Algorytmy i struktury danych

Algorytmy elementarne I

Prof. dr hab. Stanisław Gawiejnowicz

Wydział Biologii
Uniwersytet im. Adama Mickiewicza w Poznaniu

e-mail: stgawiej@amu.edu.pl

Wykład nr 2: Algorytmy elementarne I

- Algorytmy wykorzystujące jedynie zmienne
- Algorytmy wykorzystujące tablice 1D
- Literatura

Algorytmy wykorzystujące jedynie zmienne

- Istnieje dość duża grupa problemów, które można rozwiązać za pomocą algorytmów wykorzystujących tylko zmienne
- Przykładami tego typu algorytmów są algorytmy obliczania pól figur geometrycznych
- Starożytny matematyk grecki Heron (10,70) jest uważany za autora wzoru Herona na pole trójkąta: $S_\Delta = \sqrt{p*(p-a)*(p-b)*(p-c)}$, gdzie $p = \frac{a+b+c}{2}$

WZOR_HERONA_1(a,b,c)

```
p = (a+b+c)/2
s = sqrt(p*(p-a)*(p-b)*(p-c))
return s
```

Algorytmy wykorzystujące jedynie zmienne

WZOR_HERONA_2(a,b,c)

```
s = 0.25*sqrt((a+b+c )*(b+c)*(a+c)*(a+b))
return s
```

- Ten algorytm wykonuje tyle samo operacji, co poprzedni
- Poprzedni algorytm używał 2 zmiennych, ten – tylko 1 zmiennej

Algorytmy wykorzystujące jedynie zmienne

- Algorytmy obliczania objętości brył geometrycznych także wykorzystują jedynie zmienne

OBJETOSC_WALCA(r,h)

$V \leftarrow \pi * r * r * h$

return V

- Liczba π to wartość ilorazu długości okręgu i jego średnicy

Liczba π (1/2)

Wybrane własności liczby π (1/2)

- $\pi \approx \frac{22}{7}$ (Archimedes, III w. p.n.e.)
- $\pi \approx \frac{355}{113}$ (Zu Chongzhi, 500)
- $\pi \approx 3.14159265358979323846264338327950288\ldots$
  ludolfina (Ludolph van Ceulen, 1610)
- *Kuć i orać w dzień zawzięcie,*
  Bo plonów niema bez trudu!
  Złocisty szczęścia okręcie,
  Kołyszesz...
  Kuć! My nie czekajmy cudu!
  Robota to potęga ludu!
  K. Cwojdziński (1930)

Liczba π (2/2)

Wybrane własności liczby π (2/2)

- wzór Eulera $e^{i\pi} + 1 = 0$
- wzór Stirlinga $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

Algorytmy wykorzystujące jedynie zmienne

- Wiele zagadnień kalendarzowych także można rozwiązać za pomocą algorytmów wykorzystujących jedynie zmienne

ROK_PRZESTEPNY(n)

if ((n **mod** 4 == 0) **and** (n **mod** 100 $\neq$ 0)) **or** (n **mod** 400 == 0)
then return true
else return false

Algorytmy wykorzystujące jedynie zmienne

- Podane poprzednio algorytmy wykonywały stałą liczbę operacji elementarnych, tzn. niezależną od rozmiaru danych wejściowych
- Inną grupę stanowią algorytmy wykonujące liczbę operacji proporcjonalną do rozmiaru danych wejściowych
- Przykładem takiego problemu jest obliczanie reszty z dzielenia (operacja **mod**)

MOD(a,b)

```
tmp = 1
while (tmp + 1) * b ≤ a do
    tmp = tmp + 1
return a − tmp * b
```

Algorytmy wykorzystujące jedynie zmienne

- Addytywny algorytm Euklidesa, uważany za jeden z najstarszych algorytmów, także wykorzystuje jedynie zmienne

ADDYTYWNY_NWD(a,b)

while $a \neq b$ **do**
    if $a \geq b$ **then** $a = a - b$
    else $b = b - a$
return a

Algorytmy wykorzystujące jedynie zmienne

- Kolejnym przykładem problemu który można rozwiązać za pomocą algorytmu wykorzystującego jedynie zmienne jest obliczanie potęgi

Przykład

- Obliczenie x^{16} wymaga 15 mnożeń, jeśli stosujemy algorytm oparty na definicji
- Jeśli zauważymy, że $x^{16} = (((x^2)^2)^2)^2$, to liczba mnożeń będzie równa 4
- Podobnie, $x^{27} = x^{16} * x^{11} = x^{16} * x^8 * x^3 = x^{16} * x^8 * x^2 * x$, co wymaga 13 mnożeń zamiast 26

Algorytmy wykorzystujące jedynie zmienne

POTEGA_BINARNA(x,n)

```
k = n
b = 1
c = x
while k ≠ 0 do
    if k mod 2 == 0
    then k = k div 2
        c = c * c
    else k = k − 1
        b = b * c
return b
```

Algorytmy wykorzystujące jedynie zmienne

- Kolejnym przykładem problemu z tej grupy jest problem obliczania silni

SILNIA(n)

```
wart = 1
for i = 2 to n do
    wart = wart * i
return wart
```

Algorytmy wykorzystujące jedynie zmienne

- Leonardo z Pizy (1170-1250), inaczej nazywany Fibonacci, wprowadził do Europy arabski dziesiętny system pozycyjny oraz napisał o nim książkę Liber abaci

- Z jego nazwiskiem jest związany ciąg Fibonacci'ego

Ciąg Fibonacci'ego $F_0, F_1, \ldots, F_i, \ldots$, to ciąg liczb zdefiniowany następująco: $F_0 = 1$, $F_1 = 1$, $F_i = F_{i-1} + F_{i-2}$ dla $i \geq 2$

Algorytmy wykorzystujące jedynie zmienne

- Powyższa definicja jest rekurencyjna, znany jest także jej nierekurencyjny (iteracyjny) odpowiednik podany przez J.M.P. Bineta (1786-1856) w 1843 roku

$$F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$$

- Zauważmy, że z ww. wzoru wynika, że

$$F_n \approx \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n$$

- Pierwsza wersja iteracyjnego algorytmu obliczania n-tej liczby Fibonacci'ego jest następująca

Algorytmy wykorzystujące jedynie zmienne

FIBONACCI_1(n)

```
F0 = 1
F1 = 1
k = 2
while k ≤ n do
    Fk = F0 + F1
    F0 = F1
    F1 = Fk
    k = k + 1
return Fk
```

Algorytmy wykorzystujące tablice 1D

- Duża grupa algorytmów elementarnych wykorzystuje strukturę tablicy 1D
- Tablica jest złożoną strukturą danych, zbudowaną z pewnej liczby elementów tego samego typu
- Do elementu tablicy odwołujemy się podając nazwę tablicy oraz położenie tego elementu w tablicy, opisane za pomocą wyrażenia indeksowego
- **Przykłady:** A[3], B[2i-1]
- Opis tablicy zawiera jej nazwę oraz dolny i górny zakres wartości indeksu
- Mimo iż w pseudokodach algorytmów pomijamy opis typu danych elementów składowych tablic, to w przypadku implementacji tych algorytmów w języku programowania jest on konieczny
- **Przykłady:** A[1..n], B[0..m]
- Tablicę jednowymiarową można utożsamić z wektorem

Algorytmy wykorzystujące tablice 1D

- Rozpoczniemy od algorytmu, który dokonuje inicjalizacji wszystkich elementów tablicy 1D

INICJALIZACJA_TABLICY(n,A[1..n])

```
for i = 1 to n do
    A[i] = i
return
```

Algorytmy wykorzystujące tablice 1D

- Następny algorytm wypisuje wartość **true** (**false**), jeśli dwie tablice 1D są (nie są) identyczne

POROWNANIE_TABLIC(n,A[1..n],B[1..n])

```
wynik = true
i = 0
while (i<>0) and wynik do // <> ≡ ≠
    i = i + 1
    wynik = (A[i] == B[i])  //nawiasy!
write wynik
return
```

Algorytmy wykorzystujące tablice 1D

- Kolejny algorytm oblicza liczbę ujemnych elementów w danej tablicy 1D

ILE_UJEMNYCH_1(n,A[1..n])

```
ile = 0
for i = 1 to n do
    if A[i] < 0
    then ile = ile + 1
return ile
```

Algorytmy wykorzystujące tablice 1D

- Pseudokod podanego algorytmu można także zapisać następująco
- W przyszłości pokażemy, że obie postaci są równoważne

ILE_UJEMNYCH_2(n,A[1..n])

```
ile = 0
i = 1
pętla_1: if i ≤ n then
             if A[i] < 0 then ile = ile + 1
             else i = i + 1
                  goto pętla_1
         else goto pętla_2
pętla_2: return ile
```

Algorytmy wykorzystujące tablice 1D

- Klasycznym przykładem algorytmu wykorzystującego tablicę 1D jest algorytm wyszukiwania w nieuporządkowanej tablicy

ZNAJDZ_ELEMENT_1 (n,A[1..n],x)

```
i = 1
while (A[i] ≠ x) and (i ≠ n) do
    i = i+1
if A[i] ≠ x
then write "Nie znaleziono x w A"
else return i
```

- Warunek w pętli **while** można uprościć dodając wartownika

ZNAJDZ_ELEMENT_2 (n,A[1..n+1],x)

```
A[n+1] = x
i = 1
while A[i] ≠ x do
    i = i+1
if i ≤ n
then return i
else write "Nie znaleziono x w A"
     return
```

Algorytmy wykorzystujące tablice 1D

- Innym przykładem algorytmu wykorzystującego tablicę 1D jest algorytm wyszukiwania w uporządkowanej tablicy (wyszukiwanie binarne)

WYSZUKIWANIE_BINARNE (n,A[1..n],x)

```
i = 1
j = n
while (A[i] ≠ x) and (i ≤ j) do
    k = (i+j) div 2
    if x > A[k]
    then i = k + 1
    else j = k - 1
if A[k] = x
then return k
else write "Nie znaleziono x w A"
     return
```

Algorytmy wykorzystujące tablice 1D

- Wielomianem (stopnia n zmiennej rzeczywistej x) nazywamy funkcję postaci $a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$, gdzie współczynniki $a_0, a_1, \ldots, a_n \in \mathbb{R}$
- Problem obliczania wartości wielomianu w punkcie można rozwiązać za pomocą różnych algorytmów
- Następujący algorytm oblicza wartość wielomianu w punkcie wprost z definicji

WARTOSC_WIELOMIANU(n,A[0..n],x0)

```
s = A[0]
for i = 1 to n do
    pot = 1
    for j = 1 to i do
        pot = pot * x0
    s = s + A[i] * pot
return s
```

- Algorytm WARTOSC_WIELOMIANU nie jest efektywny (szybki), ponieważ wykonuje dużą (formalnie: kwadratową) liczbę mnożeń
- Tę liczbę można zmniejszyć, jeżeli wielomian przedstawimy w postaci iloczynowej

Przykład

- Niech $f(x) = 2x^3 - 4x^2 + 5x - 1$.

- Algorytm WARTOSC_WIELOMIANU nie jest efektywny (szybki), ponieważ wykonuje dużą (formalnie: kwadratową) liczbę mnożeń
- Tę liczbę można zmniejszyć, jeżeli wielomian przedstawimy w postaci iloczynowej

Przykład

- Niech $f(x) = 2x^3 - 4x^2 + 5x - 1$. Wówczas
- $f(x) = (2x^2 - 4x + 5)x - 1 = ((2x - 4)x + 5)x - 1$

Algorytmy wykorzystujące tablice 1D

- Tę postać wykorzystuje schemat Hornera, zaproponowany przez W.G. Hornera (1786-1837)
- Algorytm wykorzystujący schemat Hornera można sformułować bez zapamiętywania wyników częściowych

SCHEMAT_HORNERA_1(n,A[0..n],x0)

```
wart = A[n]
for i = n-1 downto 0 do
    wart = wart * x0 + A[i]
return wart
```

Algorytmy wykorzystujące tablice 1D

- Inna postać schematu Hornera wykorzystuje tablicę do zapamiętania wyników częściowych

SCHEMAT_HORNERA_2(n,A[0..n],x0)

```
W[n] = A[n]
for i = n-1 downto 0 do
    W[i] = W[i+1] * x0 + A[i]
return W[0]
```

Algorytmy wykorzystujące tablice 1D

- Wspomniane wcześniej liczby Fibonacci'ego także można obliczać za pomocą tablicy 1D

FIBONACCI_2(n)

```
F[0] = 1
F[1] = 1
k = 2
while k ≤ n do
    F[k] = F[k-1] + F[k-2]
    k = k + 1
return F[n]
```

Algorytmy wykorzystujące tablice 1D

- Algorytm FIBONACCI_2 jest przykładem zastosowania programowania dynamicznego
- Rozwiązywanie problemu za pomocą programowania dynamicznego polega na podziale badanego problemu na podproblemy o mniejszym rozmiarze, a następnie ich rozwiązaniu, zapamiętaniu tych rozwiązań oraz ich wykorzystaniu do rozwiązania całego problemu

Problem rozwiązywany za pomocą algorytmu wykorzystującego programowanie dynamiczne musi posiadać własność optymalnej podstruktury: optymalne częściowe rozwiązanie jest częścią optymalnego pełnego rozwiązania

- Programowanie dynamiczne opracował w latach 50-tych XX wieku Amerykanin R.E. Bellman (1920-1984)

Algorytmy wykorzystujące tablice 1D

- Liczba pierwsza to całkowita liczba dodatnia, która dzieli sie tylko przez 1 i samą siebie
- **Przykłady:** Liczby 2, 3, 5 i 7 są pierwsze, natomiast liczby 6, 21, 49 i 111 nie są pierwsze
- Podamy teraz przykład algorytmu wykorzystującego tablicę 1D do znalezienia wszystkich liczb pierwszych w przedziale [2,n]
- Ze względu na metodę znajdowania liczb pierwszych algorytm ten nosi nazwę sito Eratostenesa

Algorytmy wykorzystujące tablice 1D

- Zakładamy że dana jest tablica $A[2..n]$ oraz $A[i] = i$ dla $2 \leq i \leq n$
- Jako wynik zwracamy niezerowe elementy tablicy $A[2..n]$

SITO_ERATOSTENESA(n, A[2..n])

```
for i = 2 to n do
    if A[i] ≠ 0
    then write A[i]
        j = i
        while j ≤ n do
            j = j + i
            if A[j] ≠ 0
            then A[j] = 0
return {A[i] ≠ 0 : 2 ≤ i ≤ n}
```

Algorytmy wykorzystujące tablice 1D

Przykład zastosowania sita Eratostenesa

- Załóżmy, że chcemy znaleźć wszystkie liczby pierwsze z przedziału [2,20]
- Początkowy zbiór liczb z tego przedziału jest postaci 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
- Pierwszą liczbą która pozostanie w sicie jest 2
- Skreślamy wszystkie wielokrotności 2
- W zbiorze pozostaną liczby 3,5,7,9,11,13,15,17,19
- Kolejną liczbą pierwszą która pozostanie w sicie jest 3
- Skreślamy wszystkie wielokrotności 3
- W zbiorze pozostaną liczby 5,7,11,13,17,19
- Kolejną liczbą pierwszą która pozostanie w sicie jest 5 itd.
- Po zakończeniu pracy algorytmu w sicie pozostaną liczby pierwsze 2,3,5,7,11,13,17,19

Literatura

- A.V. Aho. J.E. Hopcroft, J.D. Ullman, Projektowanie i analiza algorytmów komputerowych, PWN, 1983, rozdz. 3.
- T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Wprowadzenie do algorytmów, Wydawnictwo Naukowe PWN, 2012, rozdz. 2, 4, 7.
- D.E. Knuth, Sztuka programowania, Tom 1: Algorytmy podstawowe, WNT, 2002, rozdz. 1.1-1.2.