Algorytmy i struktury danych

Wykład I – wstęp do algorytmów

Paweł Rembelski

PJWSTK

4 października 2009

Notacja asymptotyczna – przypomnienie

Definicja

Niech $f : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ i $g : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ będą funkcjami takimi, że:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = a,$$

gdzie $a \in \mathbb{R}^+ \cup \{0\}$. Jeżeli:

- $a = 0$, to mówimy, że *rząd funkcji f jest ściśle mniejszy* niż rząd funkcji g i oznaczamy przez $f \prec g$,
- $a > 0$, to mówimy, że *rząd funkcji f jest równy* rzędowi funkcji g (lub *funkcje f i g są tego samego rzędu*) i oznaczamy przez $f \asymp g$,
- $a = \infty$, to mówimy, że *rząd funkcji f jest ściśle większy* niż rząd funkcji g i oznaczamy przez $f \succ g$.

Wniosek

Jeżeli dla pewnej funkcji f znajdziemy funkcję g taką, że $f \asymp g$ oraz:

- $g(n) = c$, gdzie $c \geq 0$ jest pewną stałą, to f jest funkcją stałą,
- $g(n) = \log_c n$, gdzie $c > 0$ jest pewną stałą i $n \geq 1$, to f jest funkcją logarytmiczną (dla funkcji $\log_2 n$ wprowadzamy oznaczenie $\lg n$),
- $g(n) = n^{\frac{1}{c}}$, gdzie $c > 1$ jest pewną stałą, to f jest funkcją pierwiastkową (lub podwielomianową),
- $g(n) = n^c$, gdzie $c \geq 1$ jest pewną stałą, to f jest funkcją wielomianową (szczególnym przypadkiem funkcji wielomianowej jest funkcja liniowa),
- $g(n) = c^n$, gdzie $c > 1$ jest pewną stałą, to f jest funkcją wykładniczą,
- $g(n) = n^n$, to f jest funkcją ponadwykładniczą.

Wniosek

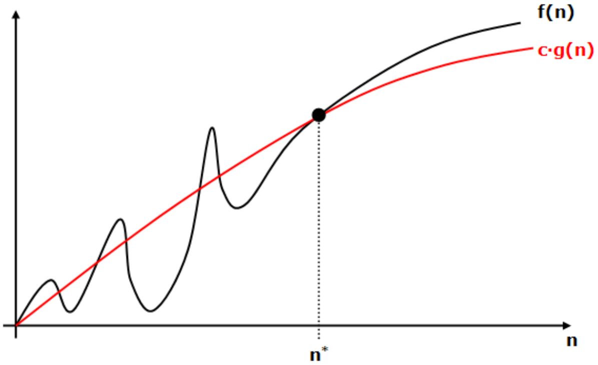
Jeżeli $c > 1$ jest pewną stałą, to:

$$c \prec \log_c n \prec n^{\frac{1}{c}} \prec n \prec n \lg n \prec n^c \prec c^n \prec n! \prec n^n,$$

dla wszystkich dostatecznie dużych n.

Definicja

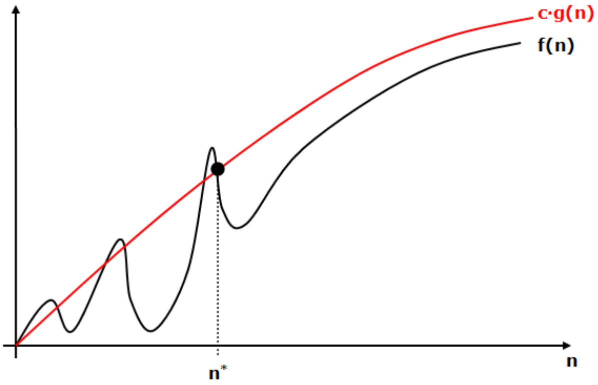
Niech $f : \mathbb{N} \rightarrow \mathbb{R}^{+} \cup \{0\}$ i $g : \mathbb{N} \rightarrow \mathbb{R}^{+} \cup \{0\}$ będą funkcjami. Jeżeli $f \succ g$ albo $f \asymp g$, to mówimy że rząd funkcji f jest *równy co najmniej* rzędowi funkcji g (lub funkcja g *ogranicza z dołu* funkcję f) i oznaczamy przez $f = \Omega(g)$.



Na przedstawionym rysunku $f = \Omega(g)$, gdzie $c > 0$ jest pewną stałą.

Definicja

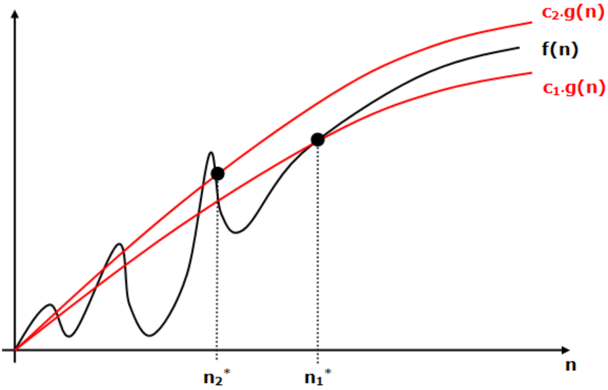
Niech $f : \mathbb{N} \rightarrow \mathbb{R}^{+} \cup \{0\}$ i $g : \mathbb{N} \rightarrow \mathbb{R}^{+} \cup \{0\}$ będą funkcjami. Jeżeli $f \prec g$ albo $f \asymp g$, to mówimy że rząd funkcji f jest *równy co najwyżej* rzędowi funkcji g (lub funkcja g *ogranicza z góry* funkcję f) i oznaczamy przez $f = O(g)$.



Na przedstawionym rysunku $f = O(g)$, gdzie $c > 0$ jest pewną stałą.

Definicja

Niech $f : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ i $g : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ będą funkcjami. Jeżeli $f \asymp g$, to mówimy że funkcja g *ogranicza z dołu i z góry* funkcję f i oznaczamy przez $f = \Theta(g)$.



Na przedstawionym rysunku $f = \Theta(g)$, gdzie $c > 0$ jest pewną stałą.

Ograniczenie z góry:

$$\lg n! = \lg \prod_{i=1}^{n} i \leq \lg \prod_{i=1}^{n} n = \lg n^n = n \lg n,$$

czyli $\lg n! \leq n \lg n$.

Ograniczenie z dołu:

$$\begin{aligned} \lg n! &= \lg \prod_{i=1}^{n} i = \sum_{i=1}^{n} \lg i \geq \int_1^n \lg x dx = \int_1^n \frac{\ln x}{\ln 2} dx \\ &= \left(\frac{x \ln x - x}{\ln 2} \right)_1^n \geq c \cdot \frac{n \ln n}{\ln 2} = c \cdot n \lg n, \end{aligned}$$

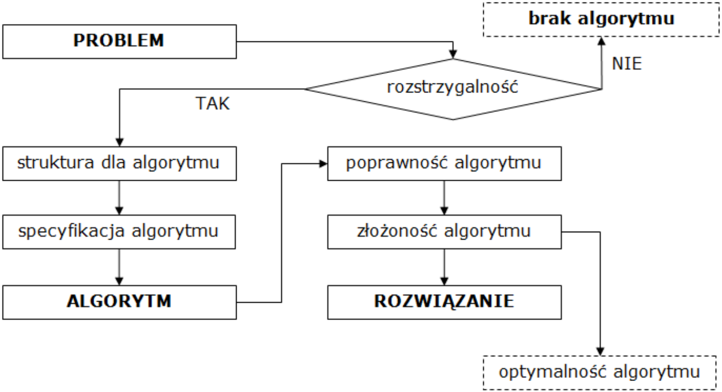
gdzie $c \leq 1$ jest pewną dodatnią stałą, czyli $\lg n! \geq c \cdot n \lg n$.

Wniosek

Ponieważ $\lg n! \leq n \lg n$ i $\lg n! \geq c \cdot n \lg n$, dla $0 < c \leq 1$, to $\lg n! \asymp n \lg n$, czyli $\lg n! = \Theta(n \lg n)$.

Algorytmy – podstawy

Algorytm i zagadnienia towarzyszące



Nieformalnie o problemach nierozstrzygalnych ...

Problem nierozstrzygalny to zagadnienie, dla którego **nie istnieje** algorytm rozwiązania, np:

- problem Hamleta: "być albo nie być, oto jest pytanie ...",
- problem stopu: zaproponować algorytm sprawdzający, czy dowolny inny algorytm jest procesem skończonym.

Zadanie. Podać uzasadnienie dla nierozstrzygalności problemu stopu.

Nieformalnie o problemach rozstrzygalnych ...

Problem rozstrzygalny (algorytmiczny) to zagadnienie, dla którego **istnieje** skończony algorytm rozwiązania, np:

- problem wyprania ubrań w pralce automatycznej: dla określonej pralki, środka piorącego i sterty brudnych ubrań zaproponować algorytm ich wyprania,
- problem elementu maksymalnego: zaproponować algorytm wyszukujący element maksymalny w danym skończonym, niepustym wektorze różnych liczb naturalnych.

Wniosek

Przedmiotem dalszych wspólnych rozważań będą jedynie **problemy algorytmiczne**.

Nieformalnie o strukturze dla algorytmu ...

Strukturą dla algorytmu nazywamy „środowisko” wykonania owego algorytmu zadane bezpośrednio charakterystyką rozwiązywanego problemu algorytmicznego.

Formalnie o strukturze dla algorytmu ...

Strukturą dla algorytmu nazywamy system algebraiczny

$$S = \langle U, o_0, o_1, o_2, \ldots, r_n, r_1, \ldots, r_m \rangle ,$$

gdzie U jest uniwersum a o_i, r_j, dla $0 \leq i \leq n$ oraz $0 \leq j \leq m$, to odpowiednio operacje i relacje w zbiorze U, który pozwala na wyrażenie charakterystyki rozwiązywanego problemu.

Przykład. Struktura dla problem elementu maksymalnego: $S = \langle \mathbb{N}, +, < \rangle$.

Uwaga! W dalszej części wykładu podczas konstrukcji algorytmów będziemy domyślnie zakładali dostępność podstawowych struktur algebraicznych i co za tym zwykle będziemy podawać jedyni ich niezbędne rozszerzenie.

Nieformalnie o algorytmie ...

Algorytm* to metoda postępowania, która w danym „środowisku” prowadzi do rozwiązania postawionego problemu algorytmicznego, czyli intuicyjnie

$$algorytm\,(problem) = rozwiązanie.$$

Przykład. Algorytm dla problemu wyprania ubrań w pralce automatyczne:

- włącz pralkę, załaduj ubrania i nasyp proszek,
- jeżeli ubrania są bardzo brudne, to nastaw program z zakresu VII-XII, w przeciwnym przypadku nastaw program z zakresu I-VI,
- odkręć zawór doprowadzający wodę do pralki,
- naciśnij przycisk START,
- dopóki nie zaświeci się kontrolka KONIEC, nie przeszkadzaj pralce w pracy,
- zakręć zawór doprowadzający wodę do pralki,
- wyjmij ubrania, wyłącz pralkę.

---

* określenie algorytm wywodzi się od nazwiska matematyka perskiego Muhammed'a ibn Musa Alchwarizmi (przełom VIII i IX wieku).

Formalnie o algorytmie ...

Algorytm (*metoda*, *procedura*) nad strukturą $S = \langle U, o_0, o_1, o_2, \ldots, r_n, r_1, \ldots, r_m \rangle$ to skończony ciąg „czynności", które pozwalają przekształcić zadany zbiór *elementów wejściowych* $IN \subseteq U$ (inaczej *informacje wejściowe*, *dane wejściowe*, *argumenty algorytmu*) w zbiór elementów wyjściowych $OUT \subseteq U$ (inaczej *informacje wyjściowe*, *dane wyjściowe*, *rezultat algorytmu*).

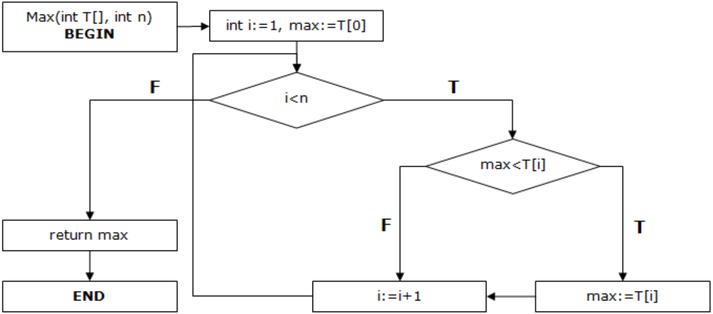
Przykład. Algorytm znajdowania elementu maksymalnego w niepustym wektorze (tablicy) T różnych liczb naturalnych długości n:

- dane wejściowe: T – wektor wejściowy, n – rozmiar (długość) wektora T,
- dane wyjściowe: *max* – element maksymalny wektora T,
- algorytm MAX:
    - przypisz zmiennej *max* wartość $T[0]$,
    - jeżeli wartość zmiennej *max* jest mniejsza niż wartość 2-go elementu wektora, to przypisz zmiennej *max* wartość owego elementu,
    - jeżeli wartość zmiennej *max* jest mniejsza niż wartość 3-go elementu wektora, to przypisz zmiennej *max* wartość owego elementu,
    - ...
    - jeżeli wartość zmiennej *max* jest mniejsza niż wartość n-go elementu wektora, to przypisz zmiennej *max* wartość owego elementu,
    - zwróć jako wynik wartość zmiennej *max*.

Definicja

Schematem blokowym algorytmu nazywamy skończony graf skierowany o ustalonej składni i semantyce krawędzi oraz wierzchołków, reprezentujący sposób działania rozważanej metody.

Przykład. Algorytm znajdowania elementu maksymalnego w niepustym wektorze (tablicy) T różnych liczb naturalnych długości n:



Definicja

Implementacją algorytmu nazywamy zapis sposobu działania rozważanej metody w danym języku (pseudokodzie) o ustalonej składni i semantyce.

Przykład. Algorytm znajdowania elementu maksymalnego w niepustym wektorze (tablicy) T różnych liczb naturalnych długości n:

```
int Max(int T[], int n) {
  int i:=1, max:=T[0];

  while (i<n) {
    if (max<T[i]) max:=T[i];
    i:=i+1;
  }
  return max;
}
```

Uwaga! W dalszej części wykładu będziemy analizowali algorytmy zapisane głównie w postaci implementacyjnej w pseudokodzie o składni i semantyce zbliżonej do języków programowania *C*, *C++*, *Java* (z wyjątkiem np. instrukcji przypisania i symbolu relacji równości).

Nieformalnie o poprawności algorytmu ...

Algorytm jest poprawny, jeżeli w przyjętym „środowisku" wykonania realizuje zamierzenia opisane relacja między danymi wejściowymi a danymi wyjściowymi, czyli rozwiązuje postawiony problem.

Definicja

Specyfikacją algorytmu nazywamy parę $\langle WP, WK \rangle$, dla WP będącego *warunkiem początkowym* i WK jest *warunkiem końcowym* algorytmu, gdzie postać obu warunków jest podyktowana sformułowaniem rozwiązywanego problemu.

Przykład. Specyfikacja algorytmu Max:

```
int Max(int T[], int n) {
  int i:=1, max:=T[0];
  ...
  return max;
}
```

WP : T jest niepustym wektorem różnych liczb naturalnych, $n \in \mathbb{N}^+$, $|T| = n$

WK : $Max\,(T, n) = max$, gdzie max jest maksymalnym elementem wektora T

Definicja

Algorytm *Alg* działający w strukturze S jest *częściowo poprawny* ze względu na specyfikację $\langle WP, WK \rangle$ wtedy i tylko wtedy, gdy dla wszystkich danych wejściowych, które spełniają warunek początkowy WP, jeżeli algorytm *Alg* zatrzyma się, to uzyskane dane wyjściowe spełniają warunek końcowy WK.

Pytanie. Czy algorytm Max jest częściowo poprawny w strukturze $S = \langle \mathbb{N}, +, < \rangle$?

Definicja

Algorytm *Alg* działający w strukturze S jest *całkowicie poprawny* (*poprawny*) ze względu na specyfikację $\langle WP, WK \rangle$ wtedy i tylko wtedy, gdy dla wszystkich danych wejściowych, które spełniają warunek początkowy WP algorytm *Alg* zatrzyma się i uzyskane dane wyjściowe spełniają warunek końcowy WK.

Pytanie. Czy algorytm Max jest całkowicie poprawny w strukturze $S = \langle \mathbb{N}, +, < \rangle$?

Wniosek

Każdy algorytm, który jest całkowicie poprawny jest także częściowo poprawny, ale nie każdy algorytm, który jest częściowo poprawny jest także całkowicie poprawny.

Definicja

Algorytm *Alg* ma (spełnia) *własność stopu* wtedy i tylko wtedy, gdy zatrzymuje się dla dowolnych danych wejściowych spełniających warunek początkowy *WP*.

Wniosek

Aby dowieść poprawności całkowitej algorytmu wystarczy:

- udowodnić jego poprawność częściową,
- wykazać własność stopu algorytmu.

Trudności

Czy ustalenie własności poprawności nawet trywialnego algorytmu iteracyjnego jest zadaniem prostym ... niestety nie! Dla niektórych algorytmów własność poprawności zależy od poprawności dotychczas nierozwiązanych hipotez.

Przykład. Rozważmy algorytm realizujący następujący ciąg operacji:

```
bool Collatz(int n) {
  int i:=n;

  while (i>1) {
    if (i mod 2=0) i:=i/2;
     else i:=3*i+1;
  }
  return TRUE;
}
```

$WP : n \in \mathbb{N}$

$WK : Collatz\,(n) = TRUE$

Hipoteza Collatza

Dla dowolnej liczby naturalnej dodatniej n prawdą jest, że $Collatz\,(n) = TRUE$.

Wniosek

Załóżmy, że hipoteza Collatz'a jest fałszywa, wtedy:

- algorytm Collatz jest częściowo poprawny,
- algorytm Collatz nie jest całkowicie poprawny (brak własności stopu).

Definicja

Operacją dominującą w algorytmie nazywamy ten jego element, którego wykonanie uważamy za najbardziej kluczowe z punktu widzenia np. implementacji realizacji w danym środowisku obliczeniowym.

Pytanie. Którą z operacji w algorytmie Max powinniśmy uznać za dominującą, jeżeli algorytm implementujemy i uruchamiamy na standardowym komputerze klasy PC?

```
int Max(int T[], int n) {
  int i:=1, max:=T[0];

  while (i<n) {            operacja porównania
    if (max<T[i]) max:=T[i];   operacja porównania, operacja przypisania
    i:=i+1;                operacja przypisania, operacja dodawania
  }
  return max;
}
```

Definicja

Niech *Alg* będzie algorytmem oraz *d* będą ustalonymi danymi wejściowymi dla tego algorytmu. *Koszt czasowy* wykonania algorytm *Alg* dla danych wejściowych *d* jest to liczba operacji dominujących jakie wykonuje rozważany algorytm na rozważanych danych wejściowych. Koszt czasowy oznaczamy przez $t(Alg, d)$.

Pytanie. Jaki jest koszt czasowy algorytmu Max dla danych wejściowych $d = (T, n)$, gdzie $n = 100$?

Definicja

Niech *Alg* będzie algorytmem oraz *d* będą ustalonymi danymi wejściowymi dla tego algorytmu. *Koszt pamięciowy* wykonania algorytm *Alg* dla danych wejściowych *d* jest to liczba dodatkowych jednostek pamięci jakie są niezbędne do wykonania rozważanego algorytmu na rozważanych danych wejściowych. Koszt pamięciowy oznaczamy przez $s(Alg, d)$.

Pytanie. Jaki jest koszt pamięciowy algorytmu Max dla danych wejściowych $d = (T, n)$, gdzie $n = 100$?

Uwaga! W dalszej części wykładu będziemy używali skróconej notacji złożoności $t(d)$ i $s(d)$ jeżeli będzie to jednoznaczne.

Definicja

Złożoność czasowa algorytmu Alg to liczba operacji dominujących jakie wykonuje rozważany algorytm na danych wejściowych rozmiaru n, wyrażona jako funkcja rozmiaru tych danych. Złożoność czasową oznaczamy przez $T(Alg, n)$.

Pytanie. Jaka jest złożoność czasowa algorytmu Max?

Pytanie. Jakie jest ograniczenie dolne złożoności czasowej algorytmu Max?

Definicja

Złożoność pamięciowa algorytmu Alg to liczba dodatkowych jednostek pamięci jakie są niezbędne do wykonania rozważanego algorytmu na danych wejściowych rozmiaru n, wyrażona jako funkcja rozmiaru tych danych. Złożoność pamięciową oznaczamy przez $S(Alg, n)$.

Pytanie. Jaka jest złożoność pamięciowa algorytmu Max?

Pytanie. Jakie jest ograniczenie górne złożoności czasowej algorytmu Max?

Uwaga! W dalszej części wykładu będziemy używali skróconej notacji złożoności $T(n)$ i $S(n)$ jeżeli będzie to jednoznaczne.

Trudności ...

Czy złożoność czasową algorytmu można zawsze podać w sposób dokładny ... niestety nie! Dla niektórych algorytmów, złożoność czasowa jest funkcją nie tylko rozmiaru danych wejściowych ale i ich postaci.

Przykład. Nieco inny algorytm dla rozważanego problemu:

```
int ModifiedMax(int T[], int n) {
```

$WP: T$ jest niepustym wektorem różnych liczb naturalnych, $n \in \mathbb{N}^+$, $|T| = n$

```
  int i:=1, max:=T[0];

  while (i<n) {
    if (max<T[i]) max:=T[i];
    if (T[i] mod 2=1) { T[i]:=T[i]-1; i:=0; } else i:=i+1;
  }
  return max;
```

WK : $ModifiedMax(T, n) = max$, gdzie max maksymalnym elementem wektora T

```
}
```

Pytanie. Jaka jest złożoność czasowa algorytmu ModifiedMax, jeżeli operacją dominującą jest czynność porównywania wartości elementów wektora wejściowego z wartością zmiennej *max*?

Definicja

Pesymistyczna złożoność czasowa algorytmu Alg, oznaczona przez $W(Alg, n)$ jest równa

$$W(Alg, n) = \max\{t(Alg, d) : d \in D_n\},$$

gdzie D_n jest zbiorem wszystkich danych wejściowych rozmiaru n dla problemu, który rozwiązuje algorytm Alg.

Definicja

Średnia (oczekiwana) złożoność czasowa algorytmu Alg, oznaczona przez $A(Alg, n)$ jest równa

$$A(Alg, n) = \sum_{d \in D_n} p(d) \cdot t(Alg, d),$$

gdzie D_n jest zbiorem wszystkich danych wejściowych rozmiaru n dla problemu, który rozwiązuje algorytm Alg, a $p(d)$ jest prawdopodobieństwem wystąpienia danych d.

Uwaga! W dalszej części wykładu będziemy używali skróconej notacji złożoności $W(n)$ i $A(n)$ jeżeli będzie to jednoznaczne.

Pytanie. Jaka jest średnia i pesymistyczna złożoność czasowa algorytmu Max oraz ModifiedMax, jeżeli operacją dominującą jest czynność porównywania wartości elementów wektora wejściowego z wartością zmiennej max?

Definicja

Algorytm nazywamy *efektywnym* jeżeli jego złożoność czasowa i pamięciowa jest co najwyżej wielomianowa względem rozmiaru danych wejściowych.

Przykład. Rozważmy trzy algorytmy Alg_1, Alg_2 i Alg_3 rozwiązujące ten sam problem, gdzie

$$T(Alg_1, n) = \lg n, \;\; T(Alg_2, n) = n^2, \;\; T(Alg_3, n) = 2^n.$$

Uruchamiamy równolegle rozważane algorytm dla identycznych danych wejściowych d rozmiaru n na trzech jednakowych komputerach, które dla uproszczenia wykonują jedną operację dominującą na sekundę. Jak długo będziemy oczekiwali na wynik obliczeń?

- dla $n = 8$, $t(Alg_1, d) = 3$ sek., $t(Alg_2, d) = 64$ sek., $t(Alg_3, d) = 256$ sek.
- dla $n = 16$, $t(Alg_1, d) = 4$ sek., $t(Alg_2, d) = 256$ sek., $t(Alg_3, d) \approx 18,2$ godz.
- dla $n = 32$, $t(Alg_1, d) = 5$ sek., $t(Alg_2, d) \approx 17,1$ min., $t(Alg_3, d) \approx 136,2$ lat!

Pytanie. Ile lat zajmie wykonania algorytmu Alg_3 dla danych d rozmiaru 64?

Definicja

Niech $T(n)$ będzie funkcją złożoności czasowej algorytmu Alg rozwiązującego pewien problem P dla danych wejściowych rozmiaru n. Algorytm Alg nazywamy *asymptotycznie optymalnym* rozwiązaniem problemu P jeżeli nie istniej algorytm Alg^*, o pesymistycznej złożoności czasowej $W^*(Alg^*, n)$, rozwiązujący problem P taki, że
$W^*(Alg^*, n) \prec T(n)$.

Definicja

Niech $T(n)$ będzie funkcją złożoności czasowej optymalnego algorytmu Alg rozwiązującego pewien problem P dla danych wejściowych rozmiaru n. Algorytm Alg^* nazywamy *asymptotycznie optymalnym*, *w przypadku średnim*, rozwiązaniem problemu P jeżeli $A(Alg^*, n) \asymp T(n)$.

Definicja

Niech $T(n)$ będzie funkcją złożoności czasowej optymalnego algorytmu Alg rozwiązującego pewien problem P dla danych wejściowych rozmiaru n. Algorytm Alg^* nazywamy *asymptotycznie optymalnym*, *w przypadku pesymistycznym*, rozwiązaniem problemu P jeżeli $W(Alg^*, n) \asymp T(n)$.

Wniosek

Każdy algorytm optymalny w przypadku pesymistycznym jest algorytmem optymalnym dla zadanej klasy problemów, ale nie każdy algorytm optymalny w przypadku średnim jest algorytmem optymalnym dla zadanej klasy problemów.

Algorytmy iteracyjne – przykład

Problem, struktura i specyfikacja algorytmu

Problem

Podać algorytm **iteracyjny** $Alg(n)$ obliczający silnię liczby naturalnej n.

Struktura dla algorytmu

Struktura dla algorytmu iteracyjnego Alg: $\langle \mathbb{N}, +, \cdot, < \rangle$.

Specyfikacja algorytmu

Specyfikację algorytmu Alg stanowi para $\langle WP, WK \rangle$, gdzie warunki początkowy i końcowy są postaci kolejno:

- $WP: n \in \mathbb{N}$,
- $WK: Alg(n) = n!$

Algorytmy iteracyjne – przykład

Algorytm iteracyjny Silnia

Rozwiązanie problemu – **algorytm iteracyjny Silnia**:

```
int Silnia(int i) {
  int i:=0, s:=1;

  while (i<n) {
    i:=i+1;
    s:=s*i;
  }
  return s;
}
```

$WP: n \in \mathbb{N}$

$WK: Silnia(n) = n!$

Dowód częściowej poprawności algorytmu **metodą niezmienników**:

`int Silnia(int n) {`	$n \in \mathbb{N}$
`  int i:=0, s:=1;`	$(i = 0 \wedge s = 1) \Rightarrow (i! = 1 \wedge s = 1) \Rightarrow s = i$
	ustalenie niezmiennika: $NZ : s = i!$
`  while (i<n) {`	**weryfikacja niezmiennika**: $s = i!$
`    i:=i+1;`	$(s = i! \wedge i := i + 1) \Rightarrow s = (i-1)!$
`    s:=s*i;`	**odtworzenie niezmiennika**: $(s = (i-1)! \wedge s := s \cdot i) \Rightarrow \frac{s}{i} = (i-1)! \Rightarrow s = i! \Rightarrow NZ : s = i!$
`  }`	$(NZ \wedge \neg(i < n)) \Rightarrow (s = i! \wedge i \geq n) \Rightarrow^{*} (s = i! \wedge i = n) \Rightarrow s = n!$
`  return s;`	$s = n! \Rightarrow Silnia\,(n) = n!$
`}`	

Wniosek

Algorytm iteracyjny Silnia jest częściowo poprawny względem specyfikacji $\langle WP, WK \rangle$.

* ponieważ z warunku początkowego $n \in \mathbb{N}$ oraz tuż po inicjalizacji zmiennych $i = 0$ jak i w każdej iteracji rozważanej pętli zmienna i inkrementowana jest o wartość równą 1, to ostatecznie $i = n$. Stąd warunek $i \geq n$ można wzmocnić do postaci $i = n$.

Dowód własności stopu algorytmu przez **indukcję** względem wartości argumentu $n \in \mathbb{N}$:

- **baza indukcji**: dla $n = 0$ i inicjalizacji zmiennej $i := 0$, nie jest spełniony warunek dozoru pętli iteracyjnej $i < n$, stąd algorytm Silnia zatrzymuje się,
- **założenie indukcyjne**: dla $0 \leq i < n$ algorytm Silnia spełnia własność stopu,
- **teza indukcyjna**: dla $n \geq 1$ algorytm Silnia spełnia własność stopu.

Dowód tezy indukcyjnej

Z założenia indukcyjnego algorytm Silnia spełnia własność stopu dla dowolnego $0 \leq i < n$. Zatem, wykonanie algorytmu $Silnia\,(n-1)$ jest skończone. Dalej z treści pętli iteracyjnej $i := i + 1$ oraz warunku dozoru pętli$i < n$ wynika, że wykonanie algorytmu $Silnia\,(n)$ jest o jeden krok iteracyjny dłuższe od wykonania algorytmu $Silnia\,(n-1)$. Stąd wykonanie algorytmu $Silnia\,(n)$ jest skończone dla dowolnego $n \geq 1$.

Wniosek

Ponieważ algorytm iteracyjny Silnia jest częściowo poprawny i spełnia własność stopu, to jest całkowicie poprawny względem specyfikacji $\langle n \in \mathbb{N}, Silnia(n) = n! \rangle$.

.

Złożoność czasowa algorytmu

- operacja dominująca: mnożenie liczb naturalnych,
- liczba operacji dominujących niezbędnych do wykonania algorytmu zależy jedynie do rozmiaru danych wejściowych, nie zaś od ich postaci, stąd: $A(n) = W(n)$.
- liczba operacji dominujących niezbędnych do wykonania algorytmu dla wartości argumentu n: $n - 1$,
- oszacowanie złożoności czasowej względem wartości argumentu n: $T(n) = n - 1 = \Theta(n)$,
- oszacowanie złożoności czasowej względem liczby bitów $\alpha = \lceil \lg(n+1) \rceil$ niezbędnych do zapisania wartości argumentu n: $T(n) = \Theta(2^{\alpha})$,

Złożoność pamięciowa algorytmu

- ilość pamięci dodatkowej niezbędnej do wykonania algorytmu dla wartości argumentu n: 2 zmienne dodatkowe,
- oszacowanie złożoności czasowej względem wartości argumentu n: $S(n) = \Theta(1)$,
- oszacowanie złożoności czasowej względem liczby bitów $\alpha = \lceil \lg(n+1) \rceil$ niezbędnych do zapisania wartości argumentu n: $S(n) = \Theta(1)$.

Algorytmy rekurencyjne – przykład

Problem, struktura i specyfikacja algorytmu

Problem

Podać algorytm **rekurencyjny** $Alg\,(n)$ obliczający silnię liczby naturalnej n.

Struktura dla algorytmu

Struktura dla algorytmu rekurencyjnego Alg: $\langle \mathbb{N}, +, \cdot, = \rangle$.

Specyfikacja algorytmu

Specyfikację algorytmu Alg stanowi para $\langle WP, WK \rangle$, gdzie warunki początkowy i końcowy są postaci kolejno:

- $WP\,:\, n \in \mathbb{N}$,
- $WK\,:\, Alg\,(n) = n!$.

Algorytmy rekurencyjne – przykład

Algorytm rekurencyjny Silnia

Rozwiązanie problemu – **algorytm rekurencyjny Silnia**:

```
int Silnia(int n) {
  if (n=0)
    return 1;
  else
    return Silnia(n-1)*n;
}
```

$WP: n \in \mathbb{N}$

$WK: Silnia(n) = n!$

$WK: Silnia(n) = n!$

Dowód częściowej poprawności algorytmu przez **indukcję** względem wartości argumentu $n \in \mathbb{N}$:

```
1  int Silnia(int n) {
2    if (n=0)
3      return 1;
4    else
5      return Silnia(n-1)*n;
6  }
```

- linia 1: $WP : n \in \mathbb{N}$
- linia 3: **baza indukcji**: $Silnia(n) = 1$, dla $n = 0$
- linia 4: **założenie indukcyjne**: $Silnia(i-1) = (i-1)!$, dla $0 \leq i < n$
- linia 5: **teza indukcyjna**: $Silnia(n) = n!$, dla $n \geq 1$

Dowód tezy indukcyjnej

W wierszu 5-tym algorytmu mamy $Silnia\,(n) := Silnia\,(n-1) \cdot n$, stąd i z założenia indukcyjnego $Silnia\,(n) := (n-1)! \cdot n$, zatem $Silnia\,(n) = n!$, dla dowolnego $n \geq 1$.

Z bazy indukcji oraz tezy indukcyjnej wynika, że $Silnia\,(n) = n!$, dla dowolnego $n \in \mathbb{N}$, co dowodzi poprawności warunku końcowego rozważanego algorytmu $WK : Silnia\,(n) = n!$.

Wniosek

Algorytm rekurencyjny Silnia jest częściowo poprawny względem specyfikacji $\langle WP, WK \rangle$.

Dowód własności stopu algorytmu przez **indukcję** względem wartości argumentu $n \in \mathbb{N}$:

- **baza indukcji**: dla $n = 0$ wykonana jest pierwsza część instrukcji warunkowej, stąd algorytm Silnia zatrzymuje się,
- **założenie indukcyjne**: dla $0 \leq i < n$ algorytm Silnia spełnia własność stopu,
- **teza indukcyjna**: dla $n \geq 1$ algorytm Silnia spełnia własność stopu.

Dowód tezy indukcyjnej

Z założenia indukcyjnego algorytm Silnia spełnia własność stopu dla dowolnego $0 \leq i < n$. Zatem, wykonanie algorytmu $\mathit{Silnia}\,(n-1)$ jest skończone. Dalej z treści algorytmu $\mathit{Silnia}\,(n) := \mathit{Silnia}\,(n-1) \cdot n$ wynika, że wykonanie algorytmu $\mathit{Silnia}\,(n)$ jest o jeden krok rekurencyjny dłuższe od wykonania algorytmu $\mathit{Silnia}\,(n-1)$. Stąd wykonanie algorytmu $\mathit{Silnia}\,(n)$ jest skończone dla dowolnego $n \geq 1$.

Wniosek

Ponieważ algorytm rekurencyjny Silnia jest częściowo poprawny i spełnia własność stopu, to jest całkowicie poprawny względem specyfikacji $\langle n \in \mathbb{N}, \mathit{Silnia}(n) = n! \rangle$.

Złożoność czasowa algorytmu

- operacja dominująca: mnożenie liczb naturalnych,
- liczba operacji dominujących niezbędnych do wykonania algorytmu zależy jedynie do rozmiaru danych wejściowych, nie zaś od ich postaci, stąd: $A(n) = W(n)$.
- liczba operacji dominujących niezbędnych do wykonania algorytmu dla wartości argumentu n: $n - 1$,
- oszacowanie złożoności czasowej względem wartości argumentu n: $T(n) = n - 1 = \Theta(n)$,
- oszacowanie złożoności czasowej względem liczby bitów $\alpha = \lceil \lg(n+1) \rceil$ niezbędnych do zapisania wartości argumentu n: $T(n) = \Theta(2^{\alpha})$,

Złożoność pamięciowa algorytmu bez uwzględnienia kosztów rekursji

- ilość pamięci dodatkowej niezbędnej do wykonania algorytmu dla wartości argumentu n: brak zmiennych dodatkowych,
- oszacowanie złożoności czasowej względem wartości argumentu n: $S(n) = \Theta(1)$,
- oszacowanie złożoności czasowej względem liczby bitów $\alpha = \lceil \lg(n+1) \rceil$ niezbędnych do zapisania wartości argumentu n: $S(n) = \Theta(1)$.

Złożoność pamięciowa algorytmu z uwzględnieniem kosztów rekursji

- ilość pamięci dodatkowej niezbędnej do wykonania algorytmu dla wartości argumentu n: stos wywołań rekurencyjnych wysokości n,
- oszacowanie złożoności czasowej względem wartości argumentu n: $S\left(Silnia, n\right) = \Theta\left(n\right)$,
- oszacowanie złożoności czasowej względem liczby bitów $\alpha = \lceil \lg\left(n+1\right)\rceil$ niezbędnych do zapisania wartości argumentu n: $S\left(Silnia, n\right) = \Theta\left(2^{\alpha}\right)$.

Uwaga! W dalszej części wykładu będziemy domyślnie analizowali złożoność algorytmów rekurencyjnych z uwzględnieniem kosztów rekursji.

Dodatek A – twierdzenie o rekursji uniwersalnej

Twierdzenie o rekursji uniwersalnej

Niech $a \geq 1$, $b > 1$ *będą stałymi,* $f : \mathbb{N} \rightarrow \mathbb{R}^+ \cup \{0\}$ *pewną funkcją i nich* $T(n)$ *równaniem rekurencyjnym złożoności pewnego algorytmu, postaci*

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

gdzie $\frac{n}{b}$ *traktujemy jako* $\left\lfloor \frac{n}{b} \right\rfloor$ *albo* $\left\lceil \frac{n}{b} \right\rceil$*, wtedy:*

- *jeżeli* $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ *dla pewnej stałej* $\epsilon > 0$*, to*

$$T(n) = \Theta\left(n^{\log_b a}\right),$$

- *jeżeli* $f(n) = \Theta\left(n^{\log_b a}\right)$ *, to*

$$T(n) = \Theta\left(n^{\log_b a} \lg n\right),$$

- *jeżeli* $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ *dla pewnej stałej* $\epsilon > 0$*, to*

$$T(n) = \Theta(f(n)),$$

pod warunkiem, że $af\left(\frac{n}{b}\right) \leq cf(n)$ *dla pewnej stałej* $c < 1$ *i wszystkich dostatecznie dużych* n*.*

Literatura

1. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Wprowadzenie do algorytmów*, WNT 2004.
2. L. Banachowski, K. Diks, W. Rytter, *Algorytmy i struktury danych*, WNT 1996.
3. A. V. Aho, J. E. Hopcroft, J. D. Ullman, *Algorytmy i struktury danych*, Helion 2003.
4. A. Dańko, T. L. Lee, G. Mirkowska, P. Rembelski, A. Smyk, M. Sydow, *Algorytmy i struktury danych – zadania*, PJWSTK 2006.
5. R. Sedgewick, *Algorytmy w C++*, RM 1999.
6. N. Wirth, *Algorytmy + struktury danych = programy*, WNT 1999.
7. A. Drozdek, D. L. Simon, *Struktury danych w języku C*, WNT 1996.
8. D. Harel, *Rzecz o istocie informatyki – Algorytmika*, WNT 2001.

Zadania ćwiczeniowe

1. Uporządkuj malejąco następujące funkcje zmiennej n względem ich rzędów:
   1. $f_1(n) = \lg n^3$, $f_2(n) = n \lg n!$, $f_3(n) = n^{1/3} + n$,
   2. $f_1(n) = n^3 \lg 3n$, $f_2(n) = 2^{\lg n^4}$, $f_3(n) = \sqrt{2}^n$,
   3. $f_1(n) = n^{\frac{1}{2}} + \lg n$, $f_2(n) = \lg n^2$, $f_3(n) = 2^{\lg n}$, $f_4(n) = n^2 \left(n + n^{\frac{2}{3}}\right)^{\frac{1}{2}}$,
   4. $f_1(n) = n(1+n)^2$, $f_2(n) = n^3 \lg^3(n)$, $f_3(n) = n^n$, $f_4(n) = 2^{2n}$.
2. Określ, które z podanych ograniczeń funkcji $f(n)$ są poprawne:
   1. $f(n) = \Theta(\sqrt{n})$, $f(n) = O\left(n^{\frac{3}{2}} \lg n\right)$, $f(n) = \Omega\left(\left|\frac{1}{n} - 1\right|\right)$,
      gdzie $f(n) = n \sin^2(n)$,
   2. $f(n) = \Theta(n \lg n)$, $f(n) = O\left(n^{\lg 3}\right)$, $f(n) = \Omega(n\sqrt{n})$,
      gdzie $f(n) = \lg n^{\sqrt{n}}$,
   3. $f(n) = \Theta\left(\frac{\sqrt{n}}{n!}\right)$, $f(n) = O\left(n \lg n^{2^{\lg n}}\right)$, $f(n) = \Omega(n)$,
      gdzie $f(n) = n^2 |\sin(n)|$,
   4. $f(n) = \Theta\left(\left(\lg \frac{3n}{2}\right)^2\right)$, $f(n) = O(\sqrt{n})$, $f(n) = \Omega\left(1 - n^{-1}\right)$,
      gdzie $f(n) = \lg n\sqrt{n}$.
3. Które z następujących własności są prawdziwe i dlaczego?
   1. $\sum_{i=1}^{n} \lg i = \Omega(n)$.
   2. $\lg \sum_{i=1}^{n} i = \Omega(\sqrt{n})$.
4. Oszacuj za pomocą notacji Θ i uporządkuj niemalejąco, ze względu na rząd wielkości, następujące funkcje zmiennej n:

1. $f_1(n) = \lg^2(3n)$, $f_2(n) = n \lg n! + n$, $f_3(n) = \sqrt{n} + n$, $f_4(n) = 3^{\sqrt{n}} + n \lg n$, $f_5(n) = n\sqrt{\cos^2(n)}$, $f_6(n) = \frac{3}{n} + 2$,
2. $f_1(n) = n^2 + \sin^2\left(\frac{n}{2}\right)$, $f_2(n) = \lg n^3 + n$, $f_3(n) = 2^n + \lg n!$, $f_4(n) = n \lg n + 2\sqrt{n}$, $f_5(n) = 8^{\lg n} + n$, $f_6(n) = \lg \lg\ n^2$,
3. $f_1(n) = 3^{2n} + 3^n$, $f_2(n) = \lg n + n^{\frac{1}{4}}$, $f_3(n) = n^3 \lg n + n^2 |\sin(2n)|$, $f_4(n) = 2^n + n2^{\lg n}$, $f_5(n) = \sqrt{\cos^2(n)} + \lg^2(n)$, $f_6(n) = \lg n^{2^{\lg n}} + \frac{1}{n^2}$,
4. $f_1(n) = n! + n \lg n$, $f_2(n) = 2n^3 + 3^n$, $f_3(n) = n \lg n! + n^2$, $f_4(n) = 3^{\sqrt{n}+2} - n^7$, $f_5(n) = 3|\sin(n)| + n$, $f_6(n) = 3^n + 3^{3n-4}$.

5. Która z wymienionych własności jest prawdziwa, jeżeli wiadomo, że:

- $g(n) = O(f(n))$ i $h(n) = O(f(n))$,
- $g(n) = \Omega(f(n))$ i $h(n) = \Omega(f(n))$,

gdzie $f(n)$, $g(n)$, $h(n)$ są funkcjami określonymi w zbiorze liczb naturalnych o wartościach w zbiorze liczb rzeczywistych dodatnich:

1. $g(n) + h(n) = O(f(n))$,
2. $g(n) - h(n) = \Omega(f(n))$,
3. $g(n) \cdot h(n) = O(f(n))$,
4. $g(n)/h(n) = \Omega(f(n))$, dla $h(n) \neq 0$.

6. Niech Alg będzie algorytmem, którego złożoność wyraża się pewną funkcją $f(n)$, gdzie n jest rozmiarem danych wejściowych. Czas wykonania algorytmu Alg dla danych rozmiaru x na komputerze K wynosi t sekund.

- Ile czasu zajmie wykonanie algorytmu Alg dla danych rozmiaru p-krotnie mniejszego na komputerze K?
- Jaki jest maksymalny rozmiar danych, jakie algorytm Alg możne przetworzyć na komputerze K w ciągu t' sekund?
- Oblicz czas, w jakim komputer K', p'-krotnie szybszy od komputera K, obliczy rezultat algorytmu dla danych wejściowych rozmiaru x',

gdzie:

1. $f(n) = \lg n,\ x = 128,\ t = 7,\ p = 2,\ t' = 20,\ p' = 8,\ x' = 512,$
2. $f(n) = n^2,\ x = 4,\ t = 10,\ p = 4,\ t' = 250,\ p' = 5,\ x' = 50,$
3. $f(n) = 2^n,\ x = 6,\ t = 64,\ p = 3,\ t' = 40,\ p' = 4,\ x' = 8.$

7. Niech Alg_1, Alg_2 i Alg_3 będą algorytmami o następującej złożoności czasowej względem danych rozmiaru n:

- $T(Alg_1, n) = \Theta(n \lg n)$,
- $A(Alg_2, n) = \Theta(n)$, $W(Alg_2, n) = O\left(n^2\right)$,
- $A(Alg_3, n) = \Theta(\sqrt{n})$, $W(Alg_3, n) = \Omega(n \lg n)$.

Określ możliwie dokładnie złożoność czasową następujących algorytmów:

1.
```
void Algorytm(int n) {
  for (i:=0;i<n;i:=i+1)
    Alg1(n);
}
```

2.
```
void Algorytm(int n) {
  for (i:=0;i<n;i:=i+1) {
    Alg2(n);
    Alg3(n);
  }
}
```

3.
```
void Algorytm(int n) {
  for (i:=0;i<n;i:=i+1)
    Alg1(n);

  for (i:=0;i<n;i:=i+1) {
```

```
    Alg2(n);
    Alg3(n);
  }
}
```

4.

```
void Algorytm(int n) {
  for (i:=0;i<n;i:=i+1) {
    Alg1(n);
    for (j:=0;j<n;j:=j+1) {
      Alg2(n);
      Alg3(n);
    }
  }
}
```

5.

```
void Algorytm(int n) {
  for (i:=0;i<n;i:=i+1)
    Alg1(n);

  for (i:=0;i<lg(n);i:=i+1) {
    Alg2(n);
    Alg3(n);
  }
}
```

6.

```
void Algorytm(int n) {
  for (i:=0;i<n;i:=i+1) {
    Alg1(n);
    for (j:=0;j<n*lg(n);j:=j+1) {
      Alg2(n);
      Alg3(n);
    }
```

```
    }
  }
```

8 Dla podanej poniżej funkcji Run, warunku początkowego $a \in \mathbb{N}$, $b \in \mathbb{N}$ uzasadnij, że formuła $s \cdot p^w = a^b$ jest niezmiennikiem pętli w tym algorytmie oraz ustal warunek końcowy. Co jest wynikiem działania tej funkcji? Jaka jest jej złożoność czasowa wyrażona względem wielkości liczb a i b?

```
int RUN(int a, int b) {
  int s:=1, p:=a, w:=b;

  while (w>0) {
    if (w mod 2=0) {
      p:=p*p;
      w:=w/2;
    } else {
      s:=s*p;
      w:=w-1;
    }
  }

  return s;
}
```

$WP: a \in \mathbb{N},\ b \in \mathbb{N}$ (przy `int RUN(int a, int b) {`)

$WK: ?$ (przy `return s;`)

9 Następujący algorytm oblicza kwadrat liczby naturalnej n. Udowodnij, że formuła $i \leq 2n-1,\ i = 2k-1,\ s = k^2$ jest niezmiennikiem pętli w podanym algorytmie.

```
int SQR(int n) {
  int s:=1, i:=1, k:=1;

  while (i<2n-1) {
    i:=i+2;
    s:=s+i;
    k:=k+1;
  }
```

$WP: n \in \mathbb{N} \setminus \{0\}$ (przy `int SQR(int n) {`)

```
  return s;  ←———————————————————————— | WK : s = n^2
}
```

10 Dla podanej poniżej funkcji Horner, warunku początkowego $n \in \mathbb{N}$, $x \in \mathbb{R}$ wykaż poprawność warunku końcowego $s = \sum_{j=0}^{n} T[j]x^j$, gdzie T jest n-elementowym wektorem liczb rzeczywistych reprezentującą współczynniki pewnego wielomianu zmiennej x (podpowiedź: uzasadnij, że formuła $s = \sum_{j=i}^{n} T[j]x^{(j-i)}$ jest niezmiennikiem pętli w tym algorytmie).

```
real HORNER(int x, int n, real T[]) {←——————— | WP : n ∈ ℕ, x ∈ ℕ
  int i:=n;
  real s:=T[n];

  while (i>0) {
    s:=s*x;
    s:=s+T[i-1];
    i:=i-1;
  }

  return s;←———————————————————————— | WK : s = Σ_{j=0}^{n} T[j]x^j
}
```

11 Rozważ poniższy algorytm, gdzie wszystkie elementy wektora T rozmiaru n są parami różne. Oceń prawdziwość następujących stwierdzeń:

1. niezmiennikiem pętli zewnętrznej jest formuła $T[k] > T[k-1]$ dla wszystkich $0 \leq k < i$,
2. jeżeli $T[0]$ jest elementem najmniejszym w tablicy T, to po wykonaniu algorytmu także tak jest,

3. liczba porównań elementów tablicy T, jakie wykona algorytm dla $n = 4$, jest równa 8.

```
void RUN(int n, int T[]) {
  int i, j, min;

  for (i:=0;i<n-1;i:=i+1) {
    min:=i;
    j:=i+1;
    while (j<=n) {
      if (T[j]<T[min]) min:=j;
      j:=j+1;
    }

    SWAP(T,i,min);
  }
}
```

$WP : n \in \mathbb{N} \setminus \{0\}$

12. Zaproponuj algorytm obliczania wartości współczynnika dwumianowego Newtona $\binom{n}{k}$, dla dowolnych wartości parametrów n i k:

1. korzystając wprost z definicji $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
2. korzystając z trójkąta Pascala.

Porównaj koszty tych algorytmów. Ustal niezmienniki pętli, tak by umożliwiły uzasadnienie poprawności przedstawionych algorytmów.

13. Zaproponuj algorytm obliczania wartości n-tej liczby Fibonacciego w wersji:

1. iteracyjnej,
2. rekurencyjnej.

Uzasadnij poprawność i oszacuj złożoność rozwiązania.

14. Zaproponuj algorytm iteracyjny dla problemu „Wierz z Hanoi” i n krążków w wersji:

    1. iteracyjnej,
    2. rekurencyjnej,
    3. iteracyjnej dla n-krążków,
    4. rekurencyjnej dla n krążków,

    Uzasadnij poprawność i oszacuj złożoność rozwiązania.

15. Dane są liczby naturalne n i k, gdzie $n, k > 0$. Zaproponuj algorytm obliczania części całkowitej logarytmu z n przy podstawie k. Uzasadnij poprawność i oszacuj złożoność rozwiązania.

16. Dane są liczby naturalne n i k zapisane w systemie dziesiętnym. Zaproponuj algorytm, który podaje reprezentację liczby n w systemie o podstawie k. Uzasadnij poprawność i oszacuj złożoność rozwiązania.

17. Dana jest liczba naturalna n. Zaproponuj algorytm badania, czy jest to liczba pierwsza. Uzasadnij poprawność i oszacuj złożoność rozwiązania.

18. Dane są dwie liczby naturalne a i b reprezentowane przez tablice A i B zawierające kolejne cyfry rozwinięcia dziesiętnego tych liczb. Zaproponuj algorytm realizujący tzw. mnożenie pisemne tych liczb. Uzasadnij poprawność i oszacuj złożoność rozwiązania.

19. Dana jest liczba n i ciąg n liczb rzeczywistych zapisanych w tablicy Tab. Zaproponuj algorytm, który znajduje najdłuższy podciąg rosnący tego ciągu. Uzasadnij poprawność i oszacuj złożoność rozwiązania.

20. Rozważ algorytm

```
int RUN(int n) {←——————————————————————————| WP : ?
  int m:=3, wynik:=0;

  while (m<n+1) {
    m:=m*3;
    wynik:=wynik+1;
  }

  return wynik;←—————————————————————————————| WK : ?
```

```
}
```

Zaproponuj taką specyfikację, względem której algorytm ten byłby całkowicie poprawny w strukturze liczb naturalnych. Uzasadnij poprawność algorytmu stosując metodę niezmienników. Napisz, jaki problem rozwiązuje ten algorytm, tzn. scharakteryzuj zależność między wartością n, a zmienną wynik. Oszacuj jego koszt ze względu na wybraną operację dominującą.

21 Niech Run będzie następującym algorytmem

```
int Run(int x, int y) {←——————————————————————| WP : x ∈ ℕ, , y ∈ ℕ
  int z:=0;

  while (y!=0) {
    if (y mod 2=1) z:=z+x;
    x:=x*2;
    y:=y div 2;
  }

  return z;←——————————————————————————————————| WK : ?
}
```

Podaj niezmiennik pętli while występującej w tym algorytmie wiedząc, że x i y są liczbami naturalnymi. Podaj warunek końcowy WK, tak by algorytm Run był poprawny względem specyfikacji. Warunek WK powinien opisywać zależność między początkowymi wartościami zmiennych x, y, a wartościami x, y, z po wykonaniu pętli. Uzasadnij poprawność algorytmu Run względem wybranej specyfikacji. Oszacuj koszt czasowy algorytmu.

22 Niech M będzie macierzą kwadratową rzędu n. Rozważ poniższy algorytm Run. Zaproponuj specyfikację i udowodnij całkowitą poprawność tego algorytmu ze względu na podaną specyfikację. Uzasadnij poprawność i oszacuj złożoność rozwiązania.

```
int Run(int M[][], int n) {←——————————————————| WP : ?
  int i, j;

  M[0,0]:=1;
  for (i:=1;i<n;i:=i+1) {
```

```
    M[i,0]:=1;
    M[i,i]:=1;
    for (j:=1;j<=i;j:=1) M[i,j]:=M[i-1,j-1]+M[i-1,j];
  }

  return M[n-1,n-1];
}
```
WK : ?

23 Dana jest funkcja Run następującej postaci:

```
// wariant I
int RUN(int n) { // WP : n ≥ 0
  if (n=0) return 1;

  return RUN(n-1)+RUN(n-1);
}

// wariant II
int RUN(int n) { // WP : n ≥ 0
  if (n=0) return 1;

  if (n mod 2=0) return RUN(n-1)+RUN(n-1);
  else return RUN(n-1);
}
```

1. Podaj ogólny wzór (zwartą postać) na wartość wynikową funkcji $Run(n)$.
2. Oszacuj za pomocą notacji Θ złożoność czasową i pamięciową rozważanej funkcji względem:
   - wartości zmiennej n,
   - liczby bitów niezbędnej do zapisania wartości zmiennej n.
3. Zaproponuj funkcję FastRun, która daje te same wyniki co funkcja Run, ale ma złożoność obliczeniową mniejszego rzędu niż funkcja Run i używa jedynie operacji arytmetycznych dodawania, odejmowania, mnożenia i dzielenia. Uzasadnij poprawność rozwiązania korzystając z metody niezmienników.

24. Korzystając z twierdzenia o rekurencji uniwersalnej podaj rozwiązania następujących równań rekurencyjnych:

1. $T(n) = 3T\left(\frac{n}{3}\right) + n^3 + n$,
2. $T(n) = 2T\left(\frac{n}{\sqrt{2}}\right) + \sqrt{n}$,
3. $T(n) = 8T\left(\frac{n}{2}\right) + \frac{1}{n}$,
4. $T(n) = 9T\left(\frac{n}{3}\right) + 3\lg n!$,
5. $T(n) = T\left(\frac{n}{2}\right) + 1$.