazać:

1. $0 \leqslant T_2(n)$
2. znaleźć c i n_0 dla których $T_2(n) \leqslant c \cdot \lg n$ dla wszystki $n \geqslant n_0$

ad 1 $2 \lg n \geqslant 0$ dla $n_0 = 1$ zgodnie z definicją funkcji logarytm

ad 2 ponieważ $2 \lg n \leqslant c \lg n$, więc $c = 2$ i nierówność jest prawdziwa dla $n_0 = 1$

Wykazaliśmy więc, że $T_2(n) = O(\lg n)$. Analogicznie możemy wykazać, że $T_1(n) = T_3(n) = O(n)$.

Patrząc na powyższe oszacowania i na ciąg (1) zauważymy że algorytm `POTEGA2` wykonuje się w czasie logarytmicznym względem wejścia i jest szybszy od algorytmów `POTEGA1` i `POTEGA3`, które wykonują się w czasie liniowym względem wejścia.

Wszystkie problemy dla których możemy znaleźć algorytm o złożoności wielomianowej nazywamy problemami łatwymi lub bardziej formalnie należącymi do klasy P. Pozostałe problemy (te dla których istnieje rozwiązujący je algorytm) nazywamy problemami trudnymi, formalnie należącymi do klasy NP. Przykładem tego ostatniego może być problem hasła.

Problem hasła. Chcemy włamać się na konto kolegi znając jego login i nie znając hasła.

Rozwiązać ten problem możemy za pomocą prostego algorytmu, który działa następująco:

Zakładamy, że do hasła zostały użyte znaki z zakresu $32 - 126$ ASCII, czyli 95 znaków. Algorytm najpierw sprawdza wszystkie hasła o długości 1, następnie o długości 2 itd. Poniższa tabela pokazuje ile haseł należy sprawdzić, oraz ile to trwa zakładając, że w ciągu 1 sekundy procesorem o częstotliwości $2.2GHz$ jesteśmy w stanie sprawdzić 2362232013 hasła.

długość hasła	liczba haseł	czas sprawdzenia
1	95	0 s
2	9 025	0 s
3	857 375	0 s
4	81 450 625	0 s
5	7 737 809 375	4 s
6	735 091 890 625	5 m
7	69 833 729 609 375	8 h
8	6 634 204 312 890 620	32 d
9	630 249 409 724 609 000	102 m
10	59 873 693 923 837 900 000	814 y

Warto zauważyć, że przy haśle składającym się z 10 znaków właścicielowi konta będzie wszystko jedno czy się włamiemy na jego konto ☺. Oprócz notacji O mamy jeszcze dwie notacje używane często do określania złożoności obliczeniowej, a mianowicie notacja Ω i notacja Θ. Poniżej definicje tych notacji

$$\Omega(g(n)) = \{f(n) : \exists_{c>0} \exists_{n_0 \in \mathbb{N}} \forall_{n \geqslant n_0} 0 \leqslant c \cdot g(n) \leqslant f(n)\} \tag{3}$$

$$\Theta(g(n)) = \{f(n) : \exists_{c_1>0} \exists_{c_2>0} \exists_{n_0 \in \mathbb{N}} \forall_{n \geqslant n_0} 0 \leqslant c_1 \cdot g(n) \leqslant f(n) \leqslant c_2 \cdot g(n)\} \tag{4}$$

Warto zauważyć, że jeżeli ciąg $f(n) = \Omega(g(n))$ i $f(n) = O(g(n))$ to $f(n) = \Theta(g(n))$

Literatura

[1] B. Kernighan, D. Ritchie (2007). Język ANSI C. Wydawnictwo Naukowo-Techniczne.

[2] E. Palka (2012). Elementy algorytmiki dla początkujących. Wydawnictwo Naukowe UAM. (http://lib.amu.edu.pl/ksiazki-elektroniczne/)

[3] K. A. Ross, C. R. B. Wright (1996). Matematyka dyskretna. Wydawnictwo Naukowe PWN.