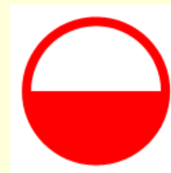
ALGORYTMY I STRUKTURY DANYCH
WYKŁAD I (materiały pomocnicze)

Wstęp



Polsko Japońska Wyższa Szkoła Technik Komputerowych

Warszawa, 12 października 2008

Plan wykładu:

- notacja asymptotyczna,
- pojęcie algorytmu,
- koszt algorytmu,
- złożoność algorytmu,
- optymalność algorytmu,
- pojęcie struktury danych,
- poprawność algorytmu,

Plan wykładu c.d.:

- grafy:
  - definicje podstawowe,
  - implementacje grafów,
- drzewa:
  - definicje podstawowe.
  - implementacje drzew binarnych,
  - implementacje drzew n-arnych,
- rekurencja:
  - metody przechodzenia drzew binarnych,
  - generowanie permutacji,
- analiza złożoności algorytmów rekurencyjnych – uogólnienie.

Notacja asymptotyczna

Notacja asymptotyczna – rzędy funkcji

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 $\lg n$),

Notacja asymptotyczna – rzędy funkcji

- $g(n) = n^{\frac{1}{c}}$, 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.

Pytanie. Jaki jest rząd funkcji $f(n) = n!$ względem przedstawionych rzędów funkcji?

Pytanie. Jaki jest rząd funkcji $f(n) = \lg n!$ względem przedstawionych rzędów funkcji?

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.

Notacja asymptotyczna – o funkcji $\lg n!$ nieco dokładniej

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:

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

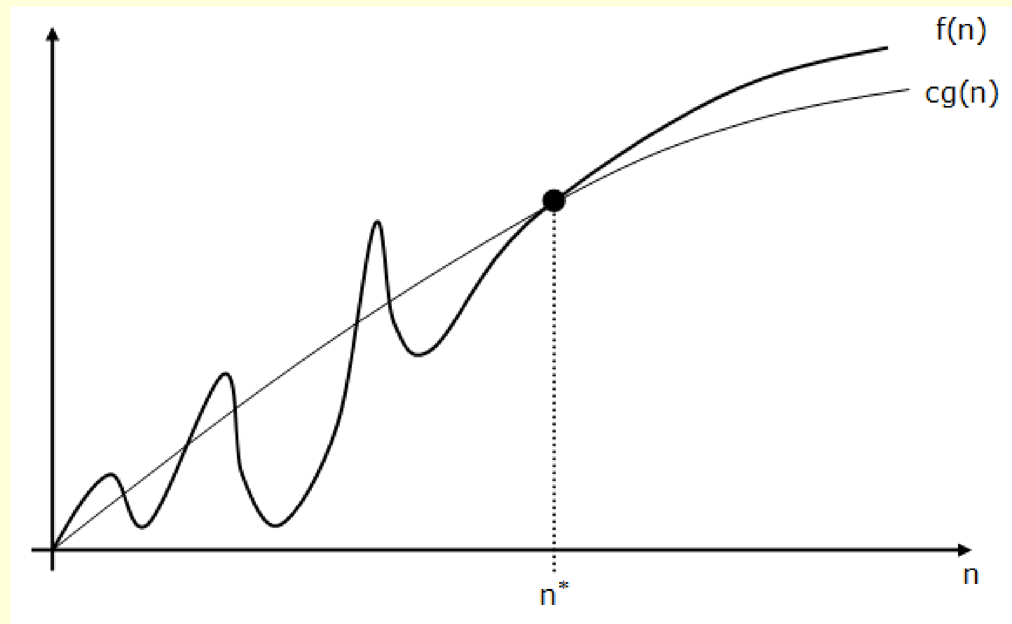
gdzie $c \leq 1$ jest pewną dodatnią stałą, czyli $\lg n! \geq c \cdot n \lg n$.

Wniosek. Ponieważ $\lg n! \leq n \lg n$ i $\lg n! \geq c \cdot n \lg n$, dla $0 < c \leq 1$, to $\lg n! \asymp n \lg n$.

Zadanie (*). Wyznacz możliwie dokładnie funkcję odwrotną do funkcji $f(n) = \lg n!$ w dziedzinie $\mathbb{R}^+ \setminus (0, 1)$.

Notacja asymptotyczna – ograniczenie z dołu

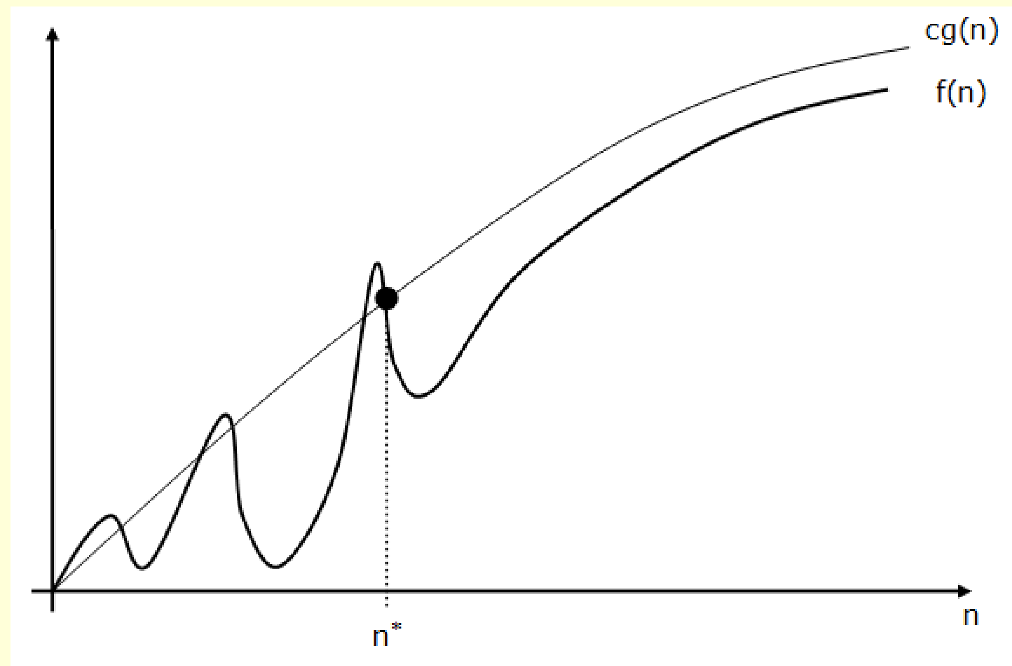
Definicja. Niech $f : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ i $g : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ będą funkcjami. Jeżeli $f \succ g$ albo $f \asymp g$, to mówimy że rząd funkcji f jest *równy co najmniej* rzędowi funkcji g (lub funkcja g *ogranicza z dołu* funkcję f) i oznaczamy przez $f =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



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

Notacja asymptotyczna – ograniczenie z góry

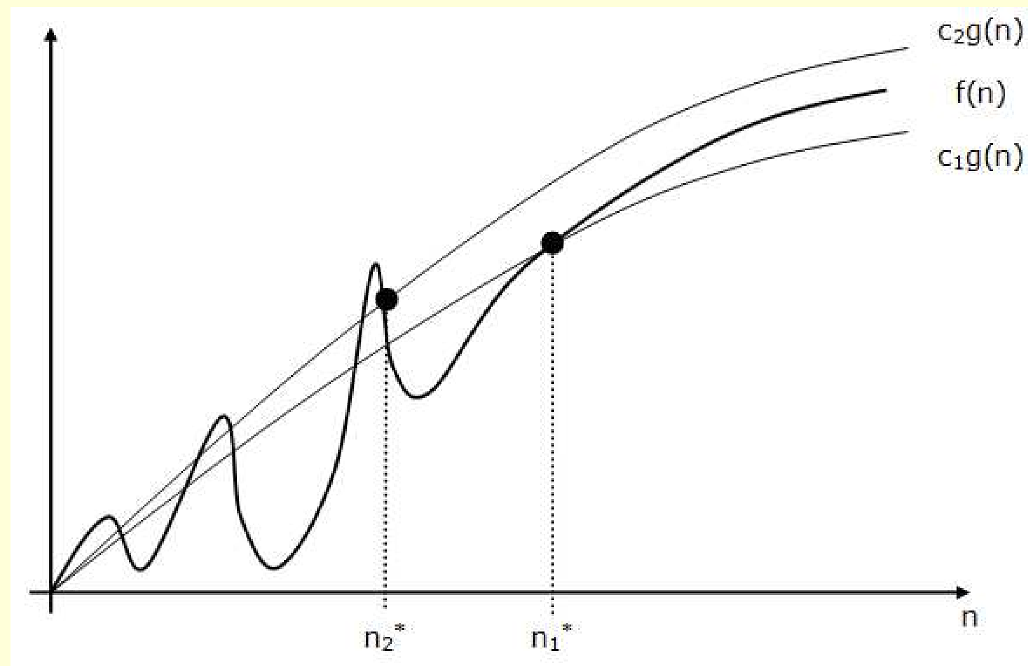
Definicja. Niech $f : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ i $g : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ będą funkcjami. Jeżeli $f \prec g$ albo $f \asymp g$, to mówimy że rząd funkcji f jest *równy co najwyżej* rzędowi funkcji g (lub funkcja g *ogranicza z góry* funkcję f) i oznaczamy przez $f =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



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

Notacja asymptotyczna – ograniczenie z dołu i z góry

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 $c_1, c_2 > 0$ są pewnymi stałymi.

Pojęcie algorytmu

Pojęcie algorytmu – wprowadzenie

Idea dla nie informatyka. Algorytm to metoda postępowania, która prowadzi do rozwiązania postawionego problemu.

Przykład. Metoda prania w pralce automatycznej:

- 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.

Pojęcie algorytmu – wprowadzenie

Idea dla informatyka. Algorytm to skończony ciąg etapów, które pozwalają przekształcić dane informacje wejściowe (dane wejściowe) w informacje wyjściowe (dane wyjściowe).

Przykład. Algorytm obliczania silni liczby naturalnej n:

- na s podstaw jeden a na i podstaw zero,
- dopóki $i < n$ wykonaj
  - zwiększ i o jeden,
  - na s podstaw wynik iloczynu s i i,
- wynikiem jest $s = n!$.

Pojęcie algorytmu – jak zapisywać algorytmy

Rozwiązanie dla nie informatyka. Język naturalny, który niestety w wielu przypadkach jest niejednoznaczny co powoduje, że konstrukcje wyrażone w tym języku także są niejednoznaczne.

Przykład. „Z życia wzięte", informacja dotycząca zniżek dla dzieci w opłacie hotelowej, tj. metoda doboru zniżki w zależności od wieku dziecka:

- jeżeli twoje dziecko ma mniej niż 10 lat, to jeżeli twoje dziecko ma więcej niż 6 lat, to przysługuje zniżka 50%, w przeciwnym przypadku przysługuje zniżka 100%, a w każdym innym przypadku przysługuje zniżka 0%.

Pytanie. Jaka zniżka przysługuje dziecku w wieku 4 lat?

Pytanie. Jaka zniżka przysługuje dziecku w wieku 8 lat?

Pytanie. Jaka zniżka przysługuje dziecku w wieku 12 lat?

Pojęcie algorytmu – jak zapisywać algorytmy

Rozwiązanie dla informatyka. Język (lub pseudojęzyk) programowania o jednoznacznej składni i semantyce.

Przykład. Algorytm obliczania silni liczby naturalnej n:

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

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

   return s;
}
```

Uwaga! W dalszej części wykładu będziemy stosowali pseudojęzyk programowania o składni i semantyce zbliżonej do języków C++ i Java.

Koszt algorytmu

Koszt algorytmu

Idea. Operacją dominującą w algorytmie nazywamy ten jego element, którego wykonanie uważamy za najbardziej kluczowe z punktu widzenia np. implementacji i uruchomienia na danym komputerze.

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

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 $t\left(Alg, d\right)$.

Pytanie. Jaki jest koszt czasowy algorytmu Silnia dla argumentu wejściowego $n = 10$?

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 $s\left(Alg, d\right)$.

Pytanie. Jaki jest koszt pamięciowy algorytmu Silnia dla argumentu wejściowego $n = 10$?

Złożoność algorytmu

Złożoność algorytmu

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 Silnia?

Pytanie. Jakie jest ograniczenie dolne złożoności czasowej algorytmu Silnia?

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 Silnia?

Pytanie. Jakie jest ograniczenie górne złożoności czasowej algorytmu Silnia?

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

Złożoność algorytmu

Problem. 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. Algorytm losowego mnożenia.

```
void LosoweMnożenie(bool A[n],int n) {
   int i,s=2;

   for (i=0;i<n;i++)
      if (A[i]==false) s=s*2;

   return s;
}
```

Pytanie. Jaka jest złożoność czasowa algorytmu LosoweMnożenie, jeżeli operacją dominującą jest działanie mnożenia?

Złożoność algorytmu

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.

Pytanie. Jaka jest złożoność czasowa algorytmu LosoweMnożenie, jeżeli operacją dominującą jest działanie mnożenia:

- w przypadku średnim,
- w przypadku pesymistycznym?

Pytanie. Jaka jest średnia i pesymistyczna złożoność czasowa algorytmu Silnia?

Złożoność algorytmu – algorytmy „efektywne”

Idea. 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?

Optymalność algorytmu

Optymalność algorytmu

Definicja. Niech $T(n)$ będzie funkcją złożoności czasowej algorytmu Alg rozwiązującego pewien problem P dla danych wejściowych rozmiaru n. Algorytm Alg nazywamy *optymalnym* (*w sensie złożoności czasowej*) 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 *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 *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.

Pojęcie struktury danych

Pojęcie struktury danych

Idea. Struktura danych, to środowisko działania algorytmu. Ten sam algorytm w rożnych środowiskach może rozwiązywać różne problemy.

Definicja. *Strukturą danych* nazywamy system algebraiczny

$$S = \langle U, o_1, o_2, \ldots, o_k, r_1, r_2, \ldots, r_n \rangle \,,$$

którego uniwersum U określa możliwe wartości zmiennych algorytmu, a operacje $o_1, o_2, \ldots, o_k$ i relacje $r_1, r_2, \ldots, r_n$ dostarczają narzędzi do realizacji algorytmu.

Przykład. Rozważmy algorytm Silnia. Domyślnie założyliśmy, że strukturą danych jest w tym przypadku następujący system algebraiczny

$$S = \langle \mathbb{N}, +, \cdot, < \rangle \,.$$

Pytanie. Jaka jest postać struktury danych dla algorytmu LosoweMnożenie?

Pytanie. Jaki problem rozwiązuje algorytm Silnia, jeżeli założymy, że struktura danych jest system algebraiczny

$$S = \langle \mathbb{N}, +, \cdot, < \rangle \,,$$

gdzie $a \cdot b =_{def} a + b$?

Poprawność algorytmu

Poprawność algorytmu

Idea. Algorytm jest poprawny, jeżeli jest zgodny zamierzeniami.

Problem. Jak formalnie opisać nasze zamierzenia?

Definicja. *Specyfikacją algorytmu* nazywamy parę

$$\langle wp, wk \rangle \,,$$

gdzie wp jest *warunkiem początkowym* a wk jest *warunkiem końcowym* algorytmu.

Przykład. Rozważmy algorytm Silnia, jego specyfikacja jest następująca:

```
int Silnia(int n) { // warunek początkowy:  n ∈ ℕ
   int s=1,i=0;

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

   return s; // warunek końcowy:  s = n!
}
```

Poprawność algorytmu

Definicja. Algorytm Alg działający w strukturze danych 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 Silnia jest częściowo poprawny w strukturze danych $S = \langle\mathbb{N}, +, \cdot, <\rangle$?

Definicja. Algorytm Alg działający w strukturze danych 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 Silnia jest całkowicie poprawny w strukturze danych $S = \langle\mathbb{N}, +, \cdot, <\rangle$?

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

Poprawność algorytmu – przykład

Przykład. Rozważmy algorytm realizujący następujący ciąg operacji:

```
bool Collatz(int n) { // warunek początkowy:  n ∈ ℕ⁺
   int i=n;

   while (i>1)
      if (i mod 2 == 0) i=i/2;
      else i=3*i+1;

   return TRUE; // warunek końcowy:  i = 1
}
```

Hipoteza Collatza. *Dla dowolnej liczby naturalnej dodatniej n prawdą jest, że* $Collatz\,(n) \equiv true$.

Wniosek. Załóżmy, że hipoteza Collatza jest fałszywa, wtedy:

- algorytm Collatz jest częściowo poprawny,
- algorytm Collatz nie jest całkowicie poprawny.

Poprawność algorytmu – poprawność algorytmów iteracyjnych

Definicja. *Niezmiennikiem pętli* nazywamy formułę, która jeżeli jest prawdziwa na początku wykonania treści pętli, to jest także prawdziwa na końcu wykonania jej treści.

Twierdzenie. *Jeżeli formuła jest prawdziwa przed wykonaniem pętli (tj. przed wejściem do pętli) i jest ona niezmiennikiem pętli, to jest także prawdziwa po wykonani pętli (tj. po wyjściu z pętli).*

Przykład. Rozważmy algorytm Silnia i formułę $s = i!$:

```
int Silnia(int n) { // warunek początkowy:  n ∈ ℕ
   int s=1,i=0;
   // 1 =_def 0! ⇒ s = i!
   while (i<n) { // s = i!
      i=i+1; // s = (i − 1)!
      s=s*i; // s/i = (i − 1)! ⇒ s = i!
   }
   // s = i! ∧ ¬(i < n) ⇒ s = i! ∧ i ≥ n ale i jest inkrementowane o 1,
   // czyli s = i! ∧ i = n ⇒ s = n!
   return s; // warunek końcowy:  s = n!
}
```

Pytanie. Która z formuł $true$, $s = n!$ jest niezmiennikiem pętli algorytmu Silnia, i która pozwala dowieść poprawności warunku końcowego $s = n!$?

Grafy

(definicje podstawowe)

Grafy – definicje podstawowe

Definicja. *Grafem* (*grafem zorientowanym*) nazywamy parę uporządkowaną $G = (V, E)$, gdzie V jest niepustym zbiorem, a E dowolnym podzbiorem produktu kartezjańskiego $V \times V$. Elementy zbioru V nazywamy *wierzchołkami* (*węzłami*) grafu, a elementy zbioru E nazywamy *krawędziami* grafu.

Pytanie. Ile rożnych grafów można utworzyć w zbiorze n wierzchołków?

Definicja. *Grafem niezorientowanym* nazywamy graf $G = (V, E)$ taki, że dla dowolnych dwóch wierzchołków $u, v \in V$ zachodzi $(u, v) \in E$ wtedy i tylko wtedy, gdy $(v, u) \in E$.

Pytanie. Ile rożnych grafów niezorientowanych można utworzyć w zbiorze n wierzchołków?

Definicja. *Grafem z wagami* nazywamy trójkę $G = (V, E, f)$, gdzie para (V, E) jest grafem, a $f : E \rightarrow \mathbb{N}^+$ jest *funkcją wag* krawędzi.

Zadanie (*). Ile różnych grafów ważonych można utworzyć w zbiorze n wierzchołków, jeżeli funkcja wag krawędzi jest postaci $f : E \rightarrow \{1, 2, 3\}$?

Grafy – definicje podstawowe

Definicja. Niech $G = (V, E)$ będzie grafem. Dwa rożne wierzchołki $u, v \in V$ nazywamy *sąsiednimi* (*incydentnymi*) wtedy i tylko wtedy, gdy $(u, v) \in E$.

Definicja. *Grafem pełnym* nazywamy graf taki, że dowolne dwa wierzchołki grafu są sąsiednie.

Pytanie. Ile krawędzi zawiera pełny graf nieskierowany o n-wierzchołkach.

Definicja. *Drogą w grafie* nazywamy ciąg wierzchołków $v_1, v_2, \ldots, v_n$ taki, że dwa wierzchołki v_i, v_{i+1} są sąsiednie, dla $i = 1, 2, \ldots, n-1$. *Długość drogi* to liczba krawędzi ją tworzących (w tym przypadku $n-1$).

Definicja. *Grafem spójnym* nazywamy graf taki, że dowolne dwa wierzchołki grafu połączone są drogą.

Definicja. Niech $v_1, v_2, \ldots, v_n$ będzie drogą w grafie. Jeżeli $v_1 = v_n$ to drogę nazywamy *drogą zamkniętą*.

Definicja. Niech $v_1, v_2, \ldots, v_n$ będzie drogą zamkniętą w grafie. Jeżeli $v_i \neq v_j$, dla $i, j = 2, 3, \ldots, n-1$, to drogę nazywamy *cyklem*.

Grafy

(implementacja grafów)

Grafy – implementacja grafów

Definicja . Niech $G = (V, E)$ będzie grafem. *Macierzą sąsiedztwa* M grafu G nazywamy macierz kwadratową rozmiaru $|V| \times |V|$, której wiersze i kolumny numerowane są, w ustalonym porządku, etykietami wierzchołków grafu i taką, że $M[u, v] = 1$ wtedy i tylko wtedy, gdy $(u, v) \in E$ oraz $M[u, v] = 0$ w przeciwnym przypadku.

Definicja. Niech $G = (V, E, f)$ będzie grafem z wagami. *Macierzą sąsiedztwa* M grafu G nazywamy macierz kwadratową rozmiaru $|V| \times |V|$, której wiersze i kolumny numerowane są, w ustalonym porządku, etykietami wierzchołków grafu i taką, że $M[u, v] = f((u, v))$ wtedy i tylko wtedy, gdy $(u, v) \in E$ oraz $M[u, v] = 0$ w przeciwnym przypadku.

Pytanie. Niech M będzie macierzą sąsiedztwa n-wierzchołkowego grafu $G = (V, E)$. Jaki jest koszt poniższych operacji na grafie G wyrażony liczbą operacji odniesień do pojedynczego elementu macierzy M:

- sprawdzenie, czy dwa wierzchołki są sąsiednie,
- wstawienie krawędzi do grafu,
- usunięcie krawędzi w grafie,
- wyznaczenie stopnia wierzchołka (stopień wierzchołka to liczba wierzchołków z nim sąsiednich)?

Grafy – implementacja grafów

Definicja. Niech $G = (V, E)$ będzie grafem. *Tablicą list incydencji* T grafu G nazywamy wektor list wierzchołków rozmiaru $|V|$, którego elementy numerowane są, w ustalonym porządku, etykietami wierzchołków grafu i taki, że $v \in T[u]$ wtedy i tylko wtedy, gdy $(u, v) \in E$.

Definicja. Niech $G = (V, E, f)$ będzie grafem z wagami. *Tablicą list incydencji* T grafu G nazywamy wektor list par (v, c), gdzie $v \in V$ i $c \in \mathbb{N}^+$, rozmiaru $|V|$, którego elementy numerowane są, w ustalonym porządku, etykietami wierzchołków grafu i taki, że $(v, c) \in T[u]$ wtedy i tylko wtedy, gdy $(u, v) \in E$ i $c = f((u, v))$.

Pytanie. Niech T będzie tablicą list incydencji n-wierzchołkowego grafu $G = (V, E)$. Jaki jest koszt poniższych operacji na grafie G wyrażony liczbą operacji przechodzenia elementów list składowych tablicy T:

- sprawdzenie, czy dwa wierzchołki są sąsiednie,
- wstawienie krawędzi do grafu,
- usunięcie krawędzi w grafie,
- wyznaczenie stopnia wierzchołka (stopień wierzchołka to liczba wierzchołków z nim sąsiednich)?

Drzewa

(definicje podstawowe)

Drzewa – definicje podstawowe

Definicja. Graf niezorientowany $G = (V, E)$ nazywamy *drzewem* wtedy i tylko wtedy, gdy graf jest spójny i acykliczny (tj. w grafie nie istnieje cykl). Jeżeli dodatkowo w grafie wyróżnimy pewien wierzchołek $v \in V$ nazywany *korzeniem*, to drzewo takie nazywamy *drzewem z ustalonym korzeniem*.

Uwaga! W dalszej części wykładu będziemy zajmowali się jedynie drzewami z ustalonym korzeniem i dla prostoty będziemy nazywali je drzewami.

Definicja. Niech r będzie korzeniem drzewa $G = (V, E)$. Wierzchołek u nazywamy *poprzednikiem* wierzchołka v wtedy i tylko wtedy, gdy w grafie G istnieje droga z wierzchołka r do wierzchołka v, na której wierzchołek u występuje tuż przed wierzchołkiem v.

Definicja. Wierzchołek v nazywamy *następnikiem* wierzchołka u wtedy i tylko wtedy, gdy wierzchołek u jest poprzednikiem wierzchołka v.

Definicja. *Wierzchołkiem wewnętrznym* nazywamy wierzchołek drzewa, który ma co najmniej jeden następnik.

Definicja. *Wierzchołkiem zewnętrznym* (*liściem*) nazywamy wierzchołek drzewa, który nie ma następnika.

Drzewa – definicje podstawowe

Definicja. *Drzewem regularnym* nazywamy drzewo n-arne, w którym każdy wierzchołek wewnętrzny ma n następników.

Definicja. *Drzewem doskonałym* nazywamy drzewo regularne n-arne, w którym wszystkie wierzchołki zewnętrzne znajdują się na jednym ustalonym poziomie.

Definicja. *Drzewem binarnym* nazywamy drzewo takie, że każdy wierzchołek drzewa ma co najwyżej dwóch następników.

Definicja. *Wysokością drzewa* nazywamy długość najdłuższej drogi w drzewie od korzenia do dowolnego z wierzchołków.

Definicja. *Poziomem wierzchołka* w drzewie nazywamy długość drogi od korzenia drzewa do tego wierzchołka.

Pytanie. Ile jest co najmniej wierzchołków w drzewie binarnym, n-arnym o wysokości h?

Pytanie. Ile jest co najwyżej wierzchołków w drzewie binarnym, n-arnym o wysokości h?

Pytanie. Ile jest co najmniej wierzchołków w drzewie binarnym, n-arnym na k-tym poziomie?

Pytanie. Ile jest co najwyżej wierzchołków w drzewie binarnym, n-arnym na k-tym poziomie?

Drzewa

(implementacja drzew)

Drzewa – implementacja drzew binarnych

Wierzchołek drzewa binarnego definiujemy jako obiekt klasy TreeNode postaci:

```
class TreeNode {

   <typ> Et;

   TreeNode left, right;

},
```

gdzie Et jest zmienną typu <typ> i reprezentuje etykietę wierzchołka drzewa, left i right to kolejno dowiązanie do lewego i prawego następnika danego wierzchołka.

Nieutworzony obiekt klasy TreeNode (tj. NULL) reprezentuje puste drzewo binarne.

Uwaga! W dalszej części wykładu mówiąc o dowiązaniu do wierzchołka drzewa binarnego, będziemy domyślnie przyjmowali przedstawioną powyżej implementację.

Zadanie. Niech zmienna $root$ reprezentuje korzeń drzewa binarnego. Podaj algorytm, który wyznaczy długość drogi od korzenia tego drzewa do „skrajnie lewego" wierzchołka.

Drzewa – implementacja drzew n-arnych

Wierzchołek drzewa n-arnego definiujemy jako obiekt klasy NTreeNode postaci:

```
class NTreeNode {
   <typ> Et;
   TreeNode successors[n];
},
```

gdzie Et jest zmienną typu <typ> i reprezentuje etykietę wierzchołka drzewa, successors[n] jest tablicą n dowiązań do następników danego wierzchołka.

Nieutworzony obiekt klasy NTreeNode (tj. NULL) reprezentuje puste drzewo n-arne.

Uwaga! W dalszej części wykładu mówiąc o dowiązaniu do wierzchołka drzewa n-arnego, będziemy domyślnie przyjmowali przedstawioną powyżej implementację.

Rekurencja

(metody przechodzenia drzew binarnych)

Rekurencja – metody przechodzenia drzew binarnych – metoda PreOrder

Zadanie. Niech $root$ będzie dowiązaniem do korzenia pewnego drzewa binarnego. Podaj algorytm, który wypisze wszystkie wierzchołki tego drzewa w czasie liniowym względem ich liczby, gdzie operacją dominującą jest czynność „przechodzania" przez wierzchołek drzewa.

Rozwiązanie.

```
void PreOrder(v TreeNode) {
   if (v==NULL) return;
   Wypisz(v); // wypisujemy wierzchołek v
   PreOrder(v.left); // odwiedzamy lewe poddrzewo
   PreOrder(v.right); // odwiedzamy prawe poddrzewo
}

PreOrder(root); // wypisujemy wszystkie wierzchołki drzewa
```

Rekurencja – metdy przechodzenia drzew binarnych – metoda PreOrder

Złożoność czasowa algorytmu. Niech $T(v)$ będzie liczbą czynności „przechodzania” przez wierzchołek drzewa binarnego o korzeniu w wierzchołku v dla funkcji PreOrder, wtedy:

$$T(v,n) = \begin{cases} 0 & \text{dla } v = NULL \\ T(v.left, n) + T(v.righ, n) + 3 & \text{dla } v \neq NULL \end{cases}.$$

Stąd $T(v,n) = 3n = \Theta(n)$.

Złożoność pamięciowa algorytmu. Niech $S(v,n)$ będzie liczbą dodatkowych jednostek pamięci niezbędnej do wykonania funkcji PreOrder, gdzie v jest dowiązaniem do korzenia drzewa binarnego o n wierzchołkach, wtedy:

- koszt pamięciowy związany z wywołaniami rekurencyjnymi rozważanej funkcji jest równy co do stałej wysokości drzewa wywołań rekurencyjnych, czyli $O(n)$,
- koszt pamięciowy związany z użyciem zmiennych pomocniczych jest stały,

stąd $S(v,n) = O(n) + O(1) = O(n)$.

Pytanie. Jaki jest dokładny koszt pamięciowy wykonania algorytmu PreOrder, jeżeli v jest dowiązaniem do korzenia doskonałego drzew binarnego o n wierzchołkach?

Rekurencja – metody przechodzenia drzew binarnych – metody InOrder, PostOrder

```
void InOrder(v TreeNode) {
   if (v==NULL) return;
   InOrder(v.left); // odwiedzamy lewe poddrzewo
   Wypisz(v)(v); // odwiedzamy wierzchołek
   InOrder(v.right); // odwiedzamy prawe poddrzewo
}

void PostOrder(v TreeNode) {
   if (v==NULL) return;
   PostOrder(v.left); // odwiedzamy lewe poddrzewo
   PostOrder(v.right); // odwiedzamy prawe poddrzewo
   Wypisz(v); // odwiedzamy wierzchołek
}
```

Rekurencja

(generowanie permutacji)

Rekurencja – generowanie permutacji

Zadanie. Niech A będzie niepustą tablicą n różnych liczb naturalnych. Podaj algorytm, który wypisze wszystkie permutacje elementów tablicy A.

Rozwiązanie.

```
void Permutacje(int A[n], int Tmp[n], int n) {
   int i;

   if (n==0) Wypisz(Tmp); // tablica Tmp reprezentuje jedną z permutacji
   else
      for (i=0;i<n;i++) {
         Tmp[n-1]=A[i];
         Permutacje(A\A[i],Tmp,n-1); // wypisujemy wszystkie permutacje,
                                        w których element A[i] znajduje się
                                        na pozycji n-tej,
      }
}
```

Rekurencja – generowanie permutacji

Złożoność czasowa algorytmu. Niech $T(n)$ będzie liczbą wywołań rekurencyjnych funkcji Permutacje, wtedy:

$$T(n) = \begin{cases} 0 & \text{dla } n = 0 \\ 1 & \text{dla } n = 1 \\ nT(n-1) & \text{dla } n > 1 \end{cases} = \begin{cases} 0 & \text{dla } n = 0 \\ n! & \text{dla } n \geq 1 \end{cases} = \Theta(n!) .$$

Złożoność pamięciowa algorytmu. Niech $S(n)$ będzie liczbą dodatkowych jednostek pamięci niezbędnej do wykonania funkcji Permutacje, wtedy:

- koszt pamięciowy związany z wywołaniami rekurencyjnymi rozważanej funkcji jest równy co do stałej wysokości drzewa wywołań rekurencyjnych, czyli $\Theta(n)$,
- koszt pamięciowy związany z użyciem tablicy pomocniczej Tmp jest równy n,

stąd $S(n) = \Theta(n) + n = \Theta(n)$.

Analiza złożoności algorytmów rekurencyjnych – uogólnienie

Analiza złożoności algorytmów rekurencyjnych – uogólnienie

Twierdzenie (o rekurencji uniwersalnej). *Niech* $a \geq 1$, $b > 1$ *będą stałymi,* $f : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ *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 to* $\left\lfloor \frac{n}{b} \right\rfloor$ *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*.*