TPI ktokolwiek widział ktokolwiek wie

Zajęcia i wykłady z dr Wołoszynem

1. *Reprezentacja liczb w systemach komputerowych*

Ogólnie, wśród systemów liczbowych można wyróżnić:

- systemy addytywne (niepozycyjne)
- systemy pozycyjne

Systemy addytywne (zwane też niepozycyjnymi) to systemy liczbowe, w których wartość przedstawionenej liczby jest sumą wartości jej znaków (cyfr).

Bit jest symbolem, który może przyjmować dwie różne postacie. Jeśli chcemy zapisać go na papierze, to stosujemy symbole pomocnicze 0 i 1. Technicznie bit realizowany jest za pomocą dwóch różnych sygnałów. W technice cyfrowej określa się poziomy napięć, które odpowiadają bezpośrednio dwóm postaciom bitu:

0,4 ... 0,8 V - stan 0 (oznaczany również L - Low - Niski)

2,0 ... 2,4 V - stan 1 (oznaczany również H - High - Wysoki)

Bajt jest to grupa 8 bitów. Komórki pamięci komputera przechowują informację w postaci bajtów. Również wiele urządzeń przystosowane zostało do danych przekazywanych w takiej formie (porcjami po 8 bitów) - np. drukarki, terminale, modemy, dyski elastyczne i twarde, itp. Dlatego bajt stał się kolejną po bicie jednostką informacji. Bajt utożsamiany jest ze znakiem, literą, ponieważ używa się często 8 bitowego kodu do przekazywania znaków (ASCII - American Standard Code for Information Interchange). 1 B pozwala przekazywać informację o 256 zdarzeniach.

Słowo 2 , podwójne słowo 4 , poczwórne słowo 8, paragraf 16 , strona 256 , segment 65 536 bitów

Reprezentacja liczb całkowitych dodatnich i ujemnych

- kod prosty
- kod uzupełnieniowy

Operacje arytmetyczne na liczbach całkowitych, nadmiar i niedobór????????

W komputerze liczby przechowywane są w pamięci lub w rejestrach procesora o ustalonej liczbie pól, np. 8 lub 16

Problemy:

- Problem przepełnienia - gdy liczba jest zbyt duża, by móc ją zapisać przy pomocy np. 8 bitów
- Problem niedopełnienia - gdy liczba jest za mała, by ją zapisać przy pomocy np. 8 bitów
- W reprezentacji zmiennoprzecinkowej nadmiarem nazywamy sytuację, gdy liczba jest tak duża (co do modułu), że nie zawiera się w przedziale liczb reprezentowalnych, a podmiarem sytuację, gdy liczba jest tak mała (co do modułu), że musi być reprezentowana przez zero

Reprezentacja liczb zmiennoprzecinkowych

- Stałopozycyjna
- Zmiennopozycyjna

Reprezentacja zera w kodzie prostym jako 10000000 <7 zer> lub 00000000

Liczby subnormalne w informatyce, brak reprezentacji liczb lub zdenormalizowane liczby (obecnie często zwane nienormalnymi liczbami) wypełniają niedomiarową lukę (wygenerowana liczba jest za mała by była reprezentowana

w urządzeniu które ma jest przechowywać.) wokół zera w arytmetyce zmiennoprzecinkowej. Jakakolwiek niezerowa liczba o wielkości mniejszej niż najmniejsza zwykła liczba jest "nienormalna".

Standard podwójnej precyzji – w c++/c double ????

Standard IEEE 754(-1985, -2008) definiuje dwie podstawowe klasy binarnych liczb zmiennoprzecinkowych:

- binary 32 - pojedynczej precyzji (ang. single-precision)
- binary 64 - podwójnej precyzji (ang. double-precision)

Format	Bit znaku	Bity cechy	Bity mantysy
32 bity - pojedyncza precyzja	1 bit	8 bitów	23 bity
64 bity - podwójna precyzja	1 bit	11 bitów	52 bity

Zapis stałoprzecinkowy albo stałopozycyjny– jeden ze sposobów zapisu liczb ułamkowych stosowanych w informatyce. Do zapisu liczby stałoprzecinkowej przeznaczona jest z góry określona ilość cyfr dwójkowych (bitów), a pozycję przecinka ustala się arbitralnie, w zależności od wymaganej dokładności.

Na przykład: mając do dyspozycji słowo 32-bitowe, można wydzielić 24 bity na część całkowitą, 8 bitów na część ułamkową, albo po 16 bitów na część całkowitą i ułamkową, albo 30 bitów na część całkowitą i zostawić tylko 2 bity do zapisu części ułamkowej.

Zakresy wartości danych różnych typów arytmetycznych:

char	-128 ... 127
signed char	-128 ... 127
unsigned char	0 ... 255
int	-32768 ... 32767
signed int	-32768 ... 32767
unsigned	0 ... 65535
unsigned int	0 ... 65535
long	-2147483648 ... 2147483647
long int	-2147483648 ... 2147483647
signed long int	-2147483648 ... 2147483647
unsigned long	0 ... 4294967295
unsigned long int	0 ... 4294967295
float	+/- ($3.4*10^{-38}$ ... $3.4*10^{38}$)
double	+/- ($1.7*10^{-308}$ ... $1.7*10^{308}$)
long double	+/- ($1.7*10^{-4932}$ ... $1.7*10^{4932}$)

Zakresy wartości typów całkowitoliczbowych i Zakresy wartości typów zmiennoprzecinkowych w c/c++

2. *Obliczalność i rozstrzygalność*

Funkcje obliczalne są podstawowym obiektem badań teorii obliczalności. Zbiór funkcji obliczalnych jest równoważny zbiorowi funkcji obliczalnych w sensie Turinga oraz funkcji częściowo rekurencyjnych. Funkcje obliczalne stanowią analogon intuicyjnego pojęcia algorytmu. Tego pojęcia używa się do dyskusji obliczalności bez odniesienia do określonego modelu obliczalności takiego jak maszyna Turinga lub maszyna von Neumana. Jednak ich definicja musi mieć odniesienie do określonego modelu obliczalności. Zgodnie z hipotezą Churcha i Turinga, funkcjami obliczalnymi są dokładnie te funkcje, które można obliczyć używając urządzenia maszynowego mając nieskończenie wiele czasu oraz przestrzeni pamięciowej. Równoważnie twierdzenie to oznacza, że każda funkcja dająca się wyrazić przez algorytm jest obliczalna.

W informatyce i logice formalnej, pojęcie funkcja rekurencyjna określa funkcję \mathbb{N}^i\rightarrow\mathbb{N} która jest obliczalna za pomocą maszyny Turinga. Klasę tych funkcji definiuje się za pomocą mniejszej klasy funkcji pierwotnie rekurencyjnych

Funkcja pierwotnie rekurencyjna [edytuj | edytuj kod]

Funkcjami pierwotnie rekurencyjnymi nazywamy funkcje:

- **Funkcja zerowa**
  $Z : \mathbb{N} \to \mathbb{N}$, zdefiniowana jako $Z(n) = 0$
- **Funkcja następnika**
  $S : \mathbb{N} \to \mathbb{N}$, zdefiniowana jako $S(n) = n + 1$
- **Funkcja rzutowania**
  $I_n^i : \mathbb{N}^n \to \mathbb{N}$, zdefiniowana jako $I_n^i(x_1, \ldots, x_n) = x_i, \ i \leqslant n$

oraz wszystkie funkcje zbudowane z funkcji pierwotnie rekurencyjnych za pomocą następujących metod kompozycji:

- **Złożenia funkcji**
  Dla danych funkcji $f : \mathbb{N}^k \to \mathbb{N}$ oraz $g_1, \ldots, g_k : \mathbb{N}^n \to \mathbb{N}$, złożeniem nazywamy funkcję $h : \mathbb{N}^n \to \mathbb{N}$, zdefiniowaną jako $h(\overline{n}) = f(g_1(\overline{n}), \ldots, g_k(\overline{n}))$
- **Rekursji prostej**
  Dla danych funkcji $g : \mathbb{N}^n \to \mathbb{N}$ oraz $h : \mathbb{N}^{n+2} \to \mathbb{N}$, złożeniem rekurencyjnym nazywamy funkcję $f : \mathbb{N}^{n+1} \to \mathbb{N}$ zdefiniowaną jako $\begin{cases} f(\overline{n}, 0) = g(\overline{n}) \\ f(\overline{n}, S(m)) = h(f(\overline{n}, m), \overline{n}, m) \end{cases}$

Problem nierozstrzygalny – w teorii obliczeń – problem decyzyjny, dla którego nie istnieje algorytm, który po skończonej liczbie kroków jednoznacznie odpowie tak lub nie dla dowolnych danych wejściowych. Turing w 1937 roku wykazał, że udzielenie odpowiedzi na pytanie, czy Maszyna Turinga o numerze n wykonując działania nad liczbą m

zakończy kiedyś swoją pracę, czy też przewidziany dla niej algorytm będzie realizowany w nieskończoność, jest problemem nierozstrzygalnym.

Zbiór rekurencyjny – podzbiór X c N (zbioru liczb naturalnych) dla którego można skonstruować algorytm, który w skończonym czasie rozstrzyga czy dana liczba należy do zbioru czy też nie. Inne nazwy tego pojęcia to zbiór obliczalny oraz zbiór rozstrzygalny.

Przykłady [edytuj | edytuj kod]

Następujące zbiory są rekurencyjne:

- Zbiór pusty
- Zbiór liczb naturalnych $\mathbb{N}$
- Każdy skończony podzbiór $\mathbb{N}$
- Zbiór liczb pierwszych

Podstawowe własności [edytuj | edytuj kod]

- Każdy zbiór rekurencyjny jest też zbiorem rekurencyjnie przeliczalnym.
- Nieskończony zbiór rekurencyjnie przeliczalny musi zawierać nieskończony podzbiór rekurencyjny.
- Istnieją zbiory rekurencyjnie przeliczalne które nie są rekurencyjne.
- Zbiór $X \subseteq \mathbb{N}$ jest rekurencyjny wtedy i tylko wtedy gdy zarówno X jak i $\mathbb{N} \setminus X$ są rekurencyjnie przeliczalne.

Funkcja obliczalna a pojęcie algorytmu ????

7. Obliczalność, rozstrzygalność

7.1 Formalne pojęcie algorytmu

Definicja

Algorytmem nazywamy uporządkowany zbiór operacji wykonywany przez maszynę Turinga, zaprogramowaną tak, że znajduje ona rozwiązanie zadania z określonej **klasy zadań.**

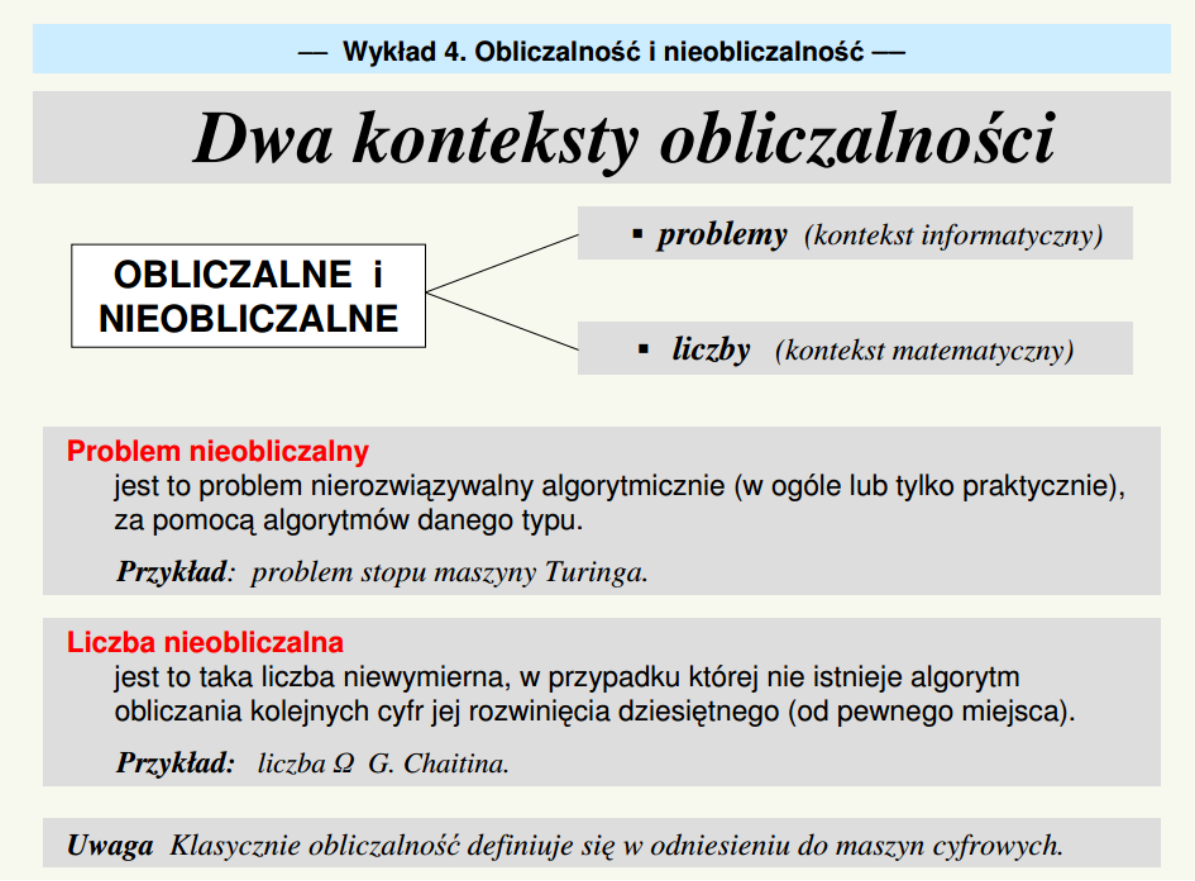
→ Formalna definicja algorytmu wyrażona została jako zadanie dla **maszyny Turinga** - uniwersalnego modelu obliczeń.

Liczba algebraiczna to liczba rzeczywista (ogólniej zespolona), która jest pierwiastkiem pewnego niezerowego wielomianu o współczynnikach wymiernych (a więc i całkowitych).

Dowodzi się, że dla każdej liczby algebraicznej α istnieje wielomian nierozkładalny nad Q, którego pierwiastkiem jest α. Stopień tego wielomianu nazywamy stopniem liczby α.

Zbiór liczb algebraicznych tworzy ciało. W 1882 Ferdinand Lindemann dowiódł, że liczba π nie jest algebraiczna, czyli jest przestępna i tym samym udowodnił, że kwadratura koła nie jest możliwa.

Liczba przestępna - to taka liczba, która nie jest pierwiastkiem żadnego wielomian o współczynnikach wymiernych. Inaczej mówiąc jest to liczba niealgebraiczna.



Twierdzenie Goedla

Każdy niesprzeczny rozstrzygalny system formalny pierwszego rzędu, zawierający w sobie aksjomaty Peana, musi być niezupełny.

Oznacza to, że żaden system formalny pierwszego rzędu nigdy nie "pokryje" w całości zbioru wszystkich twierdzeń arytmetyki. Nie oznacza to, że zbiór wszystkich twierdzeń arytmetyki nie istnieje, a jedynie, że nie może on być wygenerowany przez żaden system formalny. Inaczej mówiąc, dowodliwość jest zawsze słabsza od prawdziwości – zbiór zdań generowanych (dowodzonych) przez system formalny nigdy nie będzie równy ze zbiorem zdań prawdziwych teorii. Może on być albo mniejszy od zbioru zdań prawdziwych (system niesprzeczny, ale niezupełny), albo większy od niego (system zupełny ale sprzeczny).

Twierdzenie Gödla zachodzi także dla każdej teorii silniejszej od arytmetyki Peana (zawierającej w sobie tę arytmetykę). Oryginalne twierdzenie nic nie mówi o teoriach słabszych od arytmetyki, ale odkryto już słabsze teorie, które wystarczają do zachodzenia podobnych twierdzeń.

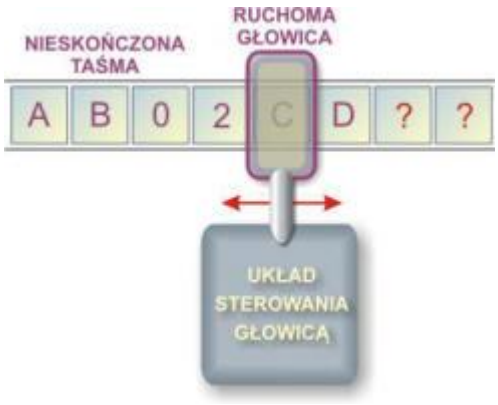
Można rozszerzyć definicję systemów formalnych tak, że twierdzenie Gödla nie będzie dla nich zachodzić. Jednak takie niestandardowe systemy nigdy nie będą rozstrzygalne, tzn. ich algorytm wnioskowania nie dałby się zaprogramować na maszynie Turinga lub ich zbiór aksjomatów nie dałby się taką maszyną wygenerować. Ponieważ każdy zbiór skończony jest rozstrzygalny, dlatego twierdzenie Gödla mówi, że nie istnieje żadna zupełna niesprzeczna, skończona aksjomatyzacja arytmetyki, ani nawet taka nieskończona, która da się wygenerować ze skończonej.

Numeracja Goedla jest arytmetyzacją logiki I rzędu (więcej lepiej nie wiedzieć, lepiej nie….)

3. *Modele prowadzenia obliczeń*

Maszyna analogowa, inform. maszyna matematyczna służąca do rozwiązywania zadań przez modelowanie zależności matematycznych za pomocą zjawisk zachodzących w układach mechanicznych, elektrycznych, elektromechanicznych lub elektronicznych.
Działanie :
odwzorowuje zjawiska zachodzące w obiekcie oryginalnym za pomocą zjawisk zachodzących w modelu (analogu), zbudo-wanym jako odpowiedni układ elektryczny. W maszynie analogowej wszystkie operacje są wykonywane jednocześnie. Jeżeli program zawiera np. kilka operacji dodawa-nia, to każda z nich jest wykonywana za pomocą innego układu operacyjnego, a więc obliczenia są wykonywane równolegle. ?????

Automat skończony (ang. finite state machine, FSM) – abstrakcyjny, matematyczny, iteracyjny model zachowania systemu dynamicznego oparty na tablicy dyskretnych przejść między jego kolejnymi stanami (diagram stanów).Ze względu na charakter przejść między stanami, wyróżnia się deterministyczne i niedeterministyczne automaty skończone.Automaty skończone są ważnym narzędziem teoretycznym w tworzeniu i testowaniu oprogramowania, a jako modele szerszych procesów znajdują także swoje zastosowanie w matematyce i logice, lingwistyce, filozofii, czy biologii. Maszyna Turinga jest generalizacją automatu skończonego operującą na nieskończonej pamięci.



Maszyna Turinga – stworzony przez Alana Turinga abstrakcyjny model komputera służącego do wykonywania algorytmów, składającego się z nieskończenie długiej taśmy podzielonej na pola. Taśma może być nieskończona jednostronnie lub obustronnie. Każde pole może znajdować się w jednym z N stanów. Maszyna zawsze jest ustawiona nad jednym z pól i znajduje się w jednym z M stanów. Zależnie od kombinacji stanu maszyny i pola maszyna zapisuje nową wartość w polu, zmienia stan, a następnie może przesunąć się o jedno pole w prawo lub w lewo. Taka operacja nazywana jest rozkazem. Maszyna Turinga jest sterowana listą zawierającą dowolną liczbę takich rozkazów. Liczby N i M mogą być dowolne, byle skończone. Czasem dopuszcza się też stan M+1, który oznacza zakończenie pracy maszyny. Lista rozkazów dla maszyny Turinga może być traktowana jako jej program.

Budowa:

- Taśma - Nieskończona taśma jest odpowiednikiem współczesnej pamięci komputera. Taśma dzieli się na komórki, w których umieszczone zostały symbole, czyli po prostu znaki przetwarzane przez maszynę Turinga. Symbole te stanowią odpowiednik danych wejściowych. Maszyna Turinga odczytuje te dane z kolejnych komórek i przetwarza na inne symbole, czyli dane wyjściowe. Wyniki obliczeń również są zapisywane w komórkach taśmy. Można definiować różne symbole dla maszyny Turinga. Najczęściej rozważa się jedynie symbole 0, 1 oraz tzw. znak pusty - czyli zawartość komórki, która nie zawiera żadnej danej do przetworzenia.

Wbrew pozorom taki prymitywny zbiór trzech symboli jest równoważny logicznie dowolnemu innemu zbiorowi. Przecież współczesne komputery wewnętrznie operują jedynie na bitach, a mimo to są w stanie przetwarzać prawie dowolną informację z obrazami, dźwiękiem i filmami włącznie

- Głowica - Aby przetwarzać dane, maszyna Turinga musi je odczytywać i zapisywać na taśmę. Do tego celu przeznaczona jest właśnie głowica zapisująco-odczytująca, która odpowiada funkcjonalnie urządzeniom wejścia/wyjścia współczesnych komputerów lub układom odczytu i zapisu pamięci.
  Głowica zawsze znajduje się nad jedną z komórek taśmy. Może ona odczytywać zawartość tej komórki oraz zapisywać do niej inny symbol - na tej zasadzie odbywa się przetwarzanie danych - z jednych symboli otrzymujemy inne. Oprócz odczytywania i zapisywania symboli w komórkach głowica wykonuje ruchy w prawo i w lewo do sąsiednich komórek na taśmie. W ten sposób może się ona przemieścić do dowolnie wybranej komórki taśmy.

Przed rozpoczęciem pracy maszyny Turinga głowica jest zawsze ustawiana nad komórką taśmy zawierającą pierwszy symbol do przetworzenia.

- Układ sterowania - Przetwarzaniem informacji zarządza układ sterowania głowicą. Jego współczesnym odpowiednikiem jest procesor komputera. Układ ten odczytuje za pomocą głowicy symbole z komórek taśmy oraz przesyła do głowicy symbole do zapisu w komórkach. Dodatkowo nakazuje on głowicy przemieścić się do sąsiedniej komórki w lewo lub w prawo.Podstawą działania maszyny Turinga są stany układu sterowania. Jest to pojęcie trudne do zrozumienia dla początkujących, jednakże gdy raz to się stanie, maszyna Turinga okaże się bardzo prostym automatem, który będziemy mogli dowolnie programować. Stan układu sterowania określa jednoznacznie jaką operację wykona, jak zareaguje maszyna Turinga, gdy odczyta z taśmy określony symbol.Zatem operacje wykonywane przez układ sterowania zależą od dwóch czynników:

  1.Symbolu odczytanego z komórki na taśmie.

  2.Bieżącego stanu układu sterującego.

  Działanie :

  Jak uruchomić formalnie zdefiniowaną maszynę Turinga?

  A.Trzeba odpowiednio przygotować taśmę – można rozważać różne reprezentacje wejścia, zakładamy, że taśma jest wypełniona symbolami pustymi # oprócz skończonego spójnego fragmentu, na którym znajduje się słowo wejściowe składające się z symboli wejściowych. Należy ustawić głowicę nad którąś z komórek taśmy - dla wygody będziemy ustawiać głowicę nad pierwszą komórką słowa wejściowego. Należy ustawić mechanizm sterujący na stan q0 i uruchomić maszyn

  B .Maszyna działa zgodnie z algorytmem zapisanym w funkcji jeśli będzie zmieniać swój stan, przesuwać głowicę i zmieniać zawartość taśmy. Tak będzie się działo dopóki nie nastąpi jeden z dwóch warunków:

  • Maszyna znajdzie się w stanie akceptującym. Mówimy wtedy, że obliczenie zakończyło się akceptująco.

  • Maszyna znajdując się w stanie A przeczyta z taśmy symbol x, a wartość funkcji częściowej nie będzie zdefiniowana. Mówimy wtedy, że obliczenie zakończyło się błędem.

  Uniwersalna Maszyna Turinga - Maszyna Turinga po wprowadzeniu w stan początkowy działa samoczynnie, wykonując zadanie dla którego została zaprojektowana. Proste Maszyny Turinga można składać w złożone i uzyskać maszynę wykonującą bardziej złożone operacje matematyczne. Do każdego zadania istnieje inna maszyna Turinga

  Stan obiektu – chwilowy - trwający w czasie, zestaw wszystkich wartości atrybutów oraz aktualnych powiązań danego obiektu z innymi obiektami, zmianę aktualnego stanu na inny może spowodować zajście pewnego zdarzenia????????

„Każdy problem algorytmiczny dla którego możemy znaleźć algorytm dający się zaprogramować w pewnym dowolnym języku, wykonujący się na pewnym dowolnym komputerze, nawet na takim, którego jeszcze nie zbudowano, ale można zbudować, i nawet na takim, który wymaga nieograniczonej ilości czasu i pamięci dla coraz większych danych, jest także rozwiązywalny przez maszynę Turinga."

To stwierdzenie jest jedną z wersji tzw. tezy ChurchaTuringa (Alonz Church, Alan M. Turing), którzy doszli do niej niezależnie w połowie lat trzydziestych.

Z tezy Churcha-Turinga wynika, że najpotężniejszy superkomputer z wieloma najwymyślniejszymi językami programowania, interpretatorami, kompilatorami nie jest potężniejszy od domowego komputera z jego uproszczonym językiem programowania. Mając ograniczoną ilość czasu i pamięci mogą obydwa rozwiązać te same problemy algorytmiczne, jak również żaden z nich nie może rozwiązać problemów nierozstrzygalnych (nieobliczalnych).

Maszyna rejestrowa – maszyna (procesor bądź maszyna wirtualna), w której podstawowe operacje prowadzi się na niewielkiej grupie rejestrów, nie zaś na stosie.Maszyny rejestrowe dysponują też stosem – dostępnym jawnie (instrukcjami dodawania do stosu push i zdejmowania ze stosu pop) bądź emulowanym (program modyfikuje rejestr bądź zmienną będącą wskaźnikiem do stosu), ale jest on używany do przechowywania i przekazywania danych, nie do obliczeń. kszość procesorów to maszyny rejestrowe. Do wirtualnych maszyn rejestrowych należy Parrot (maszyna wirtualna przeznaczona do języków dynamicznie typowanych, takich jak Perl, Ruby i Python.)

Maszyna stosowa to maszyna (procesor bądź maszyna wirtualna), w której podstawowe operacje prowadzi się na stosie, nie zaś na rejestrach. Większość maszyn wirtualnych to maszyny stosowe. Maszyną stosową był Transputer oraz polski minikomputer Mera 400. Także rejestry koprocesorów arytmetycznych z serii x86 są zorganizowane w stos.

Architektura von Neumanna – pierwszy rodzaj architektury komputera, opracowanej przez Johna von Neumanna, Johna W. Mauchly'ego oraz Johna Presper Eckerta w 1945 roku. Cechą charakterystyczną tej architektury jest to, że dane przechowywane są wspólnie z instrukcjami, co sprawia, że są kodowane w ten sam sposób.

W architekturze tej komputer składa się z czterech głównych komponentów:

-pamięci komputerowej przechowującej dane programu oraz instrukcje programu; każda komórka pamięci ma unikatowy identyfikator nazywany jej adresem

-jednostki sterującej odpowiedzialnej za pobieranie danych i instrukcji z pamięci oraz ich sekwencyjne przetwarzanie

-jednostki arytmetyczno-logicznej odpowiedzialnej za wykonywanie podstawowych operacji arytmetycznych.

-urządzeń wejścia/wyjścia służących do interakcji z operatorem

Jednostka sterująca wraz z jednostką arytmetyczno-logiczną tworzą procesor.

Architektura współczesnego komputera oparta jest na idei von Neumana

Architektura harwardzka – rodzaj architektury komputera. W odróżnieniu od architektury von Neumanna, pamięć danych programu jest oddzielona od pamięci rozkazów

Podstawowa architektura komputerów zerowej generacji i początkowa komputerów pierwszej generacji.

Prostsza (w stosunku do architektury von Neumanna) budowa przekłada się na większą szybkość działania - dlatego ten typ architektury jest często wykorzystywany w procesorach sygnałowych oraz przy dostępie procesora do pamięci cache (Pamięć podręczna (ang. cache) – mechanizm, w którym część spośród danych zgromadzonych w źródłach o długim czasie dostępu i niższej przepustowości jest dodatkowo przechowywana w

pamięci o lepszych parametrach. Ma to na celu poprawę szybkości dostępu do tych informacji, które przypuszczalnie będą potrzebne w najbliższej przyszłości. Pamięć podręczna jest elementem właściwie wszystkich systemów – współczesny procesor ma 2 albo 3 poziomy pamięci podręcznej oddzielającej go od pamięci RAM. Dostęp do dysku jest buforowany w pamięci RAM, a dokumenty HTTP są buforowane przez pośredniki HTTP oraz przez przeglądarkę.)

Separacja pamięci danych od pamięci rozkazów sprawia, że architektura harwardzka jest obecnie powszechnie stosowana w mikrokomputerach jednoukładowych, w których dane programu są najczęściej zapisane w nieulotnej pamięci ROM (EPROM/EEPROM), natomiast dla danych tymczasowych wykorzystana jest pamięć RAM (wewnętrzna lub zewnętrzna).

Niedeterministyczna maszyna Turinga to taka, dla której mechanizm sterujący może przy konkretnym stanie i odczycie z taśmy postąpić na kilka różnych sposobów. Wynik obliczeń jest pozytywny, jeśli choć jedna z możliwych dróg działania maszyny doprowadzi do sukcesu.

Hipoteza Churcha-Turinga (zwana również Tezą Churcha-Turinga) jest hipotezą określającą możliwości komputerów i innych maszyn obliczeniowych. Mówi ona, że każdy problem, dla którego przy nieograniczonej pamięci oraz zasobach istnieje efektywny algorytm jego rozwiązywania, da się rozwiązać na maszynie Turinga. Hipoteza jest niemożliwa do sprawdzenia matematycznie, ponieważ łączy w sobie zarówno ścisłe, jak i nieprecyzyjne sformułowania, których interpretacja może zależeć od konkretnej osoby. Dlatego traktowana jest bardziej jako aksjomat lub swoiste prawo.

Mniej formalnie, hipoteza pozwala dokładniej sformułować pojęcie samego algorytmu, mówiąc jednocześnie, że komputery są w stanie go wykonać. Co więcej, wszystkie komputery mają jednakowe możliwości, jeśli chodzi o zdolność do ich wykonywania, a nie dostępne zasoby oraz że nie jest możliwe zbudowanie komputera o zdolności do rozwiązywania większej liczby problemów, niż najprostsza maszyna Turinga.

Modele hiperobliczeń

Istnieją modele obliczeń, które faktycznie rozszerzają maszyny Turinga, ale zwykle są uznawane za nierealizowalne w żadnym praktycznym sensie.

Nieskończone obliczenia

Wyobraźmy sobie maszynę, której każdy kolejny krok wymaga o połowę mniej czasu niż poprzedni. Jeśli pierwszy krok wymaga jednej jednostki czasu, całe obliczenie trwa

$$1 + \frac{1}{2} + \frac{1}{4} + \cdots$$

W ciągu 2 jednostek czasu maszyna może wykonać nieskończoną liczbę kroków. Taka maszyna jest zdolna do rozwiązywania problemu stopu.

Maszyny z wyrocznią

Wyobraźmy sobie że nasza maszyna ma dostęp do "wyroczni" która może odpowiadać na pytania dotyczące wyszczególnionych nierozstrzygalnych problemów. Taki model jest często rozważany w kryptografii, gdy modelujemy urządzenia szyfrujące jako mające dostęp do nieznanego nam źródła informacji.

Ograniczenia hiperobliczeń

Okazuje się, że powyższe maszyny mają swoje ograniczenia analogiczne do ograniczeń maszyn Turinga. O ile potrafią rozstrzygać problem stopu dla maszyn Turinga, nie potrafią rozstrzygać własnych problemów stopu: przykładowo maszyna wyposażona w wyrocznię nie będzie potrafiła odpowiedzieć na pytanie czy podana maszyna z wyrocznią kiedykolwiek się zatrzyma.

W matematyce i informatyce, maszyna Zenona (w skrócie ZM, a także e przyspieszona maszyna Turinga, ATM) jest to hipotetyczny model obliczeniowy związany z maszyną Turinga pozwala na przeliczanie nieskończonej liczby algorytmicznych kroków, jakie należy przeprowadzić w skończonym czasie. Maszyny te są wykluczone w większości modeli obliczeniowych.

Komputer kwantowy – układ fizyczny, do opisu którego wymagana jest mechanika kwantowa, zaprojektowany tak, aby wynik ewolucji tego układu reprezentował rozwiązanie określonego problemu obliczeniowego. Dane w komputerach kwantowych są reprezentowane przez aktualny stan kwantowy układu stanowiącego komputer. Jego ewolucja odpowiada procesowi obliczeniowemu. Odpowiednie zaplanowanie ewolucji układu kwantowego, czyli stworzenie odpowiedniego algorytmu kwantowego pozwala teoretycznie na osiągnięcie wyników w znacznie efektywniejszy sposób, niż za pomocą tradycyjnych komputerów.

Podstawowymi elementami budowy kwantowego komputera są kwantowe bramki logiczne. Kwantowy bit, tzw. kubit, zgodnie z prawami mikroświata nie będzie miał ustalonej wartości 1 lub 0, tak jak bit w standardowym komputerze. W trakcie obliczeń będzie się znajdował w jakimś stanie pośrednim. Rządzi tym prawo prawdopodobieństwa, podobnie jak położeniem elektronu w atomie. Kubit jest kwantową superpozycją zera i jedynki. Pojedynczy wynik obliczeń komputera kwantowego będzie niepewny. Istotne staje się wykonanie całej serii obliczeń i dopiero ich średnia wartość z dużą dokładnością określi prawidłowy wynik – tym dokładniejszy, im więcej komputer dokona obliczeń. Kubit niesie w sobie naraz o wiele więcej informacji niż zero-jedynkowy bit. Dlatego jest w stanie wykonać równolegle wiele obliczeń.

Bramki kwantowe – proste elementy wykonujące podstawowe obliczenia przeprowadzane przez algorytmy kwantowe. Bramki kwantowe stanowią podstawowe operacje realizowane przez komputery kwantowe i służą do przetwarzania informacji kwantowej. Bramki kwantowe na schematach obwodów kwantowych oznaczamy za pomocą ramek, a w obliczeniach stosujemy postać macierzy unitarnych. obliczenia na bramkach kwantowych są odwracalne.

4. *Algorytmy i ich złożoność*

Algorytm – jest skończonym, uporządkowanym ciągiem jasno zdefiniowanych czynności, koniecznych do wykonania postawionego zadania.

Cechy algorytmów:

- poprawność (algorytm daje oczekiwane wyniki),
- jednoznaczność (zawsze daje te same wyniki przy takich samych danych wejściowych),
- skończoność (wykonuje się w skończonej liczbie kroków),
- sprawność (czasowa - szybkość działania i pamięciowa)

Programowanie imperatywne – paradygmat programowania(wzorzec programowania komputerów przedkładany w danym okresie rozwoju informatyki ponad inne lub ceniony w pewnych okolicznościach lub zastosowaniach.), który opisuje proces wykonywania jako sekwencję instrukcji zmieniających stan programu. Podobnie jak tryb rozkazujący w lingwistyce wyraża żądania jakichś czynności do wykonania. Programy imperatywne składają się z ciągu komend do wykonania przez komputer. Rozszerzeniem (w sensie wbudowanych funkcji) i rodzajem (w sensie paradygmatu) programowania imperatywnego jest programowanie proceduralne.

Występują następujące sposoby przedstawiania algorytmów:

-Słowny , na ogół mało dokładny.

-Lista kroków.

-Schemat blokowy.

-Drzewo algorytmu.

Różnice między programem komputerowym a algorytmem???????

Program komputerowy to algorytm napisany w języku programowania. Język programowania musi być zrozumiały dla komputera. Tekst programu może też służyć ludziom do przekazywania sobie algorytmów.

Komputery przetwarzają przekazywane im informacje z wykorzystaniem algorytmów. Program jest algorytmem zapisanym w języku zrozumiałym dla maszyny (kodzie maszynowym). Każdy poprawny kod maszynowy da się przełożyć na zestaw instrukcji dla teoretycznego modelu komputera – maszyny Turinga.

Zwykle algorytmy pracują na danych wejściowych i uzyskują z nich dane wyjściowe. Informacje zapisane w pamięci maszyny traktuje się jako jej stan wewnętrzny. Niektóre algorytmy mają za zadanie wyłącznie przeprowadzanie komputera z jednego stanu wewnętrznego do innego.

Każdy algorytm komputerowy musi być wprowadzony do komputera w bardzo rygorystycznie zdefiniowanym języku. Ludzie często komunikując się, przesyłają między sobą informację wieloznaczne. Komputery mogą reagować tylko na całkowicie jednoznaczne instrukcje. Jeżeli dany algorytm da się wykonać na maszynie o dostępnej mocy obliczeniowej i pamięci oraz akceptowalnym czasie, to mówi się, że jest obliczalny.

Poprawne działanie większości algorytmów implementowanych w komputerach opiera się na kolejnej realizacji pewnego zestawu warunków. Jeżeli któryś z nich nie zostanie spełniony, to program kończy się komunikatem o błędzie. Czasami podczas implementacji algorytmu ważny warunek zostanie pominięty. Dla przykładu, mamy program dzielący przez siebie dwie liczby. Użytkownik poleca wykonać dzielenie przez zero. Działanie aplikacji, która nie sprawdzi warunku „dzielnik nierówny zero", zostanie przerwane przez system operacyjny komputera.

Metody konstruowania algorytmów

- dziel i zwyciężaj – dzielimy problem na kilka mniejszych, a te znowu dzielimy, aż ich rozwiązania staną się oczywiste,
- programowanie dynamiczne – problem dzielony jest na kilka, ważność każdego z nich jest oceniana i po pewnym wnioskowaniu wyniki analizy niektórych prostszych zagadnień wykorzystuje się do rozwiązania głównego problemu,
- metoda zachłanna – nie analizujemy podproblemów dokładnie, tylko wybieramy najbardziej obiecującą w tym momencie drogę rozwiązania,
- programowanie liniowe – oceniamy rozwiązanie problemu przez pewną funkcję jakości i szukamy jej minimum,
- poszukiwanie i wyliczanie – kiedy przeszukujemy zbiór danych aż do odnalezienia rozwiązania,
- heurystyka – człowiek na podstawie swojego doświadczenia tworzy algorytm, który działa w najbardziej prawdopodobnych warunkach, rozwiązanie zawsze jest przybliżone.

Algorytm „naiwny" - Jest to najprostszy algorytm wyszukiwania wzorca. Algorytm tego typu jest bardzo prosty zarówno w zrozumieniu, jak i implementacji, jednak ma przy tym swoje wady. Otóż, jak łatwo się domyślić, jest on zdecydowanie wolniejszy niż bardziej złożone metody służące do osiągnięcia tego samego celu. Należy jednak pamiętać, że zdecydowana większość nowocześniejszych metod wyszukiwania wzorca opiera się właśnie na tym podstawowym sposobie wyszukiwania.

Algorytm z nawrotami (ang. backtracking) – ogólny algorytm wyszukiwania wszystkich (lub kilku) rozwiązań niektórych problemów obliczeniowych, który stopniowo generuje kandydatów na rozwiązanie jednak, gdy stwierdzi, że znaleziony kandydat c nie może być poprawnym rozwiązaniem, nawraca (ang. "backtracks") do punktu, gdzie może podjąć inną decyzje związaną z jego budową

Algorytm zachłanny (ang. greedy algorithm) – algorytm, który w celu wyznaczenia rozwiązania w każdym kroku dokonuje zachłannego, tj. najlepiej rokującego w danym momencie wyboru rozwiązania częściowego[1]. Innymi słowy algorytm zachłanny nie dokonuje oceny czy w kolejnych krokach jest sens wykonywać dane działanie, dokonuje decyzji lokalnie optymalnej, dokonuje on wyboru wydającego się w danej chwili najlepszym, kontynuując rozwiązanie podproblemu wynikającego z podjętej decyzji

Ostatnią grupą metod, którą chcielibyśmy omówić, są metody heurystyczne (heurystyki) - słowa te pochodzą od greckiego Eureka). Główna idea działania metod tego typu polega na szacowaniu rozwiązania podproblemów w „inteligentny" i zdroworozsądkowy sposób. Mówiąc "inteligentny", mamy na myśli oszacowanie na podstawie czasem niepełnych danych, jaka może być wartość wynikowa danego podproblemu (znając pewne fakty formułujemy hipotetyczne rozwiązanie) – na przykład poprzez uproszczenie pewnego modelu do minimum, zrelaksowanie ograniczeń podzadania czy przyjęcie typowych parametrów danych. Stosując te metody nie mamy pewności, ze znajdziemy optymalne rozwiązanie. Jednakże istnieją sytuacje, w których zastosowanie „normalnego" algorytmu powoduje bardzo duże koszty (czasowe i obliczeniowe) znalezienia właściwego rozwiązania, a w skrajnym przypadku nie znajduje go wcale bez użycia elementów heurystyki. Należy w tym miejscu zaznaczyć, że podejście algorytmiczne tym różni się od heurystycznego, że stosując algorytmy uruchamiamy pewien regularny, jednoznacznie określony proces gwarantujący nam znalezienie właściwego rozwiązania. Z kolei heurystyki są procesami twórczymi, zbiorami wskazówek i hipotez, które nie zapewniają wyszukania poprawnego rozwiązania.

Metodami heurystycznymi posługujemy się najczęściej w problemach, gdy brakuje nam pewnych danych, dzięki którym bylibyśmy w stanie znaleźć rozwiązanie przy użyciu klasycznych algorytmów.

Złożoność obliczeniowa algorytmu

Ilość zasobów niezbędnych do wykonania algorytmu można rozumieć jako jego złożoność. W zależności od rozważanego zasobu mówimy o złożoności czasowej czy też pamięciowej. Oczywiście w większości wypadków ilość potrzebnych zasobów będzie się różnić w zależności od danych wejściowych z zakresu danego zagadnienia.

Przykładowo rozpatrzmy rozkład liczb na czynniki pierwsze. Domyślamy się, że (niezależnie od zastosowanego algorytmu) im większa liczba, tym więcej zasobów będzie potrzebnych do jej rozłożenia. Tę cechę podziela większość zagadnień obliczeniowych: im większe rozmiary danych wejściowych, tym więcej zasobów (czasu, procesorów, pamięci) jest koniecznych do wykonania danych obliczeń. Złożoność algorytmu jest więc funkcją rozmiaru danych wejściowych.

Kolejnym problemem jest fakt, iż złożoność zwykle nie zależy wyłącznie od rozmiaru danych, ale może się znacznie różnić dla danych wejściowych o identycznym rozmiarze. Dwoma często stosowanymi sposobami podejścia są: rozpatrywanie przypadków najgorszych (złożoność pesymistyczna) oraz zastosowanie określonego sposobu uśrednienia wszystkich możliwych przypadków (złożoność oczekiwana).

5. Struktury danych

Struktura danych (ang. data structure) - sposób uporządkowania informacji w komputerze. Na strukturach danych operują algorytmy.

Przykładowe struktury danych to:

-rekord lub struktura (ang. record, struct), logiczny odpowiednik to krotka
-tablica
-lista
-stos
-kolejka

-drzewo i jego liczne odmiany (np. drzewo binarne)
-graf

Podczas implementacji (jest to proces pisania programu (kodu źródłowego), czyli programowanie lub efekt takiego procesu, czyli program.)programu programista często staje przed wyborem między różnymi strukturami danych, aby uzyskać pożądany efekt. Odpowiedni wybór może zmniejszyć złożoność obliczeniową, ale z drugiej strony trudność implementacji danej struktury może stanowić istotną przeszkodę.

Ponieważ struktury danych są w programie rzeczą szczególnie istotną, wiele języków programowania wspiera programistę, dostarczając bibliotekę standardową z zaimplementowanymi różnorodnymi strukturami danych. Można tu wymienić Standard Template Library w C++, API języka Java oraz platformę .NET. Próbą połączenia idei struktur danych i algorytmów jest pomysł programowania obiektowego.

Struktura pamięci operacyjnej komputera

RAM (Random Access Memory) - pamięć o dostępie swobodnym (bezpośrednim). Przechowywane są w niej wszystkie instrukcje wykonywane przez procesor oraz związane z nimi dane. W czasie pracy komputera można zapisywać i odczytywać z niej dane, które po odłączeniu zasilania giną bezpowrotnie.

Najważniejsze parametry pamięci:

1. pojemność
2. szybkość
    - maksymalny czas dostępu
    - czas cyklu
    - szybkość transmisji
3. pobór mocy

Podstawowym elementem układu pamięci jest **komórka pamięci**, w której można przechowywać pojedynczą cyfrę binarną. Moduł pamięci jest zorganizowany w taki sposób, że komórki tworzą matrycę, która składa się z wierszy (rows) oraz kolumn (columns). Odszukanie komórki możliwe jest przez podaniejej adresu, czyli numeru wiersza i kolumny. Adresy, które mają ten sam numer wiersza, określane są jako strony(page). Pamięć nie jest odczytywana bit po bicie, ale porcjami, np. 32 lub 64 bity jednocześnie.

DRAM (Dynamic RAM)- do przechowywania informacji używany jest jeden tranzystor na komórkę pamięci. Każda jednostka pamięci to kondensator, połączony z tranzystorem MOS używanym do realizacji odczytu i zapisu. Informacja jest pamiętana w postaci ładunków, ale niestety pozostaje zachowana tylko przez krótki czas. Dlatego też kondensator pamiętający wymaga regularnego doładowania(co ok. 2...8 ms). Operacje tę nazywamy odświeżaniem (refresh).

RAM można klasyfikować:

- ze względu na sposób wybierania elementów z matryc

- ze względu na technologie wykonania

Pamięć typu 2D - struktura liniowa (jedno wymiarowa), każda komórka pamięci posiada jedno wejście wybierające i jedno wyjście danych. Wybór elementu dokonujemy za pomocą pojedynczego sygnału.

Wada: pamięć o dużej pojemnościwymaga zastosowania skomplikowanego dekodera(wysoki koszt wykonania)

n - liczba wejść

liczba wyjść 2^n

Pamięć typu 3D - struktura liniowa (dwuwymiarowa),komórka pamięci posiada dwa wejścia wybierające - sygnały z dekodera kolumn CAS (Column Adrdress Strobe) i wierszy RAS(Row Address Strobe). Obydwa muszą być aktywne, aby element został wybrany(na obu wejściach logiczna 1) czas dostępu do układu pamięci zależy od długościcyklu obu tych sygnałów.

Wada: dużo wolniejsza od 2D

SRAM (Static RAM) - komórki pamięci przechowują dane wykorzystując, działające jako przerzutniki RS, pary układów tranzystorowych. Ponieważ do realizacji jednej komórki układu SRAM koniecznych jest sześć tranzystorów, pamięci statyczne mogą przechowywać mniej bitów na mm²krzemu niż pamięci dynamiczne i potrzebują większej mocy zasilania. W odróżnieniu od DRAM nie wymaga odświeżania, dzięki czemu jest znacznie szybsza. Z tych względów stosowana jest głównie jako pamięć podręczna (cache).

Przykładowe typy danych występujące w wielu językach programowania:

- typ całkowity (w C, C++, Javie np. int, w Pascalu np. integer) – typ reprezentujący liczbę całkowitą z jakiegoś zakresu zależnego od języka a nawet konkretnej implementacji.
- typ zmiennoprzecinkowy (w C, C++, Javie np. double, w Pascalu np. real) – typ reprezentujący przybliżoną wartość liczby rzeczywistej.
- typ stałopozycyjny (w PL/1, Cobol) – typ reprezentujący liczbę wymierną o stałym mianowniku.
- typ znakowy (w C, C++, Javie, Pascalu np. char) – typ reprezentujący pojedynczy znak (ASCII lub w nowszych implementacjach Unicode)
- typ tekstowy (w Javie string, w C++, Pascalu np. string) – typ reprezentujący cały tekst. W C jego rolę pełni wskaźnik do literału znakowego – const char*
- typ wskaźnikowy – oznacza wskaźnik na zmienną zadanego typu.
- typ referencyjny – odmiana wskaźnika, referencja jest różnie pojmowana w poszczególnych językach.

- typ wyliczeniowy – typ mogący przyjmować jedną z zadanych symbolicznych wartości, np. (czerwony, zielony, niebieski)
- typ tablicowy – ciąg zmiennych zadanego typu indeksowanych liczbą naturalną z pewnego przedziału (w większości języków programowania) lub dowolnym unikalnym kluczem który może być zarówno liczbą jak i ciągiem znaków (w PHP)
- typ strukturalny – zespół połączonych w jedną całość zmiennych zwanych polami struktury. Do każdego pola można się odwoływać oddzielnie.
- klasa (typ obiektowy) – odmiana struktury, w której oprócz zespołu danych dodane są także procedury działających na tych danych. Zmienna typu klasy nazywa się obiektem. Pojęcie klasy spowodowało powstanie nowego paradygmatu programowania, zwanego programowaniem obiektowym i zrewolucjonizowało sposób myślenia programisty, który od tej pory patrzy na program jako na zbiór autonomicznych obiektów.
- typ pusty (np. void w C i C++) – występuje np. w oznaczaniu funkcji nie zwracających żadnych wartości.
- typ logiczny (np. bool w C++) – może przyjmować wartości logiczne 1 (true, t) lub 0 (false, nil).
- typ bitowy reprezentujący ciąg bitów, (np. '01101'B – PL/I).
- typ zbiorowy reprezentujący zbiory elementów (np. [pon, wt, sr, czw, pt] – Pascal).
- typ zespolony reprezentujący liczby zespolone.
- typ etykietowy reprezentujący wartości będące etykietami instrukcji, wskazaniami instrukcji.

Typy złożone

- Struktury to specjalny typ danych mogący przechowywać wiele wartości w jednej zmiennej. Od tablic jednakże różni się tym, iż te wartości mogą być różnych typów.
- Unie - Pola w unii nakładają się na siebie w ten sposób, że w danej chwili można w niej przechowywać wartość tylko jednego typu. Unia zajmuje w pamięci tyle miejsca, ile zajmuje największa z jej składowych. W powyższym przypadku unia będzie miała prawdopodobnie rozmiar typu double czyli często 64 bity, a całkowita i znak będą wskazywały odpowiednio na pierwsze cztery bajty lub na pierwszy bajt unii (choć nie musi tak być zawsze). Dlaczego tak? Taka forma często przydaje się np. do konwersji pomiędzy różnymi typami danych. Możemy dzięki unii podzielić zmienną 32-bitową na cztery składowe zmienne o długości 8 bitów każda. Do konkretnych wartości pól unii odwołujemy się, podobnie jak w przypadku struktur, za pomocą kropki Zazwyczaj użycie unii ma na celu zmniejszenie zapotrzebowania na pamięć, gdy naraz będzie wykorzystywane tylko jedno pole i jest często łączone z użyciem struktur.
- Tablica jest zmienną złożoną z elementów tego samego typu. Obejmuje ona ciągły obszar pamięci operacyjnej dokładnie tak duży, aby zmieścić wszystkie jej elementy. W języku C i C++ nie ma żadnych wbudowanych mechanizmów zabezpieczających przed odwoływaniem się do „elementów" leżących poza zakresem indeksowym tablic.

Abstrakcyjny typ danych (z angielskiego abstract data type), typ danych zdefiniowany przez użytkownika, abstrakcyjny w tym sensie, że nie implementowany przez sprzęt komputerowy (jak np. typ całkowity lub zmiennopozycyjny). Pod każdym innym względem abstrakcyjny typ danych zachowuje się po zdefiniowaniu jak typ wbudowany. Przykładem abstrakcyjnego typu danych jest typ napisowy (string), którego dane można między sobą porównywać w porządku leksykograficznym lub złączać, nierealizowany bezpośrednio przez sprzęt. Bardziej komunikatywną nazwą abstrakcyjnego typu danych jest "typ definiowany przez użytkownika".

Abstrakcja to — w naszym rozumieniu — reprezentacja pewnego bytu, w której pominięto nieistotne w danym kontekście szczegóły. Pozwala nam to grupować byty według ich wspólnych cech i — zależnie od potrzeby — albo zajmować się całymi grupami (czyli owymi wspólnymi cechami), albo bytami wewnątrz grupy (czyli szczegółami różniącymi byty w grupie). Chodzi oczywiście o to, by poradzić sobie ze złożonością problemów.

Dwie podstawowe abstrakcje w językach programowania to:

- Abstrakcja procesu: Abstrakcjami procesów są podprogramy. Pozwalają wskazać (przez ich wywołanie), że pewna czynność ma być wykonana, bez wskazywania jak ma być wykonana. Szczegóły znajdują się w treści podprogramu, której wywołujący nie musi znać.
- Abstrakcja danych: Zamknięta całość obejmująca reprezentację pewnego typu danych wraz z podprogramami, umożliwiającymi działanie na tych danych.

Co to jest abstrakcyjny typ danych? Jak już wspominaliśmy, jest to konstrukcja języka programowania, w której definiujemy typ (w dotychczasowym rozumieniu) oraz operacje na nim w taki sposób, że inne byty w programie nie mogą manipulować danymi inaczej niż za pomocą zdefiniowanych przez nas operacji. Istotą rzeczy jest tu oddzielenie części „prywatnej” typu (czyli szczegółów reprezentacji danych i implementacji poszczególnych operacji) od części „publicznej” (tego, co można wykorzystywać w innych miejscach programu). Rozdzielenie składników abstrakcyjnego typu danych na część prywatną i publiczną jest możliwe za pomocą zawartych w języku programowania mechanizmów sterowania dostępem. Dane w typie abstrakcyjnym zwane są właściwościami lub danymi składowymi; używa się też określeń pole lub po prostu zmienna... Operacje zwane są metodami lub funkcjami składowymi.

Przywołajmy ponownie przykłady

Wbudowane w języki programowania typy pierwotne można uznać za abstrakcyjne: nie mamy dostępu do reprezentacji wewnętrznej, więc możemy posługiwać się jedynie tym, co dostarcza język.

Zdefiniowany w Pascalu typ rekordowy i kilka procedur na nim działających nie stanowi abstrakcyjnego typu danych. Ktokolwiek zadeklaruje rekord tego typu, będzie mógł działać bezpośrednio na tym rekordzie z pominięciem „oficjalnych” procedur.

Sztandarowy przykład to klasa np. z Javy lub C++. Dane schowane w części prywatnej klasy nie są dostępne na zewnątrz, stąd nie da się wykonywać żadnych operacji bezpośrednio.

Co to jest obiekt? Obiekt to instancja abstrakcyjnego typu danych, czyli pojedynczy „egzemplarz” tego typu zaalokowany na stercie (najczęściej), na stosie lub statycznie (najrzadziej). Sam abstrakcyjny typ danych w większości języków zwany jest klasą.

Jakie warunki powinien spełniać abstrakcyjny typ danych definiowany przez użytkownika? Deklaracja części publicznej (interfejsu), czyli danych i podprogramów przeznaczonych do wykorzystania przez inne byty w programie, powinna mieścić się w jednej jednostce syntaktycznej. Pozostała część, czyli ciała podprogramów publicznych (implementacja) oraz dane i podprogramy przeznaczone do wewnętrznego użytku, może mieścić się

w tej samej lub innej jednostce syntaktycznej. Inne byty w programie mogą tworzyć zmienne definiowanego typu abstrakcyjnego. Jedynymi bezpośrednimi operacjami (także w sensie dostępu do zmiennych) na obiektach tego typu są te zawarte w części publicznej.

Czy lepiej mieć interfejs i implementację w jednej jednostce syntaktycznej, czy osobno? Razem — pozwala organizować program w sposób modułowy. Osobno — umożliwia ukrycie (w sensie wizualnym, nie tylko w sensie dostępu) części implementacyjnej przed użytkownikami, co zmniejsza pokusę doraźnego manipulowania implementacją.

Implementacja w różnych językach

Co powinno być w języku?

- Jednostka syntaktyczna mieszcząca definicję typu (być może część publiczna osobno).
- Sposób wyszczególnienia danych i podprogramów publicznych.
- Kilka podstawowych, wbudowanych operacji na obiektach z typu, np. podstawienie, sprawdzenie równości.
- Pewne operacje są potrzebne niemal w każdym typie, są jednak zależne od szczegółów tego typu. Muszą zatem być implementowane przez programistę. Są to np. konstruktory, destruktory, iteratory.
- Język może oferować abstrakcyjne typy danych wprost (C++, C#, Java) lub może mieć bardziej ogólne konstrukcje (np. Ada).
- Dwa pytania implementacyjne: Czy abstrakcyjne typy danych mogą być parametryzowane? Jak realizowane jest sterowanie dostępem

C++

- Język oferuje dwie konstrukcje: class i struct, różniące się domyślnymi regułami dostępu.
- Klasy języka C++ są typami.
- Jednostka programu, która zadeklarowała instancję klasy (obiekt), ma dostęp do publicznych bytów tej klasy, ale tylko poprzez tę instancję.
- Każda instancja klasy ma własny zestaw danych, natomiast funkcje (metody) nie są powielane lecz przechowywane wspólnie dla całej klasy.
- Obiekty mogą być statyczne oraz dynamiczne, alokowane na stosie (dostęp przez „zwykłe” zmienne) lub na stercie (dostęp przez wskaźniki).
- Obiekty mogą zawierać zmienne dynamiczne alokowane na stercie, czyli poza obiektem.
- Alokacja i dealokacja na stercie są jawne; służą do tego operacje new i delete.
- Funkcje z klasy mogą być kompilowane jako inline. Kod funkcji jest wówczas kopiowany do wywołującego, eliminując czasochłonną obsługę wywołania. Mechanizm ten jest przydatny dla funkcji niewielkich objętościowo.
- Elementy klasy mogą być deklarowane jako prywatne (dostęp tylko wewnątrz obiektu), publiczne (dostęp bez ograniczeń) lub chronione (dostęp dla potomków). Osiąga się to przez umieszczenie deklaracji w częściach, odpowiednio, private, public i protected.
- Deklaracja friend pozwala dać „obcym” klasom dostęp do elementów prywatnych.
- Definicja klasy może zawierać konstruktor, który będzie niejawnie wywoływany przy tworzeniu obiektu z klasy. Konstruktor ma taką samą nazwę jak klasa. Konstruktory mogą mieć parametry i mogą być przeciążane.
- Definicja klasy może też zawierać destruktor, wołany przy dealokacji obiektu. Nazwa destruktora to nazwa klasy poprzedzona tyldą.

Kolejka (ang. queue) – liniowa struktura danych, w której nowe dane dopisywane są na końcu kolejki, a z początku kolejki pobierane są dane do dalszego przetwarzania (bufor typu FIFO, First In, First Out; pierwszy na wejściu, pierwszy na wyjściu).

Specjalną modyfikacją kolejki jest kolejka priorytetowa – każda ze znajdujących się w niej danych dodatkowo ma przypisany priorytet, który modyfikuje kolejność późniejszego wykonania. Oznacza to, że pierwsze na wyjściu niekoniecznie pojawią się te dane, które w kolejce oczekują najdłużej, lecz te o największym priorytecie.

Kolejkę spotyka się przede wszystkim w sytuacjach związanych z różnego rodzaju obsługą zdarzeń. W szczególności w systemach operacyjnych ma zastosowanie kolejka priorytetowa, przydzielająca zasoby sprzętowe uruchomionym procesom.

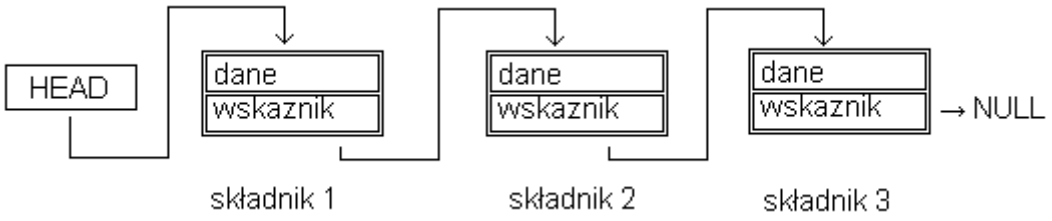
Tablica dynamiczna (ang. dynamic array) jest tworzona w czasie uruchomienia programu. Jej rozmiar może być wyliczany. Co więcej, gdy przestanie być potrzebna możemy ją usunąć z pamięci. Dzięki tym własnościom program efektywniej wykorzystuje zasoby pamięciowe komputera.

Listy, czyli struktury, które umożliwiają tworzenie ciągów danych w taki sposób, że każda dana pamięta gdzie się znajduje kolejna.

Dla przykładu weźmy na warsztat program do odtwarzania muzyki. Zazwyczaj wyposażony jest on w playlistę będącą zbiorem naszych ulubionych utworów, które chcemy odsłuchiwać. Playlista może mieć dowolną długość: można na niej trzymać zarówno 20, jak i 2000 piosenek, w dodatku w każdej chwili mamy możliwość dodania dowolnej ilości nowych. Ciężko sobie wyobrazić, by program przydzielał przy uruchomieniu kilka megabajtów "na wszelki wypadek", aby zabezpieczyć się przed sytuacją braku miejsca. Jednym ze sposobów rozwiązania tego problemu jest wykorzystanie nowej struktury danych - *list*. Ich zasadniczą cechą jest to, że zawsze zajmują w pamięci dokładnie tyle miejsca, ile potrzeba i zawsze można bezproblemowo dodać do nich nowe elementy.

Budowa listy

Każdy rekord w liście, oprócz przechowywanych przez siebie danych, zawiera również przynajmniej jedno dodatkowe pole zawierające adres/położenie następnego w kolejności rekordu. Powstaje w ten sposób łańcuch, w którym rekordy znają jedynie swoje następniki. Ilustruje to poniższy schemat:

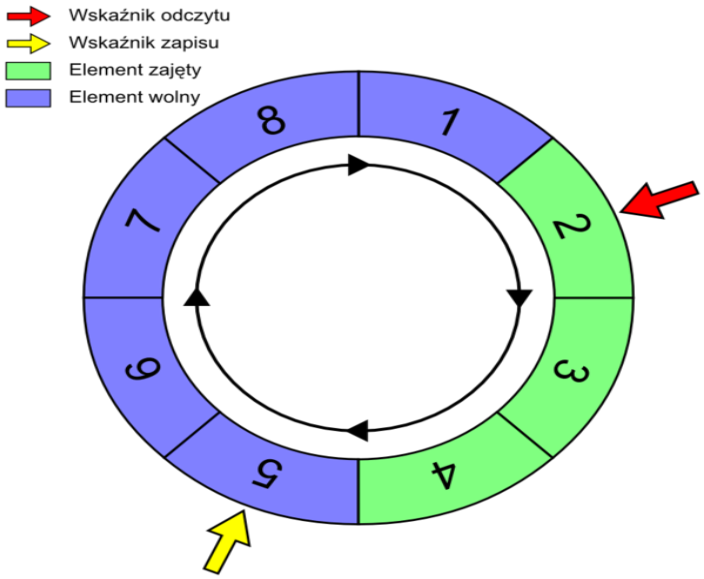


Oprócz tego musimy wydzielić też specjalny rekord zawierający tzw. *korzeń listy*, czyli odnośnik do pierwszego elementu. Ostatni rekord na liście posiada pusty wskaźnik, dzięki czemu możliwe jest powiadomienie o końcu łańcucha. Zauważmy, że w takiej strukturze bezpośredni dostęp do dowolnego elementu jest niemożliwy. Odszukanie interesującego nas rekordu zajmuje rosnącą liniowo ilość czasu. Najpierw musimy ustawić się na początku listy, a następnie przechodzić do kolejnych elementów, dopóki nie natrafimy na nasz właściwy.

Pokazana powyżej lista jest w zasadzie tylko jedną z jej odmian, zwaną *listą jednokierunkową*, ponieważ możemy po niej poruszać się tylko w jednym kierunku: do przodu. Inne rodzaje list to:

- *Listy dwukierunkowe* - każdy rekord zawiera dodatkowo drugi wskaźnik pokazujący *poprzedni* element listy.

- *Listy cykliczne* - pierwszy i ostatni rekord listy są ze sobą połączone, tworząc zamknięty cykl.



Bufor cykliczny - w informatyce bufor zorganizowany w ten sposób, że dane są przechowywane w tablicy, a dodatkowo przechowywane są dwa wskaźniki lub indeksy tablicy pokazujące pierwszy i ostatni element (albo pierwszy i puste miejsce za ostatnim). Dopisywanie nowych danych wymaga inkrementacji wskaźnika na ostatni element. W przypadku dojścia do końca tablicy jest on przemieszczany na początek. Podobnie wskaźnik odczytu po dojściu do końca tablicy przemieszcza się na początek. Bufor na ogół reprezentuje kolejkę FIFO, można też zaimplementować na nim bufor, w którym dane mogą być dopisywane i czytane z obydwu stron.

Zalety:

- prostota konstrukcji,
- szybki dostęp,
- oszczędność czasu przepisywania danych.

Wady:

- ograniczenie z góry wielkości bufora przez wielkość tablicy

Przykłady:

- bufor klawiatury w PC

Zasada działania bufora cyklicznego

Stos (ang. Stack) – liniowa struktura danych, w której dane dokładane są na wierzch stosu i z wierzchołka stosu są pobierane (bufor typu LIFO, Last In, First Out; ostatni na wejściu, pierwszy na wyjściu). Ideę stosu danych można zilustrować jako stos położonych jedna na drugiej książek – nowy egzemplarz kładzie się na wierzch stosu i z wierzchu stosu zdejmuje się kolejne egzemplarze. Elementy stosu poniżej wierzchołka można wyłącznie obejrzeć, aby je ściągnąć, trzeba najpierw po kolei ściągnąć to, co jest nad nimi.

Stos jest używany w systemach komputerowych na wszystkich poziomach funkcjonowania systemów informatycznych. Stosowany jest przez procesory do chwilowego zapamiętywania rejestrów procesora, do przechowywania zmiennych lokalnych, a także w programowaniu wysokopoziomowym.

Przeciwieństwem stosu jest kolejka, bufor typu FIFO (ang. First In, First Out; pierwszy na wejściu, pierwszy na wyjściu), w którym dane obsługiwane są w takiej kolejności, w jakiej zostały dostarczone (jak w kolejce do kasy).

Drzewa w informatyce, to bardzo szczególne struktury. Jak zobaczymy później są to pewne szczególne grafy i jako takie mają bardzo ścisłą definicję matematyczną, którą podamy przy okazji ich omawiania. Na razie wystarczy nam wiedzieć, że w drzewie istnieje pewien wyróżniony element nazywany korzeniem (na rysunku obok oznaczony przez F), od którego odchodzą poszczególne elementy drzewa – wierzchołki, które połączone są krawędziami. Ciąg krawędzi łączących poszczególne wierzchołki nazywamy ścieżką.

Istnieje dokładnie jedna ścieżka łącząca korzeń z danym wierzchołkiem.

Wszystkie wierzchołki połączone z danym wierzchołkiem, a leżące na następnym poziomie są nazywane dziećmi tego węzła (np. dziećmi wierzchołka F są B i G, natomiast wierzchołka B: A i D). Wierzchołek może mieć dowolną liczbę dzieci, jeśli nie ma ich wcale nazywany jest liściem. Liśćmi w przykładowym drzewie są A, C, E, H.

Idąc dalej tym tokiem myślowym wierzchołek jest rodzicem dla każdego swojego dziecka.

Każdy węzeł ma dokładnie jednego rodzica, wyjątkiem jest korzeń drzewa, który nie ma rodzica

Jednym z najczęściej wykorzystywanych typów drzew jest drzewo binarne, różni się ono od zwykłego drzewa dodatkowym warunkiem - każdy rodzic może mieć maksymalnie dwoje dzieci. Zwyczajowo mówi się wtedy o lewym i prawym dziecku.

Drzewo czerwono-czarne – rodzaj samoorganizującego się binarnego drzewa poszukiwań - struktury danych stosowanej w informatyce najczęściej do implementacji tablic asocjacyjnych. Została ona wynaleziona przez Rudolfa Bayera w 1972 roku, który nazwał je symetrycznymi binarnymi B-drzewami. Współczesną nazwę oraz dokładne zbadanie ich właściwości zawdzięcza się pracy "A dichromatic framework for balanced trees" z 1978 roku autorstwa Leo J. Guibasa oraz Roberta Sedgewicka.

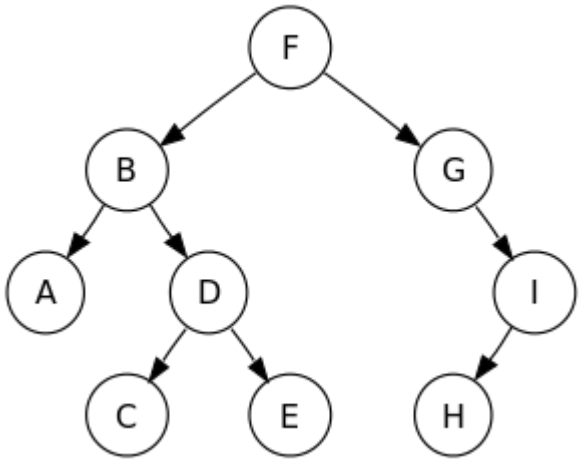
Drzewa czerwono-czarne są skomplikowane w implementacji, lecz charakteryzują się niską złożonością obliczeniową elementarnych operacji takich, jak wstawianie, wyszukiwanie czy usuwanie elementów z drzewa.

Podstawowe binarne drzewo poszukiwań pozwala na szybkie wyszukiwanie porównywalnych danych, np. liczb dzięki zorganizowaniu ich w formę drzewa binarnego, przez co czas wykonywania elementarnych operacji jest uzależniony od średniej głębokości h takiego drzewa i wynosi O(h). Jednak drzewo takie nie posiada żadnych mechanizmów, które dążą do jego zrównoważenia, przez co nietrudno jest uzyskać słabo rozgałęzioną strukturę o dużej głębokości. Czas wykonywania operacji będzie wtedy niewiele lepszy, niż dla zwykłych list.

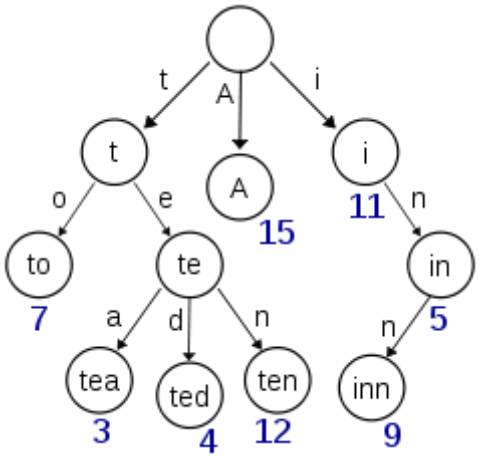
Drzewo czerwono-czarne jest rozszerzeniem podstawowej struktury o algorytm równoważenia wykonywany po każdej operacji INSERT oraz DELETE. W przypadku tej struktury elementy-liście nie przechowują żadnych informacji, dlatego często w ich miejsce wprowadza się dla zaoszczędzenia pamięci i uproszczenia kodu pojedynczego wartownika.

W drzewie czerwono-czarnym z każdym węzłem powiązany jest dodatkowy atrybut, kolor, który może być czerwony lub czarny. Oprócz podstawowych własności drzew poszukiwań binarnych, wprowadzone zostały kolejne wymagania, które trzeba spełniać:

- Każdy węzeł jest czerwony lub czarny.
- Korzeń jest czarny.
- Każdy liść jest czarny (Można traktować nil jako liść).
- Jeśli węzeł jest czerwony, to jego synowie muszą być czarni.



- Każda ścieżka z ustalonego węzła do liścia liczy tyle samo czarnych węzłów.

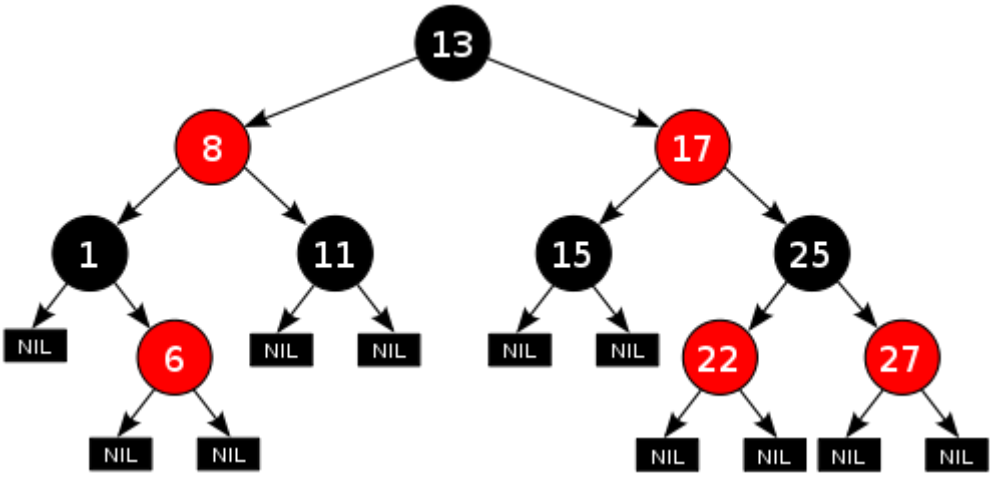


Drzewo trie (wym. tri od ang. retrieval - odczyt; lub traj by odróżnić od tree) — drzewo poszukiwań przechowujące w węzłach fragmenty kluczy ("zwykłe" drzewa poszukiwań - np. BST, AVL - przechowują w węzłach całe klucze). To pozwala przyspieszyć wyszukiwanie, gdy koszt porównania całego klucza jest duży.

Przechowując fragmenty kluczy, drzewa trie są najlepiej przystosowane do kluczy reprezentowanych jako ciąg elementów skończonego alfabetu. Przez to są stosowane do sprawdzania poprawności pisowni i dzielenia wyrazów.

Działania jakie można zrealizować za pomocą drzew trie:

- sprawdzenie, czy słowo jest w drzewie (w czasie O(k), gdzie k długość słowa);
- znalezienie najdłuższego prefiksu słowa występującego w drzewie (w czasie O(m), gdzie m długość prefiksu);



- wyszukanie wszystkich słów o podanym prefiksie.
- W słowach mogą występować również lokalne znaki wieloznaczne, co nie komplikuje znacząco wyszukiwania.

Do reprezentacji drzew trie w pamięci RAM stosuje się drzewa łączone, gdzie każdy węzeł zawiera znak klucza, a liść zawiera odnośnik do danych, drzewa indeksowane, gdzie każda gałąź to tablica zawierająca cały alfabet z odnośnikami do danych i następnych gałęzi

Graf – podstawowy obiekt rozważań teorii grafów, struktura matematyczna służąca do przedstawiania i badania relacji między obiektami. W uproszczeniu graf to zbiór wierzchołków, które mogą być połączone krawędziami w taki sposób, że każda krawędź kończy się i zaczyna w którymś z wierzchołków

Funkcja skrótu, funkcja mieszająca lub funkcja haszująca – funkcja przyporządkowująca dowolnie dużej liczbie krótką, zawsze posiadającą stały rozmiar, niespecyficzną, quasi-losową wartość, tzw. skrót nieodwracalny.

W informatyce funkcje skrótu pozwalają na ustalenie krótkich i łatwych do weryfikacji sygnatur dla dowolnie dużych zbiorów danych. Sygnatury mogą chronić przed przypadkowymi lub celowo wprowadzonymi modyfikacjami danych (sumy kontrolne), a także mają zastosowania przy optymalizacji dostępu do struktur danych w programach komputerowych (tablice haszujące).

W informatyce tablica mieszająca lub tablica z haszowaniem (ang. hash table, niekiedy błędnie tłumaczone jako "tablica haszująca") to struktura danych, która jest jednym ze sposobów realizacji tablicy asocjacyjnej, tj. abstrakcyjnego typu danych służącego do przechowywania informacji, w taki sposób aby możliwy był do nich szybki dostęp. Tablica mieszająca umożliwia również szybkie porównywanie danych, np. fragmentów tekstów, plików.

Odwołania do przechowywanych obiektów dokonywane są na podstawie klucza, który dany obiekt (informację) identyfikuje. Większość języków programowania posiada implementację tablicy mieszającej w ramach standardowej biblioteki. Ponadto większość języków interpretowanych, takich jak PHP, Ruby, czy Smalltalk posiada specjalną składnię do tworzenia tego typu struktur.

Tablica asocjacyjna (tablica skojarzeniowa, mapa, słownik, ang. associative array, map, dictionary) – nazwa dla powszechnie stosowanego w informatyce abstrakcyjnego typu danych, który przechowuje pary (unikatowy klucz, wartość) i umożliwia dostęp do wartości poprzez podanie klucza.

Formalnie typ tablicy asocjacyjnej odpowiada zbiorowi skończonych funkcji częściowych z typu klucza tablicy w typ wartości tablicy. Wiele złożonych danych jest naturalnie reprezentowanych przez tego typu tablice – np. drzewa plików, nagłówki poczty, nawet wszystkie atrybuty obiektu czy przestrzeń nazw zmiennych.

Tablice asocjacyjne realizowane są jako drzewa poszukiwań (BST, AVL, trie itp.) lub tablice mieszające. Typ danych klucza może być praktycznie dowolny. Najczęściej są to łańcuchy znaków (napisy), ale także liczby (całkowite, zmiennoprzecinkowe, zespolone), krotki (struktura danych będąca odzwierciedleniem matematycznej n-ki, tj. uporządkowanego ciągu wartości. Krotki przechowują stałe wartości o różnych typach danych - nie można zmodyfikować żadnego elementu, odczyt natomiast wymaga podania indeksu liczbowego żądanego elementu.) itp.

Zbiór jest najbardziej podstawowym modelem danych w matematyce. Wszystkie pojęcia matematyczne, od drzew po liczby rzeczywiste można wyrazić za pomocą specjalnego rodzaju zbioru. Jest wiec naturalne że jest on również podstawowym modelem danych w informatyce.

Sposoby reprezentacji znaków i łańcuchów znakowych

ASCII

We współczesnych językach programowania znaki są podstawowym typem danych. W pamięci komputera znak jest przechowywany w postaci liczby, którą nazywamy kodem znaku (ang. character code). Każdy znak posiada swój własny kod. Aby różne urządzenia systemu komputerowego mogły w ten sam sposób interpretować kody znaków, opracowano kilka standardów kodowania liter. Bardzo rozpowszechniony jest standard ASCII:

ASCII – American Standard Code for Information Interchange – Amerykański Standardowy Kod do Wymiany Informacji.

6.Składnia i translacja języków programowania

Gramatyka formalna – sposób opisu języka formalnego, czyli podzbioru zbioru wszystkich słów skończonej długości nad danym alfabetem.

Aby zdefiniować gramatykę formalną trzeba określić zbiór symboli terminalnych, zbiór symboli nieterminalnych, symbol startowy, oraz zbiór reguł które określają sposób w jaki wyprowadzamy słowa.

Symbol nieterminalny to symbol, który można definiować. Symbole nieterminalne zwane są również zmiennymi syntaktycznymi, ponieważ umożliwiają tworzenie ciągów zawierających kombinacje symboli terminalnych i nieterminalnych.

Symbol terminalny to symbol elementarny tworzący wyrazy języka formalnego. Symbole terminalne są znakami, które mogą pojawić się na wejściu lub wyjściu z reguł produkcji gramatyki formalnej. Symbol terminalny nie może być podzielony na „mniejsze" jednostki, lub ściślej: symbole terminalne nie mogą być zmieniane za pomocą reguł gramatyki formalnej, w odróżnieniu od symboli nieterminalnych.

Gramatyka formalna posiada wyróżniony symbol nieterminalny, zwany symbolem startowym, od którego, poprzez stosowanie reguł produkcji, zaczyna się wyprowadzanie wszystkich wyrazów języka formalnego. Tworzenie wyrazu języka formalnego kończy się wówczas gdy zawiera on już tylko symbole terminalne.

Gramatyka bezkontekstowa i jej związek z automatami skończonymi ????

Języki rozpoznawane przez automaty skończone nazywane są językami regularnymi. Ponieważ liczba stanów automatu jest skończona, nie może on rozpoznawać języków, które wymagają nieograniczonej liczby stanów.

Przykładem takiego języka jest zbiór słów z liter 'a' i 'b', które zawierają taką samą liczbę liter 'a' i 'b'. Aby pokazać że nie jest on regularny, załóżmy, że istnieje maszyna M która go rozpoznaje. M ma jakąś skończoną liczbę stanów n. Rozważmy teraz ciąg x zbudowany kolejno ułożonych (n+1) liter 'a' i następnie (n+1) liter 'b'. W trakcie czytania pierwszych (n+1) liter, maszyna M musi znaleźć się w którymś ze swoich stanów przynajmniej dwa razy (z powodu zasady szufladkowej). Oznacza to, że jeśli usuniemy fragment wejściowego ciągu, który maszyna przeczytała pomiędzy tymi dwoma momentami, to końcowy wynik będzie identyczny. Ale to oznacza, że maszyna zaakceptuje słowo w którym jest mniej 'a' niż 'b', co przeczy założeniu o istnieniu takiej maszyny.

Ogólnie do rozpoznawania jakie języki nie są regularne można wykorzystać lemat o pompowaniu dla języków regularnych(twierdzenie służące do dowodzenia, że dany język nie jest językiem regularnym).

Intuicyjnie wynik ten może wydawać się dziwny, ponieważ łatwo napisać program, który rozpoznaje czy w danym ciągu jest tyle samo 'a' co 'b' – a przecież wcześniej napisaliśmy, że komputery też są automatami skończonymi. Wynik ten jednak jest prawdziwy, ale wykorzystuje ważne ograniczenie komputerów: pojemność ich pamięci. Teoretycznie, czytając wystarczająco długi ciąg liter, komputer w końcu przepełni całą swoją pamięć licznikiem ile liter do tej pory przeczytał. W tym momencie będzie musiał znaleźć się ponownie w już

raz odwiedzonym stanie. Ten przykład pokazuje jak wyniki dotyczące języków regularnych odnoszą się do klasycznych komputerów.

Gramatyka bezkontekstowa to skończony zbiór zmiennych (zwanych też symbolami niekońcowymi, symbolami pomocniczymi lub kategoriami syntaktycznymi), z których każda reprezentuje pewien język. Języki reprezentowane przez zmienne opisywane za pomocą wzajemnej rekursji, z zastosowaniem pewnych symboli pierwotnych, zwanych symbolami końcowymi. Reguły wiążące ze sobą zmienne zwane są produkcjami.

Pierwotną motywacją wprowadzenia pojęcia gramatyk bezkontekstowych był opis języków naturalnych.

Notacja EBNF

Rozszerzona notacja Backusa-Naura (ang. Extended Backus-Naur Form) jest sposobem wyrażenia gramatyki bezkontekstowej, czyli opisem języków formalnych. Jest rozszerzeniem notacji BNF.Początkowo rozwijana przez Niklausa Wirtha, obecnie jest w użyciu wiele jej wariantów. Jest rozszerzeniem notacji BNF.

notacja BNF została rozszerzona o następujące symbole:

- nawiasy [a] oznaczające co najwyżej jednokrotne wystąpienie napisu a;
- nawiasy {a} oznaczające, że napis a może wystąpić dowolnie wiele razy po sobie;
- a+ – plus po symbolu a (lub po nawiasach [], {}) oznacza, że symbol lub napis może wystąpić dowolną niepustą liczbę razy;
- a* – gwiazdka po symbolu a (lub po nawiasach [], {}) oznacza, że symbol lub napis można powtórzyć dowolną liczbę razy (w szczególności – zero).
- Stosuje się też liczbowe oznaczenia dopuszczalnych krotności występowania symboli, pisane w postaci dolnych i górnych indeksów na prawo od nawiasu.

Notacja BNF, równania normalne Backusa, BNF (angielskie Backus-Naur Form), system zapisywania produkcji gramatyki zaproponowany przez J. Backusa i P. Naura przy okazji prac prowadzonych nad językiem Algol 60. Symbole metajęzykowe (alfabetu pomocniczego), służące do nazywania jednostek składniowych definiowanego języka umieszcza się w nawiasach kątowych.

Symbol: =, czytany “jest równe z definicji”, łączy strony produkcji. Pionowa kreska (|) oznacza w produkcjach spójnik logiczny “albo” i pozwala na zmniejszenie ich liczby. Symbole alfabetu końcowego pisze się w BNF bez żadnych zaznaczeń.

W diagramach składniowych stosowane są składniki specjalne opisujące składnię instrukcji i komend. Jest jednym ze sposobów reprezentowania gramatyki bezkontekstowej. Stanowią one alternatywę dla graficznej Notacji BNF lub EBNF jako metajęzyk

Reprezentacja gramatyki składa się z zestawu schematów składniowych. Każdy schemat określa nie-terminal. Istnieje główny schemat, który określa język w następujący sposób: przynależność do języka, słowo musi opisać ścieżkę do głównego wykresu.

Każdy schemat ma punkt wejścia i punkt końcowy. Schemat przedstawia możliwe ścieżki między tymi dwoma punktami, przechodząc przez innych nieterminali i terminali. Zaciski są reprezentowane przez pola okrągłe podczas nieterminale są reprezentowane przez pola kwadratowe.

Gramatyka regularna to gramatyka formalna za pomocą, której można opisać język regularny.

Istnieją dwa rodzaje gramatyk regularnych: gramatyka lewostronna, gramatyka prawostronna. Istnieje ścisły związek gramatyki lewostronnej oraz deterministycznego automatu skończonego, taki że gramatyka generuje dokładnie taki język jaki akceptuje automat. Stąd gramatyki lewostronne generują dokładnie wszystkie języki regularne. Gramatyka regularna to albo prawo albo lewostronna.

Ważne ograniczenia postaci reguł:

Po lewej stronie występuje zawsze dokładnie jeden symbol nieterminalny (tak samo jak w gramatykach bezkontekstowych)

Po prawej stronie występuje nie więcej niż jeden symbol nieterminalny i dowolny łańcuch symboli terminalnych. W gramatykach prawostronnie regularnych symbol terminalny występuje przed nieterminalnym (jeśli ten się tam znajduje), a w lewostronnie regularnych na odwrót.

Za pomocą łączenia prawo i lewostronnych gramatyk można tworzyć języki, które niekoniecznie będą regularne, z definicji jednak, tego typu gramatyki nie będą już regularne.

Wyrażenia regularne to w informatyce teoretycznej ciągi znaków pozwalające opisywać języki regularne. W praktyce znalazły bardzo szerokie zastosowanie, pozwalają bowiem w łatwy sposób opisywać wzorce tekstu, natomiast istniejące algorytmy w efektywny sposób określają, czy podany ciąg znaków pasuje do wzorca lub wyszukują w tekście wystąpienia wzorca. Wyrażenia regularne w praktycznych zastosowaniach są zapisywane za pomocą bogatszej i łatwiejszej w użyciu składni niż ta stosowana w rozważaniach teoretycznych. Co więcej, opisane niżej powszechnie wykorzystywane wsteczne referencje (czyli użycie wcześniej dopasowanego fragmentu tekstu jako części wzorca), powodują, że wyrażenie regularne je zawierające może nie definiować języka regularnego.

Wyrażenia regularne stanowią integralną część narzędzi systemowych takich jak sed, grep, wielu edytorów tekstu, języków programowania przetwarzających tekst AWK i Perl, a także są dostępne jako odrębne biblioteki dla wszystkich języków używanych obecnie.

Dwie najpopularniejsze składnie wyrażeń regularnych to składnia uniksowa i składnia perlowa. Składnia perlowa jest znacznie bardziej rozbudowana. Jest ona używana nie tylko w języku Perl, ale także w innych językach programowania: Ruby, bibliotece PCRE do C.

Podstawowe elementy wyrażeń regularnych:

- Każdy znak, oprócz znaków specjalnych, określa sam siebie, np. a określa łańcuch złożony ze znaku a.
- Kolejne symbole oznaczają, że w łańcuchu muszą wystąpić dokładnie te symbole w dokładnie takiej samej kolejności, np. ab oznacza że łańcuch musi składać się z litery a poprzedzającej literę b.
- Kropka . oznacza dowolny znak z wyjątkiem znaku nowego wiersza (zależnie od ustawień i rodzaju wyrażeń).
- Znaki specjalne poprzedzone odwrotnym ukośnikiem \ powodują, że poprzedzanym znakom nie są nadawane żadne dodatkowe znaczenia i oznaczają same siebie, np. \. oznacza znak kropki (a nie dowolny znak).

- Zestaw znaków między nawiasami kwadratowymi oznacza jeden dowolny znak objęty nawiasami kwadratowymi, np. [abc] oznacza a, b lub c. Można używać także przedziałów: [a-c]. Między nawiasami kwadratowymi:
- Daszek ^ na początku zestawu oznacza wszystkie znaki oprócz tych z zestawu.
- Aby uniknąć niejasności, znaki - (łącznik) i] (zamknięcie nawiasu kwadratowego) zapisywane są na skraju zestawu lub w niektórych systemach po znaku odwrotnego ukośnika, daszek zaś wszędzie z wyjątkiem początku łańcucha. Zasady te mogą być różne w zależności od konkretnej implementacji.
- Większość znaków specjalnych w tym miejscu traci swoje znaczenie.
- Pomiędzy nawiasami okrągłymi (i) grupuje się symbole do ich późniejszego wykorzystania.
- Gwiazdka * po symbolu (nawiasie, pojedynczym znaku) nazywana jest domknięciem Kleene'a i oznacza zero lub więcej wystąpień poprzedzającego wyrażenia.
- Znak zapytania ? po symbolu oznacza najwyżej jedno (być może zero) wystąpienie poprzedzającego wyrażenia.
- Plus + po symbolu oznacza co najmniej jedno wystąpienie poprzedzającego go wyrażenia.
- Daszek ^ oznacza początek wiersza, dolar $ oznacza koniec wiersza.
- Pionowa kreska (ang. pipeline) | to operator OR np. jeśli napiszemy a|b|c oznacza to, że w danym wyrażeniu może wystąpić a lub b lub c.
- Znaki \<, \> oznaczające początek i koniec wyrazu (w niektórych implementacjach występuje pełniący podobną funkcję metaznak \b). Np. \<al znajdzie wszystkie wyrazy zaczynające się na al. et\> znajdzie wyrazy które kończą się na et.

Rozszerzenia Perla to między innymi:

- Negacja zestawu (wszystko, co nie należy do zestawu).
- cyfry są zastępowane znakami \d (dowolna cyfra) i \D (wszystko co nie jest cyfrą)
- znaki "białe" \s i \S (przeciwieństwo)
- Rozszerzony zapis przedziałów, wprowadzenie klas znaków np.:
- [:digit:] oznacza dowolną cyfrę
- [:alpha:] literę
- [:alnum:] literę lub cyfrę
- Możliwość precyzyjnego określenia liczby wystąpień danego wyrażenia
- wyrażenie{N} oznacza dokładnie N wystąpień
- wyrażenie{N,} co najmniej N wystąpień wyrażenia
- wyrażenie{,M} co najwyżej M wystąpień wyrażenia
- wyrażenie{N,M} od N do M wystąpień wyrażenia
- Referencje wsteczne, czyli możliwość odwoływania się do odnalezionych podciągów zgrupowanych poprzez nawiasy. Np. w wyrażeniu "(.*)\1" referencją wsteczną jest "\1" i oznacza powtórzenie ciągu znalezionego w ramach pierwszej grupy nawiasów. To rozszerzenie pozwala definiować języki, które nie są regularne.

Translator – program komputerowy (lub urządzenie), dokonujący tłumaczenia (translacji) programu napisanego w określonym języku programowania, z postaci źródłowej do postaci wynikowej możliwej do wykonania przez maszynę (potocznie: „zrozumiałą dla maszyny"). Czasami zamiast kod wynikowy używa się równoważnego określenia kod obiektowy.

Translatory można podzielić na dwie grupy:

- kompilatory tłumaczące programy zapisane w językach wysokiego poziomu,
- assemblery tłumaczące programy zapisane w językach symbolicznych.

Cechą charakterystyczną translatorów jest to, że przed uruchomieniem programu musi być wykonany proces tłumaczenia jego kodu źródłowego. Innym możliwym sposobem jest interpretacja programu źródłowego „w locie" przez interpreter albo zastosowanie metody JIT – kompilacji na bieżąco.

Translator to również program, aplikacja internetowa lub urządzenie elektroniczne, tłumaczące teksty w językach naturalnych. Translatory wykorzystują różne algorytmy tłumaczenia automatycznego, jednak na obecnym etapie jakość tak wykonanych tłumaczeń znacząco ustępuje tłumaczeniom wykonywanym przez człowieka. Translatory mogą być jednak użyteczne, by szybko zorientować się w ogólnej treści tekstu, np. strony internetowej, w zupełnie nieznanym języku, w podróży lub ewentualnie do wykonania pierwszej wersji tłumaczenia, które musi potem zostać dokładnie zredagowane przez człowieka. Translatorów nie należy mylić z oprogramowaniem typu CAT.

Lekser (ang. lexer lub scanner), nazywany też analizatorem leksykalnym, to program komputerowy który dokonuje analizy leksykalnej danych wejściowych, zwykle jako pierwsza część jakiegoś większego procesu, np. kompilacji.

Programy generujące leksery to m.in.:

- lex,
- flex,
- ocamllex.

Analizator składniowy lub parser – program dokonujący analizy składniowej danych wejściowych w celu określenia ich struktury gramatycznej w związku z określoną gramatyką formalną. Nazwa analizator składniowy podkreśla analogię z analizą składniową stosowaną w gramatyce i językoznawstwie. Analizator składniowy umożliwia przetworzenie tekstu czytelnego dla człowieka w strukturę danych przydatną dla oprogramowania komputera.

Parsery wielu języków programowania bazują na gramatykach typu LALR. Najczęściej stosowane są własne standardy notacji oparte na notacji BNF. Najczęstszym zastosowaniem parserów jest analiza języków programowania. Mają one, zwykle, prostą gramatykę z nielicznymi wyjątkami. Jednakże gramatyki bezkontekstowe mają ograniczone zastosowanie, gdyż mogą one opisać jedynie ograniczony zestaw języków.

Ręczne pisanie parsera, szczególnie dla dużych języków, jest zajęciem dosyć żmudnym, dlatego powstały generatory parserów. Jednym z popularniejszych generatorów jest yacc, pozwalający na generowanie parserów w języku C. Jego odpowiednikiem rozprowadzanym na zasadach wolnego oprogramowania jest stworzony przez Free Software Foundation bison. Wśród przykładów generatorów parserów dla innych języków jest ocamlyacc dla języka OCaml oraz JavaCC i SableCC dla Javy.

Preprocesor – program komputerowy, którego zadaniem jest przetworzenie kodu źródłowego, w sposób określony przez programistę za pomocą dyrektyw preprocesora, na kod wyjściowy – tak przetworzony kod źródłowy poddawany jest następnie analizie składniowej, kompilacji, a w końcu konsolidacji. Preprocesor jest najczęściej zintegrowany z kompilatorem języka programowania.

Najbardziej znane języki, które wyposażone są w preprocesor „wbudowany w język", to C i C++. Dyrektywy preprocesora mogą występować w ogólności w dowolnym miejscu programu, a rozróżnienie ich od tekstu kodu źródłowego w językach C i C++ dokonywane jest poprzez poprzedzenie dyrektywy znakiem hash '#'.

Do najważniejszych dyrektyw należą:

- #include … - dyrektywa włączająca tekst innego pliku źródłowego w miejscu jej wystąpienia w pliku podlegającym aktualnie przetwarzaniu, przy czym możliwe jest zagłębione występowanie dyrektywy include,
- #define … - definiuje stałe i makroinstrukcje (pseudofunkcje)
- #undef … - usuwa definicje stałej lub makra
- #if … - dyrektywy kompilacji warunkowej

- #elif … - działa podobnie jak *else if* w języku C
- #endif … - oznacza koniec bloku kompilacji warunkowej
- #ifdef … - znaczy to samo co #if defined(…)
- #ifndef … - znaczy to samo co #if !defined(…)
- i inne.

Interpreter – program komputerowy, który analizuje kod źródłowy programu, a przeanalizowane fragmenty wykonuje.

Realizowane jest to w inny sposób niż w procesie kompilacji, podczas którego nie wykonuje się wejściowego programu (kodu źródłowego), lecz tłumaczy go do wykonywalnego kodu maszynowego lub kodu pośredniego, który jest następnie zapisywany do pliku w celu późniejszego wykonania.

Wykonanie programu za pomocą interpretera jest wolniejsze, a do tego zajmuje więcej zasobów systemowych niż wykonanie kodu skompilowanego, lecz może zająć relatywnie mniej czasu niż kompilacja i uruchomienie. Jest to zwłaszcza ważne przy tworzeniu i testowaniu kodu, kiedy cykl edycja-interpretacja-debugowanie może często być znacznie krótszy niż cykl edycja-kompilacja-uruchomienie-debugowanie.

Interpretacja kodu programu jest wolniejsza od uruchamiania skompilowanego kodu, ponieważ interpreter musi wpierw przeanalizować każde wyrażenie i dopiero na tej podstawie wykonać odpowiednie akcje, a kod skompilowany wykonuje wyłącznie akcje. W implementacjach będących w pełni interpreterami wielokrotne wykonanie tego samego fragmentu kodu wymaga wielokrotnej interpretacji tego samego tekstu. Ta analiza nazywana jest "kosztem interpretacji". Dostęp do zmiennych jest także wolniejszy w przypadku interpretera, gdyż odwzorowanie identyfikatorów na miejsca w pamięci operacyjnej musi zostać dokonane podczas uruchomienia lub działania, a nie podczas kompilacji, dlatego niektóre interpretery tworzą dodatkowe dane (np. adresy zmiennych) przyspieszające wykonanie programu.

Język maszynowy, kod maszynowy – zestaw rozkazów procesora, w którym zapis programu wyrażony jest w postaci liczb binarnych stanowiących rozkazy oraz ich argumenty.Był to jedyny język programowania komputerów zerowej generacji z wyjątkiem komputera Z4, a powszechnym w początkowym okresie rozwoju komputerów pierwszej generacji.Kod maszynowy może być generowany w procesie kompilacji (w przypadku języków wysokiego poziomu) lub asemblacji (w przypadku języków niskiego poziomu). W trakcie procesu tworzenia kodu maszynowego tworzony jest często kod pośredni zapisywany w pliku obiektowym. Następnie kod pośredni pobrany z pliku obiektowego poddawany jest konsolidacji (linkowaniu) w celu utworzenia ostatecznego kodu maszynowego.

Język maszynowy jest nieprzenośny, ponieważ każda architektura procesora ma swój własny język maszynowy.

Języki asemblera (zwyczajowo asemblery) to rodzina języków programowania niskiego poziomu, których jedno polecenie odpowiada zasadniczo jednemu rozkazowi procesora. Języki te powstały na bazie języków maszynowych danego procesora poprzez zastąpienie kodów operacji ich mnemonikami. Dzięki stosowaniu kilkuliterowych skrótów poleceń zrozumiałych dla człowieka pozwala to z jednej strony na tworzenie oprogramowania, z drugiej strony bezpośrednia odpowiedniość mnemoników oraz kodu maszynowego umożliwia zachowanie wysokiego stopnia kontroli programisty nad działaniem procesora. Składnia języka asemblera zależy od architektury procesora, ale i używanego asemblera, jednak zwykle autorzy asemblerów dla danego procesora trzymają się oznaczeń danych przez producenta.

Pierwotnie był to podstawowy język programowania procesorów. W wyniku poszukiwania efektywniejszych metod programowania i pojawianiem się kolejnych języków interpretowanych i kompilowanych języki asemblerów straciły na znaczeniu. Z tego powodu współcześnie nie korzysta się z nich do pisania całych programów na komputery osobiste. Jednak istnieją zastosowania, np. w przypadku programowania

mikrokontrolerów, systemów wbudowanych, sterowników sprzętu, gdzie nadal znajdują one swoje miejsce. Korzysta się z nich także do pisania kluczowych fragmentów kodu wymagających najwyższej wydajności, wyjątkowo małych rozmiarów kodu wynikowego lub również niewielkich fragmentów systemów operacyjnych.

Asembler (z ang. assembler) – termin informatyczny związany z programowaniem i tworzeniem kodu maszynowego dla procesorów. W języku polskim oznacza on program tworzący kod maszynowy na podstawie kodu źródłowego (tzw. asemblacja) wykonanego w niskopoziomowym języku programowania bazującym na podstawowych operacjach procesora zwanym językiem asemblera, popularnie nazywanym również asemblerem. W tym artykule język programowania nazywany będzie językiem asemblera, a program tłumaczący – asemblerem.

Asembler (ang. assemble – składać) to program dokonujący tłumaczenia języka asemblera na język maszynowy, czyli tzw. asemblacji. Jest to odpowiednik kompilatora dla języków wyższych poziomów. Program tworzony w innych językach programowania niż asembler jest zwykle kompilowany do języka maszynowego (wyniku pracy asemblera), a następnie zamieniany na kod binarny przez program asemblera.

Powtarzające się często schematy programistyczne oraz wstawiane fragmenty kodu doprowadziły do powstania tzw. makroasemblerów, które rozszerzają asemblery o obsługę makr przed właściwą asemblacją, co zbliża je nieco do pierwszych wersji języka C.

Kompilator – program służący do automatycznego tłumaczenia kodu napisanego w jednym języku (języku źródłowym) na równoważny kod w innym języku (języku wynikowym) . Proces ten nazywany jest kompilacją. W informatyce kompilatorem nazywa się najczęściej program do tłumaczenia kodu źródłowego w języku programowania na język maszynowy. Niektóre z nich tłumaczą najpierw do języka asemblera, a ten na język maszynowy jest tłumaczony przez asembler.

Różnica pomiędzy kompilatorem a asemblerem polega na tym, iż każde polecenie języka programowania może zostać rozbite na wiele podpoleceń języka maszynowego (przy czym nowoczesne asemblery również posiadają składnię umożliwiającą zapis wielu poleceń maszynowych jako jednego polecenia kodu źródłowego oraz opcje optymalizacji kodu). Kompilatory mogą posiadać możliwość automatycznej alokacji pamięci dla zmiennych, implementowania struktur kontrolnych lub procedur wejścia-wyjścia.

Stosowanie kompilatorów ułatwia programowanie (programista nie musi znać języka maszynowego) i pozwala na większą przenośność kodu pomiędzy platformami.

Kompilacja to proces automatycznego tłumaczenia kodu źródłowego na kod wynikowy przez kompilator. Kompilacja jest przeważnie głównym etapem ogólniejszego procesu translacji, a tworzony w jej trakcie kod wynikowy jest przekazywany do innych programów, np. do konsolidatora (linkera). Możliwe jest również tłumaczenie do postaci zrozumiałej dla człowieka.

Określenie kompilacja używana jest zazwyczaj w kontekście tłumaczenia z języka wysokiego poziomu na język niskiego poziomu, natomiast tłumaczenie w odwrotnym kierunku to dekompilacja.

Języki interpretowane działają z interpreterem, tzn są wczytywane z pliku, interpretowane przez interpereter i dopiero wykonywane. Bez interpretera nic nie zrobimy.

Języki kompilowane - kod jest kompilowany i odpalany samodzielnie.

Różnice między kompilacją a interpretacją

1. Kompilator to program, który z języka rozumianego przez człowieka tworzy program rozumiany przez komputer.

2. Interpreter to program, który wykonuje program bezpośrednio z kodu rozumianego przez człowieka. Jest tak jakby tłumaczem kodu źródłowego na komendy procesora. Różnica jest taka, że w pierwszym przypadku procesor wykonuje skompilowany program, a w drugim wykonuje kod interpretera, który wykonuje interpretowany program.

Kompilacja just-in-time

JIT (ang. just-in-time compilation) to metoda wykonywania programów polegająca na kompilacji do kodu maszynowego przed wykonaniem danego fragmentu kodu.

Cała procedura wygląda następująco:

- kod źródłowy jest kompilowany do kodu pośredniego (bajtowego),
- maszyna wirtualna przeprowadza kompilację kodu pośredniego do kodu maszynowego.

Kompilacja może się odbywać w momencie pierwszego dostępu do kodu znajdującego się w pliku lub pierwszego wywołania funkcji (stąd nazwa just-in-time).

JIT jest bardzo obiecującą techniką. Zaawansowane systemy JIT potrafią wykonywać programy szybciej niż interpretery, unikając jednocześnie komplikacji związanych z pełnym cyklem kompilacji, linkowania itd.[potrzebne źródło]

Główną konkurencją dla techniki JIT jest popularna na systemach uniksowych metoda rozpowszechniania źródeł, które dopiero na docelowej maszynie zostaną skompilowane. W metodzie rozpowszechniania źródeł, kompilacja pomimo tego, że trwa stosunkowo długo i jest skomplikowanym, a niekiedy zawodnym procesem, jest wykonywana tylko raz. W efekcie tego, że program jest skompilowany do poziomu kodu maszynowego uzyskuje się z reguły szybsze działanie wyjściowego programu.[potrzebne źródło]

JIT jest używany głównie przez maszyny wirtualne Javy oraz środowisko uruchomieniowe .NET CLR.

7.Wyrażenia i ich ewaluacja

1. Elementy języków programowania.

Na język programowania składają się:

I. Składnia

Składnia opisuje strukturę zdań języka:

- rodzaje dostępnych symboli.
- zasady, według których symbole mogą być łączone w większe struktury.
- Składnia najczęściej opisywana jest w notacji BNF (ew.EBNF) lub notacji Wirth-a.
- Notacja BNF została wymyślona przez Johna Backusa w latach 50. w czasie prac nad językiem Fortran,
- a następnie zmodyfikowana przez Petera Naura i użyta do zdefiniowania składni języka Algol. Notacja

BNF dopuszcza następujące znaki (symbole metajęzykowe):

- ::= - przypisanie

- | - alternatywa

Notacja EBNF dopuszcza dodatkowo symbole:

- {}- wybierz jeden z elementów występujących (dokładnie 1 raz)
- {}+ - element musi wystąpić co najmniej 1 raz (chociaż jeden raz)
- {}* - element może wystąpić 0 lub więcej razy (dowolnie wiele lub w ogóle)
- [] - element może wystąpić 0 lub 1 raz (co najwyżej 1 raz)

Np. Niech - zbiór symboli ASCII

Liczby całkowite w BNF:

- <zero>::=0
- <cyfra1_9>::=1|2|3|4|5|6|7|8|9
- <cyfra>::=<zero>|<cyfra1_9>
- <ciąg_cyfr>::=<cyfra>|<cyfra><ciąg_cyfr>
- <liczba_naturalna>::=<cyfra1_9>|<cyfra1_9><ciąg_cyfr>
- <liczba_całkowita>::=<liczba_naturalna>|-<liczba_naturalna>|<zero>
- Liczby całkowite w EBNF
- <zero>::=0
- <cyfra1_9>::={1|2|3|4|5|6|7|8|9}
- <cyfra>::={<zero>|<cyfra1_9>}
- <liczba_naturalna>::=<cyfra1_9>{<cyfra>}*
- <liczba_całkowita>::={[-]<liczba_naturalna>|<zero>}

II. Semantyka

Semantyka języka programowania definiuje precyzyjnie znaczenie poszczególnych symboli oraz ich

funkcję w programie. Semantykę najczęściej definiuje się słownie

III.Pragmatyka

• przydatność języka

• obszary zastosowań

• łatwość implementacji i użycia

„wszyscy wiedza, ze": systemy operacyje pisze sie w C stony w PHP i HTML *applety i programy na komórki w Javie…"*

Generacje języków

I. Języki pierwszej generacji - są to języki maszynowe, czyli języki procesorów. Kod maszynowy przeznaczony jest do

wykonania na konkretnym typie procesora. Instrukcje są w nich zapisywane w postaci liczb binarnych.

Przykładowy kod:

11101010000000000000111111111111111111000100111011000000000010100000001

II. Języki drugiej generacji - języki symboliczne, Asemblery. Języki niskiego poziomu, pod względem składni tożsame

z maszynowymi, z tą różnicą że zamiast liczb używa się tu łatwiejszych do zapamiętania mnemoników.(w językach assemblera jest to składający się z kilku liter kod-słowo, które oznacza konkretną czynność procesora. Przykładem mogą być: "add")

Przykładowy kod:

```
jmp ffff:0

mov ax, bx

add ax, 1
```

III.Języki trzeciej generacji - języki wysokiego poziomu (imperatywne). To języki w których program jest pojmowany jako ciąg wykonywanych po sobie instrukcji. W językach imperatywnych opisuje się sposób otrzymania rozwiązania a nie na właściwościach rozwiązania. Języki imperatywne -opisują algorytm otrzymania rozwiązania. Pierwszym językiem tego typu był ALGOL. Do tej grupy należą między innymi: FORTH, BASIC - języki niestrukturalne, Pascal, C, FORTRAN - języki strukturalne, C++, Java - języki zorientowane obiektowo.

Przykładowy kod:

```
if(!x)

{

printf("Nie wolno dzielić przez zero\n");

return 1;

}
```

IV. Języki czwartej generacji - języki bardzo wysokiego poziomu, deklaratywne. Korzystając z tych języków programista skupia się na problemie, a nie na sposobie jego rozwiązania. Syntaktyka wielu języków czwartej generacji przypomina składnię języka naturalnego. Są one często używane do dostępu do baz danych.

Przykładem języka z tej grupy jest: SQL.

Przykładowy kod:

```
SELECT Nazwa FROM "Faktura.db" WHERE Cena > 1000
```

V. Języki piątej generacji - języki sztucznej inteligencji, języki systemów ekspertowych. Języki najbardziej zbliżone do języka naturalnego. Przykładem języka piątej generacji jest PROLOG.”

Programowanie imperatywne: Najbardziej pierwotny sposób programowania, w którym program postrzegany jest jako ciąg poleceń dla komputera. Opisuje sposób otrzymania rozwiązania

Podstawowe elementy języka C

- Komentarze- Komentarz blokowy umieszcza się między sekwencją znaków "/*" a "*/", a komentarz liniowy rozpoczyna się sekwencją "//" a kończy znakiem końca linii

- Słowo kluczowe (ang. keyword) w języku programowania oznacza słowo (może należeć do jakiegoś języka naturalnego, jak język angielski, lub nie), stanowiące wydorębnioną jednostkę leksykalną, często w określonym kontekście mające szczególne znaczenie i oznaczające określony rozkaz, instrukcję lub deklarację w programie komputerowym. Lista słów kluczowych jest najczęściej ustalona dla danego standardu języka wraz z rozszerzeniami określonego producenta kompilatora.
  Lista słów kluczowych języka C na podstawie normy ISO/IEC 9899-1999 (C99). Istnieją zależne od implementacji rozszerzenia języka o inne słowa kluczowe jak np. asm.

auto	enum	restrict	unsigned
break	extern	return	void
case	float	short	volatile
char	for	signed	while
const	goto	sizeof	_Bool
continue	if	static	_Complex
default	inline	struct	_Imaginary
do	int	switch	
double	long	typedef	
else	register	union	

- Literał – w językach programowania, to jednostka leksykalna reprezentująca ustaloną wartość (liczbową, tekstową, itp.) wpisaną przez programistę bezpośrednio w danym miejscu w kod programu. Prawie wszystkie języki programowania mają odpowiednie formy zapisu dla podstawowych wartości takich jak liczby całkowite, liczby zmiennoprzecinkowe, łańcuchy znaków i zazwyczaj dla wartości typu logicznego i znakowego. Niektóre mają także formy zapisu dla elementów typu wyliczeniowego i złożonych wartości jak tablice, rekordy czy obiekty .

W języku C:

```c
char znak = 'c'; // 'c' jest literałem
char x;
x = getchar();
printf("%c",x); // literałem jest "%c", x jest tutaj zmienną i nie ma ustalonej wartości
```

- Identyfikator – podstawowa jednostka leksykalna określonego języka programowania, tworzona przez programistę zgodnie ze składnią danego języka programowania, służąca identyfikacji i odwoływaniu się do określonego elementu kodu źródłowego
  Identyfikatorem nazywamy ciąg liter i cyfr zaczynający się od litery. Rozróżnia się małe i wielkie litery alfabetu. Liczba znaków identyfikatora może być dowolna. Dla identyfikatorów wewnętrznych znaczenie ma co najmniej pierwszych 31 znaków, ale w niektórych implementacjach może być ich więcej. Identyfikatory o zewnętrznej łączności są bardziej ograniczone, np. w konkretnej implementacji ich znacząca długość może wynosić tylko sześć początkowych znaków bez rozróżniania małych i wielkich liter.

Identyfikatory stosuje się do nazwania i identyfikacji takich elementów programu jak:

- ✓ stałe
- ✓ zmienne
- ✓ typy
- ✓ etykiety
- ✓ pola struktur i klas
- ✓ parametry
- ✓ podprogramy (procedury, funkcje, metody, makra)
- ✓ klasy
- ✓ moduły, pakiety, biblioteki itp.
- ✓ programy
- ✓ i innych dostępnych w danym języku lub systemie programowania.

- Operator – w programowaniu konstrukcja językowa jedno-, bądź wieloargumentowa zwracająca wartość. Operatory w c :

Operatory arytmetyczne

C	Pascal	Opis
+	+	dodawanie
-	-	odejmowanie
/	/ lub div	dzielenie
*	*	mnożenie
%	mod	reszta z dzielenia dwóch liczb całkowitych

Operatory zwiększania i zmniejszania

C	Pascal	Opis
++	brak	zwiększenie o 1
--	brak	zmniejszenie o 1

Operator przypisania = przypisuje wartość wyrażenia z prawej strony do zmiennej umiejscowionej po lewej jego stronie. Operator ten to odpowiednik Pascalowego :=. Operator przypisania może wystąpić w instrukcji przypisania ale może także wystąpić wewnątrz wyrażeń, tak jak każdy inny operator, gdyż

w przeciwieństwie do instrukcji posiada tę właściwość, że zwraca określony rezultat.

Operatory porównania.

C	Pascal	Opis
>	>	większe niż
<	<	mniejsze niż
>=	>=	większe lub równe
<=	<=	mniejsze lub równe
==	=	równe
!=	<>	nie równe

Operatory logiczne

C	Pascal	Funkcja
!	not	negacja
&&	and	koniunkcja
||	or	alternatywa

Operatory porównania i logiczne zwracają zawsze wartość 1 (prawda) albo 0 (fałsz)

Operatory bitowe

C	Pascal	Operacja
~	not	negacja
&	and	iloczyn logiczny
|	or	suma logiczna
^	xor	różnica symetryczna
<<	shl	przesunięcie bitowe w lewo
>>	shr	przesunięcie bitowe w prawo

- Typy podstawowe
- Typy pochodne
- Instrukcje sterujące
- Funkcje

Zmienna (programistyczna)- konstrukcja programistyczna posiadająca trzy podstawowe atrybuty: symboliczną nazwę, miejsce przechowywania i wartość; pozwalająca w kodzie źródłowym odwoływać się przy pomocy nazwy do wartości lub miejsca przechowywania. Nazwa służy do identyfikowania zmiennej w związku z tym często nazywana jest identyfikatorem. Miejsce przechowywania przeważnie znajduje się w pamięci komputera i określane jest przez adres i długość danych. Wartość to zawartość miejsca przechowywania. Zmienna zazwyczaj posiada również czwarty atrybut: typ, określający rodzaj danych przechowywanych w zmiennej i co za tym idzie sposób reprezentacji wartości w miejscu przechowywania. W programie wartość zmiennej może być odczytywana lub zastępowana nową wartością, tak więc wartość zmiennej może zmieniać się w trakcie wykonywania programu, natomiast dwa pierwsze atrybuty (nazwa i miejsce przechowywania) nie zmieniają się w trakcie istnienia zmiennej

Typ zmiennej

W językach ze statycznym typowaniem zmienna ma określony typ danych jakie może przechowywać. Jest on wykorzystywany do określenia reprezentacji wartości w pamięci, kontrolowania poprawności operacji wykonywanych na zmiennej (kontrola typów) oraz konwersji danych jednego typu na inny.

W językach z typowaniem dynamicznym typ nie jest atrybutem zmiennej lecz wartości w niej przechowywanej. Zmienna może wtedy w różnych momentach pracy programu przechowywać dane innego typu.

Pierwsza grupa to zmienne typu całkowitego. Jak sama nazwa mówi, przechowują tylko liczby całkowite. Różnią się one rozmiarem, czyli zakresem przechowywanych liczb. Im większy rozmiar, tym większe liczby mogą być przechowane.

Typ rzeczywisty - przechowuje liczby zmiennoprzecinkowe. Gdy mamy zamiar w naszym programie wykorzystać ułamki, ten typ będzie najbardziej odpowiedni.

Typ znakowy - przechowuje znaki, które są kodowane kodem ASCII. Tzn. znak w pamięci nie może być przechowany jako znak, tylko jako pewna liczba. Dlatego każdy znak ma swój odpowiednik liczbowy z zakresu [0, 255], który nazywamy kodem ASCII.

Typ logiczny - przechowuje jedną z dwóch wartości - true (prawda) albo false (fałsz). Wartość logiczna true jest równa 1, natomiast false ma wartość 0. Dla zmiennych tego typu możemy realizować przypisanie na dwa sposoby, podając wartość true lub fałsz, albo 1 lub 0.

Zasięg zmiennej określa gdzie w treści programu zmienna może być wykorzystana, natomiast czas życia zmiennej to okresy w trakcie wykonywania programu gdy zmienna ma przydzieloną pamięć i posiada (niekoniecznie określoną) wartość. Precyzyjnie zasięg odnosi się do nazwy zmiennej i przeważnie jest aspektem leksykalnym, natomiast czas życia do zmiennej samej w sobie i związany jest z wykonywaniem programu. Ze względu na zasięg można wyróżnić podstawowe typy zmiennych:

- globalne - obejmujące zasięgiem cały program,
- lokalne - o zasięgu obejmującym pewien blok, podprogram. W językach obsługujących rekurencję zazwyczaj są to zmienne automatyczne, natomiast w językach bez rekurencji mogą być statyczne.

Zmienne zadeklarowane w module mogą być zmiennymi prywatnymi modułu - dostępne wyłącznie z jego wnętrza lub zmiennymi publicznymi (eksportowanymi) dostępnymi tam gdzie moduł jest wykorzystywany. Podobnie ze zmiennymi w klasie mogą być dostępne:

- tylko dla danej klasy (zmienna prywatna),

- dla danej klasy i jej potomków (zmienna chroniona),
- w całym programie (zmienna publiczna),
- inne ograniczenia w zależności od języka (np. friend czy internal w .net)

Zmienne mogą zmieniać swój pierwotny zasięg np. poprzez importowanie/włącznie do zasięgu globalnego modułów, pakietów czy przestrzeni nazw.

Ze względu na czas życia i sposób alokacji zmienna może być:

- Statyczna - gdy pamięć dla niej rezerwowana jest w momencie ładowania programu; takimi zmiennymi są zmienne globalne, zmienne klasy (współdzielone przez wszystkie obiekty klasy, a nawet dostępne spoza klasy), statyczne zmienne lokalne funkcji (współdzielone pomiędzy poszczególnymi wywołaniami funkcji i zachowujące wartość po zakończeniu).
- Automatyczna, dynamiczna - gdy pamięć przydzielana jest w trakcie działania programu ale automatycznie. Są to przeważnie zmienne lokalne podprogramów i ich parametry formalne. Przeważnie alokowane na stosie w rekordzie aktywacji, znikają po zakończeniu podprogramu.
- Dynamiczna - alokowanie ręcznie w trakcie wykonywania programu przy pomocy specjalnych konstrukcji lub funkcji (malloc, new). W zależności od języka zwalnianie pamięci może być ręczne lub automatyczne. Zazwyczaj nie posiada własnej nazwy, lecz odwoływać się do niej trzeba przy pomocy wskaźnika, referencji lub zmiennej o semantyce referencyjnej.

Zmienna lokalna w programowaniu, to zmienna zdefiniowana i dostępna wyłącznie w określonym bloku programu, tworzona w momencie wejścia do tego bloku oraz usuwana z pamięci w momencie wyjścia z danego bloku. Tym samym zasięg zmiennej lokalnej oraz czas jej życia pokrywaną się i obejmują blok, w którym zmienna lokalna jest zdefiniowana. Zmienna lokalna ma więc określony, ograniczony zakres istnienia i dostępności. To w jakich blokach programowych można tworzyć zmienne lokalne definiuje składnia konkretnego języka programowania. Typowymi blokami, w których można w różnych językach programowania tworzyć zmienne lokalne, są moduły, podprogramy oraz w pewnych językach programowania także instrukcje blokowe (lub inne instrukcje strukturalne, np. pętla for w języku C[1][2][3] i inne). Zmienna lokalna w danym bloku przesłania zdefiniowaną zmienną globalną, lub zmienną lokalną z bloku nadrzędnego, o tym samym identyfikatorze. Tym samym programista nie może wprost, za pomocą danego identyfikatora, w bloku o zdefiniowanej zmiennej lokalnej, odwołać się do zmiennej zewnętrznej o tym samym identyfikatorze co zdefiniowana zmienna lokalna, choć może to zrobić za pomocą innych konstrukcji, jeżeli są dostępne w danym języku programowania, np. selekcja, wskaźniki, przemianowanie, nakładanie zmiennych lub inne.

Zmienna(matematyczna) – symbol, oznaczający wielkość, która może przyjmować rozmaite wartości. Wartości te na ogół należą do pewnego zbioru, który jest określony przez naturę rozważanego problemu. Zbiór ten nazywamy zakresem zmiennej.

Przeciwieństwem zmiennej jest stała – jest to wielkość, której wartość nie może się zmieniać – konkretna liczba, wektor, macierz.

W logice zmienna, właściwie symbole zmienne stanowią drugi obok symboli stałych typ znaków charakteryzujących alfabet języka teorii sformalizowanej.

Operatory unarne (jednoargumentowe) - + ,-

Operator binarny jest operatorem, który działa na dwóch argumentach manipuluje nimi i zwraca ich wynik .Operatory są reprezentowane przez znaki specjalne lub za pomocą słów kluczowych i zapewniają łatwy sposób na porównanie wartości liczbowych lub ciągi znaków.

Przykłady:

- ✓ równość (==)
- ✓ nie równe (!=)
- ✓ mniejsze od (<)
- ✓ większe od (>)
- ✓ większe bądź równe do (>=)
- ✓ mniejsze bądź równe do (<=)
- ✓ koniunkcja (&&)
- ✓ alternatywa (||)
- ✓ Plus (+)
- ✓ Minus (-)
- ✓ mnożenie (*)
- ✓ dzielenie (/)

Operator ternarny - Istnieje jeden operator przyjmujący trzy argumenty - jest to operator wyrażenia warunkowego: a ? b : c. Zwraca on b gdy a jest prawdą lub c w przeciwnym wypadku. Operator ternarny nie jest równoważny konstrukcji if i nie jest jej skróconą formą. Wynika to z dwóch powodów:

1. To jest, jak każdy operator, funkcja. I jako funkcja przyjmuje wszystkie trzy argumenty, co oznacza, że wszystkie przypisane wyrażenia są obliczane.

2. Nie powoduje warunkowego wykonania wyrażeń.[/i]

Notacja infiksowa

Zapis infiksowy – inaczej zapis wrostkowy. Klasyczny sposób zapisywania wyrażeń z binarnymi (dwuargumentowymi) operacjami arytmetycznymi (dodawanie, mnożenie, potęgowanie, itd.).

Ogólny schemat:

<arg_A> <operator> <arg_B>

1 + 2

3 * sin(x)

Oprócz symboli i argumentów operacji stosuje się nawiasy, aby ustalić inną niż domyślna kolejność wykonywania operacji.

2 + 3 * 4 / 3 + 1 = 7

2 + 3 * 4 / (3 + 1) = 5

(2 + 3) * 4 / 3 + 1 = 7 + 2/3

Inne sposoby zapisywania wyrażeń z binarnymi operacjami arytmetycznymi pozwalają ustalić dowolną kolejność wykonywania operacji bez stosowania nawiasów.

Notacja prefiksowa

Notacja polska, zapis przedrostkowy, notacja Łukasiewicza – sposób zapisu wyrażeń logicznych (a później arytmetycznych), podający najpierw operator, a potem operandy (argumenty). Został przedstawiony w 1920 roku przez polskiego filozofa i logika Jana Łukasiewicza. Różniła się ona od zapisów nawiasowych używanych, m.in., przez klasyczne dzieło formalizmu logicznego Principia Mathematica Bertranda Russella i A. N. Whiteheada. Według Jana Woleńskiego, notacja ta pozwala na łatwiejsze przeprowadzanie operacji na formułach o znacznej długości; formuły krótsze wydają się bardziej "intuitywne"[1].

Notacja ta używana jest w logice znacznie rzadziej niż notacja nawiasowa; wśród niepolskojęzycznych naukowców używających jej znajduje się m.in. Arthur Prior. Obecnie informatyka jest jedynym polem, gdzie notacja ta jest wciąż popularna.

Notacja w logice:

Operatory logiczne notacji polskiej

- ✓ N – negacja (Np, 'nieprawda że p')
- ✓ C – implikacja (Cpq, 'jeżeli p to q')
- ✓ A – alternatywa (Apq, 'p lub q')
- ✓ D – dysjunkcja (Dpq, 'p albo q')
- ✓ K – koniunkcja (Kpq, 'p i q')
- ✓ E – równoważność (Epq, 'p wtedy, i tylko wtedy gdy q')

Notacja w arytmetyce

Wyrażenie w notacji polskiej nie wymaga nawiasów, ponieważ przypisanie argumentów do operatorów wynika wprost z ich kolejności w zapisie, o ile z góry znana jest liczba argumentów poszczególnych operatorów.

Na przykład zakładając, że operatory / i + są binarne, zapis w notacji polskiej:

/ 7 + 2 3

interpretuje się jednoznacznie jako równoważny notacji tradycyjnej (zapisowi wrostkowemu):

7 / (2 + 3)

Notacja polska jest bliska naturalnemu sposobowi wyrażania działań, w którym zazwyczaj najpierw podaje się czynność, a następnie dopełnia wyrażenie wskazaniem rzeczy, do których czynność się odnosi. Działanie z przykładu powyżej naturalnie wypowiadamy po polsku:

"podziel siedem przez sumę dwu i trzech"

/ 7 + 2 3

Notacja postfiksowa

Odwrotna notacja polska (ONP, ang. reverse polish notation, RPN) – jest sposobem zapisu wyrażeń arytmetycznych, w którym znak wykonywanej operacji umieszczony jest po operandach (zapis postfiksowy) Zapis ten pozwala na całkowitą rezygnację z użycia nawiasów w wyrażeniach, jako że jednoznacznie określa kolejność wykonywanych działań.

ONP bardzo ułatwia wykonywanie na komputerze obliczeń z nawiasami i zachowaniem kolejności działań. Zarówno algorytm konwersji notacji konwencjonalnej (infiksowej) na odwrotną notację polską (postfiksową), jak i algorytm obliczania wartości wyrażenia danego w ONP są bardzo proste i wykorzystują stos.

Wyrażenie – w językach programowania kombinacja wyrażeń stałych (literałów, stałych itp.), zmiennych, operatorów, funkcji i nawiasów, której przypisywana jest wartość zgodnie z regułami danego języka. Proces przypisywania wartości, nazywany jest wartościowaniem lub ewaluacją; podobnie jak w matematyce, wyrażenie jest reprezentacją otrzymanej wartości.

Wyrażenia mogą używać wielu wbudowanych operatorów (operators) i mogą zawierać wywołania funkcji (function calls). Nieokreślony jest porządek, w jakim są przetwarzane argumenty (arguments) dla funkcji i operandy (operands) dla większości operatorów (operators). Przetwarzanie może być przekładane. Jednak wszystkie efekty uboczne (w tym przechowywanie w zmiennych) dzieją się przed następnym punktemsekwencyjnym (sequence point). Stanowią one koniec każdej instrukcji wyrażenia (expression statement) oraz wynik tego zostaje wypisany. I tak jest ze zwrotem z każdego wywołania funkcji (function call). Punkty sekwencyjne też zdarzają się podczas przetwarzania wyrażeń zawierających pewne operatory (&&, ||, ?:oraz operator kropka .). To pozwala na wyższy poziom optymalizacji kodu obiektu przez kompilator.

Punkt sekwencyjny w programowaniu używającym instrukcji (imperative programming) określa jakikolwiek punkt w wykonaniu komputerowego programu, przy którym jest gwarantowane, że wszystkie efekty uboczne poprzednich obliczeń (evaluation) zostały wykonane, a żadne efekty uboczne z następujących obliczeń nie zostały jeszcze wykonane.

- Między obliczaniem lewych i prawych operandów && (AND), || (OR) i operatorów przecinków. Np. w wyrażeniu

*p++ != 0 && *q++ != 0

- Między obliczeniem pierwszego operandu ternerego znaku zapytania (?) a operandem drugim lub trzecim. Np. W wyrażeniu

d = (*p++) ? (*p++) : 0

na końcu pełnego wyrażenia. Ta kategoria zawiera instrukcje wyrażeń (takich jak przydzielenie a = b;), instrukcje return, instrukcje kontrolne if, switch, while,do-while oraz wszystkie trzy wyrażenia w instrukcji for.

-Zanim funkcja jest wprowadzona w wywołaniu funkcji. Porządek, w którym argumenty są obliczane nie jest określony, ale ten punkt sekwencyjny znaczy, że wszystkie ich efekty uboczne (tutaj: zwiększenie parametru o 1) są zakończone zanim funkcja jest wprowadzana przy jej wywołaniu. W wyrażeniu

f(i++) + g(j++) + h(k++), f

Na końcu inicjalizatora np. Po obliczaniu 5 w deklaracji:

int a = 5;

Inne obliczenia nastąpią, jeśli a otrzyma wartość 5.

- Między każdym deklaratorem w sekwencji każdego deklaratora – np. Między tymi dwoma obliczeniami a++ w:

int x = a++, y = a++;

Jeśli a wynosi 5, to x wyniesie 6 i dopiero potem to drugie wyniesie 6.

Wartościowanie leniwe (ang. lazy evaluation, ewaluacja leniwa) - strategia wyznaczania wartości argumentów funkcji tylko wtedy, kiedy są potrzebne (na żądanie).

Zaletami tego podejścia są możliwość obliczenia wartości funkcji nawet wtedy, gdy nie jest możliwe wyznaczenie wartości któregoś z jej argumentów, o ile tylko nie jest on używany, wzrost wydajności dzięki uniknięciu wykonywania niepotrzebnych obliczeń oraz możliwość tworzenia nieskończonych struktur danych. Wadą wartościowania leniwego jest to, że mogą nie wystąpić (być może oczekiwane) skutki uboczne procesu wyznaczania wartości argumentów.

Wartościowanie zachłanne, wartościowanie gorliwe (ang. eager evaluation) to strategia wyznaczania wartości argumentów funkcji przed jej wywołaniem.

Zaletą tego podejścia jest możliwość określenia kolejności wykonywania obliczeń, wadą – konieczność wykonania czasochłonnych obliczeń nawet w sytuacji, kiedy mogą się okazać niepotrzebne.

8.Przepływ sterowania

W informatyce przepływ sterowania oznacza kolejność, w jakiej pojedyncze wyrażenia lub instrukcje są wykonywane w paradygmacie programowania imperatywnego. W języku programowania instrukcja przepływu sterowania może zmienić przepływ sterowania tak, aby wyrażenia wykonywane były w innej kolejności, niż ta, w jakiej są wypisane w kodzie źródłowym. Pojęcie przepływu danych jest prawie zawsze ograniczane do pojedynczego wątku aplikacji, ponieważ dotyczy ono wykonywania instrukcji po jednej naraz.

Do typowych rozkazów wykonywanych przez procesor należą:

- ✓ kopiowanie danych
- ✓ z pamięci do rejestru
- ✓ z rejestru do pamięci
- ✓ z pamięci do pamięci (niektóre procesory)
- ✓ (podział ze względu na sposób adresowania danych)
- ✓ działania arytmetyczne
- ✓ dodawanie
- ✓ odejmowanie
- ✓ porównywanie dwóch liczb
- ✓ dodawanie i odejmowanie jedności
- ✓ zmiana znaku liczby
- ✓ działania na bitach
- ✓ iloczyn logiczny – AND
- ✓ suma logiczna – OR
- ✓ suma modulo 2 (różnica symetryczna) – XOR
- ✓ negacja – NOT
- ✓ przesunięcie bitów w lewo lub prawo
- ✓ skoki
    - bezwarunkowe

- warunkowe

Instrukcje procesora identyfikowane są na podstawie binarnego kodu maszynowego, jednak dany kod nie musi oznaczać wykonywania tych samych operacji przez procesor do tego samego (lub innego) zadania.

W tym celu, w procesorach niedostępnych masowo można spotkać możliwość programowania rozkazów CPU, czyli mikroprogramowania. Rozwiązanie takie daje pełniejszą kontrolę nad procesorem oraz możliwość np. zwiększenia wydajności procesora w pewnych zastosowaniach itp., w znacznie większym stopniu niż w przypadku np. dostępnych powszechnie procesorów, w których kody maszynowe są na stałe przypisane do listy wykonywanych mikroinstrukcji.

Instrukcje bezwarunkowe np. GOTO .Instrukcja skoku bezwarunkowego pozwala(po spełnieniu jakiegoś warunku) przeskoczyć bezwarunkowo do innej części programu.

INSTRUKCJE WARUNKOWE ?????

Instrukcja warunkowa prosta (IF..THEN..)

Struktura stosowana, gdy zachodzi potrzeba warunkowego wykonania (lub pominięcia) pewnych operacji. Asembler (w porównaniu do języków wysokiego poziomu) ogranicza zestaw warunków możliwych do testowania. Ponadto zwykle sprawdzany jest warunek pominięcia operacji, a nie wykonania, jak to jest w językach np. C++ czy Pascal (zaznaczyłem to na sieci działań obok).

Instrukcja warunkowa złożona (IF..THEN..ELSE..)

Jest to rozgałęzienie programu na dwie równoległe ścieżki postępowania. Wymaga to wykonania w programie dwóch skoków, w tym jednego warunkowego.

Sekwencja – uporządkowany ciąg znaków, symboli, nazw, zdarzeń itp., stanowiący strukturę układu, systemu. Występowanie kolejnych elementów wynika z określonej reguły lub formuły. Np. sekwencja wyznacza porządek, w jakim żyły kabla będą podłączone do modularnych gniazd i wtyczek. Do oznakowania sekwencji producenci okablowania używają kolorów.

Problem selekcji w informatyce polega na wyznaczeniu k-tej co do wielkości wśród n liczb. Algorytmami rozwiązującymi ten problem są algorytm Hoare'a oraz algorytm magicznych piątek, przy czym ten drugi działa w pesymistycznym czasie liniowym, co jest najlepszym możliwym rezultatem dla tego problemu.

Selekcja w programowaniu, czyli odniesienie do składowej – zawarta w kodzie źródłowym operacja odniesienia (odwołania, selekcji, wyboru) do pewnej składowej struktury danych[1][2]. Takie odwołanie umożliwia:

- przypisanie składowej określonej wartości
- odczytanie (pobranie) wartości przechowywanej w składowej
- wywołanie podprogramu stanowiącego składową.

Iteracja (łac. iteratio – powtarzanie) – czynność powtarzania (najczęściej wielokrotnego) tej samej instrukcji (albo wielu instrukcji) w pętli. Mianem iteracji określa się także operacje wykonywane wewnątrz takiej pętli.

Paradygmat programowania strukturalnego

Wiele paradygmatów jest dobrze znanych z tego, jakie praktyki są w nich zakazane, a jakie dozwolone. Ścisłe programowanie funkcyjne nie pozwala na tworzenie skutków ubocznych. W programowaniu strukturalnym nie korzysta się z instrukcji skoku (goto). Częściowo z tego właśnie powodu nowe paradygmaty są uważane za zbyt ścisłe przez osoby przyzwyczajone do wcześniejszych stylów. Jednakże takie omijanie pewnych technik znacznie ułatwia przeprowadzanie dowodów o poprawności programu, albo po prostu zrozumienia jego działania, bez ograniczania samego języka programowania.

” programowania imperatywnego.

• Tworzenie programów z kilku dobrze zdefiniowanych konstrukcji takich jak instrukcja warunkowa, pętla, zato bez skoków (go to)

• Użycie tylko trzech rodzajów struktur sterujących: – Sekwencja (lub konkatenacja) — wykonanie instrukcji w określonej kolejności. – Selekcja — wykonanie jednej z kilku instrukcji zależnie od stanu programu. Przykładem jest if-then-else i switch/case. – Cykl — powtarzanie instrukcji tak długo, jak długo spełniony lub niespełniony) jest dany warunek. Pętle, np. while, repeat-until, for itp.

• Cel: sprzyjanie pisaniu programów przejrzystych, łatwych w rozumieniu i utrzymaniu (w rozumieniu imperatywnym.

Etykieta – w informatyce w językach programowania jednostka leksykalna służąca oznaczeniu instrukcji w celu wskazania celu instrukcji skoku. W językach z numerowanymi wierszami kodu funkcje etykiety pełnią numery wierszy (zwykle w językach interpretowanych).

Najczęściej etykiety są pierwszymi wyrażeniami w danym wierszu i mają postać łańcucha znaków (liter lub cyfr) oddzielonych od instrukcji znakiem dwukropka:

Instrukcja warunkowa jest elementem języka programowania, które pozwala na wykonanie różnych obliczeń w zależności od tego czy zdefiniowane przez programistę wyrażenie logiczne jest prawdziwe, czy fałszywe. Możliwość warunkowego decydowania o tym, jaki krok zostanie wykonany w dalszej kolejności jest jedną z podstawowych własności współczesnych komputerów – dowolny model obliczeń zdolny do wykonywania algorytmów (tj. równoważny maszynom Turinga) musi ją posiadać[1].

W imperatywnych językach programowania używa się terminu instrukcja warunkowa, podczas gdy w programowaniu funkcyjnym preferowane są nazwy wyrażenie warunkowe lub konstrukcja warunkowa, gdyż posiadają one inną zasadę działania.

Rodzaje instrukcji warunkowych

- If-The
- Wyrażenia warunkowe
- Operator trójargumentowy

Instrukcja wyboru – instrukcja decyzyjna – jest to instrukcja w określonym języku programowania, umożliwiająca wybór instrukcji do wykonania spośród wielu opcji. Instrukcję wyboru stosuje się w programach, w których należ Składnia instrukcji wyboru różni się w zależności od języka programowania, lecz można wyróżnić w niej charakterystyczne elementy:

- nagłówek instrukcji wyboru:
- słowo kluczowe rozpoczynające instrukcję (np. case, select, switch)
- opcjonalnie wyrażenie, na podstawie którego następuje wybór
- słowo łączące (np. of, do)
- ciało instrukcji wyboru:
- kolejne instrukcje podlegające selekcji
- opcjonalnie poprzedzone frazami zawierającymi wartości porównywane z wyrażeniem z nagłówka instrukcji
- opcjonalnie poprzedzone słowem kluczowym (np. case, when)
- wyrażenie stałe lub lista takich wyrażeń, albo zakresów wartości (np. 1, zm, 7..12)
- słowo lub symbol łącznika (np. :, =>, do)

- opcjonalnie fraza domyślna, wykonywana gdy żadna z fraz nie spełni warunku (np. else, otherwise, other, default, when else)
- koniec instrukcji wyboru – słowo zamykające blok (np. end, end case, end select itp.). y wybrać jedną spośród wielu opcji, np w Pascalu.

W języku C występuje instrukcja decyzyjna **switch** o składni:

```
  switch([warunek]) instrukcja;
  // gdzie instrukcja jest najczęściej [[Instrukcja
blokowa|instrukcją grupującą]] zawierającą frazy case i default:
  {
    case wyr-stałe-1 : instrukcje-1
    [...
    case wyr-stałe-n : instrukcje-n]
    [default : instrukcje-default]
  }
```

W odróżnieniu od instrukcji wyboru implementowanych w większości języków programowania, w instrukcji decyzyjnej języka C wykonywane są wszystkie instrukcje występujące po właściwej frazie **case**, także instrukcje występujące w następnych frazach **case** i **default**. Z tego względu jeżeli występuje konieczność ograniczenia zakresu wykonywania instrukcji należy zastosować instrukcję opuszczenia **break**.

```
  switch(x)
    {
      case 1 : instr-1;
      case 2 : instr-2; break;
      case 3 : instr-3;
    }
```

W powyższym przykładzie instrukcji **switch**, jeśli x=1 to wykonane zostaną *instr-1* i *instr-2*, a jeżeli x=2 to tylko *instr-2*, gdyż instrukcja opuszczenia **break** spowoduje zakończenie wykonywania instrukcji zawartych w grupie switch.

Pętla iteracyjna (pętla licznikowa) w programowaniu, to rodzaj pętli, w ramach której następuje wykonanie określonej liczby iteracji. Do kontroli przebiegu wykonania pętli iteracyjnej stosuje się specjalną zmienną, w odniesieniu do której używa się określeń: zmienna sterująca, a także zmienna kontrolna, czy też zmienna licznikowa. W ramach pętli przejście do kolejnej iteracji wiąże się ze zmianą wartości zmiennej sterującej o określoną wielkość i sprawdzenie warunku, czy nowa wartość zmiennej sterującej znajduje się nadal w dopuszczalnym zakresie wartości, określonym dla tej zmiennej w ramach definicji pętli – co jest równoznaczne z powtórzeniem pętli (wykonaniem kolejnej iteracji), czy też wartość ta znajduje się już poza zakresem – co jest równoznaczne z zakończeniem wykonywania pętli i przejściem do wykonania kolejnej – następnej – instrukcji w programie. W pewnym uogólnieniu działanie pętli iteracyjnej polega na powtarzaniu kolejnych iteracji dla kolejnych wartości przyjmowanych przez zmienną sterującą, aż do osiągnięcia przez tą zmienną wartości spoza dopuszczalnego zakresu. Kolejność wykonywania działań będzie więc następująca:

1. przypisz wartość początkową zmiennej sterującej,
2. sprawdź, czy wartość zmiennej sterującej mieści się w dopuszczalnym zakresie wartości, tzn. jej wartość albo jest mniejsza lub równa od wartości granicznej, albo jest większa lub równa od wartości granicznej, w zależności od rodzaju pętli iteracyjnej,
    - jeżeli wartość zmiennej sterującej mieści się w dopuszczalnym zakresie wartości to przejdź do wykonywania iteracji

- jeżeli wartość zmiennej sterującej nie mieści się w dopuszczalnym zakresie wartości to zakończ wykonywanie pętli

3. wykonaj iterację
4. zmień wartość zmiennej sterującej o zadany krok, wartość ta może być zwiększana lub zmieszana.

Pętla repetycyjna (pętla warunkowa) w programowaniu, to rodzaj pętli, w której wykonanie kolejnej iteracji uzależnione jest od pewnego, zdefiniowanego przez programistę warunku. Warunek zawarty w definiowanej pętli jest pewnym wyrażeniem, które zwraca wartość typu logicznego (np. Pascal, Visual Basic, VBA) Istnieją języki programowania, w których składnia nie przewiduje takiego typu danych. W językach tych stosuje się wyrażenia zwracające pewną wartość innego typu, która następnie podlega odpowiedniej interpretacji, np. wartość zero może być utożsamiana z wartością false typu logicznego, a pozostałe wartości z wartością true, lub inne rozwiązania (np. PL/I,C, C++i pochodne). W zależności do tego, czy wartość logiczna, uzyskana w wyniku ewaluacji wyrażenia reprezentującego warunek, jest równa wartości logicznej true (prawda), czy false (fałsz), wykonywanie pętli jest kontynuowane, bądź przerywane.

Pomijanie i przerywanie iteracji

Słowo kluczowe break przerywa aktualnie wykonywaną pętlę. Innymi słowy: jeżeli w pętli dowolnego typu zostanie wywołane słowo kluczowe break to program 'wie', że pętla ma się natychmiast zakończyć. Kolejną instrukcją, jaka będzie wykonana po wyjściu z pętli za pomocą słowa kluczowego break będzie ta, która znajduje się bezpośrednio poza przerwaną pętlą.

Słowo kluczowe continue wykorzystuje się wtedy, gdy chcemy pominąć wykonywanie kodu znajdującego się w pętli - od miejsca wystąpienia słowa kluczowego continue aż do samego końca pętli. Mówiąc prościej: jeżeli w danym kroku pętli zostanie wywołane słowo kluczowe continue to krok aktualnie wykonywany zostanie zakończony, a następnie pętla przejdzie do wykonywania kolejnego kroku.

Podprogram (inaczej funkcja lub procedura) - termin związany z programowaniem proceduralnym. Podprogram to wydzielona część programu wykonująca jakieś operacje. Podprogramy stosuje się, aby uprościć program główny i zwiększyć czytelność kodu.

Wywołanie podprogramu, to przekazanie sterowania w programie komputerowym do wybranego podprogramu. Taka operacja definiowana jest w kodzie źródłowym w odpowiedni sposób określony w składni konkretnego języka programowania. Wywołanie podprogramu, w skompilowanym, gotowym programie komputerowym, wiąże się z niejawnym wykonaniem szeregu akcji dodatkowych, takich między innymi jak skojarzenie argumentów zawartych w wywołaniu z odpowiednimi parametrami wyspecyfikowanymi w deklaracji podprogramu, a także umieszczenie na stosie niezbędnego adresu umożliwiającego powrót z podprogramu do miejsca wywołania oraz wiele innych kwestii i szczegółów technicznych.

Instrukcja powrotu (wyjścia) to instrukcja w określonym języku programowania powodująca opuszczenie aktualnie wykonywanego bloku programu (modułu, podprogramu: procedury, funkcji, metody, lub innych segmentów – bloków programowych – występujących w określonym języku programowana, a także całego programu, procesu) i przejście do następnej instrukcji występującej po instrukcji wywołania danego podprogramu.

Instrukcja wyjścia:

- niejawna: występuje na końcu danego bloku przed słowem zamykającym segment,
- jawna:

  -bez wartości przekazywanej do miejsca wywołania

  -z wartością przekazywaną do miejsca wywołania

- dla segmentu – podprogramu – będącego funkcją, wartość przekazywana zwykle zostaje użyta w określonym wyrażeniu do obliczeń (istnieją języki dopuszczające wywołanie funkcji poza wyrażeniem i tym samym ignorowanie tej wartości),
- przy wyjściu z programu, wartość przekazywana jest do systemu (lub innego procesu wywołującego program), jako kod zakończenia programu.

Stosowanie instrukcji wyjścia

Blok, segment programu definiuje pewną część algorytmu do wykonania, program definiuje całość danego zagadnienia. Rozpoczęcie wykonywania instrukcji następuje po wykonaniu instrukcji wywołania:

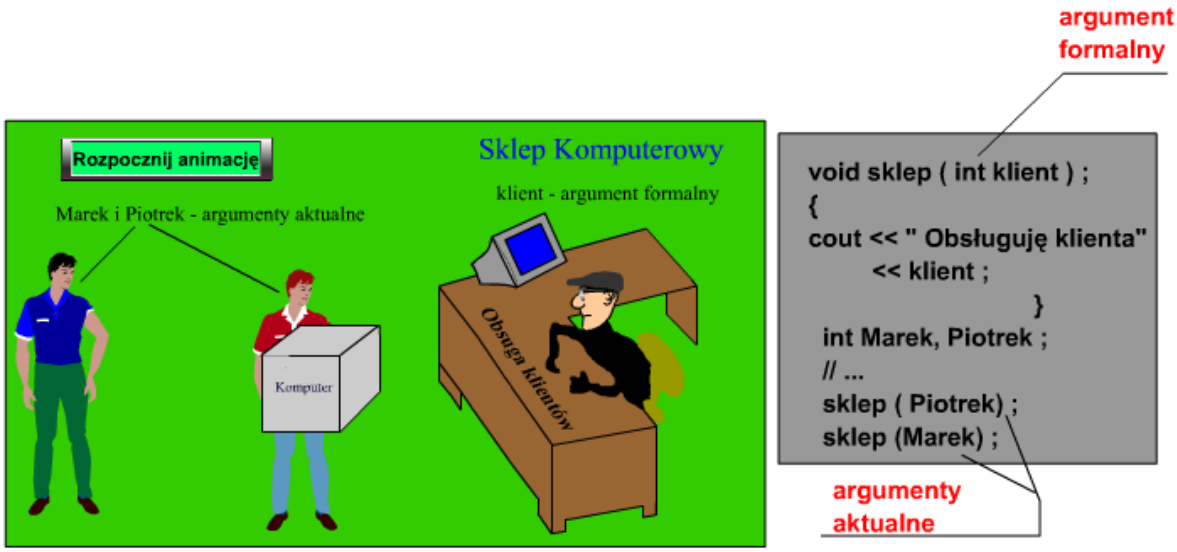
- dla programu w systemie operacyjnym,
- dla podprogramu w instrukcji wywołania lub w wyrażeniu.

Zakończenie natomiast następuje po ostatniej instrukcji zawartej w definicji bloku, co implikuje niejawne (o ile nie umieszczono jawnie instrukcji wyjścia) wystąpienie odpowiedniej instrukcji wyjścia. Często jednak występuje konieczność wcześniejszego zakończenia bloku, co wymaga jawnego użycia instrukcji wyjścia.

Argument formalny to argument, który jest zadeklarowany w nagłówku funkcji i używany w treści funkcji)

Argumenty aktualne to to, co aktualnie stosujemy w konkretnym wywołaniu funkcji.

Załóżmy, że mamy funkcje:
Void sklep (int klient) ;
Załóżmy też, że w programie wywołujemy tę funkcję tak:
int m ;
sklep (3) ;
sklep (m);
A zatem możemy powiedzieć, że nazwa 'klient' to tzw. argument formalny funkcji natomiast to, co pojawia się w nawiasie w momencie wywołania funkcji (u nas 3, m) to tak zwane argumenty (parametry).



Argument (parametr aktualny), w informatyce, to element składni w określonym języku programowania, który w wyniku wywołania podprogramu, zostaje utożsamiony (skojarzony) z określonym parametrem podprogramu. Można stwierdzić, że argument konkretyzuje pewien abstrakcyjny parametr (nadając mu konkretną wartość, kojarząc parametr z określonym miejscem pamięci, zwracając konkretną wartość do miejsca wywołania itp.)

Przekazywanie argumentów

Argumenty przekazywane przez wartość

W tym przypadku parametr zachowuje się w podprogramie jak zmienna lokalna, której przypisano w momencie wywołania podprogramu wartość początkową, równą wartości przekazywanego argumentu. Oznacza to, że argument przekazywany do podprogramu może być wyrażeniem. Po zakończeniu podprogramu, wartość odpowiedniego parametru, nie zostanie zachowana.

Przykładowe języki programowania: Ada ,Pascal, C, i inne.

Argumenty przekazywane przez wynik

Parametr zachowuje się jak zmienna lokalna, bez wartości początkowej (a więc należy dokonać przypisania wartości w podprogramie), a po zakończeniu podprogramu, argument (który powinien być zmienną, wskazaniem na zmienną lub innym wyrażeniem dopuszczalnym po lewej stronie operatora przypisania w danym języku programowania) otrzyma wartość nadaną w podprogramie, przekazaną do miejsca wywołania.

Argumenty przekazywane przez wartość i wynik Jest to połączenie metod przekazywania argumentów:

- przez wartość, oraz
- przez wynik.

Parametr zachowuje się więc jak zmienna lokalna o nadanej w momencie wywołania podprogramu wartości początkowej, równej wartości przekazywanego argumentu, a po zakończeniu podprogramu argument zachowa ostatnio nadaną parametrowi wartość.

Argumenty przekazywane przez referencję

Argumenty przekazywane przez referencję (przez zmienną, przez adres), to nałożenie zmiennej reprezentowanej przez parametr na zmienną reprezentowaną przez argument. Każda zmiana wartości parametru oznacza w rzeczywistości zmianę wartości skojarzonego z nim argumentu. Różnica z przekazywaniem argumentu przez wartość i wynik polega więc na tym, że w tym drugim przypadku istnieje zmienna lokalna, na której wykonywane są wszystkie operacje, przy przekazywaniu przez referencję wszystkie operacje wykonywane są bezpośrednio na argumencie.

Argumenty przekazywane przez nazwę

Ten sposób oznacza użycie argumentu w miejscu parametru, w takiej postaci, w jakiej wystąpił w miejscu wywołania. W uproszczeniu (nie jest to do końca prawdziwe stwierdzenie), obrazowo, można by utożsamić przekazywanie argumentu przez nazwę z tekstowym podstawieniem zapisu argumentu w każdym miejscu podprogramu, w którym odwołano się do parametru.

Dokładniej: w ALGOLu przekazywanie przez nazwę zdefiniowane jest następująco:

- Procedura jest traktowana jak podobnie jak makro, czyli jej treść jest wklejana w miejscu wywołania, z zastąpieniem parametrów przekazywanych przez nazwę argumentami.
- Dla zachowania jednoznaczności parametry aktualne otaczane są nawiasami.
- Jeśli zmienne lokalne procedury kolidują ze zmiennymi widocznymi w miejscu wywołania, to ich nazwy zostają zmienione na unikatowe.

Rekurencja w dowolnym języku programowania to nic innego jak wywołanie przez funkcję samej siebie. Na myśl przychodzi mi porównanie rekurencji do pętli, ponieważ jest ona również wywoływana określoną liczbę

razy (do momentu, gdy zajdzie warunek stopu). I tak jest w rzeczywistości – rekurencja dość często zastępuje pętle w programowaniu.

Wyjątek (ang. exception) jest mechanizmem przepływu sterowania używanym w mikroprocesorach oraz współczesnych językach programowania do obsługi zdarzeń wyjątkowych, a w szczególności błędów, których wystąpienie zmienia prawidłowy przebieg wykonywania programu. W momencie zajścia niespodziewanego zdarzenia generowany jest wyjątek, który musi zostać obsłużony poprzez zapamiętanie bieżącego stanu programu i przejście do procedury jego obsługi. W niektórych sytuacjach po obsłużeniu wyjątku można powrócić do wykonywania przerwanego kodu, korzystając z zapamiętanych informacji stanu. Przykładowo obsługa błędu braku strony pamięci polega najczęściej na pobraniu brakującej strony z pliku wymiany, co umożliwia kontynuowanie pracy programu, natomiast błąd dzielenia przez zero powoduje, że wykonywanie dalszych obliczeń nie ma sensu i musi zostać definitywnie przerwane.

W językach programowania wsparcie dla wyjątków realizowane jest na poziomie składni i semantyki danego języka. Zgłoszenie sytuacji wyjątkowej możliwe jest w dowolnym miejscu kodu poprzez instrukcje zwane raise lub throw. Od ich angielskich nazw w języku polskim proces ten nazywany jest podnoszeniem lub rzucaniem wyjątku. Dla dowolnej partii kodu możliwe jest zdefiniowanie bloku obsługi, który przechwytuje (ang. catch) określone rodzaje wyjątków.

Blok finally -Istotnym problemem w obsłudze wyjątków jest to, że wewnątrz bloku try mogły zostać tymczasowo zaalokowane jakieś zasoby, które po zakończeniu wykonywania powinny zostać zwolnione. Jeśli rzucanie i przechwytywanie wyjątku zachodzi w obrębie tej samej funkcji, odpowiedni kod można umieścić za sekcją try ... catch, lecz funkcja rzuca wyjątek, który powinien przechwycić kod wywołujący, programista sam musi zadbać, by zwolnić wszystkie tymczasowe zasoby przed jego rzuceniem. Dlatego w niektórych językach wprowadzony jest dodatkowy, opcjonalny blok finally, który musi się wykonać niezależnie od tego, czy wewnątrz try został rzucony wyjątek, czy nie

9.Klasyczne algorytmy w informatyce

Przeszukiwanie liniowe (lub wyszukiwanie sekwencyjne) to najprostszy algorytm wyszukiwania informacji w ciągu danych, np. zapisanych w tablicy lub na liście. Polega na porównywaniu żądanego klucza z kolejnymi kluczami z sekwencji danych – wyszukiwanie kończy się powodzeniem, gdy zostanie znaleziony klucz, albo niepowodzeniem, gdy zostaną przejrzane wszystkie klucze.

Wyszukiwanie binarne jest algorytmem opierającym się na metodzie dziel i zwyciężaj, który w czasie logarytmicznym stwierdza, czy szukany element znajduje się w uporządkowanej tablicy i jeśli się znajduje, podaje jego indeks. Np. jeśli tablica zawiera milion elementów, wyszukiwanie binarne musi sprawdzić maksymalnie 20 elementów (\log_2{1\,000\,000} \approx 20) w celu znalezienia żądanej wartości. Dla porównania wyszukiwanie liniowe wymaga w najgorszym przypadku przejrzenia wszystkich elementów tablicy.

Sortowanie – jeden z podstawowych problemów informatyki, polegający na uporządkowaniu zbioru danych względem pewnych cech charakterystycznych każdego elementu tego zbioru. Szczególnym przypadkiem jest sortowanie względem wartości każdego elementu, np. sortowanie liczb, słów itp.

Algorytmy sortowania są stosowane w celu uporządkowania danych, umożliwienia stosowania wydajniejszych algorytmów (np. wyszukiwania) i prezentacji danych w sposób czytelniejszy dla człowieka.

Algorytmy sortowania są zazwyczaj klasyfikowane według

- złożoności (pesymistyczna, oczekiwana) – zależność liczby wykonanych operacji w stosunku od liczebności sortowanego zbioru (n). Typową, dobrą złożonością jest średnia O(n log n) i pesymistyczna $\Omega(n^2)$. Idealną złożonością jest O(n). Algorytmy sortujące nie przyjmujące żadnych wstępnych założeń dla danych wejściowych wykonują co najmniej O(n log n) operacji w modelu obliczeń, w którym wartości są "przezroczyste" i dopuszczalne jest tylko ich porównywanie (w niektórych bardziej ograniczonych modelach istnieją asymptotycznie szybsze algorytmy sortowania);

- złożoność pamięciowa
- sposób działania: algorytmy sortujące za pomocą porównań to takie algorytmy sortowania, których sposób wyznaczania porządku jest oparty wyłącznie na wynikach porównań między elementami; Dla takich algorytmów dolne ograniczenie złożoności wynosi Ω(n log n);
- stabilność: stabilne algorytmy sortowania utrzymują kolejność występowania dla elementów o tym samym kluczu (klucz – cecha charakterystyczna dla każdego elementu zbioru, względem której jest dokonywane sortowanie). Oznacza to, że dla każdych dwóch elementów R i S o tym samym kluczu, jeśli R wystąpiło przed S to po sortowaniu stabilnym R nadal będzie przed S;

Sortowanie przez wstawianie

Metoda powszechnie stosowana przez grających w karty...

W każdym kroku (iteracji) metody, poczynając od elementu 2 do elementu n, porównuje się i-ty element z elementami poprzednimi i jeśli mają one większe klucze, przesuwa się je w prawo, robiąc miejsce na bieżący element. W każdej iteracji, przesuwanie elementów kończy się z chwilą znalezienia elementu z kluczem mniejszym niż klucz danego elementu. Jeśli takiego elementu nie ma, to przesuwanie należy zakończyć po osiągnięciu „lewego" brzegu zbioru.

Sortowanie przez wybieranie

W każdym kroku (iteracji) metody wyszukuje się najmniejszy element i zamienia go się z elementem na pierwszej pozycji. Operację tę powtarza się następnie dla n-1 elementów i zamienia się element na drugiej pozycji. Iteracje takie powtarza się, aŝ zostanie tylko jeden element – o największym kluczu. Wykonuje się więc n-1 iteracji.

Metoda sortowania przez wybieranie jest w pewnym sensie odwrotna do metody sortowania przez wstawianie:

• w metodzie wstawiania w każdym kroku rozważa się tylko jeden, kolejny element ciągu źródłowego i wszystkie elementy tablicy wynikowej aby znaleźć miejsce wstawienia nowego elementu,

• w metodzie wybierania rozważa się wszystkie elementy tablicy źródłowej, aby znaleźć element z najmniejszym kluczem i ustawić go jako kolejny element ciągu wynikowego.

Sortowanie przez zamianę (sortowanie bąbelkowe)

W każdym kroku (iteracji) metody porównuje się klucze wszystkich sąsiadujących elementów i jeśli trzeba zamienia się elementy miejscami (gdy ich kolejność nie jest zgodna z kluczem). W ten sposób po n-1 przejściach przez zbiór dochodzi się do stanu uporządkowania. Popularna nazwa metody „sortowanie bąbelkowe" wynika z podobieństwa przemieszczania się elementów „w górę" tabeli, jak bąbelków powietrza w wodzie (jeśli tabela jest ustawiona tak jak w przykładach, tj. kolejne permutacje zbioru są kolumnami.

Sortowanie bąbelkowe jest najmniej efektywne ze wszystkich omawianych metod, lecz ma bardzo prostą implementację. Jej stosowanie ma sens tylko gdy sortowany zbiór jest bardzo mały.

Przeszukiwanie grafu lub inaczej przechodzenie grafu to czynność polegająca na odwiedzeniu w jakiś usystematyzowany sposób wszystkich wierzchołków grafu w celu zebrania potrzebnych informacji. Często podczas przejścia grafu rozwiązujemy już jakiś problem, ale przeważnie jest to tylko wstęp do wykonania właściwego algorytmu.

Powszechnie stosuje się dwie podstawowe metody przeszukiwania grafów:

- przeszukiwanie wszerz (BFS)
- przeszukiwanie w głąb (DFS)

Algorytm Dijkstry, opracowany przez holenderskiego informatyka Edsgera Dijkstrę, służy do znajdowania najkrótszej ścieżki z pojedynczego źródła w grafie o nieujemnych wagach krawędzi. Mając dany graf z wyróżnionym wierzchołkiem (źródłem) algorytm znajduje odległości od źródła do wszystkich pozostałych wierzchołków. Łatwo zmodyfikować go tak, aby szukał wyłącznie (najkrótszej) ścieżki do jednego ustalonego wierzchołka, po prostu przerywając działanie w momencie dojścia do wierzchołka docelowego, bądź transponując tablicę incydencji grafu.

Algorytm Dijkstry znajduje w grafie wszystkie najkrótsze ścieżki pomiędzy wybranym wierzchołkiem a wszystkimi pozostałymi, przy okazji wyliczając również koszt przejścia każdej z tych ścieżek.

Algorytm Dijkstry jest przykładem algorytmu zachłannego

Z algorytmu Dijkstry można skorzystać przy obliczaniu najkrótszej drogi do danej miejscowości. Wystarczy przyjąć, że każdy z punktów skrzyżowań dróg to jeden z wierzchołków grafu, a odległości między punktami to wagi krawędzi.

Jest często używany w sieciach komputerowych

Algorytm A* – algorytm heurystyczny znajdowania najkrótszej ścieżki w grafie ważonym z dowolnego wierzchołka do wierzchołka spełniającego określony warunek zwany testem celu. Algorytm jest zupełny i optymalny, w tym sensie, że znajduje ścieżkę, jeśli tylko taka istnieje, i przy tym jest to ścieżka najkrótsza. Stosowany głównie w dziedzinie sztucznej inteligencji do rozwiązywania problemów i w grach komputerowych do imitowania inteligentnego zachowania.

Algorytm został opisany pierwotnie w 1968 roku przez Petera Harta, Nilsa Nilssona oraz Bertrama Raphaela. W ich pracy naukowej został nazwany algorytmem A. Ponieważ jego użycie daje optymalne zachowanie dla danej heurystyki, nazwano go A*.

Algorytm min-max , Minimax (czasami minmax) jest metodą w teorii decyzji do minimalizowania maksymalnych możliwych strat. Alternatywnie można je traktować jako maksymalizację minimalnego zysku (maximin). Wywodzi się to z teorii gry o sumie zerowej, obejmujących oba przypadki, zarówno ten, gdzie gracze wykonują ruchy naprzemiennie, jak i ten, gdzie wykonują ruchy jednocześnie. Zostało to również rozszerzone na bardziej skomplikowane gry i ogólne podejmowanie decyzji w obecności niepewności.

Problem komiwojażera (TSP - ang. travelling salesman problem) jest to zagadnienie optymalizacyjne, polegające na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym.

Nazwa pochodzi od typowej ilustracji problemu, przedstawiającej go z punktu widzenia wędrownego sprzedawcy (komiwojażera): dane jest n miast, które komiwojażer ma odwiedzić, oraz odległość/cena podróży/czas podróży pomiędzy każdą parą miast. Celem jest znalezienie najkrótszej/najtańszej/najszybszej drogi łączącej wszystkie miasta zaczynającej się i kończącej się w określonym punkcie.

Symetryczny problem komiwojażera (STSP) polega na tym, że dla dowolnych miast A i B odległość z A do B jest taka sama jak z B do A. W asymetrycznym problemie komiwojażera (ATSP) odległości te mogą być różne.

Rozwinięciem problemu komiwojażera jest problem marszrutyzacji.

Odległość Hamminga (ang. Hamming distance) D_H – w teorii informacji jest to wprowadzona przez Richarda Hamminga miara odmienności dwóch ciągów o takiej samej długości, wyrażająca liczbę miejsc (pozycji), na których te dwa ciągi się różnią. Innymi słowy jest to najmniejsza liczba zmian (operacji zastępowania elementu innym), jakie pozwalają przeprowadzić jeden ciąg na drugi.Własności:

Dla ustalonej długości n, odległość Hamminga jest metryką na przestrzeni wektorowej słów o tej długości, ponieważ spełnia warunki: jest nieujemna, symetryczna, a stosując metodę indukcji zupełnej można pokazać, że spełnia również nierówność trójkąta. Odległość Hamminga dwóch słów a i b można także interpretować jako ciężar Hamminga słowa a−b dla odpowiedniego wyboru operatora −.

Dla ciągów binarnych a i b odległość Hamminga jest równa liczbie jedynek w słowie a XOR b. Przestrzeń metryczna słów binarnych o długości n, z odległością Hamminga, jest nazywana kostką Hamminga. Słowa binarne o długości n moża traktować jako wektory w przestrzeni \mathbb{R}^n przyjmując każdy symbol w łańcuchu jako współrzędną rzeczywistą; przy tym zanurzeniu takie łańcuchy stanowią wierzchołki n-wymiarowej hiperkostki, a odległość Hamminga słów jest równoważna metryce taksówkowej pomiędzy wierzchołkami.

Odległość Levenshteina (edycyjna) – miara odmienności napisów (skończonych ciągów znaków), zaproponowana w 1965 roku przez Władimira Lewensztejna. Formalnie jest to metryka w przestrzeni ciągów znaków, zdefiniowana następująco:

- działaniem prostym na napisie nazwiemy:
  -wstawienie nowego znaku do napisu
  -usunięcie znaku z napisu
  -zamianę znaku w napisie na inny znak
- odległością pomiędzy dwoma napisami jest najmniejsza liczba działań prostych, przeprowadzających jeden napis na drugi.

Miara ta znajduje zastosowanie w przetwarzaniu informacji, danych w postaci ciągów symboli: w maszynowym rozpoznawaniu mowy, analizie DNA, rozpoznawaniu plagiatów, korekcie pisowni (np. wyszukiwanie w spisie telefonicznym błędnie podanego nazwiska), itp.

Dyskretny problem plecakowy (ang. discrete knapsack problem) jest jednym z najczęściej poruszanych problemów optymalizacyjnych. Nazwa zagadnienia pochodzi od maksymalizacyjnego problemu wyboru przedmiotów, tak by ich sumaryczna wartość była jak największa i jednocześnie mieściły się w plecaku. Przy podanym zbiorze elementów o podanej wadze i wartości, należy wybrać taki podzbiór by suma wartości była możliwie jak największa, a suma wag była nie większa od danej pojemności plecaka.

Problem plecakowy często przedstawia się jako problem złodzieja rabującego sklep – znalazł on N towarów; j–ty przedmiot jest wart c_{j} oraz waży w_{j}. Złodziej dąży do zabrania ze sobą jak najwartościowszego łupu, przy czym nie może zabrać więcej niż B kilogramów. Nie może też zabierać ułamkowej części przedmiotów (byłoby to możliwe w ciągłym problemie plecakowym).

10. Informacja a złożoność

Złożoność Kołmogorowa – dla łańcucha symboli, skończonego lub nieskończonego, to długość najkrótszego programu, który generuje dany łańcuch. Nazwa pojęcia pochodzi od nazwiska Andrieja Kołmogorowa.

Rozwinięcie dziesiętne liczby pi, choć nieskończone, ma bardzo niską złożoność Kołmogorowa, ponieważ istnieje bardzo prosty program, który generuje dowolną liczbę jej cyfr. Złożoność Kołmogorowa jest różna dla różnych komputerów (ściślej – maszyn Turinga lub obiektów izomorficznych z nimi). Ze względu na problem stopu nie może istnieć algorytm obliczający złożoność Kołmogorowa z gwarancją sukcesu.

Entropia – termodynamiczna funkcja stanu, określająca kierunek przebiegu procesów spontanicznych (samorzutnych) w odosobnionym układzie termodynamicznym. Entropia jest miarą stopnia nieuporządkowania układuJest wielkością ekstensywną[2]. Zgodnie z drugą zasadą termodynamiki, jeżeli układ termodynamiczny przechodzi od jednego stanu równowagi do drugiego, bez udziału czynników zewnętrznych (a więc spontanicznie), to jego entropia zawsze rośnie. Pojęcie entropii wprowadził niemiecki uczony Rudolf Clausius.

Redundancja (łac. redundantia – powódź, nadmiar, zbytek) – nadmiarowość w stosunku do tego, co konieczne lub zwykłe. Określenie może odnosić się zarówno do nadmiaru zbędnego lub szkodliwego, niecelowo zużywającego zasoby, jak i do pożądanego zabezpieczenia na wypadek uszkodzenia części systemu.

Kompresja danych (ang. data compression) – polega na zmianie sposobu zapisu informacji tak, aby zmniejszyć redundancję i tym samym objętość zbioru. Innymi słowy chodzi o wyrażenie tego samego zestawu informacji, lecz za pomocą mniejszej liczby bitów.

Kompresja dzieli się na bezstratną – w której z postaci skompresowanej można odzyskać identyczną postać pierwotną oraz stratną – w której takie odzyskanie jest niemożliwe, jednak główne właściwości, które nas interesują, zostają zachowane, np. jeśli kompresowany jest obrazek, nie występują w postaci odtworzonej widoczne różnice w stosunku do oryginału. Pomimo to może się już nie nadawać zbyt dobrze np. do dalszej przeróbki czy do wydruku, gdyż w tych zastosowaniach wymaga się zachowania innych właściwości.

Algorytmy kompresji dzieli się na algorytmy zastosowania ogólnego oraz algorytmy do danego typu danych. Z definicji nie istnieją algorytmy kompresji stratnej zastosowania ogólnego, ponieważ dla różnych typów danych konieczne jest zachowanie różnych właściwości. Na przykład kompresja dźwięku używa specjalnego modelu psychoakustycznego, który nie ma sensu w zastosowaniu do obrazu, poza bardzo ogólnymi przesłankami dotyczącymi sposobu postrzegania rzeczywistości przez człowieka.

Większość algorytmów bezstratnych to algorytmy zastosowania ogólnego oraz ich drobne przeróbki, dzięki którym lepiej działają z określonymi typami danych. Nawet drobne poprawki mogą znacząco polepszyć wyniki dla pewnych typów danych.

Algorytmy kompresji stratnej często jako ostatniej fazy używają kompresji bezstratnej. W takim przypadku poprzednie fazy mają za zadanie nie tyle kompresować, ile przygotować dane do łatwiejszej kompresji.

Run-Length Encoding (RLE, kodowanie długości serii) – prosta metoda bezstratnej kompresji danych, której działanie polega na opisywaniu ciągów tych samych liter (bitów, bajtów, symboli, pikseli itp.) za pomocą licznika powtórzeń (długości ciągu), a dokładniej przez pary: licznik powtórzeń litery, litera.

Kodowanie Huffmana (ang. Huffman coding) – jedna z najprostszych i łatwych w implementacji metod kompresji bezstratnej.Została opracowana w 1952 roku przez Amerykanina Davida Huffmana.

Algorytm Huffmana nie należy do najefektywniejszych systemów bezstratnej kompresji danych, dlatego też praktycznie nie używa się go samodzielnie. Często wykorzystuje się go jako ostatni etap w różnych systemach kompresji, zarówno bezstratnej, jak i stratnej, np. MP3 lub JPEG. Pomimo że nie jest doskonały, stosuje się go ze względu na prostotę oraz brak ograniczeń patentowych. Jest to przykład wykorzystania algorytmu zachłannego.

Lempel-Ziv-Welch, LZW – metoda strumieniowej bezstratnej kompresji słownikowej, będąca modyfikacją metody LZ78.

Pomysłodawcą algorytmu jest Terry A. Welch. Metodę opisał w 1984 roku, w artykule A technique for high-performance data compression opublikowanym w numerze 6. Computer (str. 8-19).

Metoda LZW jest względnie łatwa do zaprogramowania, daje bardzo dobre rezultaty. Wykorzystywana jest m.in. w programach ARC, PAK i UNIX-owym compress, w formacie zapisu grafiki GIF, w formatach PDF i PostScript (filtry kodujące fragmenty dokumentu) oraz w modemach (V.42bis). LZW było przez pewien czas algorytmem objętym patentem, co było przyczyną podjęcia prac nad nowym algorytmem kompresji obrazów, które zaowocowały powstaniem formatu PNG.

Kontrola parzystości – metoda wykrywania przekłamań w transmitowanych wiadomościach. Polega na dodawaniu do wysyłanej wiadomości bitu kontrolnego. W zależności od przyjętej konwencji bit ten nazywany jest bitem parzystości lub bitem nieparzystości. Kontrola parzystości opiera się na parzystości sumy bitów wiadomości, której nie należy mylić z parzystością wiadomości potraktowanej jako liczba dwójkowa. Tę drugą parzystość można odczytać wprost z najmniej znaczącego bitu wiadomości.

Bitem parzystości nazywamy bit kontrolny, który przyjmuje wartość 1, gdy liczba jedynek w przesyłanej wiadomości jest nieparzysta lub 0, gdy odwrotnie. Innymi słowy – bit parzystości sprawia, że wiadomość ma

zawsze parzystą liczbę jedynek. W tym wariancie bit parzystości możemy obliczyć wykonując sumę modulo dwa na wszystkich bitach wiadomości:

Cykliczny kod nadmiarowy, inaczej: cykliczna kontrola nadmiarowa (ang.) Cyclic Redundancy Check, CRC – jest to system sum kontrolnych wykorzystywany do wykrywania przypadkowych błędów pojawiających się podczas przesyłania i magazynowania danych binarnych.

Stratna kompresja danych multimedialnych

W wyniku kompresji część danych (mniej istotnych) jest bezpowrotnie tracona, dane po dekompresji nieznacznie różnią się od oryginalnych danych przed kompresją.

Próbkowanie (dyskretyzacja, kwantowanie w czasie) - proces tworzenia sygnału dyskretnego, reprezentującego sygnał ciągły za pomocą ciągu wartości nazywanych próbkami. Zwykle jest jednym z etapów przetwarzania sygnału analogowego na sygnał cyfrowy.

Kwantyzacja to nieodwracalne nieliniowe odwzorowanie statyczne zmniejszające dokładność danych przez ograniczenie ich zbioru wartości. Zbiór wartości wejściowych dzielony jest na rozłączne przedziały. Każda wartość wejściowa wypadająca w określonym przedziale jest w wyniku kwantyzacji odwzorowana na jedną wartość wyjściową przypisaną temu przedziałowi, czyli tak zwany poziom reprezentacji. W rozumieniu potocznym proces kwantyzacji można przyrównać do "zaokrąglania" wartości do określonej skali.

11. Determinizm i chaos w systemach komputerowych

Determinizm przyczynowy (łac. determinare — oddzielić, ograniczyć, określić) — koncepcja filozoficzna, według której wszystkie zdarzenia w ramach przyjętych paradygmatów są połączone związkiem przyczynowo-skutkowym, a zatem każde zdarzenie i stan jest zdeterminowane przez swoje uprzednio istniejące przyczyny (również zdarzenia i stany). W wypadku najpowszechniejszego tzw. determinizmu fizycznego (zwanego też nomologicznym lub naukowym) oznacza to powszechne obowiązywanie praw natury w całej przyrodzie bez wyjątku (także w człowieku): a zatem, zgodnie z tzw. postulatem Laplace'a (zob. demon Laplace'a), hipotetyczne całkowite poznanie stanu wszechświata w dowolnym momencie wraz z całością reguł jego działania gwarantowałoby bezbłędną prognozę dalszych jego losów.

?????Ergodyczność (gr. érgon ‘praca, dzieło’) fiz. właściwość przestrzeni powalająca na zastępowanie jej średniej czasowej średnią fazową funkcji dowolnej wielkości fizycznej ją opisującej.

???Chaos deterministyczny − w matematyce i fizyce, własność równań lub układów równań, polegająca na dużej wrażliwości rozwiązań na dowolnie małe zaburzenie parametrów. Dotyczy to zwykle nieliniowych równań różniczkowych i różnicowych, opisujących układy dynamiczne.

?????Zachowanie układów chaotycznych

Ścisłym kryterium chaotyczności jest określenie wartości wykładników Lapunowa. Układ jest chaotyczny, jeśli ma co najmniej jeden dodatni wykładnik Lapunowa. W takim wypadku w przestrzeni fazowej blisko leżące trajektorie mogą po pewnym czasie dowolnie się od siebie oddalić (to oddalanie się "z czasem" różnych trajektorii obliczonych dla odmiennych wartości parametru leży w definicji chaotyczności względem tego parametru). Choć dla idealnie dokładnie zadanych parametrów początkowych jesteśmy w stanie dokładnie przewidzieć zachowanie się układu, w praktyce, gdzie warunki początkowe znane są zawsze ze skończoną dokładnością, w krótkim czasie układ staje się nieprzewidywalny.

Szczególną cechą układów chaotycznych jest tzw. mieszanie topologiczne. Oznacza ono, że jeśli weźmiemy dowolny region (zbiór otwarty) w przestrzeni fazowej układu, to w miarę jego ewolucji w czasie pokryje się on częściowo z dowolnym innym wybranym regione

Warto nadmienić, że niektóre równania i układy liniowe posiadają także rozwiązania niestabilne. Tym samym niestabilność rozwiązań jest własnością słabszą niż chaotyczność. W przypadkach liniowych niestabilność

dotyczy jednak jedynie specjalnie dobranych warunków początkowych. Układ jest uważany za chaotyczny, gdy niestabilność dotyczy prawie wszystkich warunków początkowych (formalnie zestawy tych warunków tworzą zbiór gęsty).

Dowiedzenie, że dany, konkretny układ równań jest chaotyczny dla pewnych wartości parametrów modelu, jest na ogół procesem złożonym. Dlatego niewłaściwe jest nazywanie chaotycznym każdego układu przejawiającego skomplikowane zachowania. Przykładami układów mylnie nazywanych chaotycznymi są turbulencje i zachowanie giełdy. Nie udowodniono chaotyczności dla pełnego układu równań Naviera-Stokesa, a dla zachowania giełdy nie znamy nawet równań różnicowych czy różniczkowych, opisujących ją w zadowalający sposób. Tym samym nie potrafimy się wypowiedzieć o chaotyczności ich rozwiązań.

Funkcja skrótu, funkcja mieszająca lub funkcja haszująca – funkcja przyporządkowująca dowolnie dużej liczbie krótką, zawsze posiadającą stały rozmiar, niespecyficzną, quasi-losową wartość, tzw. skrót nieodwracalny.

W informatyce funkcje skrótu pozwalają na ustalenie krótkich i łatwych do weryfikacji sygnatur dla dowolnie dużych zbiorów danych. Sygnatury mogą chronić przed przypadkowymi lub celowo wprowadzonymi modyfikacjami danych (sumy kontrolne), a także mają zastosowania przy optymalizacji dostępu do struktur danych w programach komputerowych

Własności funkcji skrótu

- własność kompresji
- własność łatwości obliczeń, mając dane M łatwo jest obliczyć h
- własność jednokierunkowości, mając dane h, trudno jest obliczyć M
- własność słabej odporności na kolizje, mając dane M, trudno jest znaleźć inną wiadomość M' taką, że H(M)=H(M')
- własność odporności na kolizje jest obliczeniowo trudne znalezienie dwóch dowolnych argumentów M ≠ M', dla których H(M) = H(M').

Kolizja funkcji skrótu H to taka para różnych wiadomości m1, m2, że mają one taką samą wartość skrótu, tj. H(m1) = H(m2).

Ponieważ funkcja skrótu zwraca skończenie wiele wartości, a przestrzeń argumentów jest nieskończona (w przypadku funkcji akceptujących dowolnie długie argumenty), lub przynajmniej znacznie większa od przestrzeni wyników, dla każdej funkcji skrótu istnieją kolizje.

W wielu zastosowaniach zależy nam na tym, żeby nie była znana żadna kolizja funkcji skrótu. Jest to jednak bardzo silne wymaganie, i często zależy nam na słabszej właściwości funkcji (od silniejszych do słabszych właściwości):

- niemożliwość łatwego generowania nowych kolizji
- niemożliwość znalezienia, dla danego m1 takiego m2, że H(m1) = H(m2), czyli second preimage resistance
- niemożliwość znalezienia, dla danego h takiego m że H(m) = h, czyli preimage resistance

Zasada szufladkowa Dirichleta – twierdzenie mówiące, że jeżeli m przedmiotów włożymy do n różnych szufladek, przy czym m>n, to co najmniej w jednej szufladce znajdą się co najmniej dwa przedmioty.

Sformułowanie twierdzenia przypisuje się Dirichletowi, a w bardziej formalnym języku można wysłowić je na przykład tak:

jeżeli zbiór X liczy n elementów i X=X_1\cup X_2\cup\dots \cup X_k i n>k, to któryś ze zbiorów X_i musi liczyć przynajmniej dwa elementy.

Inna wersja formalna brzmi następująco:

Jeżeli zbiór X liczy n elementów, zbiór Y – m elementów i n>m, to nie istnieje funkcja różnowartościowa ze zbioru X do zbioru Y.

Wydaje się, że ta oczywista obserwacja nie może mieć poważnych zastosowań, ale jest akurat przeciwnie. Zasada szufladkowa bywa wykorzystywana w dowodach wielu głębokich twierdzeń matematycznych i często samo zauważenie, że można ją zastosować, jest kluczem do rozwiązania problemu.

Funkcja jednokierunkowa – funkcja, która jest łatwa do wyliczenia, ale trudna do odwrócenia. "Łatwa do wyliczenia" oznacza tu, że istnieje algorytm wielomianowy, który ją wylicza. "Trudna do odwrócenia" oznacza, że żaden wielomianowy algorytm probabilistyczny nie potrafi znaleźć elementu przeciwobrazu f(x) z prawdopodobieństwem większym niż zaniedbywalne, jeśli x jest wybrane losowo. Trudność dotyczy zatem średniego przypadku, a nie pesymistycznego, jak w większości problemów w teorii złożoności obliczeniowej (np. w problemach NP-trudnych).

Zastosowania

- ✓ Zabezpieczanie haseł w komputerach
- ✓ Ochrona integralności danych
- ✓ Ochrona przed wirusami
- ✓ Podpis cyfrowy
- ✓ Generatory ciągów pseudolosowych (kluczy), tworzonej na podstawie tajnej danej.
- ✓ MIC(mesage integrity checks)
- ✓ MAC (message authentication code) – funkcjeodwracalne ale tylko w przypadku posiadania klucza

*Kryptologia (z gr. κρυπτός – kryptos – "ukryty" i λόγος – logos – "rozum", "słowo") – dziedzina wiedzy o przekazywaniu informacji w sposób zabezpieczony przed niepowołanym dostępem. Współcześnie kryptologia jest uznawana za gałąź zarówno matematyki, jak i informatyki; ponadto jest blisko związana z teorią informacji, inżynierią oraz bezpieczeństwem komputerowym.

12.Losowość i pseudolosowość

Generator liczb pseudolosowych (Pseudo-Random Number Generator, lub PRNG) to program lub podprogram, który na podstawie niewielkiej ilości informacji (ziarno, zarodek, ang. seed) generuje deterministycznie ciąg bitów, który pod pewnymi względami jest nieodróżnialny od ciągu uzyskanego z prawdziwie losowego źródła.

Generatory liczb pseudolosowych nie generują ciągów prawdziwie losowych – generator inicjowany ziarnem, które może przyjąć k różnych wartości, jest w stanie wyprodukować co najwyżej k różnych ciągów liczb. Co więcej, ponieważ rozmiar zmiennych reprezentujących wewnętrzny stan generatora jest ograniczony (zwykle decyzją programisty, do kilkudziesięciu lub kilkuset bitów; a rzadziej, po prostu rozmiarem pamięci komputera), i ponieważ w związku z tym może on znajdować się tylko w ograniczonej liczbie stanów, bez dostarczania nowych danych z zewnątrz musi po jakimś czasie dokonać pełnego cyklu i zacząć generować te same wartości. Teoretyczny limit długości cyklu wyrażony jest przez 2^n, gdzie n to liczba bitów przeznaczonych na przechowywanie stanu wewnętrznego. W praktyce, większość generatorów ma znacznie krótsze okresy.

Do wielu zastosowań, opisany powyżej rodzaj deterministycznej pseudolosowości jest w zupełności wystarczający. W grach komputerowych czy algorytmach probabilistycznych (takich jak np. całkowanie Monte Carlo) potrzebne jest jedynie źródło wartości o cechach przybliżonych do liczb prawdziwie losowych, chociaż jakość losowości może być decydująca dla dokładności obliczeń. Dlatego przy zastosowaniu każdego nowego

generatora do celów obliczeń numerycznych należy sprawdzić jego własności statystyczne. W przypadku skorzystania z jednego z dobrze zbadanych generatorów można czasem bezpośrednio obliczyć długość cyklu, a pozostałe właściwości (jak np. równomierność rozkładu) są najczęściej znane. Można też skorzystać z jednego ze standardowych testów (test pokerowy, test serii itp).

???Ziarno początkowy zestaw parametrów generatora .

Zastosowanie w kryptografii

Szczególną klasę PRNG stanowią generatory uznane za bezpieczne do zastosowań kryptograficznych. Kryptografia opiera się na generatorach liczb pseudolosowych przede wszystkim w celu tworzenia unikalnych kluczy stałych oraz sesyjnych. Ze względu na fakt, że bezpieczeństwo komunikacji zależy od jakości klucza, od implementacji PRNG stosowanych w takich celach oczekuje się między innymi, że:

- ✓ Generowane wartości będą każdorazowo praktycznie nieprzewidywalne dla osób postronnych (np. przez wykorzystanie odpowiednich źródeł danych przy tworzeniu ziarna).
- ✓ Nie będzie możliwe ustalenie ziarna lub stanu wewnętrznego generatora na podstawie obserwacji dowolnie długiego ciągu wygenerowanych bitów.
- ✓ Znajomość dowolnej liczby wcześniej wygenerowanych bitów nie będzie wystarczała, by odgadnąć dowolny przyszły bit z prawdopodobieństwem istotnie wyższym od \begin{matrix}\frac 1 2\end{matrix}.
- ✓ Dla wszystkich możliwych wartości ziarna, zachowana będzie pewna minimalna, dopasowana do zastosowania długość cyklu PRNG (aby uniknąć ponownego wykorzystania takiego samego klucza).

** Wyrażenie – w językach programowania kombinacja wyrażeń stałych (literałów, stałych itp.), zmiennych, operatorów, funkcji i nawiasów, której przypisywana jest wartość zgodnie z regułami danego języka[1]. Proces przypisywania wartości, nazywany jest wartościowaniem lub ewaluacją; podobnie jak w matematyce, wyrażenie jest reprezentacją otrzymanej wartości.

Wyrażenia mogą (choć nie muszą) mieć skutków ubocznych; ich brak jest jednym z założeń programowania funkcyjnego.

**Dlaczego komputery liczą binarnie

Dwójkowy system liczenia, choć nieintuicyjny dla człowieka, ma trzycechy czyniące go idealnym z punktu widzenia elektroniki cyfrowej informatyki. Są nimi:

1. Łatwość implementacji elektrycznej i elektronicznej,

2. Odporność na zakłócenia,

3. Możliwość interpretacji wartości 0 i 1 jako wartości logicznych „prawda” i „fałsz” (algebra Boole’a).

** Liczba Pi, a razem z nią wiele innych wielkości niewymiernych (choćby pierwiastek z dwóch) ma tę własność, że da się ją obliczyć z dowolną zadaną dokładnością.

** w programie dev C++ co oznacza /

**Iteracja (łac. iteratio – powtarzanie) – czynność powtarzania (najczęściej wielokrotnego) tej samej instrukcji (albo wielu instrukcji) w pętli. Mianem iteracji określa się także operacje wykonywane wewnątrz takiej pętli.

**W programowaniu obiektowym iteratorem nazywamy obiekt pozwalający na sekwencyjny dostęp do wszystkich elementów lub części zawartych w innym obiekcie, zwykle kontenerze lub liście. Iterator jest czasem nazywany kursorem, zwłaszcza w zastosowaniach związanych z bazami danych.

**Rekurencja, zwana także rekursją (ang. recursion, z łac. recurrere, przybiec z powrotem) to w logice, programowaniu i w matematyce odwoływanie się np. funkcji lub definicji do samej siebie.

W logice wnioskowanie rekurencyjne opiera się na założeniu istnienia pewnego stanu początkowego oraz zdania (lub zdań) stanowiącego podstawę wnioskowania (przy czym, aby cały dowód był poprawny, zarówno reguła, jak i stan początkowy muszą być prawdziwe). Istotą rekurencji jest tożsamość dziedziny i przeciwdziedziny reguły wnioskowania, wskutek czego wynik wnioskowania może podlegać tej samej regule zastosowanej ponownie.

Na prostym przykładzie:

reguła: każdy ojciec jest starszy od swojego syna; każdy ojciec jest czyimś synem

stan początkowy: jestem 22-letnim mężczyzną

teza: ojciec ojca mojego ojca jest starszy ode mnie

dowód:

mój ojciec jest starszy ode mnie

mój ojciec jest czyimś synem

ojciec mojego ojca jest starszy od mojego ojca

ojciec mojego ojca jest czyimś synem

itd.

** czy istnieją problemy nierozwiązywalne?

Nawet w dziedzinie problemów jasno określonych (a tylko te są domeną informatyki) istnieją problemy algorytmicznie nierozwiązywalne:

f1) bezwzględnie – gdy nie istnieje (w sensie obiektywnym) algorytm rozwiązujący wszystkie szczególne przypadki danego problemu; lub

f2) praktycznie – gdy dla danego problemu nie istnieje algorytm o dostatecznie niskiej złożoności czasowej.

Problemy nierozwiązywalne bezwzględnie nazywa się nieobliczalnymi.

****

1. Czym są komputery kwantowe?
Tradycyjne pecety działają w systemie binarnym, rozróżniając dwa stany - "tak" albo "nie", włączony lub wyłączony, "0" lub "1". Natomiast w świecie kwantów rządzi kubit, który nie ma ustalonej wartości jak bit w komputerze. Aby wynik obliczeń komputera kwantowego był dokładny, trzeba wykonać całą serię pomiarów - im więcej, tym rezultat dokładniejszy. Kubitami są cząstki elementarne - fotony lub elektrony.

2. Jak szybko liczą komputery kwantowe?
To zależy od ilości kwantowych bitów (kubitów): im więcej, tym szybciej. W ciągu sekund potrafią rozwiązać zadania, które normalnemu komputerowi zajęłyby lata. Jest jedno ale: to dotyczy tylko określonych obliczeń.

3. Czy są już takie komputery?
Dotychczas komputery kwantowe występowały tylko w laboratoriach i instytutach badawczych. Te jednak tylko udowadniają, że technologia działa.

4. Dlaczego tak trudno jest budować komputery kwantowe?
Stany kwantowe są ekstremalnie delikatne, w żadnym razie nie wolno zakłócić nakładania stanów podczas procesu obliczeniowego. Bitom kwantowym zagrażają także inne niebezpieczeństwa: działanie pola elektromagnetycznego, najmniejsze wstrząsy albo minimalne wahania temperatury skutkują błędami obliczeniowymi. Dlatego procesory wymagają chłodzenia do blisko zera absolutnego, a więc do -273 stopni Celsjusza.

Kubit – najmniejsza i niepodzielna jednostka informacji kwantowej.
Z fizycznego punktu widzenia kubit jest to kwantowomechaniczny układ opisany dwuwymiarową przestrzenią Hilberta, w związku z tym, różni się od klasycznego bitu tym, że może znajdować się w dowolnej superpozycji DWÓCH STANÓW KWANTOWYCH. Jako model fizyczny kubitu najczęściej podaje się przykład cząstki o spinie ½, np. elektronu.

Stan kwantowy — informacja o układzie kwantowym pozwalająca przewidzieć prawdopodobieństwa wyników wszystkich pomiarów, jakie można na tym układzie wykonać. Stan kwantowy jest jednym z podstawowych pojęć mechaniki kwantowej.

Mechanika kwantowa (teoria kwantów) – teoria praw ruchu obiektów poszerzająca zakres mechaniki na sytuacje, dla których przewidywania mechaniki klasycznej nie sprawdzały się. Opisuje przede wszystkim świat mikroskopowy – obiekty o bardzo małych masach i rozmiarach, np. atom, cząstki elementarne itp., ale także takie zjawiska makroskopowe jak nadprzewodnictwo i nadciekłość. Jej granicą dla średnich rozmiarów, energii czy pędów zwykle jest mechanika klasyczna.

****

6. Obliczalność i rozstrzygalność

Rekurencja, zwana także rekursją to w logice, programowaniu i w matematyce odwoływanie się np. funkcji lub definicji do samej siebie.

W informatyce, teoria obliczalności to dział teorii obliczeń zajmujący się badaniem jakie problemy są rozwiązywalne przy użyciu komputerów. Nie należy mylić teorii obliczalności z teorią złożoności obliczeniowej, zajmującej się badaniem jak efektywnie da się rozwiązywać różne problemy.

Rozstrzygalność (decydowalność) problemu matematycznego to następująca jego właściwość: istnieje algorytm, który oblicza odpowiedź na dowolne pytanie stawiane przez problem. Problem może być nierozstrzygalny, jeśli jego rozstrzygalność prowadziłaby do powstania paradoksów.

****

Języki naturalne i formalne

- Cechy języka naturalnego - duża swoboda konstruowania zdań (brak ścisłych reguł gramatycznych), duża ilość wyjątków.
- Języki formalne - ścisły i jednoznaczny opis konstrukcji.

Przykłady języków programowania:
Język maszynowy,asembler, kompilator

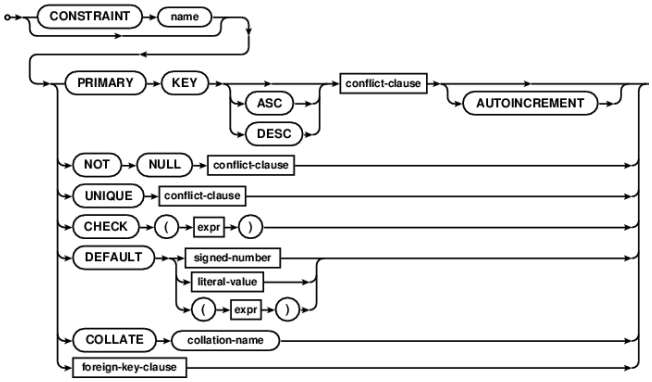
Notacja EBNF= jest sposobem wyrażenia gramatyki bezkontekstowej, czyli opisem języków formalnych. Jest rozszerzeniem notacji BNF.

Notacja BNF jest zestawem reguł produkcji o następującej postaci:
<symbol> ::= <wyrażenie zawierające symbole>
Znaczenie użytych tu symboli jest następujące:
< – lewy ogranicznik symbolu
> – prawy ogranicznik symbolu
Notacja ta jest powszechnie używana w informatyce do zapisu składni (syntaktyki) języków programowania i protokołów komunikacyjnych.

Railroad Diagram Generator
written by Gunther Rademacher grd@gmx.net
v1.39.959 Feb 3, 2015
SAXON XQUERY

RDG - Railroad Diagram Generator
Narzędzie do tworzenia, edycji lub przeglądania juz istniejących diagramów syntaktycznych.

Można w nim wybrać już istniejący ze strony (EBNF Notification) lub ze specyfikacji W3C (np. XML, XPath), a następnie przeglądnąć / zedytować daną gramatykę.



gramatyka bezkontekstowa = gramatyka formalna