

Wstęp do programowania

Złożoność obliczeniowa, poprawność programów

Paweł Daniluk

Wydział Fizyki

Jesień 2013



Złożoność obliczeniowa

Problem

Ile czasu zajmuje wykonanie programu/algorytmu?

Czas zależy od

- zastosowanego algorytmu
- jakości implementacji
- danych wejściowych
- języka programowania, kompilatora/interpretera
- wydajności maszyny

Co właściwie należy mierzyć, aby miarodajnie porównywać algorytmy?

Złożoność obliczeniowa

Założmy, że maszyna wykonuje operacje O_1, O_2, \dots, O_k , a t_i to czas wykonania pojedynczej operacji. Czas pracy programu na danych D :

$$T(D) = N_1(D)t_1 + N_2(D)t_2 + \dots + N_k(D)t_k$$

Warto przyjąć pewne uproszczenia.

Niech T będzie funkcją rozmiaru danych ($n = s(D)$). Mamy wtedy różne rodzaje złożoności obliczeniowej ($D(n)$ – zbiór zestawów danych wejściowych o rozmiarze n):

- pesymistyczna – $T(n) = \max_{D \in D(n)} T(D)$
- średnia (oczekiwana) – $T(n) = \frac{1}{|D(n)|} \sum_{D \in D(n)} T(D)$

Notacja asymptotyczna

Wydajność procesorów podwaja się co półtora roku (prawo Moore'a).
Czasy wykonywania pojedynczych operacji nie mają istotnego znaczenia.

$$f(n) = O(g(n)) \iff \exists_{c>0} f(n) \leq cg(n)$$

Przykłady

$$2n = O(n)$$

$$3n^2 + n + 3 = O(n^2)$$

$$\log_k n = O(\log_2 n)$$

Notacja asymptotyczna – caveat emptor

Dla małych n algorytm asymptotycznie bardziej kosztowny może być wydajniejszy.

Przykład

$$n + 20 = O(n^2)$$

ale:

$$2 + 20 > 2^2$$

Przykład – wyszukiwanie

Liniowe

Należy przebiec kolejno wszystkie elementy, aż do znalezienia wymaganego.

- złożoność pesymistyczna $n = O(n)$ – ostatni element jest poszukiwanym
- złożoność średnia $\frac{n}{2} = O(n)$ – przy założeniu, że poszukiwany element może być z jednakowym prawdopodobieństwem na każdej pozycji

Binarne

Złożoność dana równaniem rekurencyjnym:

$$T(n) = 1 + T(n/2)$$

$$T(n) = O(\log n)$$

Przykład – sortowanie

BubbleSort



QuickSort



HeapSort



Poprawność programów

Program jest częściowo poprawny, jeżeli daje poprawne wyniki, o ile się kończy.

Program jest w pełni poprawny, jeżeli jest częściowo poprawny i obliczenie kończy się dla dowolnych danych.

Uwaga

Program

```
while True :  
    skip
```

jest częściowo poprawny niezależnie od rozwiązywanego problemu.

Dowodzenie/badanie poprawności

Sposoby

- dowód poprawności algorytmu
 - formalny dowód poprawności programu
 - testy (eksperyment)
-
- “To widać!”
 - poprawność algorytmu wynika wprost (niemal wprost) z twierdzenia matematycznego
 - metoda z niezmiennikiem i warunkiem końca pętli
 - logika Hoare'a

Przykład – liczby Fibbonaciego

```
def fib(n):  
    if n<=1:  
        return n  
  
    return fib(n-1)+fib(n-2)
```

Przykład – liczby Fibbonaciego

```
def fib(n):  
    if n<=1:  
        return n  
  
    return fib(n-1)+fib(n-2)
```

To widać!

Przykład – problem plecakowy

Złodziej dysponuje workiem, który może pomieścić przedmioty o łącznej wadze w . W jaki sposób może zmaksymalizować wartość łupu, jeżeli ma do wyboru n rodzajów przedmiotów o wagach i wartościach zadanych ciągami w_i, v_i . Liczba dostępnych przedmiotów każdego rodzaju jest nieograniczona.

$K(w)$ – maksymalny łup osiągalny przy pomocy plecaka o pojemności w

Twierdzenie

$$K(w) = \max_{i:w_i \leq w} K(w - w_i) + v_i$$

Program obliczający $K(w)$ dla kolejnych liczb naturalnych jest banalnie łatwo napisać.

Przykład – flaga polska

```
l=[random.randint(0,1) for i in range(10)]  
  
a=0  
b=len(l)-1  
  
while a<b:  
    if l[a]==0:  
        a+=1  
    elif l[b]==1:  
        b-=1  
    else:  
        l[a], l[b]=l[b], l[a]
```

Niezmiennik pętli (N)

$$\forall_{i < a} l[i] = 0 \wedge \forall_{i > b} l[i] = 1 \wedge a \leq b$$

$$N \wedge \neg(a < b) \implies \text{tablica uporządkowana}$$

Przykład – flaga polska

```
l=[random.randint(0,1) for i in range(10)]  
  
a=0  
b=len(l)-1  
  
while a<b:  
    if l[a]==0:  
        a+=1  
    elif l[b]==1:  
        b-=1  
    else:  
        l[a], l[b]=l[b], l[a]
```

Niezmiennik pętli (N)

$$\forall_{i < a} l[i] = 0 \wedge \forall_{i > b} l[i] = 1 \wedge a \leq b$$

$$N \wedge \neg(a < b) \implies \text{tablica uporządkowana}$$

To jest dowód częściowej poprawności.

Logika Hoare'a

$$\{P\} C \{Q\}$$

- P – warunek wstępny
- Q – warunek końcowy
- C – instrukcja

Powyższy zapis jest formułą logiczną oznaczającą, że jeżeli przed wykonaniem instrukcji C jest spełniony warunek P , to po jej wykonaniu spełniony jest warunek Q .

Wnioskowanie

przesłanki oddzielone przecinkami
wniosek

Instrukcja pusta

$$\overline{\{P\} \text{ skip } \{P\}}$$

Przypisanie

$$\overline{\{P[E/x]\} x \leftarrow E \{P\}}$$

$P[E/x]$ oznacza warunek P , w którym wszystkie wystąpienia x zostały zastąpione przez E .

Przykłady

- $\{x + 1 = 24\} y \leftarrow x + 1 \{y = 24\}$
- $\{x + 1 \leq N\} x \leftarrow x + 1 \{x \leq N\}$

Instrukcja złożona

$$\frac{\{P\} S \{Q\}, \{Q\} T \{R\}}{\{P\} S; T \{R\}}$$

Instrukcja warunkowa

$$\frac{\{B \wedge P\} S \{Q\}, \{\neg B \wedge P\} T \{Q\}}{\{P\} \text{ if } B \text{ then } S \text{ else } T \{Q\}}$$

Pętla

$$\frac{\{P \wedge B\} S \{P\}}{\{P\} \text{ while } B \text{ do } S \{\neg B \wedge P\}}$$

Ta reguła pozwala wnioskować wyłącznie poprawność częściową.

Pętla – pełna poprawność

$$\frac{\begin{array}{l} < \text{ jest dobrze ufundowanym porządkiem na } D, \\ P \wedge B \implies t \in D \wedge t \text{ nie jest elementem minimalnym,} \\ [P \wedge B \wedge t = z] S [P \wedge t < z] \end{array}}{[P] \text{ while } B \text{ do } S [\neg B \wedge P]}$$

t jest licznikiem pętli. W przykładzie z flagą polską można wziąć $t = b - a$.

Problem stopu

Rozstrzygnąć czy program P zatrzymuje się dla dowolnych danych wejściowych.

Problem stopu

Rozstrzygnąć czy program P zatrzymuje się dla dowolnych danych wejściowych.

Nie istnieje algorytm rozwiązujący problem stopu.

Problem stopu

Dowód

Założmy, że istnieje w pełni poprawny program S , który dla dowolnego programu P i danych D rozstrzyga, czy P uruchomiony na danych D zatrzymuje się.

Weźmy program T :

$T(P)$

```
1  if  $S(P, P) = true$ 
2    then loop
3    else stop
```

Jaki jest rezultat uruchomienia $T(T)$?

Klasy złożoności obliczeniowej

Dwa rodzaje problemów

- Problem decyzyjny – odpowiedź na pytanie “Czy?”
- Problem optymalizacyjny – odpowiedź na pytanie “Ile?”

W analizie złożoności obliczeniowej zazwyczaj wystarczy zajmować się problemami decyzyjnymi.

Klasa złożoności wielomianowej (P)

Istnieje algorytm o złożoności obliczeniowej $O(n^k)$ dla pewnego k .

Klasa niedeterministycznej złożoności wielomianowej (NP)

Istnieje algorytm o złożoności obliczeniowej $O(n^k)$ dla pewnego k , który wymaga podania dodatkowej informacji z wyroczni. W praktyce: da się w czasie wielomianowym sprawdzić, czy rozwiązanie jest poprawne, o ile się je zna.

http://bioexploratorium.pl/wiki/Wstę_p_do_programowania_2013z