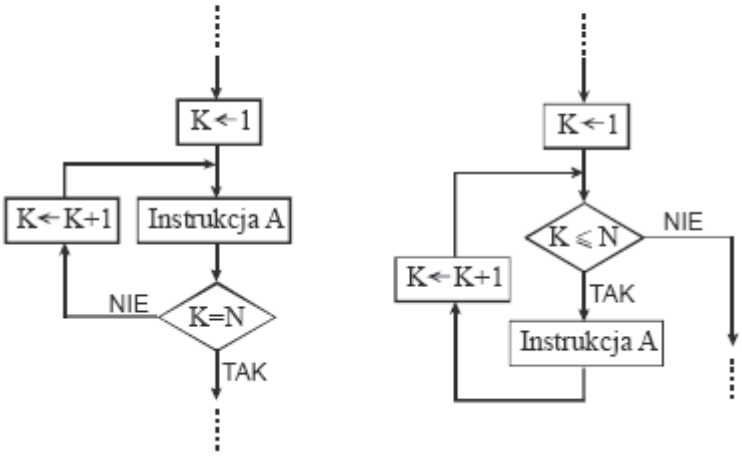
Zestawienie tematów objętych zakresem egzaminu z BAL-u:

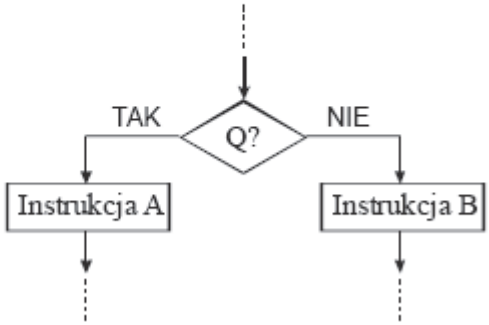
Dział 1:

- Schematy blokowe podstawowych struktur sterujących: wyboru warunkowego, pętli „dopóki”, pętli „aż do”, pętli ograniczonej.

➢ **pętla (iteracja) ograniczona –**
„wykonaj A dokładnie N razy”

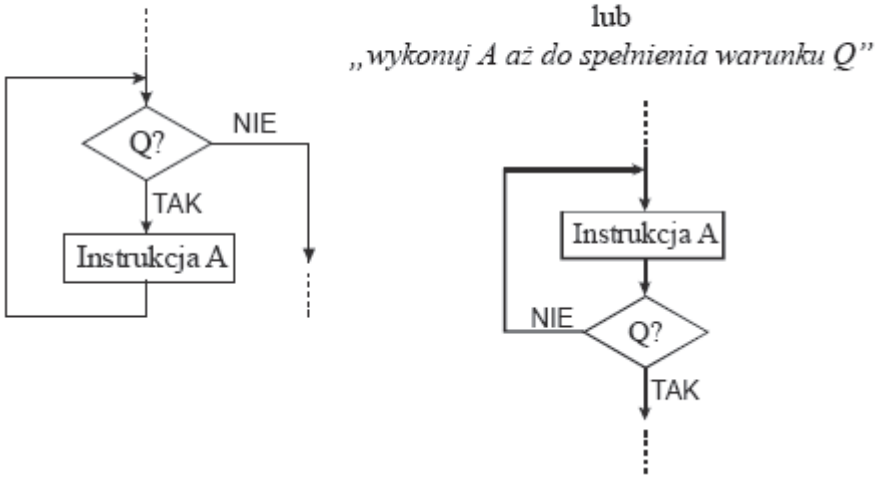


➢ **wybór warunkowy –**
„jeśli warunek Q jest spełniony, to wykonaj A,
w przeciwnym przypadku wykonaj B”



➢ **iteracja warunkowa –**
„dopóki warunek Q jest spełniony, wykonuj A”

lub
„wykonuj A aż do spełnienia warunku Q”



- Przerabianie schematu algorytmu opartego na pętli „dopóki” na schemat oparty na pętli „aż do” i odwrotnie.
- Budowanie pętli ograniczonej w oparciu o pętlę warunkową i zmienną licznikową.

- Podstawowe struktury danych: tablice jedno- i wielowymiarowe, rekordy, wskaźnikowe listy rekordów, ukorzenione drzewa rekordów (rozróżnianie struktur statycznych i dynamicznych).

> Jeśli w trakcie działania algorytmu struktura danych, utworzona przed rozpoczęciem jego działania, nie może być zmieniana w trakcie wykonywania algorytmu ani w zakresie liczby zmiennych, ani sposobu ich powiązania, to nazywamy ją **strukturą statyczną**

> W przeciwnym przypadku, jeśli dysponujemy operacjami, za pomocą których możemy modyfikować strukturę (np. usuwać jej fragmenty albo dodawać nowe) w trakcie działania algorytmu, to nazywamy ją **strukturą dynamiczną**

Struktury statyczne: tablice jednowymiarowe (wektory), tablice dwu- i więcej wymiarowe (macierze).
Struktury dynamiczne: zmienne wskaźnikowe, stos, kolejka, drzewo.

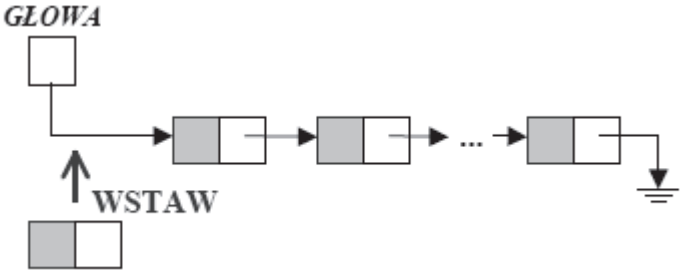
- Budowa prostych algorytmów wykorzystujących podstawowe struktury sterujące i struktury danych – organizowanie współpracy pomiędzy strukturą danych a strukturą sterującą, np. budowa algorytmów wyszukiwania wartości ekstremalnych (wielkości lub położenia), sumowania, mnożenia i normalizacji liczbowych elementów zbioru danych wejściowych umieszczonych w różnych strukturach danych.
- Opisywanie algorytmu za pomocą schematu blokowego i pseudojęzyka programowania – przechodzenie z jednego sposobu opisu na drugi.

Dział 2:

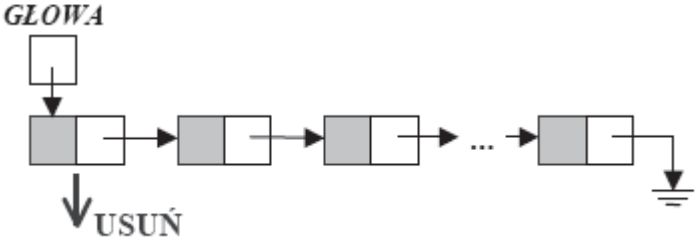
- Schemat działania stosu i kolejki, które zbudowano w oparciu o listę wskaźnikową (realizacja operacji wstawiania i usuwania rekordów z tych struktur).

STOS (zrealizowany na liście jednokierunkowej):

> Ustalamy, że nowy rekord może być wstawiony tylko jako pierwszy na liście, czyli operacja **WSTAW** nie wymaga parametru!

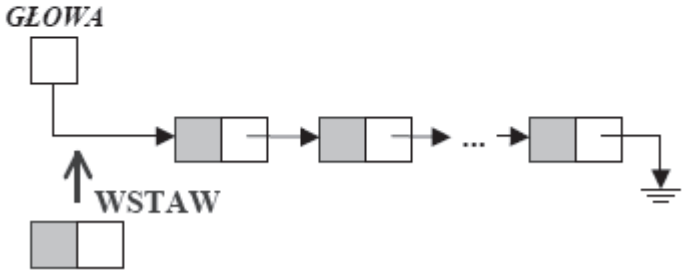


> Ustalamy, że usuwany z listy może być tylko rekord, który jest pierwszy na liście, czyli operacja **USUŃ** nie wymaga parametru!

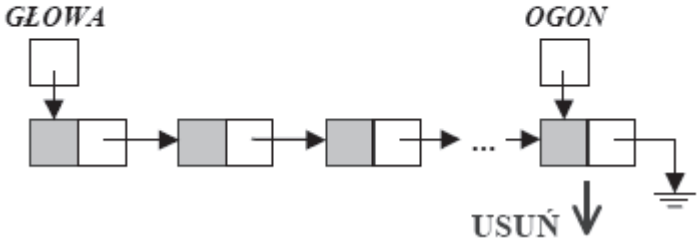


KOLEJKA (zrealizowana na liście jednokierunkowej):

Ustalamy, że nowy rekord może być wstawiony tylko jako pierwszy na liście, czyli operacja **WSTAW** nie wymaga parametru!



Ustalamy, że usuwany z listy może być tylko rekord, który jest ostatni na liście, czyli operacja **USUŃ** nie wymaga parametru!



- Zasada działania algorytmów rekurencyjnych.
REKURENCJA – wywołanie procedury wewnątrz niej samej.

- Schematy algorytmów przeszukiwania podstawowych struktur danych (współpraca struktur sterujących ze strukturami danych): pętle zagnieżdżone dla tablic wielowymiarowych, iteracyjne algorytmy przeglądu drzewa w głąb i wszerz, rekurencyjny algorytm przeglądu drzewa binarnego.

Algorytm przeglądu drzewa w głąb

(budowanie ciągu zawierającego wszystkie wierzchołki drzewa)

1. wstaw korzeń jako pierwszy element ciągu,
2. powtarzaj co następuje, aż do nadania korzeniowi etykiety „*zamknięty*”:
   - 2.1. wybierz z aktualnego ciągu ostatni wierzchołek, który nie ma etykiety „*zamknięty*”,
   - 2.2. jeśli wybrany wierzchołek nie ma potomstwa, które jeszcze nie zostało dopisane do ciągu, to nadaj mu etykietę „*zamknięty*”,
   - 2.3. w przeciwnym przypadku dopisz do ciągu pierwszego (licząc od lewej) jego potomka, który jeszcze nie występuje w ciągu.

Algorytm przeglądu drzewa wszerz

(budowanie ciągu zawierającego wszystkie wierzchołki drzewa)

1. nadaj wszystkim wierzchołkom drzewa etykietę „*nowy*”,
2. wstaw korzeń jako pierwszy element ciągu,
3. dopóki w tworzonym ciągu występuje wierzchołek z etykietą „nowy”, powtarzaj co następuje:
   - 3.1. wybierz z aktualnego ciągu pierwszy wierzchołek, który ma etykietę „*nowy*”,
   - 3.2. dodaj do ciągu wszystkich jego potomków w kolejności od lewej,
   - 3.3. usuń z wierzchołka wybranego w kroku 3.1. etykietę „*nowy*” .

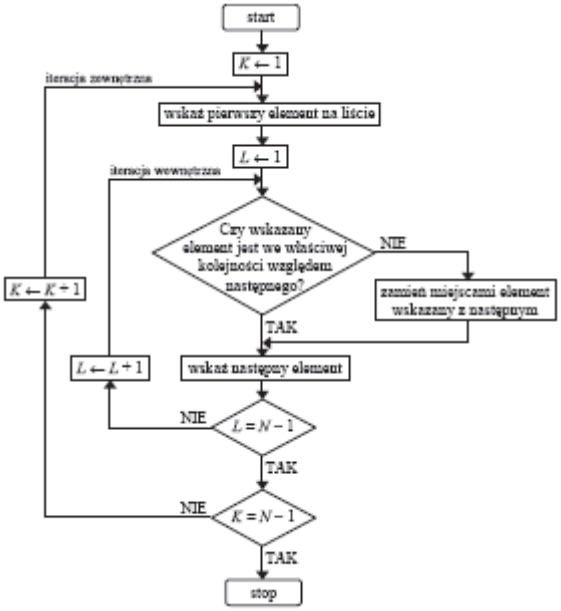
- Budowa drzewa BST.

Drzewo BST to takie drzewo binarne, w którym dla dowolnie wskazanego wierzchołka spełnione są dwa warunki:
żaden z elementów zapisanych w wierzchołkach jego lewego poddrzewa nie jest większy od elementu zapisanego w tym wierzchołku i żaden z elementów zapisanych w wierzchołkach jego prawego poddrzewa nie jest mniejszy od tego elementu .

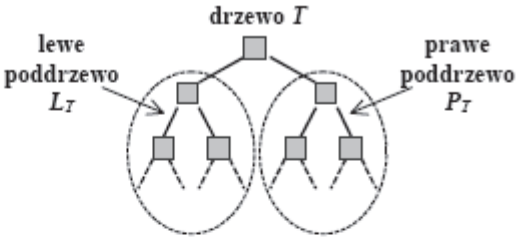
- Podstawowe algorytmy sortowania: bąbelkowego, ze scalaniem i drzewiastego (szczegóły działania).

Algorytm sortowania bąbelkowego:

1. wykonaj co następuje $n-1$ razy:
   - 1.1. wskaż pierwszy element listy,
   - 1.2. wykonaj co następuje $n-1$ razy:
     - 1.2.1. porównaj wskazany element listy z następnym
     - 1.2.2. jeśli te dwa elementy są w niewłaściwej kolejności, to zamień je miejscami,
     - 1.2.3. wskaż następny element z listy.



Procedura rekurencyjna realizująca 2. etap algorytmu sortowania drzewiastego:



Procedura ***obejdź***(T)
1. jeśli drzewo T jest puste, to wróć do poziomu wywołania,
2. w przeciwnym przypadku wykonaj co następuje:
   2.1. wywołaj ***obejdź*** (L_T),
   2.2. wypisz element umieszczony w korzeniu drzewa T,
   2.3. wywołaj ***obejdź*** (P_T),
3. wróć do poziomu wywołania.

Reguła wypisywania **IN-ORDER**

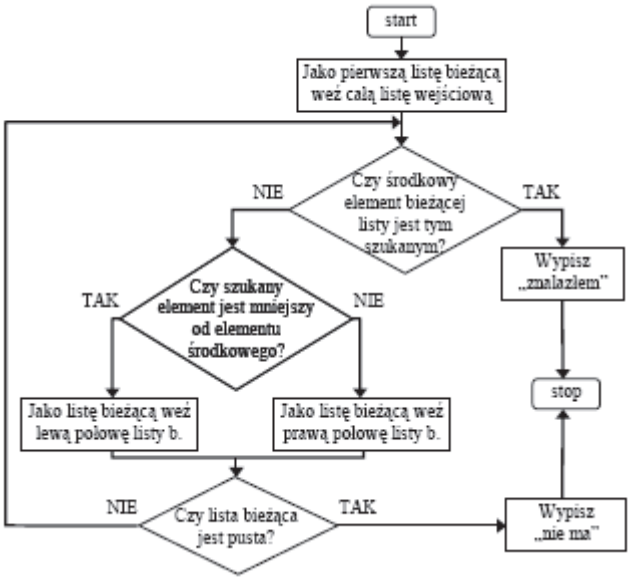
Algorytm sortowania przez scalanie

Dane wejściowe: nieuporządkowana lista (wskaźnikowa)

Procedura ***sortuj*** (L)
1. jeśli lista L zawiera tylko jeden element, to jest posortowana,
2. w przeciwnym przypadku wykonaj co następuje:
   2.1. podziel listę L na dwie połowy L_1 i L_2,
   2.2. wywołaj ***sortuj*** (L_1),
   2.3. wywołaj ***sortuj*** (L_2),
   2.4. scal posortowane listy L_1 i L_2 w jedną posortowaną listę,
3. wróć do poziomu wywołania.

- Algorytm binarnego wyszukiwania (przez połowienie) elementu z listy uporządkowanej (szczegóły działania).

Algorytm wyszukiwania binarnego (z listy uporządkowanej)



- Przybliżony algorytm wyznaczania upakowania plecaka oparty na metodzie zachłannej (szczegóły działania).

Przykład algorytmu przybliżonego
dla problemu załadunku plecaka

Problem: znajdź wartości $x_1, x_2, ..., x_N$, dla których

$$\max \sum_{i=1,...,N} c_i \cdot x_i \quad \text{i} \quad \sum_{i=1,...,N} a_i \cdot x_i \le b$$

Dane: $c_1, c_2, ..., c_N$ i $a_1, a_2, ..., a_N$ oraz b

Algorytm:

1. Uporządkuj pakowane przedmioty według wartości $\frac{c_i}{a_i}$ posortowanych od największej do najmniejszej;
2. W wyznaczonym porządku sprawdzaj kolejno, czy przedmiot zmieści się jeszcze w plecaku – jeśli tak, to go zapakuj, a jeśli nie, to go pomiń i przejdź do następnego – aż do końca listy.

Dział 3:

- Cztery metody budowania algorytmów: „wędruj i sprawdzaj", „dziel i zwyciężaj", „zachłanna", „programowanie dynamiczne" – schemat działania i przykładowe realizacje wśród algorytmów omówionych na wykładach (zapis w pseudojęzyku).

WĘDRUJ I SPRAWDZAJ

„Bądź cierpliwy, szukaj, to w końcu znajdziesz"

Algorytmy zbudowane w oparciu o jej schemat wymagają często dokonania przeglądu całej struktury danych:

- za pomocą pojedynczej pętli (iteracji), np. dla tablicy jednowymiarowej lub listy wskaźnikowej;
- za pomocą pętli (iteracji) zagnieżdżonych, np. dla tablicy wielowymiarowej lub listy list wskaźnikowych;
- za pomocą rekurencji, np. dla struktur drzewiastych.

Wędruj i sprawdzaj – przykłady:
- przegląd drzewa w głąb
- przegląd drzewa wszerz
- wyznaczanie najdłuższej przekątnej wielokąta

DZIEL I ZWYCIĘŻAJ

Jeśli nie możesz uporać się od razu z rozwiązaniem zadania w całości, to spróbuj dzielić je na mniejsze o takiej samej strukturze i stosuj rekurencyjnie procedurę rozwiązywania.

Uzyskane rozwiązania małych fragmentów zadań łącz w rozwiązania tych zadań, które były wcześniej dzielone.

Dziel i zwyciężaj – przykłady:
- sortowanie przez scalanie

METODA ZACHŁANNA

Bywają zadania, których rozwiązanie może być budowane z fragmentów dobieranych kolejno według zasady:
„idź naprzód, bierz najlepsze z tego co pod ręką i już nie oddawaj tego, co masz”.

Metoda zachłanna – przykłady:
- algorytm do wyznaczania najkrótszej sieci kolejowej

1. Wybierz najkrótszy odcinek ze wszystkich podanych,
2. Powtarzaj co następuje, aż do połączenia wszystkich punktów:
   2.1. Wyznacz zbiór odcinków, które przyłączają do sieci jakiekolwiek jeszcze nie przyłączone miasto,
   2.2. Dołącz do sieci najkrótszy odcinek z tego zbioru

PROGRAMOWANIE DYNAMICZNE

Metody algorytmiczne c.d.

Dwuetapowa metoda polegająca najpierw na stopniowym gromadzeniu dodatkowej wiedzy o wszystkich możliwych cząstkowych rozwiązaniach zadania, a potem na wykorzystaniu tej wiedzy do wybrania najlepszego rozwiązania.

Programowanie dynamiczne – przykłady:
- wyznaczanie najkrótszej drogi przejścia z punktu początkowego do końcowego.

Algorytm

Faza 1:

1. Przypisz punktowi docelowemu G wartość $L(G) = 0$,
2. Powtarzaj co następuje, aż do wyznaczenia $L(A)$:
   2.1. Wybierz taki punkt Y, dla którego wszystkie odcinki z niego wychodzące prowadzą do punktów o już wyznaczonych wartościach $L(X)$,
   2.2. Wyznacz dla wybranego punktu wartość $L(Y)$ wybierając najmniejszą sumę długości odcinka z niego wychodzącego i wartości $L(X)$ dla punktu, do którego ten odcinek prowadzi
   2.3. Zapamiętaj oprócz wartości $L(Y)$ dla jakiego punktu ta suma była najmniejsza

Faza 2:

1. Wyznacz najkrótszą drogę przejścia zaczynając od punktu A i odczytując kolejno, które kolejne punkty związane były z najmniejszymi sumami wybieranymi w 1. fazie

- Demonstracja działania przykładowych algorytmów na podanych zestawach danych wejściowych (krok po kroku).

Dział 4:

- Definicje podstawowe dla asymptotycznej analizy złożoności czasowej algorytmów prowadzonej w najgorszym przypadku.

ZŁOŻONOŚĆ CZASOWA ALGORYTMU

zależność pomiędzy rozmiarem danych wejściowych a liczbą wybranych operacji elementarnych wykonywanych w trakcie przebiegu algorytmu

(zależność podawana jako funkcja, której argumentem jest rozmiar danych, a wartością liczba operacji).

- Formalne porównywanie asymptotycznych rzędów złożoności.

W asymptotycznej analizie złożoności przyjęto, że porównując algorytmy badamy wartość ilorazu $s(N)$ dla bardzo dużych wartości N, czyli formalnie jego granicę dla $N \to \infty$.

Zatem o wyniku porównania złożoności dwóch algorytmów decyduje wartość:

$$\lim_{N\to\infty} s(N) = \lim_{N\to\infty} \frac{F_1(N)}{F_2(N)}$$

W przypadku porównywania dwóch funkcji liniowych będzie to wartość:

$$\lim_{N\to\infty} s(N) = \frac{L_1}{L_2}$$

- Posługiwanie sie rachunkiem O(×) w zakresie: mnożenia rzędu złożoności przez wartość niezależną od rozmiaru danych wejściowych, mnożenia go przez wartość funkcji zależnej od rozmiaru danych, dodawania i mnożenia rzędów złożoności przez siebie.

- Powiązanie elementów rachunku O(×) ze strukturami sterującymi analizowanych algorytmów.

- Wyznaczanie rzędu asymptotycznej złożoności algorytmu w notacji O(×) na podstawie podanego schematu blokowego lub opisu w pseudojęzyku (algorytm może zawierać podprogramy o podanych rzędach złożoności).

Asymptotyczna analiza złożoności algorytmów

Dwa algorytmy opisane funkcjami $F_1(N)$ i $F_2(N)$ mają **złożoność tego samego rzędu**, jeśli

$$\lim_{N\to\infty} \frac{F_1(N)}{F_2(N)} = C \quad \text{i zachodzi} \quad 0 < C < \infty$$

Sytuację, w której dwie funkcje złożoności są tego samego rzędu zapisujemy

$$F_1(N) = \Theta(F_2(N)) \qquad \text{(notacja theta)}$$

Algorytm opisany funkcją $F_1(N)$ ma **złożoność nie wyższego rzędu** niż algorytm opisany funkcją $F_2(N)$, jeśli

$$\lim_{N\to\infty} \frac{F_1(N)}{F_2(N)} = C \quad \text{i zachodzi} \quad C < \infty$$

Sytuację, w której pierwsza funkcja złożoności jest nie wyższego rzędu niż druga zapisujemy

$$F_1(N) = \mathrm{O}(F_2(N)) \qquad \text{(notacja O duże)}$$

Algorytm opisany funkcją $F_1(N)$ ma **złożoność niższego rzędu** niż algorytm opisany funkcją $F_2(N)$, jeśli

$$\lim_{N\to\infty} \frac{F_1(N)}{F_2(N)} = 0 \quad \text{, czyli } \boxed{C = 0}$$

Sytuację, w której pierwsza funkcja złożoności jest niższego rzędu niż druga zapisujemy

$$\boxed{F_1(N) \prec F_2(N)}$$

Dział 5:

- Rozstrzyganie całkowitej i częściowej poprawności algorytmu: definicje i rozróżnienie, istota metody niezmienników i zbieżników.

Algorytmem **całkowicie poprawnym** nazywamy algorytm, dla którego **udowodniono**, że nie zawiera on ani błędów logicznych, ani algorytmicznych.

Twierdzenie, które należy udowodnić brzmi:

„Dany algorytm dla każdego zestawu dopuszczalnych danych wejściowych samoistnie przestaje działać po wykonaniu skończonej liczby operacji elementarnych, dając poprawny (spełniający założone warunki) wynik końcowy"

Etapem pośrednim dla wykazania całkowitej poprawności może być **udowodnienie częściowej poprawności** algorytmu.

„Dla każdego zestawu dopuszczalnych danych wejściowych z faktu, że dany algorytm samoistnie przestał działać po wykonaniu skończonej liczby operacji elementarnych wynika, że dał poprawny (spełniający założone warunki) wynik końcowy"

Podstawową metodą badania **częściowej poprawności** algorytmu jest **metoda niezmienników**, polegająca na:

- wybraniu w schemacie algorytmu **punktów kontrolnych**,
- związaniu z każdym punktem kontrolnym tzw. **asercji**, czyli warunku logicznego zależnego od stanu realizacji procesu wyznaczania założonego wyniku końcowego,
- ustaleniu w obrębie każdej z iteracji takiej asercji, której prawdziwość będzie można wykazać po dowolnej liczbie powtórzeń tej iteracji (tzw. **niezmiennik** iteracji),
- wykazaniu, że z prawdziwość jednej asercji wynika prawdziwość następnej, że niezmienniki pozostają prawdziwe po kolejnych iteracjach i pociągają za sobą prawdziwość ostatniej asercji, która oznacza osiągnięcie założonego wyniku.

Po wykazaniu częściowej poprawności algorytmu podstawową metodą badania jego **całkowitej poprawności** jest **metoda zbieżników**, polegająca na:

- ustaleniu dla każdej iteracji **zbieżnika**, czyli takiej zmiennej, której wartości zależą od stanu realizacji algorytmu po wykonaniu kolejnych powtórzeń tej iteracji i tworzą ograniczony ciąg monotoniczny,
- wykazaniu, że każdy ze zbieżników nie może zmieniać się nieskończoną liczbę razy i algorytm po wykonaniu skończonej liczby powtórzeń w każdej z iteracji zatrzyma się w ostatnim punkcie kontrolnym.

Aby wykazać **częściową poprawność** algorytmu należy kolejno udowodnić prawdziwość następujących implikacji:

1. Jeżeli asercja 1. jest prawdziwa, to asercja 2. przed rozpoczęciem iteracji jest prawdziwa,
2. Jeżeli w pewnym kroku iteracji asercja 2. jest prawdziwa, to w następnym kroku też jest ona prawdziwa (warunek z asercji 2. stanie się wtedy niezmiennikiem iteracji),
3. Jeżeli w ostatnim kroku iteracji asercja 2. jest prawdziwa, to asercja 3. jest też prawdziwa (osiągnięto założony wynik).

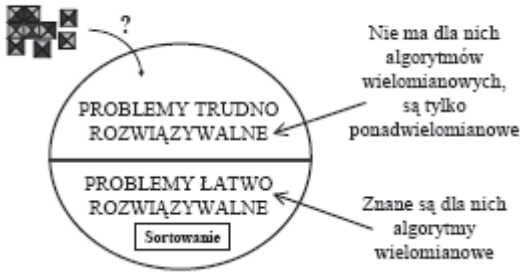
- Relacja pomiędzy analizą złożoności algorytmu a analizą złożoności problemu algorytmicznego.
Analiza złożoności problemu algorytmicznego polega na wyznaczeniu złożoności dla problemów danego typu, a analiza złożoności algorytmu dotyczy konkretnego przypadku.

- Podział na problemy o złożoności wielomianowej i ponad wielomianowej: formalne określenie, teoretyczne i praktyczne znaczenie podziału.

Funkcje złożoności dzielimy generalnie na:

- **wielomianowe**, dla których istnieje takie $k < \infty$, że są one ograniczone z góry przez funkcję N^k,
- **ponadwielomianowe**, dla których takie k nie istnieje (np. 2^N).

Dwie z klas problemów algorytmicznych:

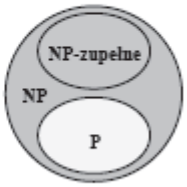


- Podział na klasy problemów *P*, *NP* i *NP-zupełne*: cechy charakterystyczne problemów z tych klas, relacje pomiędzy wymienionymi klasami, rozstrzyganie podstawowego problemu algorytmiki „*P=NP*”.

Klasa NP obejmuje problemy algorytmiczne, dla których istnieją niedeterministyczne algorytmy o złożoności wielomianowej.

Klasa P obejmuje problemy posiadające zwykłe (deterministyczne) algorytmy o złożoności wielomianowej, czyli problemy łatwo rozwiązywalne.

Klasa NP-zupełne obejmuje „wzorcowe" problemy z klasy NP „szybko" (wielomianowo) redukowalne jeden do drugiego.

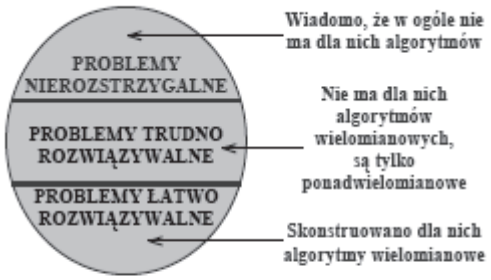


Czy P = NP ?

Zawieranie się na rysunku klasy P w klasie NP wynika ze stwierdzenia, że algorytm deterministyczny jest szczególnym przypadkiem algorytmu niedeterministycznego, w którym nie trzeba korzystać z „wyroczni".

- Podział problemów algorytmicznych na rozstrzygalne (obliczalne) i nierozstrzygalne. Przykłady nierozstrzygalnych problemów algorytmicznych.

Trzy klasy problemów algorytmicznych:



- nieograniczona liczba przypadków do sprawdzenia nie jest dostatecznym warunkiem nierozstrzygalności problemu
- nierozstrzygalność wynika z wewnętrznej struktury problemu i jest często sprzeczna z intuicją

Nierozstrzygalne problemy algorytmiczne:
- problem nieskończonego domina
- problem „węża domino" dla półpłaszczyzny
- problem „zgodności słowników"
- problem równoważności składniowej dwóch języków programowania
- problem stopu algorytmu
- problem „okresowego domina"

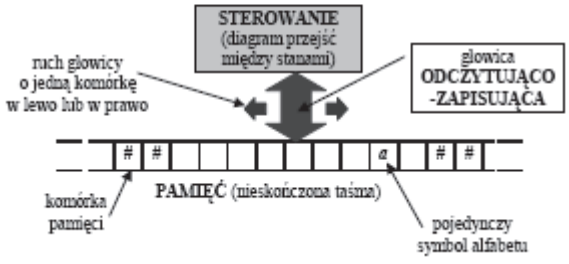
- Maszyna Turinga: czym jest, budowa i zastosowanie.

Na **MT** składa się:

- skończony **alfabet** symboli do zapisywania danych,
- skończony zbiór **stanów**, w których może się znajdować proces realizacji algorytmu zapisanego w postaci MT,
- nieskończona **taśma** podzielona na **komórki** przechowujące pojedyncze symbole alfabetu,
- krokowo poruszająca się **głowica** odczytująco-zapisująca,
- **diagram przejść** miedzy stanami, który steruje głowicą tak, że po każdym odczytaniu zawartości komórki następuje zapisanie do niej podanego symbolu, głowica jest przesuwana w lewo lub w prawo o jedną komórkę i następuje przejście do kolejnego stanu, co rozpoczyna znów tę samą sekwencję operacji,
- jeden stan **początkowy** i stany **końcowe**, których może być kilka.

Maszyna Turinga (MT)

Jeden z wielu modeli prostej MO



- Teza Churcha-Turinga i jej znaczenie dla analizy problemów algorytmicznych.

Teza Churcha-Turinga (tzw. Teza CT)

Na maszynie Turinga można zrealizować rozwiązanie każdego efektywnie rozwiązywalnego problemu algorytmicznego!

MT jest uniwersalną MO

Konsekwencje tezy CT

MT jest uniwersalną MO

⇩

Można zbudować **uniwersalną maszynę Turinga**, która może symulować działanie każdej innej maszyny Turinga z dowolnymi dopuszczalnymi danymi zapisanymi na jej taśmie (trzeba w tym celu opisać na taśmie zlinearyzowany diagram przejść, reprezentując każde przejście jako parę stanów z podaną etykietą przejścia)

Zatem konsekwencją tezy CT jest istnienie **programów uniwersalnych**, które mogą symulować działanie każdego innego programu zapisanego w dowolnym języku dla dowolnych dopuszczalnych dla tego programu danych, tzn. kończyć działanie w taki sam sposób jak symulowany program i podawać taki sam wynik, jak gdyby rzeczywiście ten program został uruchomiony dla tych danych.

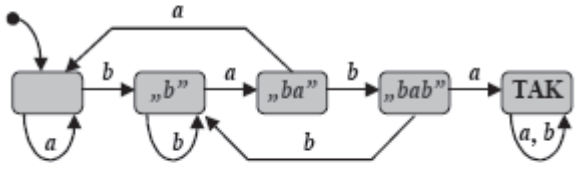
A jak teza CT ma się do rozstrzygnięcia, czy P = NP?

Jeśli chcemy formalnie zdefiniować klasy problemów P i NP, to najlepiej jest to zrobić w kategoriach obliczeń na maszynie Turinga:

- rozmiarem danych wejściowych jest liczba komórek na taśmie MT potrzebna do zapisania zakodowanych danych wejściowych,
- czas działania algorytmu mierzony jest liczbą przejść pomiędzy stanami jaka jest potrzebna do wyznaczenia zamierzonego wyniku,
- problemy z klasy **P** są rozwiązywalne w czasie wielomianowym przez zwykłe maszyny Turinga,
- problemy z klasy **NP** są rozwiązywalne w czasie wielomianowym przez niedeterministyczne maszyny Turinga.

Przykład automatu skończonego

Automat pracuje na dwuelementowym alfabecie $\{a, b\}$ i rozstrzyga czy w ciągu danych wejściowych pojawiła się choć raz sekwencja „baba".



a	b	b	a	a	b	a	b	b	a	b	a	b	a	a	#

Złożoność algorytmów

- **Złożoność pamięciowa algorytmu**
  wynika z liczby i rozmiaru struktur danych wykorzystywanych w algorytmie.
- **Złożoność czasowa algorytmu**
  wynika z liczby operacji elementarnych wykonywanych w trakcie przebiegu algorytmu.

ZŁOŻONOŚĆ CZASOWA ALGORYTMU - **zależność** pomiędzy rozmiarem danych wejściowych a liczbą operacji elementarnych wykonywanych w trakcie przebiegu algorytmu (podawana jako funkcja rozmiaru danych, której wartości podają liczbę operacji)

np.

a) w alg. sortowania bąbelkowego dla listy o długości N:
   $\boldsymbol{F}(N)$ = liczba porównań par sąsiednich elementów (?)
b) w alg. rozwiązywania problemu wież Hanoi dla N krążków:
   $\boldsymbol{F}(N)$ = liczba przeniesień pojedynczego krążka z kołka na kołek (?)
c) w alg. wyznaczania najdłuższej przekątnej wielokąta wypukłego o N wierzchołkach:
   $F(N)$ = liczba porównań długości dwóch odcinków-przekątnych (?)
d) w alg. wyznaczania „najtańszej sieci kolejowej" dla N węzłów sieci i M możliwych do zbudowania odcinków:
   $\boldsymbol{F}(N, M)$ = liczba porównań kosztów budowy dwóch odcinków (?).

W praktyce złożoność czasowa decyduje o przydatności algorytmów

Ponieważ

- trzeba rozwiązywać algorytmicznie coraz większe zadania:
  - w komputerowych systemach wspomagania decyzji,
  - przy komputerowych symulacjach i prognozach złożonych zjawisk.
- rozwijane są komputerowe systemy czasu rzeczywistego:
  - sterujące automatycznie złożonymi układami (transport, produkcja)

Chcemy zmniejszać czas wykonania algorytmu!

Przykład 1

Normalizacja wektora (tablicy jednowymiarowej) względem wartości maksymalnej; dane wejściowe zapisane w $V(1), V(2), ..., V(N)$

Algorytm 1

1. wyznacz w zmiennej *MAX* największą z wartości ;
2. **dla** *I* **od** 1 **do** *N* **wykonuj** :
   2.1. $V(I) \leftarrow V(I) \cdot 100 / MAX$

Algorytm 2

1. wyznacz w zmiennej *MAX* największą z wartości ;
2. $ILORAZ \leftarrow 100 / MAX$;
3. **dla** *I* **od** 1 **do** *N* **wykonuj** :
   3.1. $V(I) \leftarrow V(I) \cdot ILORAZ$

Jeśli operacją elementarną jest wyznaczenie wartości iloczynu lub ilorazu dwóch zmiennych,
to $F_1(N) = 2 \cdot N$ i $F_2(N) = N + 1$

Przykład 2

Wyszukiwanie liniowe elementu z listy o długości N ; dane wejściowe to lista i element do wyszukania:

Algorytm 1

1. weź pierwszy element listy ;
2. **wykonuj**:
   2.1. sprawdź czy bieżący element jest tym szukanym ;
   2.2. sprawdź czy osiągnąłeś koniec listy ;
   2.3. weź następny element z listy

   aż znajdziesz szukany element lub przejrzysz całą listę

Algorytm 2

1. dopisz szukany element na końcu listy ;
2. weź pierwszy element listy ;
3. **wykonuj**:
   - 3.1. sprawdź czy bieżący element jest tym szukanym ;
   - 3.2. weź następny element z listy

   aż znajdziesz ;
4. sprawdź czy jesteś na końcu listy

Jeśli operacją elementarną jest sprawdź , to $F_1(N) = 2 \cdot N$ i $F_2(N) = N + 1$

Poprawianie złożoności algorytmu jest czymś więcej niż tylko zmniejszaniem czasu jego wykonania!

Jeżeli np. porównujemy dwa algorytmy wykonujące to samo zadanie za pomocą jednej pętli ograniczonej, w której liczba iteracji N jest wprost proporcjonalna do rozmiaru danych wejściowych, to liczby operacji elementarnych (czasy wykonania) tych algorytmów w funkcji rozmiaru danych wynoszą odpowiednio:

$$\boldsymbol{F}_1(N) = \mathrm{K}_1 + \mathrm{L}_1 \cdot N$$
$$\boldsymbol{F}_2(N) = \mathrm{K}_2 + \mathrm{L}_2 \cdot N$$

do ich porównania możemy wykorzystać iloraz $s(N) = \dfrac{F_1(N)}{F_2(N)}$

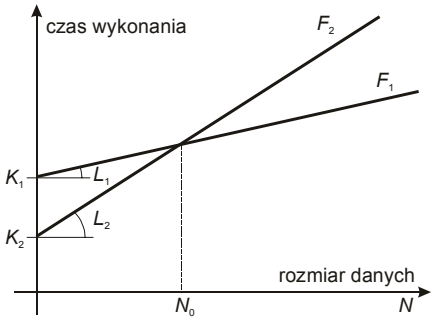
i wtedy spełnienie warunków:

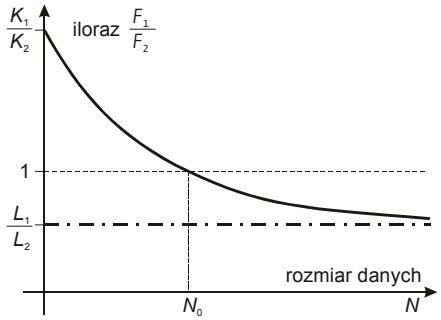
$s(N) = 1$ oznaczałoby jednakową szybkość działania
$s(N) < 1$ oznaczałoby, że 1. algorytm jest szybszy.

Ale zauważmy, że $s(N) = 1$ zachodzi tylko wtedy, kiedy $\mathrm{K}_1 = \mathrm{K}_2$ i $\mathrm{L}_1 = \mathrm{L}_2$, co oznacza, że oba algorytmy są praktycznie identyczne.
A co mamy powiedzieć, kiedy $s(N) \geq 1$ dla $N \leq N_0$, ale $s(N) < 1$ dla $N > N_0$.
Który algorytm jest lepszy?

Przyjęto, że porównując algorytmy badamy wartość ilorazu $s(N)$ dla bardzo dużych wartości N, czyli formalnie dla $N \to \infty$.

Zatem o wyniku porównania czasów działania dwóch algorytmów decyduje wartość: $\boxed{\lim_{N\to\infty} s(N) = \dfrac{L_1}{L_2}}$; jest to tzw. *analiza asymptotyczna*.





Dwa algorytmy o czasach wykonania $F_1(N)$ i $F_2(N)$ mają **złożoność tego samego rzędu**, jeśli $\lim_{N\to\infty} \dfrac{F_1(N)}{F_2(N)} = C$,

gdzie $0 < C < \infty$

Spełnienie takiego warunku oznaczamy $\boxed{F_1(N) = \Theta(F_2(N))}$

Jeśli zachodzi $F_1(N) = \Theta(F_2(N))$, to także $F_2(N) = \Theta(F_1(N))$

Algorytm o czasie wykonania $F_1(N)$ ma **nie wyższy rząd złożoności** od algorytmu o czasie wykonania $\boldsymbol{F}_2(N)$,

jeśli $\lim_{N\to\infty} \dfrac{F_1(N)}{F_2(N)} = C < \infty$

Spełnienie takiego warunku oznaczamy $\boxed{F_1(N) = \mathrm{O}(F_2(N))}$

Jeśli zachodzi jednocześnie $F_1(N) = O(F_2(N))$ i $F_2(N) = O(F_1(N))$, to $F_1(N) = \Theta(F_2(N))$.
Jeśli zachodzi $F_1(N) = \Theta(F_2(N))$, to także $F_1(N) = O(F_2(N))$.

- jeśli $C = 0$, to algorytm o czasie wykonania $F_1(N)$ ma niższy rząd złożoności (ma lepszą złożoność) od algorytmu o czasie wyk. $F_2(N)$; oznaczamy to symbolicznie $\boxed{F_1(N) \prec F_2(N)}$

- algorytm o czasie wykonania $F(N)$ ma **złożoność liniową**, jeśli $\lim_{N\to\infty} \frac{F(N)}{N} = C$, dla $0 < C < \infty$;
  złożoność rzędu N oznaczana jest symbolem $\boldsymbol{\Theta(N)}$
  (ma on złożoność co najwyżej liniową, jeśli $C < \infty$; ozn. $\boldsymbol{O(N)}$).

- algorytm o czasie wykonania $F(N)$ ma **złożoność kwadratową**, jeśli $\lim_{N\to\infty} \frac{F(N)}{N^2} = C$, dla $0 < C < \infty$;
  złożoność rzędu N^2 oznaczana jest symbolem $\boldsymbol{\Theta(N^2)}$
  (ma on złożoność co najwyżej kwadratową, jeśli $C < \infty$; ozn. $\boldsymbol{O(N^2)}$;
  zauważmy jeszcze, że $N \prec N^2$).

- sytuację, w której zachodzi warunek $\forall N : 0 \le F(N) \le C < \infty$ oznaczamy $F(N) = O(1)$.

Dopiero zmniejszenie rzędu złożoności algorytmu jest istotnym ulepszeniem rozwiązania problemu algorytmicznego!

Jeżeli algorytm wykonuje różną liczbę operacji elementarnych w zależności od konkretnych danych wejściowych (o tym samym rozmiarze) możemy badać czas wykonania **w najgorszym przypadku** (czyli skupiając się na takich przypadkach dopuszczalnych danych wejściowych, dla których ta liczba jest największa) - tzw. *analiza najgorszego przypadku* lub *pesymistyczna*.

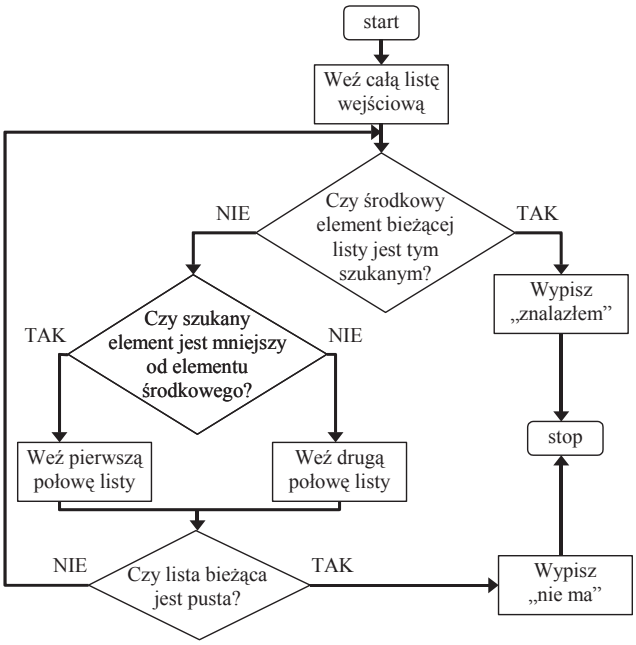
W przykładach normalizacji wektora i wyszukiwania liniowego

◊ czas wykonania Algorytmu 1. wynosi $F_1(N) = 2 \cdot N$: $F_1(N) = O(N)$
◊ czas wykonania Algorytmu 2. wynosi $F_2(N) = N + 1$: $F_2(N) = O(N)$
◊ oba mają pesymistyczny czas wykonania $O(N)$ - mogą zdarzyć się takie dane wejściowe dla wyszukiwania liniowego, dla których trzeba będzie przejrzeć cała listę o długości N
(jest tak, kiedy szukanego elementu nie ma w ogóle na liście)

Czy można zaproponować algorytm o lepszej złożoności dla wyszukiwania elementu na liście?

Szukamy zatem algorytmu o (pesymistycznym) czasie wykonania $F(N) \prec N$.

Wyszukiwanie **binarne** (przez połowienie) elementu z listy uporządkowanej:
$Y_1, Y_2, ..., Y_N$ (dla każdego $i < j$ zachodzi $Y_i \le Y_j$)



Jeśli przyjmiemy, że operacją elementarną jest porównanie elementu szukanego z jednym z elementów listy (środkowym), to analiza złożoności polega na znalezieniu odpowiedzi na pytanie:
ile razy jest powtarzana w najgorszym przypadku iteracja w algorytmie?

Odpowiedź: $1 + \log_2 N$ ($\log_2 N$ będziemy oznaczali $\lg N$)

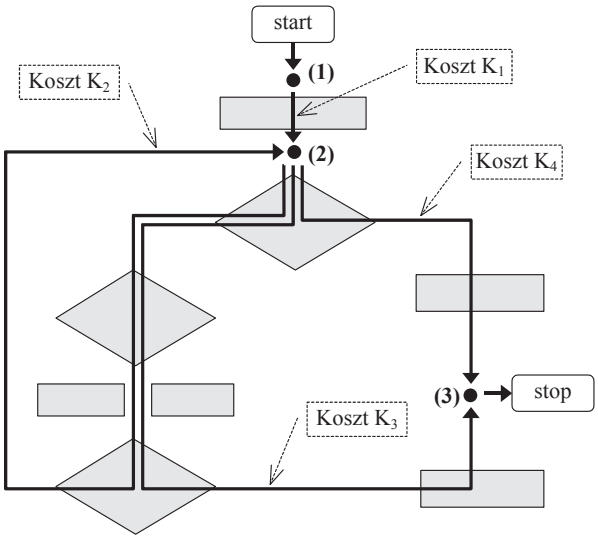
Zatem pesymistyczna złożoność algorytmu wyszukiwania binarnego wynosi **O(lg *N*)**

(mówimy, że algorytm ten ma złożoność logarytmiczną); $\lg N \prec N$

Skala ulepszenia:

N	$1 + \lfloor \lg N \rfloor$
10	4
100	7
1 000	10
10 000	14
1 000 000	20
1 000 000 000	30
1 000 000 000 000	40
1 000 000 000 000 000	50
1 000 000 000 000 000 000	60

O złożoności czasowej algorytmu decyduje liczba wykonywanych iteracji



Całkowity koszt czasowy w najgorszym przypadku:

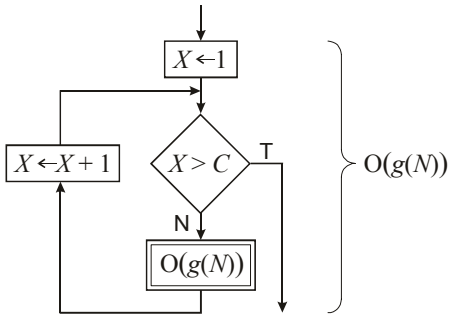
$F(N) = K_1 + \max(K_3, K_4) + K_2 \cdot \lg N$, a to oznacza $F(N) = O(\lg N)$

Właściwości notacji O(·) (tzw. rachunek O(·))

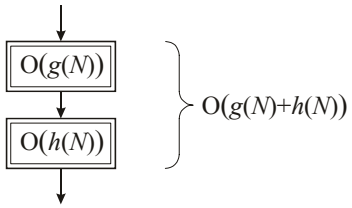
- jeśli $F(N)$ jest funkcją złożoności algorytmu, to spełnienie warunku $\lim_{N\to\infty} \frac{F(N)}{g(N)} = C < \infty$, jest zapisywane

  $F(N) = O(g(N))$; odczytujemy „algorytm ma złożoność rzędu nie wyższego niż $g(N)$”
  lub krócej „złożoność algorytmu jest $O(g(N))$”
- równość w zapisie $F(N) = O(g(N))$ powinna być rozumiana w ten sposób, że funkcja $F(N)$ jest jedną z funkcji, które spełniają powyższy warunek lub precyzyjniej, że funkcja $F(N)$ należy do zbioru wszystkich funkcji spełniających powyższy warunek
- $F(N) = O(F(N))$,
- $O(O(g(N))) = O(g(N))$,

- $C \cdot O(g(N)) = O(C \cdot g(N)) = O(g(N))$ (dla dowolnego $0 < C < \infty$),



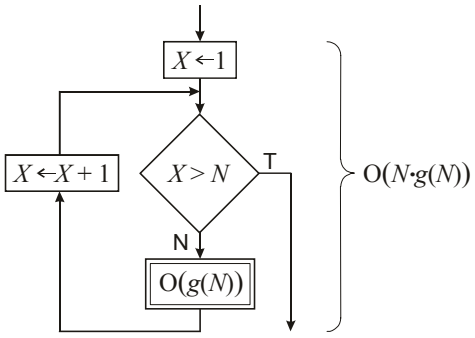
- $O(g(N)) + O(h(N)) = O(g(N) + h(N))$,

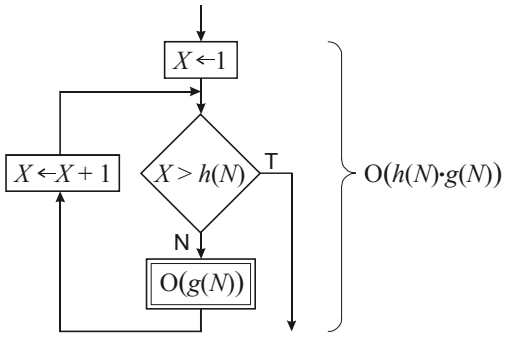


- jeśli zachodzi $\lim_{N\to\infty} \frac{g(N)}{h(N)} = 0$, czyli $g(N) \prec h(N)$,

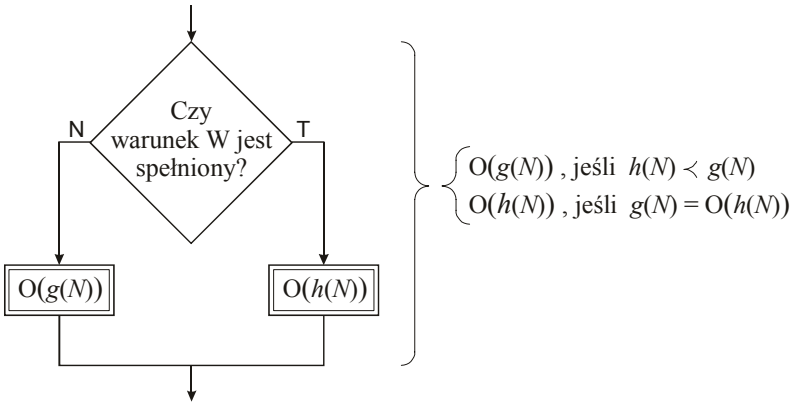
  to $O(g(N)) + O(h(N)) = O(g(N) + h(N)) = O(h(N))$

- $O(g(N)) \cdot O(h(N)) = O(g(N) \cdot h(N)) = g(N) \cdot O(h(N)) = h(N) \cdot O(g(N))$,





- przy analizie złożoności w najgorszym przypadku (dla warunku zależnego od danych wejściowych) zachodzi:



Złożoność czasowa przykładowych algorytmów

- sortowanie bąbelkowe w pierwotnej wersji (zagnieżdżone iteracje):

1. wykonuj co następuje N - 1 razy:
   1.1. ... ;
   1.2. wykonuj co następuje N - 1 razy:
   1.2.1. ... ;

Całkowity koszt czasowy w najgorszym przypadku wynosi z dokładnością do stałych:
$(N-1)\cdot(N-1) = N^2 - 2N + 1$; $1 \prec 2N \prec N^2$, czyli złożoność jest $\mathbf{O(N^2)}$

- sortowanie bąbelkowe ulepszone (coraz krótsze przebiegi):
  Całkowity koszt czasowy w najgorszym przypadku wynosi:
  $(N-1) + (N-2) + (N-3) + ... + 2 + 1 = 0{,}5\cdot N^2 - 0{,}5\cdot N$, czyli także złożoność jest $\mathbf{O(N^2)}$
- sumowanie zarobków pracowników - złożoność $\mathbf{O(N)}$
- znajdowanie największej przekątnej w wielokącie wypukłym metodą naiwną - złożoność $\mathbf{O(N^2)}$
- znajdowanie największej przekątnej w wielokącie wypukłym metodą jednokrotnego obiegu z parą prostych równoległych - złożoność $\mathbf{O(N)}$
- rekurencyjny algorytm dla problemu wież Hanoi

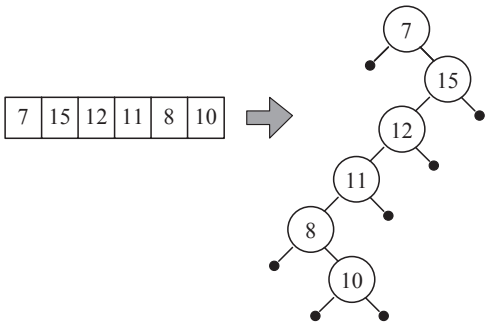
procedura ***przenieś*** N ;
1. jeśli $N = 1$, to wypisz ruch i koniec;
2. w przeciwnym razie (tj. jeśli $N > 1$) wykonaj co następuje:
   2.1. wywołaj ***przenieś*** $N-1$;
   2.2. wypisz ruch ;
   2.3. wywołaj ***przenieś*** $N-1$;

Oznaczmy nieznany koszt czasowy przez $T(N)$ i ułóżmy równania, które $T(N)$ musi spełniać (tzw. **równania rekurencyjne**):

$$T(1) = 1$$
$$T(N) = 2 \cdot T(N-1) + 1$$

Spełnia je koszt $T(N) = 2^N - 1$, czyli złożoność jest $\mathbf{O(2^N)}$

- sortowanie drzewiaste bez samoorganizacji drzewa:
  ma pesymistyczną (w najgorszym przypadku) złożoność $\mathbf{O(N^2)}$, bo wprawdzie lewostronne obejście drzewa ma złożoność liniową $\mathbf{O(N)}$, ale konstrukcja binarnego drzewa poszukiwań ma pesymistyczną złożoność kwadratową



- sortowanie drzewiaste z samoorganizacją drzewa:
  ma złożoność $\mathbf{O(N \cdot \lg N)}$, co daje wyraźną poprawę sprawności algorytmu sortowania w porównaniu z algorytmem bąbelkowym

N	N^2	$N \cdot \lg N$
10	100	33
100	10 000	664
1 000	1 000 000	9 965
1 000 000	1 000 000 000 000	19 931 568
1 000 000 000	1 000 000 000 000 000 000	29 897 352 853

- sortowanie przez scalanie (rekurencyjne):

procedura ***sortuj-listę*** *L;*

1. jeśli *L* zawiera tylko jeden element, to jest posortowana;
2. w przeciwnym razie wykonaj co następuje:
   2.1. ... ;
   2.2. wywołaj ***sortuj-listę*** *połowa L* ;
   2.3. wywołaj ***sortuj-listę*** *połowa L* ;
   2.4. scal posortowane połowy listy *L* w jedną posortowaną listę;

Oznaczmy nieznany koszt czasowy przez $T(N)$ i ułóżmy równania rekurencyjne, które $T(N)$ musi spełniać:

$$T(1) = 0$$
$$T(N) = 2 \cdot T(0{,}5 \cdot N) + N$$

Spełnia je koszt $T(N) = N \cdot \lg N$, czyli złożoność jest $\mathbf{O(N \cdot \lg N)}$

Sortowanie przez scalanie jest jednym z najlepszych algorytmów sortowania (choć ma nie najlepszą złożoność pamięciową O(*N*))

Średnia złożoność

Analiza najgorszego przypadku ***lub*** analiza średniego przypadku

W analizie średniego przypadku istotną rolę odgrywają założenia o rozkładzie prawdopodobieństwa w zbiorze dopuszczalnych danych wejściowych.

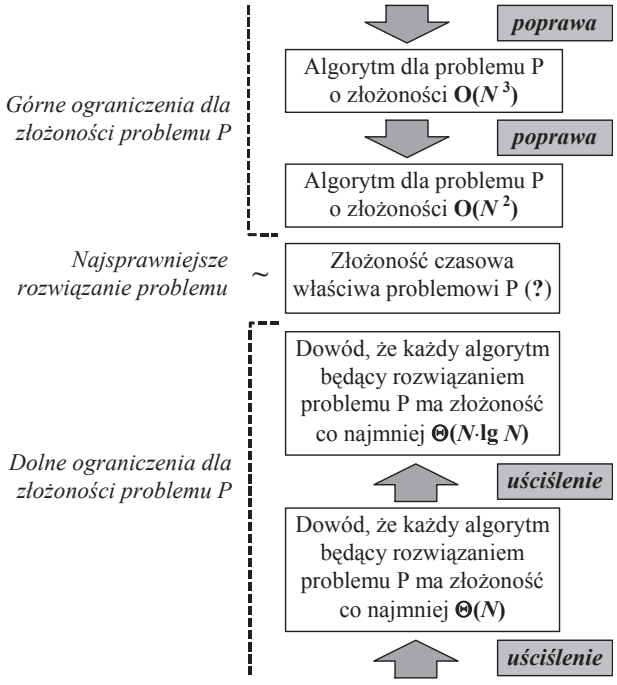
algorytm	*średni przyp.*	*najgorszy przyp.*
sumowanie zarobków	$\mathbf{O(N)}$	$\mathbf{O(N)}$
sortowanie bąbelkowe	$\mathbf{O(N^2)}$	$\mathbf{O(N^2)}$
sortowanie drzewiaste z samoorganizacją drzewa	$\mathbf{O(N \cdot \lg N)}$	$\mathbf{O(N \cdot \lg N)}$
sortowanie przez scalanie	$\mathbf{O(N \cdot \lg N)}$	$\mathbf{O(N \cdot \lg N)}$
Quicksort	$\mathbf{O(N \cdot \lg N)}$	$\mathbf{O(N^2)}$

Średni koszt czasowy algorytmu Quicksort wynosi tylko $1{,}4 \cdot N \cdot \lg N$

Dolne i górne ograniczenia złożoności problemów algorytmicznych

Czy można skonstruować jeszcze lepszy algorytm?

- złożoność poprawnego algorytmu znajdującego rozwiązanie danego problemu ustanawia **górne ograniczenie** złożoności dla tego problemu.
- **dolne ograniczenie** złożoności problemu (otrzymane w wyniku analizy samego problemu) określa zakres dalszej poprawy rzędu złożoności algorytmów rozwiązujących ten problem



Problemy zamknięte i luki algorytmiczne

problem	*dolne ogr.*	*górne ogr.*

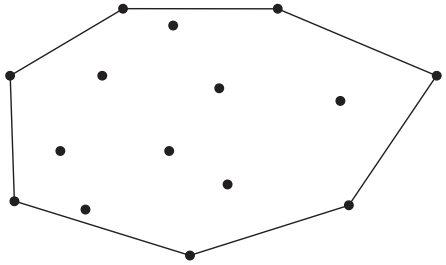
przeszukiwanie listy nieuporządkowanej	$\Theta(N)$	$O(N)$
przeszukiwanie listy uporządkowanej	$\Theta(\lg N)$	$O(\lg N)$
sortowanie	$\Theta(N \cdot \lg N)$	$O(N \cdot \lg N)$
wyznaczanie najtańszej sieci kolejowej	$\Theta(N)$	$O(f(N) \cdot N)$ [1)]

[1)] $f(N)$ - bardzo wolno rosnąca funkcja, np. dla $N = 64\,000$ ma wartość 4

- problem zamknięty - problem z luką algorytm.

Przykład analizy złożoności algorytmu

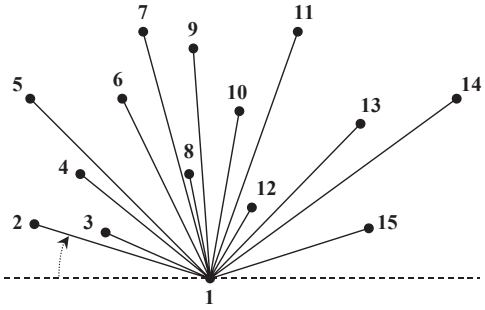
Problem wyznaczania powłoki wypukłej dla zbioru N punktów na płaszczyźnie:



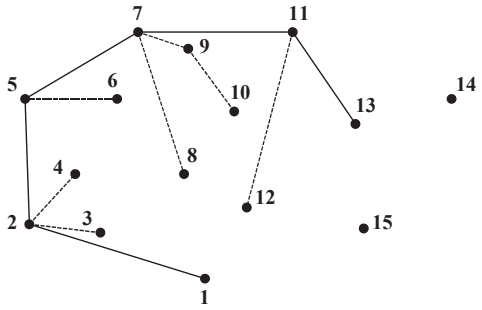
- algorytm „naiwny" ma złożoność $O(N^3)$;
  trzeba dla każdego z N punktów dobierać pozostałe N–1 do pary, która wyznacza prostą i dla każdej z tych prostych sprawdzić czy N–2 punkty leżą wszystkie po tej samej stronie prostej.
- algorytm o niższej złożoności (?):

1. znajdź punkt „najniżej" położony - P_1 ;
2. posortuj pozostałe punkty rosnąco według kątów tworzonych przez odcinki $\overline{P_1P_J}$ z linią poziomą przechodzącą przez P_1 - powstanie lista P_2, ... ,P_N ;
3. dołącz do powłoki punkty P_1 i P_2 ;
4. **dla** J **od** 3 **do** N **wykonuj** ;
   4.1. dołącz do powłoki punkt P_J ;
   4.2. cofając się wstecz po odcinkach aktualnej powłoki, usuwaj z niej te punkty P_K , dla których prosta przechodząca przez P_K i P_{K-1} przecina odcinek $\overline{P_1P_J}$, aż do napotkania pierwszego punktu nie dającego się usunąć ;

krok 2.



kolejne iteracje w kroku 4.



Podsumowanie złożoności algorytmu krok po kroku:

Krok 1.	$O(N)$	(wybór pierwszego)
Krok 2.	$O(N \cdot \lg N)$	(sortowanie)
Krok 3.	$O(1)$	(dołączenie 1 odcinka)
Krok 4.	$O(N)$	(dołączanie z usuwaniem)
Razem	$O(N+N \cdot \lg N+1+N) = O(N \cdot \lg N)$	

1. Wymień o opisz struktury sterujące stosowane do budowy algorytmów.
Podstawowe struktury sterujące to:
1.1. bezpośrednie następstwo – wykonaj instrukcję A, potem instrukcję B, potem instrukcję C, itd.
1.2. wybór warunkowy – jeśli warunek Q jest spełniony wykonaj instrukcję A, jeśli nie to wykonaj instrukcję B.
1.3. iteracja ograniczona – wykonaj instrukcję A dokładnie N razy.
1.4. iteracja warunkowa „dopóki" – dopóki warunek Q jest spełniony wykonuj instrukcję A.
1.5. Iteracja warunkowa „aż do" – wykonuj instrukcję A dopóki warunek Q jest spełniony.

2. Jaka jest konstrukcja algorytmu sortowania bąbelkowego?
Sortowanie bąbelkowe polega na przestawianiu sąsiednich par elementów stojących w niewłaściwej kolejności. Istotne jest iż ciąg elementów przeglądany jest zawsze w tym samym kierunku, a przeglądanie to trwa dopóki mogą się w nim pojawić elementy w nieodpowiedniej kolejności.
Zapis słowny algorytmu sortowania bąbelkowego:
1. wykonaj co następuje N-1 razy;
    1.1. wskaż na pierwszy element;
    1.2. wykonaj co następuje N-1 razy;
        1.1.1. porównaj ze sobą wskazany element i element następny;
        1.1.2. jeśli elementy stoją w złej kolejności to zamień je miejscami;
        1.1.3. wskaż na następny element;

3. Narysuj schemat blokowy: wyboru warunkowego, iteracji warunkowych typu: „dopóki" i „aż do".

4. Zapisz słownie i naszkicuj schemat blokowy algorytmu sumowania wektora (jednowymiarowej tablicy) – n elementowego. Wykorzystaj zmienną indeksującą i znajomość długości wektora.
Zapis słowny algorytmu sumowania n -elementowego wektora:
1. zanotuj na boku liczbę zero;
2. wskaż na pierwszy element wektora;
3. wykonaj co następuje n-1 razy;
    a. dodaj wartość wskazanego elementu do liczby zanotowanej na boku;
    b. wskaż na kolejny element wektora;
4. dodaj wartość wskazanego elementu do liczby na boku;
5. wypisz wartość liczby na boku;

5. Jakie korzyści przynosi stosowanie procedur (podprogramów) w algorytmach?
Zalety stosowania procedur są następujące:
1. oszczędność tekstu
2. znacznie większa przejrzystość i czytelność struktury algorytmu
3. znacznie większa kontrola nad poprawnością algorytmu
4. uproszczenie we wprowadzaniu poprawek i usuwaniu błędów
5. możliwość programowania analitycznego i syntetycznego

6. Na czym polega rekurencja i jak można ją wykorzystać w konstruowaniu algorytmów?
Rekurencja – to zdolność podprogramu (procedury) do wywoływania samej siebie. Przykładem zastosowania procedury rekurencyjnej jest algorytm przenoszenia krążków znany z Wież Hanoi. Tam aby wykonać pewne przeniesienie należy przy okazji wykonać inne, czyli wywołać tę samą procedurę wewnątrz procedury wywoływanej na początku.
Inne przykłady wykorzystania procedur rekurencyjnych to:
- przeglądanie lewostronne struktur drzewiastych
- obliczanie wartości n! liczby n.

7. Jakie znasz podstawowe typy danych? Jak są one kodowane binarnie?
Podstawowe typy danych to:
- liczby (całkowite, dziesiętne, dwójkowe, szesnastkowe)
- słowa (układy liter z różnych alfabetów)
- wskaźniki (dane tego typu zawierają adresy - wskazania na inne elementy w pamięci operacyjnej – dane tego typu wymagają specjalnego traktowania)

Kodowanie liczb – przeliczenie ich wartości na wartości binarne, czyli zero – jedynkowe.
Kodowanie słów - odbywa się za pomocą standardu ASCII (American Standard Code for Information Interchange). Zgodnie z nim każdemu znakowi przypisana jest liczba od 0 do 127 – kodowanie na ośmiu bitach.

8. Jakie znasz statyczne struktury danych?
Do statycznych struktur danych należą:
1. zmienne – podstawowe obiekty w pamięci, posiadające własną nazwę i zdolność przechowywania pojedynczego elementu.

2. wektory – czyli tablice jednowymiarowe – są to obiekty w pamięci mające nadaną własną nazwę i posiadające zdolność przechowywania określonej mnogości elementów, z których każdy oznaczony jest odpowiednim, unikalnym indeksem.
3. tablice dwuwymiarowe – macierze - są to obiekty w pamięci mające nadaną własną nazwę i posiadające zdolność przechowywania określonej mnogości elementów, z których każdy oznaczony jest dwoma indeksami.
4. Tablice wielowymiarowe – są to obiekty w pamięci mające nadaną własną nazwę i posiadające zdolność przechowywania odpowiedniej mnogości elementów, z których każdy oznaczony jest n – indeksami.

9. Jaka struktura sterująca byłaby właściwa do przejrzenia tablicy dwuwymiarowej?

Odpowiednią do tego zadania strukturą sterującą jest iteracja zagnieżdżona. Iteracja zewnętrzna odpowiada za przeglądanie kolumn a iteracja wewnętrzna za przegląd wierszy.

10.Z jakich obiektów są zbudowane dynamiczne struktury danych?

Dynamiczne struktury danych budowane są z dwóch głównych rodzajów obiektów:
1. zmiennych kluczowych i dodatkowych (przechowujących odpowiednie dane)
2. zmiennych wskaźnikowych (wskazujących na kolejne elementy tych struktur, lub przechowujące wartość NIL)
Rozróżniając dokładniej wyróżniamy:
a.) listy jednokierunkowe – każdy element tej struktury posiada pola kluczowe, dodatkowe i jedno pole wskaźnikowe, odwołujące się do następnego elementu struktury.
b.) listy dwukierunkowe – każdy element tej struktury posiada pola kluczowe, dodatkowe i dwa pola wskaźnikowe, odwołujące się do następnego i poprzedniego elementu struktury.
c.) drzewa – każdy element tej struktury posiada pola kluczowe, dodatkowe, pola wskaźnikowe na potomków (w liczbie n, np.: drzewa binarne 2) i pole wskaźnikowe na rodzica.

11.Jak jest zorganizowana struktura danych zwana kolejką?

Kolejka (zwana także strukturą FIFO z ang. first in first out) to specjalna struktura dynamiczna, o ograniczonych możliwościach modyfikacji. Operacja dodawania elementu do struktury (insert) odbywa się zawsze na początku, a operacja odłączania (delete) elementu od struktury odbywa się zawsze na końcu tejże struktury. Zachowana zostaje kolejność dołączania i odcinania elementów od struktury – pierwszy przyłączony będzie pierwszym odłączonym.

12.Jak jest zorganizowana struktura danych zwana stosem?

Stos (zwany także strukturą LIFO z ang. last in first out) to specjalna struktura dynamiczna, o ograniczonych możliwościach modyfikacji. W strukturze tego typu zarówno operacje dołączania (insert) jak i odłączania (delete) elementów struktury odbywają się z tej samej strony struktury – na początku. Porządek dołączania i odcinania elementów jest taki, że element położony na strukturze jako ostatni pierwszy będzie z niej zdjęty.

13.Jak jest zorganizowana struktura danych zwana listą?

Lista to struktura składająca się z wielu odpowiednio ze sobą połączonych ze sobą elementów. Każdy element jest identyczny jak pozostałe i składa się z odpowiedniej ilości pól kluczowych, dodatkowych i wskaźnikowych (lista jednokierunkowa 1 pole wskaźnikowe, dwukierunkowa 2 pola wskaźnikowe). Pola kluczowe i dodatkowe przechowują informacje, natomiast pola wskaźnikowe zawierają adresy do elementów: następnego (jednokierunkowa) i poprzedniego (dwukierunkowa).

14.Narysuj schemat blokowy sumowania pól kluczowych obiektów z listy. Wykorzystaj pola wskaźnikowe i wartość NIL.

15.Jakie znasz typy list?

a.) lista jednokierunkowa – każdy węzeł zawiera pole kluczowe, pola dodatkowe i jedno pole wskaźnikowe z adresem następnego węzła struktury.
b.) lista dwukierunkowa – każdy węzeł zawiera pole kluczowe, pola dodatkowe i dwa pola wskaźnikowe z adresami następnego i poprzedniego węzła struktury.
c.) lista z wartownikiem – jest to dwukierunkowa lista, z której wyeliminowano wskazanie NIL. Pierwszy węzeł jako wskazanie na węzeł poprzedni odnosi się do węzła ostatniego. Węzeł ostatni jako wskazanie na węzeł następny odnosi się do węzła pierwszego. (Jest to jak gdyby zapętlenie struktury.)
d.) lista jako kolejka
e.) lista jako stos

16.Jak jest zorganizowana struktura danych zwana drzewem?

Drzewo składa się z odpowiedniej mnogości identycznych elementów - zwanych węzłami. Każdy węzeł zawiera:
- pole kluczowe (zawiera dane)
- pola dodatkowe (zawierają dane)
- pola wskaźnikowe na potomstwo (zawierają adresy potomstwa – elementów następnych)
- pole wskaźnikowe na rodzica (zawiera adres rodzica – elementu poprzedniego)

Opis biologiczny drzewa:

1. wskazanie na korzeń – element wskaźnikowy (na grafie często pomijany), zawierającym adres pierwszego węzła drzewa.
2. korzeń – pierwszy węzeł drzewa (zazwyczaj w grafie umieszczony na górze), nie posiadający rodzica.
3. węzły – wszystkie elementy struktury leżące pomiędzy korzeniem a liśćmi.
4. liście – elementy leżące na najniższym poziomie drzewa i nie posiadające potomstwa (ich pola wskaźnikowe przechowują wartość NIL).
Połączenia pomiędzy poszczególnymi węzłami określa się mianem gałęzi drzewa.
W celu ustanowienia hierarchii w strukturze drzewiastej przyjęto iż elementy leżące na gałęziach danego węzła określa się mianem potomstwa danego węzła. Wskazany element jest zatem rodzicem elementów jemu podległych.
W drzewie wyróżnia się także poziomy, przy czym ich odliczanie zaczyna się od góry: korzeń stanowi poziom pierwszy, a poziom ostatni tworzą liście.
Wszystkie drzewa dzieli się na:
BST – drzewo, którego rząd wyjściowy potomków ograniczony jest przez 2.
PEŁNE – drzewo, w którym wszystkie wierzchołki poza liśćmi mają jednakową liczbę potomków i wszystkie liście są na tym samym poziomie.

17.Co to jest drzewo binarne?

Drzewo binarne to drzewo, którego rząd wyjściowy węzłów jest ograniczony przez 2. Każdy węzeł może zatem posiadać maksymalnie 2 potomków.

18.Jaki obiekt w drzewie nazywany jest liściem a jaki korzeniem?

Korzeń – jest to wierzchołek drzewa (w grafie zazwyczaj umieszczany na górze) - nie posiadający rodzica. W korzeniu mają początek wszystkie procesy algorytmiczne oparte na strukturze danych tego typu.
Liść – wierzchołek końcowy drzewa (w grafie umieszczony zazwyczaj na dole) - nie posiadający potomstwa. Odpowiada on różnym wynikom zakończenia obliczeń w algorytmie.

19.Podaj na przykładzie pierwszego etapu algorytmu sortowania drzewiastego zasadę budowy drzewa BST.

Aby przekształcić nieuporządkowaną listę wejściową w drzewo poszukiwań binarnych musimy wziąć pierwszy element drzewa i zapisać go jako korzeń nowopowstającego drzewa. Następnie biorąc każdy kolejny element z listy porównujemy go z elementami już należącymi do drzewa i:
-jeśli element jest mniejszy umieszczamy go po lewej stronie korzenia.
-jeśli element jest większy umieszczamy go po prawej stronie korzenia.
Postępujemy tak aż do wyczerpania listy.

20.Z jaką strukturą sterującą związane są drzewa? Ilustrując ten związek opisz zasadę przeglądania drzewa w algorytmie sortowania drzewiastego.

Drzewa związane są z rekurencją.
Opis słowny algorytmu przeglądu drzewa w algorytmie sortowania drzewiastego:
procedura OBEJDŹ drzewo T:
1. jeśli T jest puste to wróć
2. w przeciwnym razie wykonaj co następuje:
    a. wywołaj obejdź T w lewo
    b. wypisz element umieszczony w korzeniu
    c. wywołaj obejdź T w prawo

Zasada przeglądu drzewa w algorytmie sortowania drzewiastego jest taka, że najpierw wywołana zostaje procedura obejdź pod-drzewo z lewej strony, wypisz wartość korzenia i obejdź pod-drzewo z prawej strony. W drzewie BST z lewej umieszczone są elementy mniejsze niż w korzeniu, a z prawej większe. Mamy więc pewność, że dzięki takiej organizacji wypisane zostaną elementy od najmniejszego do największego.
Związek pomiędzy drzewem i rekurencją jest tak silny dlatego, że nawet sam graf kolejnych wywołań rekurencyjnych ma strukturę drzewiastą.

21.Według jakich zasad można systematycznie przeglądać strukturę drzewiastą?

Przegląd struktury = budowanie ciągu zawierającego wszystkie elementy tej struktury.
Strukturę drzewiastą można przeglądać na trzy sposoby:
- wędruj i sprawdzaj (metoda rekurencyjna)
Jest to prosta metoda polegająca na przejrzeniu wszystkich elementów struktury i wypisaniu kolejno wszystkich jej elementów.
- przegląd w głąb (metoda iteracyjna)
Schemat działania algorytmu:
1. Wypisany zostaje korzeń (jako pierwszy element ciągu)

2. Algorytm wybiera ostatni element ciągu nie posiadający etykiety „zamknięty” i postępuje zgodnie z zasadą: jeśli wierzchołek nie ma potomstwa – nadaj mu etykietę „zamknięty”, jeśli ma, wypisz potomków sprawdzając od lewej
Wszystko powtarzane jest dopóki wszystkie elementy ciągu nie mają etykiety „zamknięty”.
- przegląd w szerz (metoda iteracyjna)
Schemat działania algorytmu:
1. Nadanie wszystkim elementom drzewa etykiety „nowy”
2. Wypisanie korzenia jako pierwszego elementu ciągu
3. Algorytm wybiera z ciągu pierwszy element z etykietą „nowy” , dopisuje do ciągu wszystkich jego potomków (nadając im etykietę „nowy”) i zdejmuje etykietę „nowy” z danego elementu.
Wszystko powtarzane jest dopóki w ciągu znajdują się elementy z etykietą „nowy”.
Po utworzeniu odpowiedniego ciągu, do jego przeszukiwania wykorzystujemy metodę: „wędruj i sprawdzaj”. Metoda ta polega na przechodzeniu kolejnych elementów ciągu i porównywaniu ich ze wzorcem elementu poszukiwanego. Po natrafieniu na odpowiedni element zostaje on wypisany.

22.Jakie znasz elementy składowe typowego języka programowania?

Elementy składowe języka programowania to:
-symbole (alfabet danego języka programowania – określa jakich znaków wolno używać podczas tworzenia programów i jakie jest ich znaczenie – np.: alfabet łaciński, grecki, znaki matematyczne itp.)
-słowa kluczowe (jest to zbiór ciągów symboli o specjalnym znaczeniu – np.: komendy i instrukcje)
-składnia – (zawiera: opis dostępnych struktur sterujących np.: bezpośrednie następstwo, wybór warunkowy, iteracja ograniczona – decyduje jakich struktur można używać a jakich nie; opis sposobu definiowania rozmaitych struktur sterujących – mówi w jaki sposób prawidłowo korzystać z poszczególnych instrukcji; schematy podstawowych instrukcji)
-semantyka (jest to zbiór przepisów odpowiadających na pytania: „co znaczy to zdanie” lub „co znaczy ta instrukcja” – określa prawidłowość składniową zapisywanych wyrażeń)

23.Czym różni się kompilacja programu od jego interpretacji?

Kompilacja – jest wykonywana jednorazowo po zakończeniu pisania programu. Jest to przełożenie całego programu napisanego w języku wysokiego poziomu na program w języku niższego poziomu. Programy napisane w językach kompilowanych np.: w C, C++ są o wiele szybsze niż identyczne programy zapisane w językach interpretowanych.
Interpretacja – jest to proces tłumaczenia poszczególnych komend z języka wysokiego poziomu na kod maszynowy zrozumiały dla maszyny, w czasie ich wykonywania. Ta translacja wykonywana jest za każdym razem gdy uruchamiany jest dany program, przez co wykonywanie takiego programu jest o wiele wolniejsze od programów kompilowanych.

24.Na czym polega metoda algorytmiczna zwana „wędruj i sprawdzaj”?

Metoda algorytmiczna typu „wędruj i sprawdzaj” polega na prostym przeglądzie struktury danych w celu odnalezienia zadanego elementu w tej strukturze.
Przykładem wykorzystania metody „wędruj i sprawdzaj” jest algorytm wyszukiwania największej przekątnej w wielokącie wypukłym. Najpierw tworzymy tabelę dwuwymiarową zawierającą dane o odległościach dzielących wszystkie wierzchołki. Potem przeglądamy kolejno wszystkie elementy tej struktury w celu znalezienia największego elementu.
Istotne dla tej metody jest uwzględnienie odpowiedniej struktury sterującej w zależności od struktury danych, którą chcemy przejrzeć, i tak:
wektor, lista - iteracja
tablice wielowymiarowe, listy list itp. - iteracje zagnieżdżone
drzewa - rekordy

25.Na czym polega metoda algorytmiczna zwana „dziel i zwyciężaj”?

Metoda „dziel i zwyciężaj” postępuje zgodnie z zasadą, że jeśli jakiś problem jest za duży by uporać się z nim w całości, to należy spróbować podzielić go na mniejsze części o takiej samej strukturze. Ta metoda algorytmiczna związana jest ściśle z procedurami rekurencyjnymi.
Przykładem wykorzystania metody „dziel i zwyciężaj” jest algorytm sortowania przez scalania. Najpierw dzielimy n -elementową listę wejściową na połowy, potem otrzymane połowy znów na połowy i tak do chwili kiedy nie otrzymamy n pojedynczych elementów. Potem porównujemy ze sobą pary elementów i tworzymy uporządkowane ciągi dwuelementowe, porównujemy ciągi dwuelementowe i tworzymy posortowane ciągi cztero elementowe, i tak do chwili kiedy nie otrzymamy jednej porządkowanej n elementowej listy.
Opis słowny rekurencyjnego algorytmu sortowania przez scalanie:
Procedura SORTUJ LISTĘ L:
1. jeśli lista zawiera 1 element to jest posortowana
2. w przeciwnym razie wykonaj co następuje:
    a. podziel listę na L1 i L2
    b. wywołaj SORTUJ L1

c. wywołaj SORTUJ L2
d. scal posortowane listy L1 i L2 w jedną posortowaną listę
3. wróć do poziomu wywołania

26.Na czym polega metoda algorytmiczna zwana „zachłanną”?

Istnieją zadania, których rozwiązanie można budować z dobieranych kolejno elementów i do takich właśnie zadań przeznaczona jest metoda zachłanna. Postępuje ona zawsze zgodnie z zasadą „łap co masz najlepszego pod ręką i nigdy nie oddawaj tego co już masz”. Jest to metoda efektywna tylko w pewnej grupie problemów.
Przykładem zastosowania metody „zachłannej” jest algorytm poszukiwania najtańszej sieci wiążącej wszystkie podane punkty (czyli budowanie minimalnego drzewa rozpinającego w zadanym grafie). Mając daną sieć punktów połączonych odcinkami o podanej wadze wybieramy najtańszy. Potem wybieramy najtańszy odchodzący od punktów które już mamy itd. Czynność powtarzamy do chwili kiedy nie połączymy wszystkich oznaczonych na płaszczyźnie punktów.
Opis algorytmu:
1. wybierz odcinek o najmniejszej wadze
2. powtarzaj co następuje aż do połączenia wszystkich punktów na płaszczyźnie:
- wybierz najtańszy odcinek łączący punkt który już masz, z tym dowolnym który nie jest jeszcze przyłączony

Metoda ta daje bardzo dobre wyniki gdy poszukiwana jest najtańsza wagowo ścieżka połączeń wszystkich punktów na płaszczyźnie (tzw. minimalne drzewo rozpinające). Nie daje jednak poprawnych wyników gdy wykorzystuje się ją do np.: poszukiwania najtańszej drogi łączącej dwa wskazane punkty na płaszczyźnie.

27.Opisz schemat działania algorytmu sortowania przez scalanie.

Sortowanie przez scalanie polega na podzieleniu za pomocą procedury rekurencyjnej n –elementowej listy wejściowej na n pojedynczych elementów. Następnie algorytm wykonuje kolejno porównania: najpierw par elementów (tworząc posortowane ciągi 2 –elementowe), potem ciągów 2 elementowych (tworząc posortowane ciągi 4 elementowe), i tak do chwili gdy nie powstanie posortowana n – elementowa lista.
Najistotniejszym etapem tego algorytmu jest etap scalania. W etapie tym następuje porównanie dwóch identycznych części (n –elementowych ciągów) i przekształcenie ich tak by utworzyły 2n –elementowy, posortowany ciąg wynikowy. Porównanie odnosi się zawsze do dwóch pierwszych elementów ciągów wejściowych, a do nowopowstającego, posortowanego ciągu wynikowego przepisywany jest mniejszy element porównania.
Opis słowny algorytmu:
Procedura SORTUJ listę L;
1. jeśli lista zawiera tylko jeden element to jest posortowana
2. w przeciwnym razie wykonaj co następuje
    a. podziel listę L na L1 i L2
    b. wywołaj SORTUJ listę L1
    c. wywołaj SORTUJ listę L2
    d. scal posortowane listy L1 i L2 w jedną posortowaną liste L (etap iteracyjny)
3.wróć do poziomu wywołania

28.Na czym polega metoda algorytmiczna zwana „programowaniem dynamicznym”?

Metoda algorytmiczna zwana „programowaniem dynamicznym” opiera się o ***zasadę optymalności Bellmana***, która mówi: „jeśli znamy najlepszą drogę przejścia z punktu początkowego do punktu końcowego prowadzącą przez kolejne punkty, to każdy fragment tej drogi pomiędzy dowolnym punktem a punktem końcowym jest najlepszą drogą przejścia od tego punktu do punktu końcowego”.
Przykładem zastosowania metody „dynamicznej” jest algorytm wyszukiwania najkrótszej ścieżki łączącej dwa wskazane punkty na płaszczyźnie – grafie skierowanym (problem strudzonego wędrowca). W zadaniu tym w przeciwieństwie do metody zachłannej nie chwyta się tych elementów które w danej chwili są najlepsze lecz próbuje się ustalić jakie wyjście da najlepszy efekt końcowy. Analizując poszczególne fragmenty grafu poruszamy się od końca – czyli od punktu oznaczonego flagą „meta”. Badając kolejne ścieżki grafu poznajemy wagę wszystkich możliwych dróg spośród których wybieramy najlepszą.

29.Jakie znasz rodzaje błędów popełniane przy konstrukcji i zapisie algorytmów? Jakie mogą być ich konsekwencje?

Najczęściej spotykanymi rodzajami błędów są:
<u>Błędy językowe (składniowe).</u>
Powstają w wyniku naruszenia składni używanego języka programowania.
np.:
- użycie for(i=0, i<n, i++)
- zamiast for(i=0; i<n; i++)

Konsekwencje:
-zatrzymanie kompilacji lub interpretacji z komunikatem lub bez
-przerwanie realizacji programu nawet jeśli kompilator nie wykrył błędu
-nieprzyjemne ale zwykle niezbyt poważne błędy, stosunkowo łatwe do poprawienia
Błędy semantyczne.
Powstają w wyniku niezrozumienia semantyki używanego języka programowania.
np.:
- przyjęliśmy, że po zakończeniu iteracji for(i=0; i<n; i++) i ma wartość n, a nie n+1
Konsekwencje:
-program realizuje algorytm w sposób niepoprawny
-są to błędy trudne do wykrycia i usunięcia, ale można ich uniknąć przy poświęceniu większej uwagi zrozumieniu używanych instrukcji
Błędy logiczne.
Powstają najczęściej na etapie projektowania algorytmu i są przyczyną złego podejścia do rozwiązania danego zadania.
np.:
- przyjęliśmy, że algorytm zliczający zdania w tekście będzie zwiększał zmienną „suma" po odnalezieniu sekwencji „. „. Jednak sekwencja ta może pojawić się np. w środku zdania lub zdanie może zostać zakończone innymi znakami np.: wykrzyknikiem.
Konsekwencje:
-algorytm przestaje być poprawnym sposobem rozwiązania zadania
-dla pewnych danych wejściowych wyniki są niezgodne z oczekiwaniami
-procesor może nie być w stanie wykonać pewnych instrukcji
-są to błędy bardzo groźne, mogą długo pozostawać w ukryciu i być bardzo trudne do usunięcia
Błędy algorytmiczne.
Wynikają ze złego zaprojektowania algorytmu:
np.:
- na etapie projektowania algorytmu źle zaadresowano w grafie połączenia iteracji lub wyborów warunkowych.
Konsekwencje:
- algorytm dla pewnych dopuszczalnych danych daje niepoprawne rozwiązanie
- wykonanie programu zostaje przerwane
- program realizujący algorytm nie kończy swego działania

30.Jaka jest różnica pomiędzy całkowitą, a częściową poprawnością algorytmu?

Algorytm jest całkowicie poprawny względem warunku początkowego i końcowego, jeżeli dla każdych danych spełniających warunek początkowy, obliczenia kończą się i wyniki spełniają warunek końcowy.
Algorytm jest częściowo poprawny jeśli dla każdych obliczeń, które się kończą wynik jest poprawny względem warunku początkowego i końcowego.
Różnica polega na tym, ze w przypadku poprawności częściowej nie zawsze obliczenia prowadzą do zakończenia pracy algorytmu.
Weryfikując częściową poprawność algorytmu musimy wykazać, że obliczenia kończą się dla wszystkich poprawnych danych wejściowych.

31.Co nazywamy niezmiennikiem a co zbieżnikiem?

Niezmienników używa się w obrębie iteracji do sprawdzenia częściowej poprawności algorytmu. (Metoda niezmienników polega na ustawieniu w danym algorytmie punktów kontrolnych, tzw. asercji. Jeśli za każdym razem, dla różnego rodzaju danych wejściowych algorytm osiągnie dany punkt kontrolny to znaczy, że jest częściowo poprawny. Metodę tę dlatego nazwano metodą niezmienników, gdyż zgodnie z jej schematem działania, bez względu na wprowadzone dane wejściowe, dla poprawnego algorytmu przebieg przez punkty kontrolne powinien być niezmienny.)
Zbieżników używa się do ustalania całkowitej poprawności algorytmów. (Metoda zbieżników polega na ustaleniu w danym algorytmie zbieżników – zmiennych, które będą przyjmowały różne wartości, ale zawsze, bez względu na wprowadzone dane, będą dążyły do pewnej wartości maksymalnej lub minimalnej i nigdy jej nie przekroczą. Np.: zbieżnikiem w algorytmie sortowania może być zmienna, której przypiszemy, wartość równą wartości elementów do posortowania.)

32.Jak określona jest złożoność algorytmów? Jakie znasz jej rodzaje?

Złożoność algorytmów możemy badać pod kontem:
-pamięci
-czasu
Złożoność pamięciowa to zależność pomiędzy ilością potrzebnej do działania algorytmu pamięci (wielkością i liczbą struktur danych), a rozmiarami danych wejściowych wprowadzanych do algorytmu.

Złożoność czasowa to zależność pomiędzy liczbą elementarnych operacji wykonywanych w trakcie przebiegu algorytmu a wielkością danych wejściowych. Algorytmy zazwyczaj bada się pod kontem złożoności czasowej, gdyż ta przeważa o użyteczności algorytmu.

33.Jak formalnie stwierdzamy, że dwa algorytmy mają tą samą złożoność?

Dwa algorytmy o czasach wykonania F1(N) i F2(N) mają złożoność tego samego rzędu, jeśli:

$$\lim_{N\to\infty}\frac{F_1(N)}{F_2(N)}=C$$

Jeśli C = 0 algorytm o czasie wykonania F1(N) ma niższy rząd złożoności.
Jeśli C = ∞ algorytm o czasie wykonania F2(N) ma niższy rząd złożoności.
Jeśli 0<C<∞ złożoność czasowa jest tego samego rzędu.

34.Podaj definicję złożoności logarytmicznej algorytmu.

Oznaczenie złożoności logarytmicznej: O(logN).
Algorytm o czasie wykonania F(N) ma złożoność logarytmiczną jeżeli: lim F(N)/N→0.

35.Podaj definicję złożoności liniowej algorytmu.

Oznaczenie złożoności liniowej: F(N)=O(N).
Algorytm o czasie wykonania F(N) ma złożoność liniową jeżeli: lim F(N)/N=C, gdzie 0<C<∞.

36.Podaj definicję złożoności kwadratowej algorytmu.

Oznaczenie złożoności kwadratowej: $F(N)=O(N^2)$.
Algorytm o czasie wykonania F(N) ma złożoność kwadratową jeżeli: lim $F(N)/N^2=C$, gdzie 0<C<∞.

37.Podaj definicję złożoności wykładniczej algorytmu.

Oznaczenie złożoności wykładniczej: $F(N)=O(2^N)$.
Algorytm o czasie wykonania F(N) ma złożoność wykładniczą jeżeli: lim $F(N)/2^N=C$, gdzie 0<C<∞.

38.Co oznacza zapis F(N)=O(g(N)) w odniesieniu do funkcji F(N).

Zapis F(N)=O(g(N)) oznacza spełnienie warunku lim F(N)/g(N)=C, gdzie 0<C<∞ i można odczytywać: algorytm ma złożoność rzędu g(N), lub czas wykonania algorytmu jest O(g(N)).
Równość w zapisie F(N)=O(g(N)) powinna być rozumiana w ten sposób, że funkcja F(N) jest jedną z funkcji, które spełniają powyższy warunek lub precyzyjniej, że należy do zbioru wszystkich funkcji spełniających powyższy warunek.

39.Opisz schemat działania algorytmu wyszukiwania binarnego z listy uporządkowanej. Jaką ten algorytm ma złożoność?

Działanie algorytmu wyszukiwania binarnego z listy uporządkowanej rozpoczyna się od porównania ze wzorcem poszukiwanego elementu, elementu znajdującego się w samym środku listy. Jeżeli nie trafiono w poszukiwany element wiadomo już, w której połowie listy mamy go szukać (bo lista jest uporządkowana). Wtedy rozpoczynamy badanie odpowiedniej połowy listy od porównania wzorca elementu poszukiwanego z jej elementem środkowym. Jeśli znów nie trafiliśmy powtarzamy działania dalej. W pewnym momencie algorytm zawsze trafi na poszukiwany element, chyba że nie ma go na liście.
Złożoność czasowa tego algorytmu wynosi F(N)=O(logN), gdzie N jest zależne od ilości elementów.
Długość listy:
-20 elementów (5 porównań)
-1000000 elementów (20 porównań)

40.Wymień znane ci algorytmy sortowania i podaj ich złożoność.

Wartości podane dla średniego przypadku:
-sortowanie drzewiaste bez samoorganizacji drzewa: $O(N^2)$
-sortowanie drzewiaste (lewostronne obejście): O(N)
-sortowanie bąbelkowe: $O(N^2)$
-sortowanie z samoorganizacją drzewa: O(N*logN)
-sortowanie rekurencyjne (przez scalanie): O(N*logN)
-sortowanie typu quicksort: O(N*logN) - w najgorszym przypadku $O(N^2)$

41.Czym różni się analiza złożoności algorytmu w średnim i w najgorszym przypadku?

W analizie najgorszego przypadku badamy jaka będzie złożoność algorytmu w przypadku wystąpienia najniekorzystniejszych danych (np.: iteracyjny przegląd n –elementowej macierzy w poszukiwaniu elementu, którego na niej nie ma). Badanie takie daje nam pewną wiedzę o zachowaniu się algorytmu w takim przypadku.
W analizie średniego przypadku staramy się przewidzieć jakie rozwiązanie będzie pojawiało się najczęściej i właśnie dla niego staramy się obliczyć poziom złożoności algorytmicznej. W obliczeniach średniego przypadku istotną rolę odgrywają założenia o rozkładzie prawdopodobieństwa w zbiorze dopuszczalnych danych wejściowych.

Jeżeli mamy pewność, że szanse wystąpienia najgorszego wypadku są znikome, a dany algorytm ma zadowalającą złożoność średniego przypadku można zastanowić się nad jego zastosowaniem w danym programie.

42.Jaki problem nazywamy zamkniętym z punktu widzenia złożoności obliczeniowej?

Problemy zamknięte, to problemy dla których ustanowione dolne i górne ograniczenia złożoności czasowej schodzą się do tego samego przedziału. Znaczy to po prostu, że górne ograniczenie złożoności danego algorytmu ma ten sam rząd co jego dolne ograniczenie złożoności.
Jeśli dolna i górna ograniczoność czasowa się nie schodzą, mówimy wówczas o istnieniu luk algorytmicznych.

43.Podaj przykłady algorytmów zamkniętych z punktu widzenia złożoności obliczeniowej i przykłady luk.

Przykłady algorytmów zamkniętych z punktu widzenia złożoności obliczeniowej:
- proste sumowanie zarobków - dolne i górne ograniczenie złożoności wynosi: O(N)
- przeszukiwanie uporządkowanej listy – dolne i górne ograniczenie złożoności wynosi: O(logN).

Przykłady luk algorytmicznych:
- problem minimalnego drzewa rozpinającego – górne ograniczenie złożoności wynosi: $O(N^2)$, a dolne O(N)

44.Jaką złożoność ma problem Wieży Hanoi?

Problem Wieży Hanoi jest zamknięty: dolne i górne ograniczenie złożoności jest równe (zarówno dla algorytmu iteracyjnego jak i rekurencyjnego) i wynosi: $O(N^2)$.

45.Co znaczy, że jedna funkcja złożoności jest ograniczona z góry przez drugą?

Oznacza to, że:
- dana funkcja ma złożoność o rząd niższą od funkcji, będącej jej górnym ograniczeniem
- istnieje lepsze rozwiązanie danego problemu

46.Co znaczy, że problem ma złożoność wielomianową?

Problem ma złożoność wielomianową jeżeli, funkcja jego złożoności jest ograniczona z góry przez N^K dla pewnego K, czyli $F(N)=O(N^K)$

47.Jakiego rodzaju problemy tworzą klasę problemów NP – zupełnych? Podaj przykłady.

Klasę problemów NP – zupełnych tworzą problemy o następujących cechach:
-każdy taki problem da się w czasie wielomianowym przekształcić do innego
-dla wszystkich istnieją wątpliwe (wykładnicze) rozwiązania
-dla żadnego nie znaleziono rozsądnego (wielomianowego) rozwiązania
-dla żadnego nie udowodniono, że wymaga on czasu wykładniczego
-wszystkie te problemy są ze sobą powiązane (jeśli dla choćby jednego uda się znaleźć wielomianowe rozwiązanie to będzie wiadomo, że inne także da się rozwiązać w czasie wielomianowym i odwrotnie, jeśli wykaże się, że dla choćby jednego nie istnieje wielomianowe rozwiązanie, dla innych także go nie będzie)

Przykłady problemów NP – zupełnych:
-ścieżka Hamiltona
-ułożenia dwuwymiarowe (małpia układanka, wąż domino, figury geometryczne)
-problem komiwojażera
-załadunek plecaka

48.Jakie byłyby konsekwencje udowodnienia, że wybrany problem NP – zupełny nie może być rozwiązany deterministycznym algorytmem o złożoności czasowej wielomianowej?

Gdyby udowodniono, że choćby jeden z problemów NP – zupełnych z pewnością nie ma rozwiązania deterministycznie wielomianowego, znaczyłoby to, że inne problemy tej klasy także go nie mają. Wynika to z zależności, że jeden problem tej klasy można w czasie wielomianowym przekształcić w inny.

49.Jakie byłyby konsekwencje skonstruowania dla wybranego problemu NP – zupełnie deterministycznego algorytmu o złożoności wielomianowej?

Gdyby udowodniono, że choć jeden z algorytmów klasy NP – zupełnej posiada wielomianowe rozwiązanie udowodniono by równocześnie, że inne także go posiadają. Wynika to z faktu, że każdy problem tej klasy da się w czasie wielomianowym przekształcić w dowolny inny.

50.Na czym polega idea konstruowania algorytmów przybliżonych?

Główna idea konstruowania takich algorytmów bazuje na założeniu, że w wielu przypadkach wynik gorszy od optymalnego jest i tak lepszy od całkowitego braku rozwiązania dla danego zadania. Takie „zastępcze” algorytmy noszą nazwę algorytmów aproksymacyjnych.

Przykład:
Problem komiwojażera jest problemem trudno-rozwiązywalnym (NP – zupełnym), ale można w czasie wielomianowym wyznaczyć całkiem niezłe jego przybliżenie, obchodzące wszystkie wierzchołki grafu. Miarą dobroci takiego rozwiązania jest współczynnik Sa = otrzymany wynik / najlepsze rozwiązanie.

W przypadku rozwiązania zastępczego dla problemu komiwojażera Sa≤1,5, a złożoność czasowa tego rozwiązania wynosi zaledwie $O(N^3)$.

51.Jakie problemy nazywamy nierozstrzygalnymi? Opisz przykładowy problem.

Problem, dla którego nie ma żadnego poprawnie działającego algorytmu nazywamy problemem nieobliczalnym. Jeśli problemem tym jest problem decyzyjny to nazywamy go problemem nierozstrzygalnym.

Przykładem problemu nierozstrzygalnego jest problem domina.

Problem domina polega na sprawdzeniu i udzieleniu odpowiedzi „tak" lub „nie", na pytanie: czy danym zbiorem kart o wymiarach 1 na 1 da się poryć odpowiednio duży teren. Problem jest o tyle trudny, że każda krawędź karty ma inny kolor, a założenie jest takie, że kolory stykających się krawędzi muszą być identyczne.

Dla każdego algorytmu (zapisanego w dowolnym, dającym się efektywnie wykonać języku programowania), który przeznaczony jest do rozstrzygnięcia problemu domina, istnieje nieskończenie wiele zestawów danych wejściowych, dla których algorytm ten będzie działał w nieskończoność lub da błędną odpowiedź.

52.Co to jest problem stopu w algorytmie? Co wiadomo o tym problemie?

Problem stopu jest problemem decyzyjnym, mającym dać odpowiedź „tak" lub „nie" na pytanie: czy dany algorytm R napisany w języku L zatrzyma się dla danych wejściowych X. Jak dotąd ustalono, że problem stopu zalicza się do problemów nierozstrzygalnych, czyli nie istnieje taki algorytm, który w racjonalnym czasie potrafiłby dać odpowiedź na to pytanie.

53.Co to jest maszyna Turinga, do czego służy i jak jest zbudowana?

Maszyna Turinga to uniwersalny model obliczeniowy, przeznaczony do efektywnego wykonywania dających się efektywnie wykonać algorytmów.

Maszyna Turinga składa się z:
-skończonego alfabetu symboli
-skończonego zbioru stanów
-nieskończonej taśmy podzielonej na komórki
(każda komórka może zawierać jeden symbol)
-ruchomej głowicy czytająco – zapisującej
(głowica porusza się krokowo co jedną komórkę)
-diagramu przejść między stanami
(czasem zwanego po prostu diagramem przejść)

W zależności od rodzaju maszyny Turinga może się ona być:
-z taśmą jednostronną ograniczoną
-z wieloma taśmami i wieloma głowicami
-z taśma dwuwymiarową
-z głowicą bez funkcji zapisu

54.Jak wygląda sterowanie w maszynie Turinga?

Sterowanie maszyną odbywa się za pomocą diagramu przejść, który jest po prostu grafem skierowanym, którego wierzchołki reprezentują stany, w jakich może znaleźć się maszyna. Do oznaczenia stanów używa się często zaokrąglonych czworokątów. Krawędź prowadząca ze stanu x do stanu y nazywa się przejściem międzystanowym i etykietuje się ją kodem (a/b, kierunek), gdzie a/b są symbolami a kierunek to albo lewo, albo prawo. Część a to wyzwalacz, a część b, kierunek to akcja.
(a/b, lewo) – jeśli trafisz na „a" zapisz tam „b" i idź o jedno pole w lewo.
(b/a, prawo) – jeśli trafisz na „b" zapisz tam „a" i idź o jedno pole w prawo.

55.O czym mówi teza Churcha-Turinga i jakie ma znaczenie dla analizy złożoności problemów algorytmicznych?

Maszyna Turinga potrafi rozwiązać każdy efektywnie rozwiązywalny problem.

Rozwijając tę tezę, można dojść do wniosku, że jeśli istnieje jakiś szybki komputer, który potrafi rozwiązać dany problem w czasie O(f(N)), to istnieje równoważna mu maszyna Turinga, która potrzebuje na rozwiązanie tego problemu nie więcej niż O(p(f(N))) czasu, dla pewnej ustalonej funkcji wielomianowej p.

56.Co to jest automat skończony i jak jest zbudowany?

Automat skończony to zdegenerowana maszyna Turinga. Posiada (tak jak pełna maszyna Turinga) taśmę (pełniącą rolę pamięci) podzieloną na części (pseudo-komórki) i głowicę wędrującą po zadanej taśmie. Różnice polegają na tym, że taśma jest jednostronnie ograniczona, głowica może poruszać się tylko i wyłącznie w prawą stronę i nie ma możliwości zapisu (zapis jest zbędny bo z ograniczenia ruchu w jedną stronę wynika, że i tak maszyna ta nie mogłaby powrócić do miejsca, w którym coś zapisała).

Sterowanie automatem odbywa się w ten sam sposób co pełną maszyną Turinga, czyli za pomocą diagramu przejść międzystanowych zapisanego na taśmie. Diagram ten jest jednak również zdegenerowany do jedynie dwóch elementów: <b,kierunek>

Automat skończony wykorzystywany jest przede wszystkim w problemach decyzyjnych.

57.Czy automat skończony może służyć do przedstawiania algorytmów obliczeniowych?
Automat skończony nie może być stosowany do wykonywania obliczeń, bo nie potrafi liczyć. Wynika to z jego ograniczeń konstrukcyjnych – porusza się on tylko w jedną stronę (zatem nie może powracać do miejsc w których już była) i nie posiada funkcji zapisu (bo i tak nie mogłaby powrócić do zapisanych danych). Bez możliwości zapisu maszyna nie jest wstanie wykonywać działań matematycznych a jedynie może rozwiązywać problemy decyzyjne.
58.Jakie znasz modele współpracy procesorów pracujących równolegle?
Pamięć dzielona:
-dzielenie dostępu tylko podczas odczytywania
-dzielenie dostępu podczas zapisywania
Nieograniczona pamięć dzielona jest praktycznie niemożliwa do zrealizowania, ze względu na bardzo dużą złożoność połączeń elementów pamięci i procesorów
Sieci o ustalonej konfiguracji połączeń:
-każdy procesor może być połączony z co najwyżej pewną stałą liczbą (w zależności od sieci) liczbą procesorów sąsiadujących
-sieci są konstruowane często jako równoległe maszyny rozwiązujące szczególne problemy algorytmiczne
Dobrze znanym przykładem są sieci logiczne (boolowskie): bramki – procesory realizujące proste funkcje logiczne na bitach (AND, OR, NOT, NAND, itp.)
59.Wyjaśnij pojęcie zasobu krytycznego w systemach pracujących współbieżnie (problem prysznica i chińskich filozofów).
Zasób krytyczny to taki zasób systemowy, który jest ograniczony ilościowo i jednocześnie przeznaczony do użytku wszystkich pracujących w danym systemie procesorów. Zasobem krytycznym może być pamięć operacyjna, linie lub łącza transmisyjne.
Analogią do zasobu krytycznego w problemie „prysznica hotelowego" jest sam prysznic. Jest on tylko jeden, a wszyscy chcą z niego korzystać. W przykładzie „chińskich filozofów" zasobem krytycznym są pałeczki. Dla N filozofów jest zaledwie N/2+1 pałeczek, a każdy z filozofów potrzebuje dwóch by się posilić.
60.Wyjaśnij pojęcie zastoju w systemach pracujących współbieżnie.
Zastój w systemach pracujących współbieżnie to sytuacja, w której następuje zakleszczenie wszystkich procesorów i żaden z procesorów, nigdy już nie otrzyma dostępu do zasobu krytycznego.
Analogia w przykładzie „prysznica hotelowego" jest sytuacja, gdy jeden z gości hotelowych zasłabł pod prysznicem, jeden z gości hotelowych nie przychodzi skorzystać z prysznica gdy nadchodzi jego kolej lub po prostu jeden pokój jest pusty. W takim wypadku numer na tablicy informacyjnej umieszczonej na drzwiach prysznica nigdy już się nie zmieni i nigdy żaden inny mieszkaniec hotelu nie będzie mógł skorzystać z prysznica.
61.Wyjaśnij pojęcie zagłodzenia w systemach pracujących współbieżnie.
Zagłodzenie procesora w systemie pracującym współbieżnie to sytuacja, w której jednemu z procesorów brakuje dostępu do zasobu krytycznego zawsze gdy ten jest mu potrzebny.
Analogia w przykładzie „prysznica hotelowego" jest taka, że w pierwotnej wersji jeden z gości zameldowanych w ów hotelu, przychodzi sprawdzić czy prysznic jest wolny, i zawsze trafia na sytuację kiedy jest zajęty.
62.Wyjaśnij pojęcie aktywnego czekania w systemach pracujących współbieżnie.
Aktywne czekanie to czekanie z ciągłym sprawdzaniem, czy dostęp do zasobu krytycznego przypadkiem nie został zwolniony.
Analogią do aktywnego czekania w przypadku „prysznica hotelowego" było ciągłe sprawdzanie mieszkańców hotelu, czy na tablicy wstępu pod prysznic nie pojawił się przypadkiem numer ich pokoju. W przypadku „chińskich filozofów" było to ciągłe sprawdzanie czy przypadkiem dany filozof nie ma w danej chwili dostępu do dwóch pałeczek.