

Przypisy

- [1] Aktualny poziom bezpieczeństwa funkcji skrótu (<http://blog.securitystandard.pl/news/342130.html>). Security Standard, 20 marca 2009.
- [2] Trzydziestu kandydatów do SHA-3 (<http://ipsec.pl/kryptografia/2008/trzydziestu-kandydatow-do-sha-3.html>). IPsec.pl, 4 listopada 2009.

Bibliografia

- Kutylowski M., Strothmann W.B., *Kryptografia*, OW Read Me 1999 ISBN 83-7147-092-4
- Sklyarov D., *Łamanie zabezpieczeń programów* OW Read Me 2004 ISBN 83-7243-418-2
- Cormen T. H., Leiserson C. E., Rivest R. L., *Wprowadzenie do algorytmów* WNT 2001 ISBN 83-204-2556-5

Linki zewnętrzne

- The Hash Function Lounge (<http://www.larc.usp.br/~pbarreto/hflounge.html>) ([ang.](#)). Paulo S. L. M. Barreto.
[dostęp 2011-03-06].

Algorytm Knutha-Morrisa-Pratta

Algorytm Knutha-Morrisa-Pratta to algorytm wyszukiwania wzorca w tekście, który wykorzystuje fakt, że w przypadku wystąpienia niezgodności ze wzorcem, sam wzorec zawiera w sobie informację pozwalającą określić gdzie powinna się zacząć kolejna próba dopasowania, pomijając ponowne porównywanie już dopasowanych znaków. Dzięki temu właściwy algorytm działa w czasie liniowym od długości przeszukiwanego tekstu i wzorca (co dla dużych wzorców ma znaczenie).

Algorytm został wynaleziony przez Donalda Knutha i Vaughana Pratta i niezależnie przez J. H. Morrisa w 1977, ale wszyscy trzej opublikowali go wspólnie.

Algorytm KMP

Przykład działania

W tym artykule przyjęto konwencję tablic indeksowanych od zera do przechowywania łańcuchów znaków. Zapis taki jest zgodny ze składnią języka C.

By lepiej zobrazować, jak działa algorytm, przeanalizujmy jego przebieg. Niech dane będą wzorec W i tekst S. W dowolnej chwili stan algorytmu opisują dwie zmienne: m oraz i, które oznaczają odpowiednio pozycję w S, od której rozpoczyna się aktualne częściowe dopasowanie, oraz indeksu w W oznaczającego następny rozpatrywany znak wzorca. Niech w naszym przykładzie wyglądają tak:

```
m: 01234567890123456789012
S: ABC ABCDAB ABCDABCDABDE
W: ABCDABD
i: 0123456
```

Rozpoczynamy porównując kolejne znaki W do "równoległych" znaków S, przemieszczając się dalej, jeśli pasują. Niemniej jednak, w czwartej iteracji, dostajemy w S[3] spację, zamiast W[3]='D' - niezgodność. Ponieważ 'A' nie występuje na pozycjach 1-3, wiemy, że próby dopasowania wzorca nie powiodą się wcześniej niż na czwartej pozycji. Zatem przesuwamy się tam, ustawiając m=4 oraz i=0.

```
m: 01234567890123456789012
S: ABC ABCDAB ABCDABCDABDE
W:      ABCDABD
```

```
i: 0123456
```

Szybko otrzymujemy niemal kompletne dopasowanie "ABCDAB", jednak dla $W[6]$ ($S[10]$) znów mamy niezgodność. Niemniej jednak, przed końcem bieżącego dopasowania częściowego, natrafiliśmy na "AB", które mogłoby być początkiem nowego dopasowania, musimy więc wziąć je pod uwagę. Jak już wiemy litery te odpowiadają dwóm znakom przed bieżącą pozycją, nie musimy ich ponownie sprawdzać; po prostu ustawiamy $m=8$, $i=2$ i kontynuujemy dopasowywanie kolejnego znaku. W ten sposób omijamy nie tylko znaki poprzednio dopasowane w S , ale również te poprzednio dopasowane z W .

```
m: 01234567890123456789012
S: ABC ABCDAB ABCDABCDABDE
W:      ABCDABD
i:      0123456
```

Próba odnalezienia wzorca natychmiast kończy się porażką, trafiamy bowiem na spację. Podobnie jak po pierwszej porażce, wracamy na początek W i zaczynamy szukanie od kolejnego znaku ustawiając $m = 11$ oraz resetujemy $i=0$.

```
m: 01234567890123456789012
S: ABC ABCDAB ABCDABCDABDE
W:      ABCDABD
i:      0123456
```

Jeszcze raz, natychmiast natrafiamy na dopasowanie "ABCDAB", lecz kolejny znak, 'C', nie odpowiada ostatniemu znakowi 'D' słowa W . Rozumując jak poprzednio, ustawiamy $m=15$ oraz (by rozpocząć od dwuznakowego łańcucha "AB") $i=2$, i rozpoczynamy dopasowywanie od bieżącej pozycji.

```
m: 01234567890123456789012
S: ABC ABCDAB ABCDABCDABDE
W:      ABCDABD
i:      0123456
```

Tym razem udało się odnaleźć pełne wystąpienie wzorca; jego pozycja to $S[15]$.

Opis części szukającej algorytmu

Powyższy przykład zawiera wszystkie elementy algorytmu. Chwilowo opiszemy tylko wyszukiwanie, przy założeniu, że dysponujemy tabelą "częstkowego dopasowania" T . Jak stworzyć taką tablicę opisane jest nieco niżej. Tablica ta wskazuje, gdzie musimy szukać początku nowego dopasowania w przypadku, gdy próba odnalezienia wystąpienia wzorca na obecnej pozycji zakończy się niepowodzeniem. Tablicy T używamy w następujący sposób: jeżeli mamy zaczynające się w $S[m]$ dopasowanie, które zawiedzie podczas porównywania $S[m + i]$ z $W[i]$, wtedy następne możliwe dopasowanie rozpocznie się w indeksie $m + i - T[i]$ w S (tj. $T[i]$ jest ilością kroków wstecz, które musimy wykonać, gdy nie istnieje dopasowanie). Dla wygody ustawiamy więc $T[0] = -1$ - jeśli $W[0]$ nie zostanie dopasowane, nie możemy się cofnąć i musimy po prostu sprawdzić następny znak. Należy też zauważyć, że skoro po niepowodzeniu na pozycji $m+i$ kolejne możliwe dopasowanie rozpocznie się w indeksie $m + i - T[i]$, kontynuujemy więc szukanie od $W[T[i]]$. Poniżej znajduje się opis części szukającej algorytmu KMP zapisany w pseudokodzie.

```
algorytm kmp_search:
  wejście:
    tablica znaków, S (przeszukiwany tekst)
    tablica znaków, W (szukane słowo)
  wyjście:
```

```

    liczba całkowita (liczona od zera pozycja w S, na której znaleziono W)
zdefiniowane zmienne:
    liczba całkowita,          m = 0 (początek bieżącego dopasowania w S)
    liczba całkowita,          i = 0 (pozycja bieżącego znaku w W)
    tabela liczb całkowitych,  T (tabela liczona gdzie indziej)
dopóki m + i jest mniejsze niż długość S, wykonuj:
    jeżeli W[i] = S[m + i], niech i = i + 1
    jeżeli i równe jest długości W, zwróć m
    w przeciwnym przypadku,
    niech m = m + i - T[i], oraz jeśli i > 0, niech i = T[i]
    (jeśli dotrzemy tu, tzn., że przeszukaliśmy bezskutecznie cały S)
zwróć długość S

```

Wydajność części szukającej algorytmu

Przy założeniu, że tablica T była już wyliczona, zaamortyzowany koszt algorytmu Knutha-Morrisa-Pratta wynosi $O(k)$, gdzie k jest długością tekstu S. Ponieważ większa część obliczeń wykonywana jest w pętli while, wystarczy, że oszacujemy maksymalną ilość jej przebiegów. Rozpatrzmy najpierw najgorszy przypadek. Łatwiej będzie opisać go na przykładzie. Gdy dla danych

```

S=AAAAAABAC....
W=AAAAAAA

```

algorytm po znalezieniu częściowego dopasowania A^6 (zaczynającego się na pozycji 0) natrafi na B (na pozycji 6), spróbuje odrzucić pierwsze A i rozpocznie z A^5 (na pozycji 1), gdzie spróbuje dopasować A^6 . Wciąż jednak natrafia na B, będzie więc sukcesywnie cofał się krok po kroku aż do A^1 (na pozycji 5). Dopiero potem, gdy i ta próba dopasowania zakończy się porażką, przejdzie do następnego znaku. Taka sytuacja wymaga $O(n)$ operacji, gdy n jest długością wzorca. Gdy pomnożymy n przez k wyjdzie na to, że algorytm KMP wcale nie jest szybszy od naiwnego algorytmu wyszukiwania wzorca. Zauważmy jednak, że algorytm, aby móc cofnąć się o jedno miejsce, musi wcześniej przeczytać przynajmniej jeden znak, zatem sytuacje takie jak ta mogą wystąpić jedynie w k/m przypadkach. Z tej prostej obserwacji wynika, że zamortyzowany koszt działania algorytmu wynosi około $2*k$, a więc mieści się w $O(k)$.

Tablica częściowych dopasowań

Celem tej tabeli jest umożliwienie algorytmowi nie dopasowywania żadnego znaku z S więcej niż raz. Kluczową obserwacją natury liniowego poszukiwania, która na to pozwala, polega na tym, że mając porównany pewien segment głównego ciągu znaków z początkowym fragmentem wzorca, wiemy dokładnie, w których miejscach mogłoby zacząć się nowe potencjalne dopasowanie przed bieżącą pozycją. Inaczej mówiąc, występuje tutaj "wstępne poszukiwanie" samego wzoru i zestawu wszelkich możliwych pozycji do powrotu, które pomijają złe wybory, nie zabierając zasobów.

Chcemy mieć możliwość wyszukania, dla każdej pozycji w W, długości najdłuższego możliwego początkowego segmentu W wskazującego (ale nie zawierającego) tę pozycję, inną niż całe segmenty zaczynające się w $W[0]$, których nie dało się dopasować; to wyznacza nam, jak daleko mamy się cofnąć w znajdowaniu kolejnego wzoru. Zatem $T[i]$ jest dokładnie długością najdłuższego możliwego początkowego segmentu W, który jest także podciągami kończącym się w $W[i-1]$. Zakładamy, że pusty ciąg ma długość 0. Kiedy występuje rozbieżność na samym początku wzoru, to jest to szczególna okoliczność (nie mamy już żadnej możliwości wycofywania się), ustawiamy $T[0] = -1$, jak już wcześniej to przedyskutowaliśmy.

Opracowany przykład algorytmu budowania tabel

Najpierw rozważmy przykład $W = \text{"ABCDABD"}$. Zobaczymy, że zachowuje się w dużym stopniu według schematu głównego wyszukiwania i jest z podobnych powodów wydajny. Ustawiamy $T[1] = 0$. Podobnie $T[2] = 0$. Kontynuując do $T[3]$, zauważymy, że jest krótszy sposób sprawdzenia wszystkich końcowych wzorców: powiedzmy, że odkryliśmy właściwy wzorec początkowy zakończony w $W[2]$ o długości 2 (maksymalna możliwa); wtedy jego pierwszy znak jest właściwym początkowym wzorcem właściwego początkowego wzorca W , więc samego siebie, i kończy się w $W[1]$, który - jak już określiliśmy - nie może wystąpić. Zatem nie potrzebujemy nawet zajmować się wzorcami o długości 2, i jak w poprzednim przypadku zawodzi jedynie ten o długości 1, więc $T[3] = 0$.

Przekazujemy następny w kolejności $W[4]$, 'A'. Ta sama logika pokazuje, że najdłuższy wzorec, który musimy rozważyć ma długość 1 i pomimo że w tym przypadku 'A' zadziała, należy pamiętać, że szukamy segmentów kończących się przed bieżącym znakiem, zatem również $T[4] = 0$.

Rozważając teraz kolejny znak, $W[5]$, którym jest 'B', przećwiczymy następującą logikę: jeśli mielibyśmy znaleźć wzorec zaczynający się przed poprzednim znakiem $W[4]$, kontynuując do bieżącego $W[5]$, wtedy w szczególności on sam zawierałby właściwy segment początkowy kończący się w $W[4]$, zaczynający się przed nim, co zaprzecza faktowi, że już znaleźliśmy to 'A', które jest najwcześniejszym wystąpieniem właściwego segmentu kończącego się w $W[4]$. Zatem nie musimy patrzeć za łańcuch końcowy dla $W[5]$. Faktycznie sprawdzając go, odkryjemy, że jest to 'A', to samo jak w $W[0]$. Zatem $T[5] = 1$.

Ostatecznie zauważamy, że następny znak w kolejnym segmencie, rozpoczynającym się w $W[4] = \text{'A'}$ byłby 'B', i w rzeczy samej jest to także $W[5]$. Dalej, ten sam argument co powyżej pokazuje, że nie musimy zaglądać za $W[4]$, by odnaleźć segment dla $W[6]$, więc to jest to, i otrzymujemy $T[6] = 2$.

W ten sposób otrzymujemy następującą tabelę:

i	0	1	2	3	4	5	6
W[i]	A	B	C	D	A	B	D
T[i]	-1	0	0	0	0	1	2

Opis i pseudokod algorytmu budowania tabel

Powyższy przykład ilustruje ogólną technikę zapełniania tabeli przy najmniejszym wysiłku. Zasadą takiego wyszukiwania jest to, że większość pracy została wykonana podczas docierania do aktualnej pozycji, więc nie trzeba wiele robić aby ją opuścić. Jedyną drobną komplikacją jest to, że logika, która jest poprawna w dalszej części łańcucha błędnie generuje niewłaściwe ciągi znaków na początku. Wymaga to odrobiny kodu inicjującego

algorytm kmp_table:

Dane wejściowe:

tablica znaków, W (słowo, które będzie analizowane)

tablica liczb całkowitych, T (która ma być zapełniona)

Dane wyjściowe:

nic (ale podczas działania zapełniana jest tablica wejściowa)

Zdefiniowane zmienne:

liczba całkowita, $i = 2$ (aktualna pozycja, jaką przetwarzamy w tablicy T)

liczba całkowita, $j = 0$

(liczony od zera indeks tablicy W , w której ma być umieszczona kolejna litera szukanego ciągu znaków)

(pierwszych kilka wartości to stałe, ale inne, niż algorytm mógłby sugerować)

niech $T[0] = -1$, $T[1] = 0$

dopóki i jest mniejsze od długości w , **rób:**

```

(pierwsza opcja: ciąg znaków jest dłuższy)

    jeżeli  $W[i - 1] = W[j]$ , niech  $T[i] = j + 1$ ,  $i = i + 1$ ,  $j = j + 1$ 

(druga opcja: ciąg znaków nie jest dłuższy, ale nie możemy się cofnąć)

w przeciwnym przypadku,

    jeżeli  $j > 0$ , niech  $j = T[j]$ 

(trzeci przypadek: wyczerpuje się zasób kandydatów, Zauważ, że  $j=0$ )

w przeciwnym przypadku,

    niech  $T[i] = 0$ ,  $i = i + 1$ 

```

Złożoność obliczeniowa algorytmu opartego na tablicach jednowymiarowych

Złożoność algorytmu opartego na tablicy wynosi $O(n)$, gdzie n jest długością tablicy W . Poza czynnościami inicjalizacyjnymi, cała praca jest wykonywana w pętli. To wystarczy, aby pokazać że ta pętla wykonuje się w czasie $O(n)$, jednocześnie badając wartości i oraz $i-j$. W pierwszej pętli wartość różnicy $i - j$ jest zachowana, jako że i oraz j są zwiększane jednocześnie, ale naturalnie i jest zwiększane. W drugiej pętli j jest zastępowane przez $T[j]$, które - jak widzieliśmy wcześniej - jest zawsze mniejsze od j , zatem zwiększa się różnica $i - j$. W trzeciej pętli i jest zwiększane, j pozostaje bez zmian, a więc zarówno i oraz $i - j$ wzrasta. Ponieważ $i \geq i - j$, to znaczy, że na każdym etapie albo i albo dolna granica i się zwiększa; zatem, gdy algorytm przerwie się, gdy $i = n$, musi przerwać się po najwyżej $2n$ iteracjach pętli, bo $i - 1$ zaczyna się od 1. Więc złożoność algorytmu to $O(n)$.

Wydajność algorytmu Knutha-Morrisa-Pratta

Skoro algorytm składa się z dwóch części, z czego pierwsza ma złożoność $O(k)$ a druga $O(n)$, to całkowita złożoność jest równa sumie złożoności częściowych, czyli $O(k+n)$.

Jest oczywiste, że w opracowanym przykładzie algorytm będzie miał największą przewagę nad algorytmem naiwnym, bo może naraz opuszczać większą część znaków. To znaczy: im mniej musi się cofać, tym szybciej się wykonuje, co jest odzwierciedlone przez obecność zer w tabeli T . Słowo takie jak "ABCDEFGF" dobrze działa z tym algorytmem, bo nie zawiera powtórzeń swojego początku, więc jego tabelka to po prostu same zera z -1 na początku. Dla kontrastu, $W = "AAAAAA"$ zadziała strasznie, bo jego tabela ma postać:

i	0	1	2	3	4	5	6
$W[i]$	A	A	A	A	A	A	A
$T[i]$	-1	0	1	2	3	4	5

Jest to najgorszy możliwy wzorec dla T , i może zostać wyczerpany przez słowo takie, jak: $S = "AAAAABAAAAABAAAAAA"$, gdzie algorytm będzie próbował dopasować każde 'A' do 'B' przed zakończeniem działania. To skutkuje w maksymalnej liczbie powtórzeń, podwajając liczbę S , gdyż ilość powtórzeń "AAAAAB" wzrasta. Chociaż dla tego słowa metoda tablicowa jest szybka (nie ma tutaj żadnego cofania), to przebiega tylko raz dla danego słowa W , kiedy to procedura poszukiwania potencjalnie może przebiegać wiele razy. Jeżeli za każdym razem wyraz główny będzie przeszukiwany w poszukiwaniu wzorca S , to ogólna wydajność będzie najgorsza z możliwych. Wybierając sposób porównywania, to szczególne połączenie będzie znakomitym przypadkiem do zastosowania algorytmu Boyera-Moore'a w przeszukiwaniu ciągu znaków. Jakkolwiek, algorytm KMP gwarantuje, że poszukiwanie zostanie wykonane w liniowym czasie, zarówno w najlepszym i najgorszym przypadku, podczas gdy algorytm Boyera-Moore'a osiąga najlepszy przypadek wielkim kosztem, w najgorszym czasie.

Linki zewnętrzne

- Demonstracja działania algorytmu KMP ^[1] (pol.)
- Sprawdź swój program na SPOJ-u ^[2] (pol.)
- An explanation of the algorithm ^[3] (ang.)
- Knuth-Morris-Pratt algorithm ^[4] (ang.)

Bibliografia

- Donald Knuth; James H. Morris, Jr, Vaughan Pratt (1977). "Fast pattern matching in strings". SIAM Journal on Computing 6 (2): 323-350.
- Thomas H. Cormen; Charles E. Leiserson, Ronald L. Rivest, Clifford Stein (2001). "Section 32.4: The Knuth-Morris-Pratt algorithm", Introduction to Algorithms, Second edition, MIT Press and McGraw-Hill, 923-931. ISBN 0-262-03293-7.

Przypisy

- [1] <http://megrez.mizar.org/dydaktyka/algorytmy/KMP>
 [2] <http://pl.spoj.pl/problems/KMP/>
 [3] <http://www.ics.uci.edu/~eppstein/161/960227.html>
 [4] <http://www-igm.univ-mlv.fr/~lecroq/string/node8.html>

Algorytm Karpa-Rabina

Algorytm Karpa-Rabina jest algorytmem dopasowania wzorca - służy do lokalizowania w tekście określonego podciągu.

Dany jest wzorec $x[1 \dots m]$ (złożony z m znaków) oraz przeszukiwany ciąg $y[1 \dots n]$ (złożony z n znaków; $n > m$). Problem dopasowania wzorca polega na znalezieniu takiego indeksu i , dla którego $y[i \dots i + m - 1] = x[1 \dots m]$.

W algorytmie Karpa-Rabina wykorzystuje się funkcję mieszającą h . Przeglądane są wszystkie podciągi $y_i := y[i \dots i + m - 1]$ dla $i \in [1 \dots n - m]$, ale bezpośrednie porównania podciągu y_i z x (które jest bardzo kosztowne) ma miejsce tylko wtedy, gdy $h(y_i) = h(x)$.

O efektywności algorytmu decyduje konstrukcja funkcji mieszającej h - powinna istnieć funkcja N która **niskim kosztem** wyznacza $h(y_{i+1})$ na podstawie znanej już wartości $h(y_i)$. W praktyce traktuje się tekst, jako liczbę zapisaną w systemie o określonej podstawie (najczęściej biorąc jako wartości cyfr kody ASCII znaków).

Oczekiwana złożoność obliczeniowa jest rzędu $O(m + n)$.

Pseudokod (dane x, y, n, m):

```

 $Sx \leftarrow h(x[1 \dots m])$ 
 $Sy \leftarrow h(y[1 \dots m])$ 
for  $i \leftarrow 1$  to  $n - m$  do
  begin
    if  $Sx = Sy$  then
      begin
        porównaj ciąg  $x[1 \dots m]$  z podciągiem  $y[i \dots i + m - 1]$ 
        if wynik porównania prawdziwy then zwróć indeks  $i$ 
      end
    end
  end

```

$$Sy \leftarrow N(Sy, y[i], y[i + m])$$

end

Zastosowanie

Jednym z najprostszych praktycznych zastosowań algorytmu Rabina-Karpa jest wykrywanie plagiatu. Powiedzmy, na przykład, że student pisze wypracowanie na temat Pana Tadeusza. Profesor mógłby wyszukać opracowania na ten sam temat i automatycznie porównać je z zawartością wypracowania. Za pomocą algorytmu Rabina-Karpa można wykazać, z którego opracowania zostało skopiowane dane zdanie. Aby zapobiec oszukaniu systemu poprzez niewielkie przeróbki tekstu algorytm może zostać ustawiony tak, aby ignorował detale, takie jak znaki przestankowe, poprzez ich uprzednie usunięcie lub wprowadzenie marginesu błędu przy porównywaniu skrótu przeszukiwanego tekstu ze wzorcem (ponieważ podobne ciągi znaków mają podobne skróty). Ponieważ liczba ciągów przez nas poszukiwanych jest znaczna algorytmy pojedynczego wyszukiwania byłyby niepraktyczne.

Różne algorytmy poszukiwania ruchomych podciągów

Podstawowym zadaniem algorytmu jest znalezienie podciągu długości m , nazywanego wzorem, wbudowanego w tekst o długość n ; na przykład znajdując wyraz "zegar" w zdaniu "Zegarmistrz wziął za naprawę 20 złotych." Najprostszy algorytm wykonujący to zadanie po prostu wyszukuje podciągi na wszystkie możliwe sposoby.

```
1 function NaiveSearch(string s[1..n], string sub[1..m])
2     for i from 1 to n
3         for j from 1 to m
4             if s[i+j-1] ≠ sub[j]
5                 jump to next iteration of outer loop
6     return i
7     return not found
```

Taka praca algorytmu jest dobra w wielu praktycznych przypadkach, ale w pewnych przykładach, takich jak szukanie tekstu składającego się z „b” i 10,000 „a” w tekście z 10 milionów „a”, owocuje wystąpieniem pesymistycznego czasu przebiegu równego $\Theta(mn)$.

Algorytm Knutha-Pratta-Morrisa zmniejsza ten czas do $\Theta(n)$ za pomocą prekumputacji, czyli badania każdego znaku w tekście tylko raz. Algorytm skoku Boyer-Moore’a przesuwają wskaźnik nie o 1 miejsce, ale tak dużo jak tylko to możliwe, ponieważ wskaźnik przesuwają się o tyle miejsc ile znalazło elementów zgodnych z wzorem licząc od początku wzoru, skutecznie zmniejszając liczbę powtórzeń zewnętrznej pętli, w najlepszym przypadku liczba ta może być rzędu n/m . Algorytm Rabina-Karpa skupia się zamiast tego przyspieszeniu działania linii kodu od 3 do 6, co będzie rozważane w następnych paragrafach.

Użycie funkcji skrótu do zmiennego przeszukiwania podciągu

Zamiast dążyć do bardziej wyrafinowanego przeszukiwania, algorytm Rabina-Karpa przyspiesza porównywanie szukanego wzorca do podciągów w tekście za pomocą funkcji skrótu. Funkcja skrótu przerabia każdy ciąg liter na postać numeryczną nazywaną skrótem (ang. *hash*). Algorytm Rabina-Karpa wykorzystuje fakt, że jeżeli dwa teksty są równe, ich skróty też są równe. Implikacja odwrotna nie zachodzi jednak – dla dwóch różnych tekstów mogą się zdarzyć dwa równe skróty. Trzeba zatem porównać także teksty, jeśli wartości skróty są równe:

```
1 function RabinKarp(string s[1..n], string sub[1..m])
2     hsub := hash(sub[1..m])
3     hs := hash(s[1..n])
4     for i from 1 to n
```

```

5         if hs = hsub
6             if s[i..i+m-1] = sub
7                 return i
8         hs := hash(s[i+1..i+m])
9     return not found

```

Linie 2, 3, 6 i 8 wymagają $\Omega(m)$ czasu każda. Jednakże linie 2 i 3 są wykonane tylko raz a linia 6 jest uruchamiana tylko gdy wartości skrótu są równe. Linia 5 jest wykonywana n razy, ale wymaga stałego czasu. Więc jedynym punktem mogącym negatywnie wpłynąć na złożoność algorytmu jest linia 8.

Jeśli wyliczymy w sposób naiwny wartość skrótu dla podciągu $s[i+1..i+m]$, zajmie to $\Omega(m)$ czasu a ponieważ linia ta jest wykonywana przy każdym obiegu pętli, cały algorytm będzie miał złożoność czasową $\Omega(mn)$ czyli tyle ile najmniej wydajne algorytmy naiwne. Sztuczka polega na wyliczeniu aktualnego skrótu na podstawie skrótu wyliczonego w poprzednim obiegu pętli. Do tego służy algorytm postępującego skrótu.

Najprostsza wersja działa według takiej zasady: wyznacza się wartość każdego znaku w podciągu, następnie wylicza się skróty według wzoru

$$hs[i+1 \dots i+m] = hs[i \dots i+m-1] - s[i] + s[i+m]$$

Ta prosta funkcja skrótu ma niestety niezbyt równomierny rozkład, co powoduje, że linia 6 jest wykonywana o wiele częściej niż w innych bardziej wyrafinowanych algorytmach postępującego skrótu.

Funkcje skrótu będące w użyciu

Kluczem wydajności algorytmu Robina-Karpa jest efektywne wyliczanie wartości skrótu kolejnych podciągów tekstu. Jeden z wydajnych algorytmów postępującego skrótu traktuje podciągi jako numery o pewnej bazie. Bazą zazwyczaj jest jakaś duża liczba pierwsza. Np: dla podciągu „hi” wartość skrótu wynosi $104 * 101^1 + 105 * 101^0 = 10609$ (kod ASCII dla h wynosi 104 a dla i – 105). Główną zaletą takiego sposobu hashowania jest możliwość wyznaczenia wartości skrótu kolejnego podciągu na podstawie podciągu poprzedniego poprzez wykonanie stałej liczby operacji niezależnej od długości tego podciągu.

Wielowzorcowe wyszukiwanie Rabina-Karpa

Algorytm Robina-Karpa jest gorszy w wyszukiwaniu pojedynczego wzorca od algorytmów takich jak Knutha-Morrisa-Pratta czy Boyer’a-Moore’a, ponieważ jest najwolniejszy w przypadku pesymistycznego ułożenia ciągu. Jednak nie ma sobie równych przy wyszukiwaniu wielowzorcowym. Jeżeli chcemy wyszukać w tekście wielokrotnie powtarzający się wzorec, wystarczy stworzyć prostą wersję algorytmu Rabina-Karpa używającą filtru Blooma do sprawdzania czy skróty sprawdzanego podciągu jest równy skrótom podciągów przez nas poszukiwanemu.

```

function RabinKarpSet(string s[1..n], set of string subs, m) {
    set hsubs := emptySet
    for each sub in subs
        insert hash(sub[1..m]) into hsubs
    hs := hash(s[1..m])
    for i from 1 to n
        if hs ∈ hsubs
            if s[i..i+m-1] = a substring with hash hs
                return i
            hs := hash(s[i+1..i+m])
    return not found
}

```


W przykładzie tym zakładamy, że wszystkie podciągi mają narzuconą długość m jednak można to wyeliminować. Po prostu porównujemy obecną wartość skrótu z wartościami skrótów wszystkich podciągów jednocześnie, a następnie porównujemy znalezione pary z wszystkimi podciągami o danej wartości skrótu.

Inne algorytmy przy wyszukiwaniu pojedynczego wzorca mają złożoność czasową $O(n)$ a dla wielokrotnie (k -razy) powtarzającego się wzorca $O(n*k)$, w przeciwieństwie do algorytmu Rabina-Karpa, który wyszuka k wzorów w czasie $O(n+k)$ ponieważ sprawdzenie czy skrót podciągu jest równy skróтови wzorca zabiera $O(1)$ czasu.

Linki zewnętrzne

- Richard M. Karp, Michael O. Rabin. *Efficient randomized pattern-matching algorithms* ^[1]. „IBM Journal of Research and Development”. 31 (2), marzec 1987.
- Karp-Rabin algorithm ^[2] na stronie EXACT STRING MATCHING ALGORITHMS ^[3] (ang.)

Przypisy

- [1] <http://portal.acm.org/citation.cfm?id=1012171>
 [2] <http://www-igm.univ-mlv.fr/~lecroq/string/node5.html>
 [3] <http://www-igm.univ-mlv.fr/~lecroq/string/index.html>

Algorytm Boyera i Moore'a

Algorytm Boyera i Moore'a – algorytm poszukiwania wzorca w tekście. Polega na porównywaniu, zaczynając od ostatniego elementu wzorca.

Zalety:

- jeżeli okaże się, że znak, który aktualnie sprawdzamy, nie należy do wzorca możemy przeskoczyć w analizie tekstu o całą długość wzorca
- z reguły skoki wzorca są większe od 1 (można porównać z KMP)

Przykłady

tekst: WIKIPEDIA, WOLNA ENCYKLOPEDIA
 wzorzec: CYKL

```

WIKIPEDIA, WOLNA ENCYKLOPEDIA
  |   |   |   |   |   |
CYKL  |   |   |   |   |
      CYKL  |   |   |   |
          CYKL  |   |   |
              CYKL  |   |
                  CYKL  |
                      CYKL
  
```

Pierwsze 4 porównania trafiają na litery: i, i, w, a, które nie występują we wzorcu, możemy więc za każdym razem skoczyć do przodu o całą długość wzorca, a więc 4 znaki. Kolejne porównanie trafia jednak na literę 'c', która występuje we wzorcu. Należy wówczas przesunąć wzorzec tak, by litery te nałożyły się na siebie. W tym przypadku okazuje się, że natrafiliśmy na szukane słowo.

tekst: ALGORYTMY I STRUKTURY DANYCH
 wzorzec: TUR

ALGORYTMY I STRUKTURY DANYCH

```

| | | | | |
TUR | | | | |
  TUR | | | |
    TUR | | |
      TUR | |
        TUR |
          TUR
            TUR

```

Pierwsze 4 porównania trafiają na litery: g, y, y, 'spacja', które nie występują we wzorcu, możemy więc skoczyć o całą długość wzorca do przodu. Przy 5 porównaniu litera, którą sprawdzamy, znajduje się we wzorcu. W tym przypadku nie przesuwamy wzorca do przodu, ponieważ litery już się pokrywają. Sprawdzamy kolejne litery. Okazuje się jednak, że już się nie pokrywają, więc znów możemy skoczyć o całą długość wzorca. Natrafiamy na literę 't', która znajduje się we wzorcu. Przesuwamy wzorzec tak by litery pokrywały się. Tym razem znaleźliśmy szukane słowo.

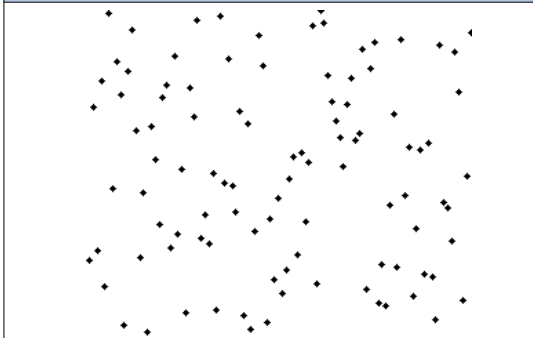
Linki zewnętrzne

- Artykuł na temat algorytmu (en) ^[1]

Przypisy

[1] <http://www.cs.utexas.edu/~moore/publications/fstrpos.pdf>

Sortowanie bąbelkowe

Sortowanie bąbelkowe	
 <p>Przykład działania algorytmu sortowania bąbelkowego.</p>	
Rodzaj	Sortowanie
Struktura danych	Tablica, lista
Złożoność	
Czasowa	$O(n^2)$
Pamięciowa	$O(1)$

Sortowanie bąbelkowe (ang. *bubble sort*) - prosta metoda sortowania o złożoności czasowej $O(n^2)$ i pamięciowej $O(1)$.

Polega na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności, jeżeli zaburza ona porządek, w jakim się sortuje tablicę. Sortowanie kończy się, gdy podczas kolejnego przejścia nie dokonano żadnej zmiany.

Dowód matematyczny

Algorytm opiera się na zasadzie maksimum, tj. każda liczba jest mniejsza lub równa od liczby maksymalnej. Porównując kolejno liczby można wyznaczyć największą z nich. Następnie ciąg częściowo posortowany (mający liczbę maksymalną), można skrócić o tę liczbę i ponowić szukanie maksimum, już bez elementów odrzuconych i tak długo, aż zostanie nam jeden element. Otrzymane kolejne maksima są coraz mniejsze przez co ciąg jest uporządkowany.

Złożoność obliczeniowa

Algorytm wykonuje n przejść a każdym przejściu wykonuje $n-1$ porównań. Przez co jego teoretyczna złożoność czasowa wynosi $O(n^2)$. Niestety w podstawowej wersji algorytmu nie można tego czasu polepszyć. Mało tego, każda permutacja powoduje, że algorytm jest wykonywany w czasie pesymistycznym.

Modyfikacje powodujące ulepszenie czasu

Algorytm można rozbudować tak, by czas optymistyczny był lepszy. Najłatwiejsze jest dodanie flagi informującej, czy w danej iteracji doszło do zmiany. Flaga jest zerowana na wejściu w przebiegu pętli, w przypadku natrafienia na zmianę jest podnoszona, a po wykonaniu przejścia sprawdzana. Jeśli nie było zmian, to sortowanie jest zakończone. Modyfikacja ta wprawdzie wydłuża czas wykonania jednego przejścia przez pętlę (gdyż trzeba wyzerować flagę, podnieść ją i sprawdzić), jednakże w wariantcie optymistycznym (ciąg częściowo posortowany) może zaoszczędzić iteracji, przez co algorytm będzie działał szybciej.

Przykład działania

Ciąg wejściowy $[4, 2, 5, 1, 7]$. Każdy wiersz symbolizuje wypchnięcie kolejnego największego elementu na koniec ("wypłynięcie największego bąbelka"). Niebieskim kolorem oznaczono końcówkę ciągu już posortowanego.

$$\begin{array}{l}
 \underbrace{[4, 2, 5, 1, 7]}_{4 > 2} \rightarrow \underbrace{[2, 4, 5, 1, 7]}_{4 < 5} \rightarrow \underbrace{[2, 4, 5, 1, 7]}_{5 > 1} \rightarrow \underbrace{[2, 4, 1, 5, 7]}_{5 < 7} \\
 \underbrace{[2, 4, 1, 5, 7]}_{2 < 4} \rightarrow \underbrace{[2, 4, 1, 5, 7]}_{4 > 1} \rightarrow \underbrace{[2, 1, 4, 5, 7]}_{4 < 5} \\
 \underbrace{[2, 1, 4, 5, 7]}_{2 > 1} \rightarrow \underbrace{[1, 2, 4, 5, 7]}_{2 < 4} \\
 \underbrace{[1, 2, 4, 5, 7]}_{1 < 2}
 \end{array}$$

Pseudokod


Pseudokod wersji podstawowej algorytmu dla tablicy o rozmiarze "n" (elementy tablicy są numerowane od 0 do n-1):

```

procedure bubbleSort( A : lista elementów do posortowania )
  n = liczba_elementów(A)
  do
    for (i = 0; i < n-1; i++) do:
      if A[i] > A[i+1] then
        swap(A[i], A[i+1])
      end if
    end for
    n = n-1
  while n > 1
end procedure

```

Implementacja

 Zobacz przykłady implementacji tego algorytmu na stronie Wikiźródła

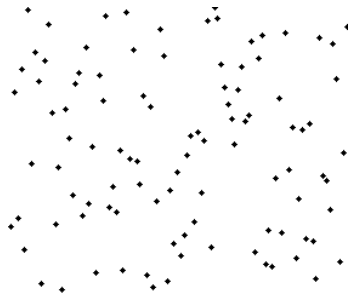
Linki zewnętrzne

- Algorytm przedstawiony z wykorzystaniem węgierskiego tańca ^[1]

Przypisy

[1] <http://www.youtube.com/watch?v=lyZQPjUT5B4>

Sortowanie przez wstawianie

Sortowanie przez wstawianie	
 <p>Przykład działania sortowania przez wstawianie.</p>	
Rodzaj	Sortowanie
Struktura danych	Tablica, lista
Złożoność	
Czasowa	$O(n^2)$

Sortowanie przez wstawianie (ang. *Insert Sort*, *Insertion Sort*) - jeden z najprostszych algorytmów sortowania, którego zasada działania odzwierciedla sposób w jaki ludzie ustawiają karty - kolejne elementy wejściowe są ustawiane na odpowiednie miejsca docelowe. Jest efektywny dla niewielkiej liczby elementów, jego złożoność wynosi $O(n^2)$ ^[1]. Pomimo tego, że jest znacznie mniej wydajny od algorytmów takich jak quicksort czy heapsort, posiada pewne zalety:

- jest wydajny dla danych wstępnie posortowanych
- jest wydajny dla zbiorów o niewielkiej liczebności
- jest stabilny

Schemat działania algorytmu

1. Utwórz zbiór elementów posortowanych i przenieś do niego dowolny element ze zbioru nieposortowanego.
2. Weź dowolny element ze zbioru nieposortowanego.
3. Wyciągnięty element porównuj z kolejnymi elementami zbioru posortowanego póki nie napotkasz elementu równego lub elementu większego (jeśli chcemy otrzymać ciąg niemalejący) lub nie znajdziemy się na początku/końcu zbioru uporządkowanego.
4. Wyciągnięty element wstaw w miejsce gdzie skończyłeś porównywać.
5. Jeśli zbiór elementów nieuporządkowanych jest niepusty wróć do punkt 2.

Algorytm - pseudokod

Poniższy kod przedstawia algorytm zapisany w pseudokodzie, gdzie^[1]:

- A - tablica danych, przeznaczonych do posortowania (indeksowana od 1 do n)
- n - liczba elementów w tablicy A

```
<syntaxhighlight lang="cpp"> 0. Insert_sort(A, n)
1. for i=2 to n : 2. klucz = A[i]
3 > Wstaw A[i] w posortowany
ciąg A[1 ... i-1]
4. j = i - 1
5. while j>0 and A[j]>klucz:
6. A[j + 1] = A[j]
7. j = j - 1
8. A[j+1] = klucz
</syntaxhighlight>
```

Analiza algorytmu

Złożoność

Niech $\delta(i)$ oznacza liczba sprawdzeń warunku pętli (4.) w i -tym przebiegu pętli (1.). Wtedy:

- instrukcja 1. jest wykonana n razy
- instrukcja 2. jest wykonana $n-1$ razy
- instrukcja 3. jest wykonana $n-1$ razy
- instrukcja 4. jest wykonana $\sum_{i=1}^n \delta(i)$ razy
- instrukcja 5. jest wykonana $\sum_{i=1}^n \delta(i) - 1$ razy
- instrukcja 6. jest wykonana $\sum_{i=1}^n \delta(i) - 1$ razy
- instrukcja 7. jest wykonana $n-1$ razy

Jeżeli przez c_1, c_2, \dots, c_7 oznaczmy czas pojedynczego wywołania instrukcji 1, 2, ... 7, to całkowity czas wykonania algorytmu sortowania przez wstawianie będzie równy:

$$T(n) = c_1 n + (c_2 + c_3 + c_7)(n-1) + \sum_{i=1}^n \delta(i) c_4 + \left(\sum_{i=1}^n \delta(i) - 1 \right) c_5 + \left(\sum_{i=1}^n \delta(i) - 1 \right) c_6$$

Przypadek optymistyczny

Gdy tablica danych wejściowych jest uporządkowana rosnąco, to w każdym przebiegu pętli (1.) przy pierwszym sprawdzeniu warunków pętli (4.) zachodzi $A[j] < \text{klucz}$ - pętla (4.) nie zostaje wykonana ani razu, a więc liczba sprawdzeń warunku pętli (4.) $\delta(i)$ wynosi 1 dla wszystkich j od 0 do n . Całkowity czas $T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$ wykonania algorytmu jest więc wyrażony funkcją liniową, z czego wynika, że dla posortowanych tablic o długości n algorytm insert sort działa w czasie $O(n)$.

Przypadek pesymistyczny

Gdy tablica danych wejściowych jest uporządkowana malejąco, w każdym przebiegu pętli (1.) przy sprawdzaniu warunków pętli (4.) zachodzi $A[j] > \text{klucz}$ - pętla (4.) w każdym i -tym przebiegu pętli (1.) jest wywoływana $i-1$ razy. Po podstawieniu do wzoru na $T(n)$: $\delta(i) = i$, otrzymuje się kwadratową złożoność czasową $T(n) \in O(n^2)$.

Przypadek średni

Przy założeniu, że wszystkie możliwe wartości tablicy A występują z jednakowym prawdopodobieństwem, można dowieść, że oczekiwana liczba elementów $A[0..i]$ większych od wstawianego klucza wynosi $i/2$. W tym wypadku podstawienie $\delta(i) = i/2$ daje podobnie jak w przypadku pesymistycznym złożoność $T(n) \in O(n^2)$.

Poprawność

Prawidłowość algorytmu sortowania przez wstawianie można dowieść, udowadniając poniższe twierdzenie:

w i -tym wykonaniu pętli for tablica $A[0..i-1]$ składa się z posortowanych elementów znajdujących się w pierwotnej tablicy na pozycjach $[0..i-1]$.

Dowód:

W pierwszej iteracji, dla $i=1$ zdanie jest prawdziwe (rozważamy posortowany ciąg $A[0..0]$).

W i -tej iteracji, jeżeli elementy $A[0..i-2]$ są posortowane, to nowy klucz zostanie wstawiony na pozycję, w której nie będzie tworzył inwersji z żadnym z elementów w zbiorze do którego zostanie dołączony, a więc zbiór $A[0..i-1]$ będzie posortowany.

W ostatniej iteracji pętli ostatni wolny klucz zostaje wstawiony w posortowany ciąg $A[0..n-2]$ w wyniku czego powstaje posortowana tablica $A[0..n-1]$

Bezpośredni wniosek z powyższego twierdzenia: algorytm sortowania przez wstawianie dla każdej tablicy A dokona jej prawidłowego posortowania.

Przykłady implementacji

 Zobacz przykładowe implementacje sortowania przez wstawianie na Wikiźródłach

Przypisy

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: *Wprowadzenie do algorytmów*. WNT, 2007.

Linki zewnętrzne

- Algorytm przedstawiony z wykorzystaniem węgierskiego tańca (<http://www.youtube.com/watch?v=ROaIU379I3U>)

Sortowanie przez scalanie

W informatyce **sortowanie przez scalanie** (ang. *merge sort*), to rekurencyjny algorytm sortowania danych, mający zastosowanie przy danych dostępnych sekwencyjnie (po kolei, jeden element na raz), na przykład w postaci listy jednokierunkowej (tj. łączonej jednostronnie) albo pliku sekwencyjnego. Odkrycie algorytmu przypisuje się Johnowi von Neumannowi.

Algorytm

Algorytm ten jest dobrym przykładem algorytmów typu Dziel i zwyciężaj (ang. *divide and conquer*), których ideą działania jest podział problemu na mniejsze części, których rozwiązanie jest już łatwiejsze.

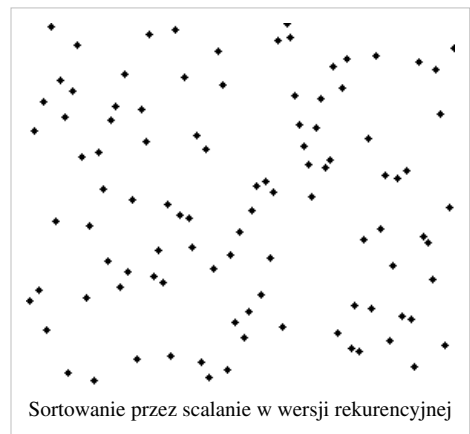
Wyróżnić można trzy podstawowe kroki:

- Podziel zestaw danych na dwie, równe części (w przypadku nieparzystej liczby wyrazów jedna część będzie o 1 wyraz dłuższa);
- Zastosuj sortowanie przez scalanie dla każdej z nich oddzielnie, chyba że pozostał już tylko jeden element;
- Połącz posortowane podciągi w jeden.

Procedura scalania dwóch ciągów $A[1..n]$ i $B[1..m]$ do ciągu $C[1..m+n]$:

- Utwórz wskaźniki na początki ciągów A i $B \rightarrow i=1, j=1$
- Jeżeli ciąg A wyczerpany ($i > n$), dołącz pozostałe elementy ciągu B do C i zakończ pracę.
- Jeżeli ciąg B wyczerpany ($j > m$), dołącz pozostałe elementy ciągu A do C i zakończ pracę.
- Jeżeli $A[i] \leq B[j]$ dołącz $A[i]$ do C i zwiększ i o jeden, w przeciwnym przypadku dołącz $B[j]$ do C i zwiększ j o jeden
- Powtarzaj od kroku 2 aż wszystkie wyrazy A i B trafią do C

Scalanie wymaga $O(n+m)$ operacji porównań elementów i wstawienia ich do tablicy wynikowej.



Sortowanie przez scalanie w wersji rekurencyjnej

Złożoność czasowa

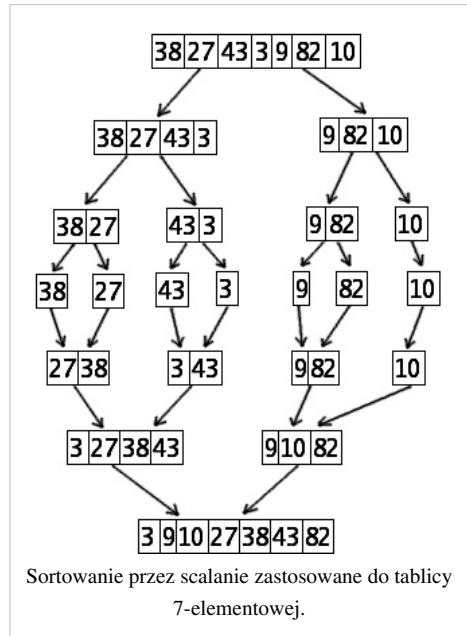
Bez straty ogólności założmy, że długość ciągu, który mamy posortować jest potęgą liczby 2 (patrz Złożoność obliczeniowa)

Obrazek obok przedstawia drzewo rekursji wywołania algorytmu mergesort.

Mamy więc drzewo o głębokości $\log_2 n$, na każdym poziomie dokonujemy scalenia o łącznym koszcie $n \times c$, gdzie c jest stałą zależną od komputera. A więc intuicyjnie, tzn. nieformalnie możemy dowieść, że złożoność algorytmu mergesort to $\log_2 n \times n$

Formalnie złożoność czasową sortowania przez scalanie możemy przedstawić następująco:

$$T(1) = O(1)$$



$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

Ciągi jednoelementowe możemy posortować w czasie stałym, czas sortowania ciągu n -elementowego to scalenie dwóch ciągów $\frac{n}{2}$ -elementowych, czyli $O(n)$, plus czas potrzebny na posortowanie dwóch o połowę krótszych ciągów.

Mamy:

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + n = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n \\ &= 2\left(2\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + \frac{n}{2}\right) + n \\ &= 2\left(2\left(\dots 2\left(T\left(\frac{n}{2^{2^i}}\right) + \frac{n}{2^i}\right) + \dots\right) + \frac{n}{2}\right) + n \\ &= 2\left(2\left(\dots 2\left(T(1) + 2\right)\dots\right) + \frac{n}{2}\right) + n \end{aligned}$$

gdzie $n = 2^k$

Po rozwinięciu nawiasów otrzymamy:

$$T(n) = 2n \log n$$

A więc asymptotyczny czas sortowania przez scalanie wynosi $O(n \log n)$ (zobacz: notacja dużego O).

Dowód poprawności algorytmu

Dowód przez indukcję względem długości n tablicy elementów do posortowania.

1) $n=2$

Algorytm podzieli dane wejściowe na dwie części, po czym zastosuje dla nich scalanie do posortowanej tablicy

2) Zał.: dla ciągów długości k , $k < n$ algorytm mergesort prawidłowo sortuje owe ciągi.

Dla ciągu długości n algorytm podzieli ten ciąg na dwa ciągi długości $n/2$. Na mocy założenia indukcyjnego ciągi te zostaną prawidłowo podzielone i scalone do dwóch posortowanych ciągów długości $n/2$. Ciągi te zostaną natomiast scalone przez procedurę scalającą do jednego, posortowanego ciągu długości n .

Pseudokod

Struktura tablica jest tablicą, której elementy mogą być zmieniane, argumenty start, koniec są całkowitoliczbowe.

```
procedure merge(tablica, start, środek, koniec);

var tab_pom : array [0..koniec-start] of integer;
    i, j, k   : integer;

begin

    i := start;
    k := 0;
    j := środek + 1;

    while (i <= środek) and (j <= koniec)
    begin
        if tablica[j] < tablica[i] then
            begin
                tab_pom[k] := tab[j];
                j := j + 1;
            end
        else
            begin
                tab_pom[k] := tab[i];
                i := i + 1;
            end;
        k := k + 1;
    end;


    if (i <= środek)
    while (i <= środek)
    begin
        tab_pom[k] := tab[i];
        i := i + 1;
        k := k + 1;
    end
    else
    while (j <= koniec)
    begin
        tab_pom[k] := tab[j];
        j := j + 1;
        k := k + 1;
    end;

    for i:= 0 to koniec-start do
        tab[start + i] := tab_pom[i];

    end;
```

```
procedure merge_sort(tablica, start, koniec);  
  
var środek : integer;  
  
begin  
  
    if start <> koniec then  
    begin  
        środek := (start + koniec) div 2;  
        merge_sort(tablica, start, środek);  
        merge_sort(tablica, środek + 1, koniec);  
        merge      (tablica, start, środek, koniec);  
    end;  
  
end;
```

Implementacja

 Zobacz przykłady implementacji tego algorytmu na stronie Wikiźródeł

Wersja nierekurencyjna

Podstawową wersję algorytmu sortowania przez scalanie można uprościć. Pomysł polega na odwróceniu procesu scalania serii. Ciąg danych możemy wstępnie podzielić na n serii długości 1, scalić je tak, by otrzymać $\frac{n}{2}$ serii długości 2, scalić je otrzymując $\frac{n}{4}$, serii długości 4...

Złożoność obliczeniowa jest taka sama jak w przypadku klasycznym, tu jednak nie korzystamy z rekursji, a więc zaoszczędzamy czas i pamięć potrzebną na jej obsłużenie.

Linki zewnętrzne

- Przyspieszony MergeSort ^[1]
- Algorytm przedstawiony z wykorzystaniem tańca ^[2]

Przypisy

[1] <http://kicia.ift.uni.wroc.pl/algorytmy/mergesortpaper.pdf>

[2] http://www.youtube.com/watch?v=XaqR3G_NVoo

Sortowanie przez zliczanie

Sortowanie przez zliczanie – metoda sortowania danych, która polega na sprawdzeniu ile wystąpień kluczy mniejszych od danego występuje w sortowanej tablicy.

Algorytm zakłada, że klucze elementów należą do skończonego zbioru (np. są to liczby całkowite z przedziału 0..100), co ogranicza możliwości jego zastosowania.

Zalety i wady

Główną zaletą tej metody jest liniowa złożoność obliczeniowa algorytmu – $O(n+k)$ (n – oznacza liczebność zbioru, k – rozpiętość danych, czyli w przypadku liczb: powiększoną o 1 różnicę między maksymalną, a minimalną wartością, np. rozpiętość liczb w Dużym Lotku wynosi $(49-1) + 1 = 49$).

Największymi ograniczeniami algorytmu są konieczność uprzedniej znajomości zakresu danych i złożoność pamięciowa (wymaga dodatkowo $O(k)$ lub $O(n+k)$ pamięci).

Implementacje

Istnieją dwie implementacje algorytmu:

- prostsza – sortująca *in situ* (w miejscu), zakłada, że elementy o równych kluczach są nierozróżnialne, nie mogą zatem być to klucze danych (każdy z nich jest bowiem powiązany z przenoszoną wartością – zatem, mimo iż są one równe, muszą pozostawać rozróżnialne);
- standardowa – gwarantuje stabilność i nie wymaga dodatkowego założenia. Potrzebuje natomiast $O(n)$ więcej pamięci;

Przykładowa implementacja w języku C++

Wersja ta sortuje n -elementową tablicę liczb całkowitych.

```
const int k = 77; // elementami tablicy T są liczby całkowite z
                  // z przedziału 0..76

const int n = 1000;
int T[n]; // tablica zawierająca elementy do posortowania
int Tp[n]; // tablica zawierająca elementy posortowane
int TPom[k]; // zawiera liczbę elementów o danej wartości

int i; // zmienna pomocnicza

for(i = 0 ; i < k ; ++i)
    TPom[i] = 0; // zerowanie tablicy

for(i = 0 ; i < n ; ++i)
    ++TPom[T[i]]; // po tych operacjach TPom[i] będzie
// zawierała
// liczbę wystąpień elementów o
kluczach mniejszych od i
for(i = 1 ; i < k ; ++i)
    TPom[i] += TPom[i-1]; // teraz TPom[i] zawiera pozycje w
posortowanej
```

```
// tablicy ostatniego elementu o kluczu
i
for(i = n-1 ; i >= 0 ; --i)
    Tp[--TPom[T[i]]] = T[i]; // wstawienie elementu na odpowiednią
    pozycję
                                // i aktualizacja TPom
```

Linki zewnętrzne

- Aplet w języku Java demonstrujący działanie algorytmu ^[1]

Przypisy

[1] <http://users.cs.cf.ac.uk/C.L.Mumford/tristan/CountingSort.html>

Sortowanie kubełkowe

Sortowanie kubełkowe (ang. *bucket sort*) – jeden z algorytmów sortowania. Jest on najczęściej stosowany, gdy liczby w zadanym przedziale są rozłożone jednostajnie, ma on wówczas złożoność $\Theta(n)$. W przypadku ogólnym pesymistyczna złożoność obliczeniowa tego algorytmu wynosi $O(n^2)$.

Pomysł takiego sortowania podali po raz pierwszy w roku 1956 E. J. Issac i R. C. Singleton.

Sposób działania

1. Podziel zadany przedział liczb na n podprzedziałów (*kubełków*) o równej długości.
2. Przypisz liczby z sortowanej tablicy do odpowiednich kubełków.
3. Sortuj liczby w niepustych kubełkach.
4. Wypisz po kolei zawartość niepustych kubełków.

Zazwyczaj przyjmuje się, że sortowane liczby należą do przedziału od 0 do 1. Jeśli tak nie jest, to można podzielić każdą z nich, przez największą możliwą (jeśli znany jest przedział) lub wyznaczoną. Należy tu jednak zwrócić uwagę, że wyznaczanie największej możliwej liczby w tablicy m -elementowej ma złożoność obliczeniową $O(m)$.

Pseudokod

```
function bucket-sort(array, n) is
    buckets ← new array of n empty lists
    for i = 0 to (length(array)-1) do
        insert array[i] into buckets[msbits(array[i], k)]
    for i = 0 to n - 1 do
        next-sort(buckets[i])
    return the concatenation of buckets[0], ..., buckets[n-1]
```

Literatura

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, "Wprowadzenie do algorytmów", WNT 2001

Sortowanie pozycyjne

Sortowanie pozycyjne (ang. *radix sort*) to algorytm sortowania porządkujący stabilnie ciągi wartości (liczb, słów) względem konkretnych cyfr, znaków itp, kolejno od najmniej znaczących do najbardziej znaczących pozycji. Złożoność obliczeniowa jest równa $O(d(n+k))$, gdzie k to liczba różnych cyfr, a d liczba cyfr w kluczach.

Wymaga $O(n+k)$ dodatkowej pamięci.

Przewagą sortowania pozycyjnego nad innymi metodami jest fakt, iż nie wykonuje ono żadnych operacji porównania na danych wejściowych. Załóżmy że mamy dużą liczbę bardzo długich liczb, bardzo do siebie podobnych – w tym sensie, że większość z nich ma takie same cyfry na początkowych pozycjach. Nie jest łatwo powiedzieć która jest większa, gdyż za każdym razem musimy porównać dużo cyfr zanim trafimy na różnicę. Czas porównania takich liczb jest zatem proporcjonalny do ich długości. Gdybyśmy do posortowania tych liczb zastosowali algorytm porównujący liczby, np. sortowanie szybkie, otrzymalibyśmy dla niego złożoność $O(dn \log n)$ gdzie d to liczba cyfr w liczbach.

Algorytmy pozycyjne sprawdzają się także w roli algorytmów sortujących listy.

Implementacja w pseudojęzyku programowania

- `tab[]` – tablica ciągów (cyfr, liter itp.) gdzie pozycja 1 oznacza najbardziej znaczącą pozycję ciągu
- `d` – długość ciągów

```
procedure RadixSort (tab[], d)
begin
  for i:=d downto 1 do
    posortuj stabilnie ciągi według i-tej pozycji;
end;
```

Dowód poprawności algorytmu sortowania pozycyjnego

Założmy, że przed i -tym przebiegiem pętli `for`, wszystkie ciągi są posortowane według $(i-1)$ tej cyfry/litery. Po kolejnej iteracji ciągi będą posortowane według i -tej. Jeżeli dla dwóch, lub więcej ciągów, ich i -ta cyfra/litera jest taka sama, stabilność sortowania zapewni nam zachowanie dobrego porządku. Po ostatnim przebiegu pętli `for` ciągi będą uporządkowane według najbardziej znaczących cyfr, oraz kolejnych w przypadku identyczności na ostatnich pozycjach.

Powyższy algorytm zakłada, że ciągi są tej samej długości. W przypadku gdy tak nie jest, możemy uzupełnić ciągi do tej samej długości zerami z lewej strony (dla liczb) lub zerowymi znakami z prawej (dla napisów). Jeżeli ciągów długich jest niewiele, metoda ta jest nieefektywna, jednak istnieją modyfikacje oryginalnego algorytmu działające ściśle w czasie liniowym względem rozmiaru danych.

Przykład działania algorytmu sortowania pozycyjnego

[^] oznacza aktualną pozycję.

523	472	523	266
266	523	349	349
783	--> 783	--> 266	--> 472

472	266	472	523
349	349	783	783
^	^	^	^

Sortowanie biblioteczne

Sortowanie biblioteczne (ang. *Library sort*) – algorytm sortowania, który bazuje na algorytmie sortowania przez wstawianie, ale z dodawaniem pustych miejsc w tablicy w celu przyspieszenia wstawiania elementów.

Nazwa wywodzi się z następującej analogii:

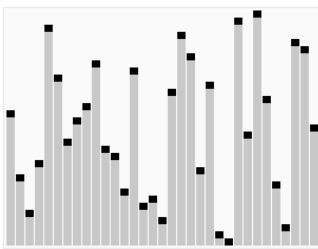
Wyobraźmy sobie bibliotekarza, który układa książki alfabetycznie na półkach. Zaczynając od książek na literę A ustawia jedną przy drugiej, aż dojdzie do litery Z. Jeśli bibliotekarz postanowi dodać nową książkę na półkę, której nazwa zaczyna się na literę B, to będzie musiał przesunąć wszystkie książki od miejsca, gdzie pasuje nowa książka aż do ostatniej książki. Ponieważ litera B występuje blisko początku alfabety, więc praktycznie całą półkę książek należy przesunąć. W celu przyspieszenia pracy bibliotekarz powinien zostawiać wolne miejsce po każdej książce i wówczas może wstawić jedną nową książkę na półkę bez przesuwania. Na tym polega algorytm Sortowania bibliotecznego. Wikipedia:Weryfikowalność

Algorytm zaproponowali: Michael A. Bender, Martín Farach-Colton oraz Miguel Mosteiro w 2004 roku^[1]. Podobnie jak sortowanie przez wstawianie, na którym bazuje, sortowanie biblioteczne jest algorytmem stabilnym (czas wykonania w najgorszym przypadku nie odbiega znacząco od średniego czasu wykonania). Wykazano, że jego złożoność czasowa wynosi najczęściej $O(n \log n)$ (podobnie jak algorytmu quicksort), rzadziej $O(n^2)$ (jak przy sortowaniu przez wstawianie). Mankamentem algorytmu jest zwiększone zapotrzebowanie na pamięć.

Przypisy

- [1] M.A. Bender, M. Farach-Colton, M. Mosteiro *"Insertion Sort is $O(n \log n)$ "* (<http://www.cs.auckland.ac.nz/~mcw/Teaching/refs/sorting/library-sort.pdf>)

Sortowanie szybkie

Sortowanie szybkie	
	
Rodzaj	Sortowanie
Struktura danych	Różne
Złożoność	
Czasowa	$O(n \log n)$
Pamięciowa	zależnie od implementacji

Sortowanie szybkie (ang. *quicksort*) – jeden z popularnych algorytmów sortowania działających na zasadzie "dziel i zwyciężaj".

Sortowanie QuickSort zostało wynalezione w 1962 przez C.A.R. Hoare'a^[1].

Zasada

Algorytm działa rekursywnie - wybierany jest pewien element tablicy, tzw. element osiowy, po czym na początek tablicy przenoszone są wszystkie elementy mniejsze od niego, na koniec wszystkie większe, a w powstałe między tymi obszarami puste miejsca trafia wybrany element. Potem sortuje się osobno początkową i końcową część tablicy. Rekursja kończy się, gdy kolejny fragment uzyskany z podziału zawiera pojedynczy element, jako że jednoelementowa podtablica nie wymaga sortowania.

Jeśli przez **l** oznacza się indeks pierwszego, a przez **r** – ostatniego elementu sortowanego fragmentu tablicy, zaś przez **i** – indeks elementu, na którym tablica została podzielona, to procedurę sortowania można (w dużym uproszczeniu) przedstawić następującym pseudokodem:

```

PROCEDURE Quicksort(l, r)
BEGIN
  IF l < r THEN                                { jeśli fragment dłuższy niż 1 element }
    BEGIN
      i = PodzielTablice(l, r); { podziel i zapamiętaj punkt
podziału }
      Quicksort(l, i-1);        { posortuj lewą część }
      Quicksort(i, r);          { posortuj prawą część }
    END
  END

```

Algorytm **sortowania szybkiego** jest bardzo wydajny: jego **średnia** złożoność obliczeniowa jest rzędu $O(n \cdot \log n)$ (zob. hasło Notacja dużego O). Jego szybkość i prostota implementacji powodują, że jest powszechnie używany; jego implementacje znajdują się również w standardowych bibliotekach języków programowania - na przykład w bibliotece standardowej języka C (funkcja `qsort`), w implementacji klasy `TList` w Delphi, jako procedura standardowa w PHP itp.

Złożoność

Algorytm ten dość dobrze działa w praktyce, ale ma bardzo złą pesymistyczną złożoność.

Przypadek optymistyczny

W przypadku optymistycznym, jeśli mamy szczęście za każdym razem wybrać medianę z sortowanego fragmentu tablicy, to liczba porównań niezbędnych do uporządkowania n -elementowej tablicy opisana jest rekurencyjnym wzorem

$$T(n) = (n - 1) + 2T\left(\frac{n - 1}{2}\right)$$

Dla dużych n :

$$T(n) \approx n + 2T\left(\frac{n}{2}\right)$$

co daje w rozwiązaniu liczbę porównań (a więc wskaźnik złożoności czasowej):

$$T(n) \approx n \log_2 n$$

Równocześnie otrzymuje się minimalne zagnieżdżenie rekursji (czyli głębokość stosu, a co za tym idzie, złożoność pamięciową):

$$M(n) \approx \log_2 n$$

Przypadek przeciętny

W przypadku przeciętnym, to jest dla równomiernego rozkładu prawdopodobieństwa wyboru elementu z tablicy:

$$T(n) \approx 2n \ln n \approx 1.39n \log_2 n$$

złożoność jest zaledwie o 39% wyższa, niż w przypadku optymistycznym.

Przypadek pesymistyczny

W przypadku pesymistycznym, jeśli zawsze wybierzemy element najmniejszy (albo największy) w sortowanym fragmencie tablicy, to:

$$T(n) = n - 1 + T(n - 1)$$

skąd wynika kwadratowa złożoność czasowa:

$$T(n) = \frac{n^2 - n}{2} \approx \frac{n^2}{2}$$

W tym przypadku otrzymuje się też olbrzymią, liniową złożoność pamięciową:

$$M(n) = n - 1$$

Usprawnienia algorytmu

Wybór elementu

Najprostsza, "naiwna" metoda podziału – wybieranie zawsze skrajnego elementu tablicy – dla danych już uporządkowanych daje katastrofalną złożoność $O(n^2)$. Trywialne na pozór zadanie posortowania posortowanej tablicy okazuje się dla tak zapisanego algorytmu zadaniem skrajnie trudnym. Aby uchronić się przed takim przypadkiem stosuje się najczęściej randomizację wyboru albo wybór "środkowy z trzech". Pierwszy sposób opiera się na losowaniu elementu osiowego, co sprowadza prawdopodobieństwo zajścia najgorszego przypadku do wartości zanedbywalnie małych. Drugi sposób polega na wstępnym wyborze trzech elementów z rozpatrywanego fragmentu tablicy, i użyciu jako elementu osiowego tego z trzech, którego wartość leży pomiędzy wartościami pozostałych

dwu. Można również uzupełnić algorytm o poszukiwanie przybliżonej mediany (patrz poniżej: Gwarancja złożoności).

Ograniczenie rekursji

Wysoka wydajność algorytmu sortowania szybkiego predestynuje go do przetwarzania dużych tablic. Takie zastosowanie wymaga jednak zwrócenia szczególnej uwagi na głębokość rekursji. Głębokość rekursji wiąże się bowiem z wykorzystaniem stosu maszynowego.

W najgorszym przypadku, jeśli algorytm będzie dzielił tablicę zawsze na część jednoelementową i resztę, to rekursja osiągnie głębokość $n-1$. Aby temu zapobiec, należy sprawdzać, która część jest krótsza – i tę porządkować najpierw. Z dłuższą zaś nie wchodzić w rekursję, lecz ponownie dzielić na tym samym poziomie wywołania:

```

PROCEDURE Quicksort( l, r )
BEGIN
  WHILE l < r DO                                { dopóki fragment dłuższy niż 1 element }
  BEGIN
    i := PodzielTablice( l, r );
    IF (i-l) ≤ (r-i) THEN                          { sprawdź, czy lewa część krótsza }
    BEGIN                                           { TAK? }
      Quicksort( l, i-1 );                          { posortuj lewą, krótszą część }
      l := i+1                                       { i kontynuuj dzielenie dłuższej }
    END
  ELSE
    BEGIN                                           { NIE }
      Quicksort( i, r );                            { posortuj prawą, krótszą część }
      r := i-1                                       { i kontynuuj dzielenie dłuższej }
    END
  END
END

```

Przy takiej organizacji pracy na stosie zawsze pozostają zapamiętane (w zmiennych **i**, **r** albo **l**, **i**) indeksy ograniczające dłuższą, jeszcze nie posortowaną część tablicy, a wywołanie rekurencyjne zajmuje się częścią krótszą. To znaczy, że na każdym poziomie wywołań algorytm obsługuje fragment będący co najwyżej połową fragmentu z poprzedniego poziomu. Stąd wynika, że poziomów wywołań nie będzie więcej, niż $\log_2 n$, gdzie n oznacza długość całej tablicy. Zatem usprawnienie to zmienia asymptotyczne wykorzystanie pamięci tego algorytmu z $O(n)$ do $O(\log_2 n)$.

Quicksort w miejscu

Istnieje modyfikacja czyniąca algorytm quicksort działającym w miejscu. W oryginalnym sortowaniu szybkim używa się rekursji lub stosu (*de facto* oba sposoby niewiele się różnią – rekursja w uproszczeniu jest niejawnym stosem) do zapamiętywania miejsc podziału. Więc chociaż algorytm w obu wersjach nie korzysta z dodatkowych tablic o rozmiarze zależnym od rozmiaru danych wejściowych, to nie można nazywać go działającym w miejscu, gdyż wysokość, a więc i wymagania pamięciowe wywołań rekursji/stosu są ściśle zależne od rozmiaru danych początkowych.

Załóżmy, że dla sortowanej tablicy A funkcja $PodzielTablice(l,p)$ zwróciła wartość s . W oryginalnym algorytmie quicksort powinno zostać wykonane wywołania $Quicksort(l,s-1)$ i $Quicksort(s+1,p)$. Zamiast tego "zajmujemy się" tylko $Quicksort(l,s-1)$, a pozycje $s+1$ i p zapamiętujemy w następujący sposób:

- $s+1$: jest jednoznacznie wyznaczona przez koniec ciągu $(l,s-1) \rightarrow (s+1) = (s-1) + 2$

- p : znajdujemy maksimum ciągu $(s+1, p)$ i zamieniamy tę pozycję z pozycją s . Aby odtworzyć pozycję p , wystarczy przeszukiwać ciąg w prawo do znalezienia elementu większego od wartości stojącej na pozycji s . Wtedy indeks elementu o najmniejszej wartości większej od $A[s]$ to $p+1$.

Metoda ta sprowadza koszt pamięciowy algorytmu do wartości stałej, $O(1)$, wymaga jednak dodatkowego nakładu czasu na wyszukiwanie maksymalnych elementów kolejno sortowanych fragmentów tablicy. Ujmuje więc algorytmowi jego główną zaletę, wpisaną nawet w jego nazwę – szybkość działania.

Drobne fragmenty

U podstaw kolejnego usprawnienia leży spostrzeżenie, że około połowa wszystkich rekurencyjnych wywołań procedury dotyczy jednoelementowych fragmentów tablicy – a więc fragmentów z definicji posortowanych. Co więcej, po wliczeniu dodatkowej pracy potrzebnej na wybór elementu dzielącego i zorganizowanie pętli dzielącej tablicę okazuje się, że sortowanie tablic nawet kilkuelementowych algorytmem **quicksort** jest bardziej pracochłonne, niż jakimś algorytmem prostym, na przykład przez wstawianie.

Warto więc zaniechać dalszych podziałów, gdy uzyskane fragmenty staną się dostatecznie krótkie – rzędu kilku lub kilkunastu elementów. Otrzymuje się w wyniku tablicę "prawie posortowaną", w której większość elementów może nie znajduje się na właściwych miejscach, ale są *blisko* właściwych miejsc. Taką tablicę ostatecznie sortuje się algorytmem wstawiania, który bardzo efektywnie radzi sobie z tego rodzaju danymi. Jak pokazuje praktyka, wybór granicznej długości fragmentu (progu "odcięcia" rekursji, ang. *cutoff*) nie wymaga szczególnych rygorów – algorytm działa niemal równie dobrze dla wartości od 5 do 25. W większości zastosowań pozwala to osiągnąć oszczędność łącznego czasu wykonania rzędu 20% ^[2].

Gwarancja złożoności


Pomimo wszelkich usprawnień, pozostaje jednak, zazwyczaj znikome, prawdopodobieństwo zajścia przypadku pesymistycznego, w którym złożoność czasowa wynosi $O(n^2)$. Jeśli chcemy mieć pewność wykonania sortowania w czasie nie dłuższym niż $O(n \log_2 n)$, należy uzupełnić algorytm o poszukiwanie przybliżonej mediany, czyli elementu dzielącego posortowaną tablicę na tyle dobrze, że pesymistyczne oszacowanie złożoności zrówna się z optymistycznym.

Jeżeli prawdopodobieństwo wystąpienia przypadku pesymistycznego w praktyce jest duże, to można skorzystać ze specjalnych algorytmów znajdowania dobrej mediany. Niestety algorytmy te mają dość dużą złożoność, dlatego w takiej sytuacji należy też rozważyć skorzystanie z innych algorytmów sortowania, takich jak np. sortowanie stogowe (ang. *heap sort*), sortowanie pozycyjne (*radix sort*), czy sortowanie przez scalanie (*mergesort*).

W większości praktycznych zastosowań algorytm sortowania szybkiego jest bezkonkurencyjny. W praktyce pozostaje on zdecydowanie najczęściej używanym algorytmem sortowania. Opracowano też wiele modyfikacji i usprawnień tego algorytmu, poprawiając niektóre właściwości, lub dostosowując go do konkretnych wymagań.

Wadą sortowania szybkiego jest brak stabilności. Z tego powodu w niektórych zastosowaniach jest używany mergesort.

Przykładowe implementacje

 *Zobacz przykładowe implementacje sortowania szybkiego na Wikiźródłach*

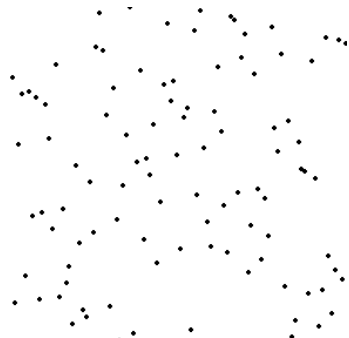
Przypisy

- [1] C.A.R. Hoare: *Quicksort*. Computer Journal, Vol. 5, 1, 10-15 (1962)
- [2] Jon Bentley: *Perłki oprogramowania*. tłum. Agata Tomaszewska. Wyd. 2. Warszawa: WNT, 2001, ss. 151,277, seria: Klasyka Informatyki. ISBN 83-204-2627-8.

Linki zewnętrzne

- Algorytm przedstawiony z wykorzystaniem węgierskiego tańca (<http://www.youtube.com/watch?v=ywWBy6J5gz8>)

Sortowanie przez wybieranie

Sortowanie przez wybieranie	
 <p>Przykład działania sortowania przez wybieranie.</p>	
Rodzaj	Sortowanie
Struktura danych	Tablica, lista
Złożoność	
Czasowa	$O(n^2)$

Sortowanie przez wybieranie - jedna z prostszych metod sortowania o złożoności $O(n^2)$. Polega na wyszukaniu elementu mającego się znaleźć na zadanej pozycji i zamianie miejscami z tym, który jest tam obecnie. Operacja jest wykonywana dla wszystkich indeksów sortowanej tablicy.

Algorytm przedstawia się następująco:

1. wyszukaj minimalną wartość z tablicy spośród elementów od $i+1$ do końca tablicy
2. zamień wartość minimalną, z elementem na pozycji i

Gdy zamiast wartości minimalnej wybierana będzie maksymalna, wówczas tablica będzie posortowana od największego do najmniejszego elementu.

Algorytm jest niestabilny. Przykładowa lista to: $[2a, 2b, 1] \rightarrow [1, 2b, 2a]$ (gdzie $2b=2a$)

Przykład

Posortowana zostanie tablica 8-elementowa $[9, 1, 6, 8, 4, 3, 2, 0]$. W tablicy pogrubione zostaną te elementy wśród których wyszukuje się wartość minimalną.

nr iteracji (wartość i)	tablica	minimum
0	[9, 1, 6, 8, 4, 3, 2, 0]	0
1	[0, 1, 6, 8, 4, 3, 2, 9]	1 (element znajduje się na właściwej pozycji)
2	[0, 1, 6, 8, 4, 3, 2, 9]	2
3	[0, 1, 2, 8, 4, 3, 6, 9]	3
4	[0, 1, 2, 3, 8, 6, 9]	4 (...)
5	[0, 1, 2, 3, 4, 8, 6, 9]	6
6	[0, 1, 2, 3, 4, 6, 8, 9]	8 (...)

Algorytm można nieco przyspieszyć, gdy tablica jest wypełniana z obu końców, tj. wyszukiwane jest równocześnie minimum i maksimum.

Implementacja

Sortowanie przez wybieranie w C++:

- przez wyszukiwanie największego składnika:

```
int Max_element_indeks(int n)
{
    int max = 0;
    for (int i = 1; i < n; i++)
        if (t[i] > t[max])
            max = i;
    return max;
}

void Selection_sort(int n)
{
    for (int i = n; i >= 2; i--)
    {
        int max = Max_element_indeks(i);
        if (max != i - 1)
            swap(t[i - 1], t[max]);
    }
}
```

```
template<typename It>
void selection_sort(It begin, It end)
{
    for (; begin != end; ++begin)
        std::iter_swap(begin, std::min_element(begin, end));
}
```

- przez wyszukiwanie najmniejszego składnika:

```
#include <cstdlib>
#include <iostream>

using namespace std;

void selection_sort(int n, int t[]);

int main(void)
{
    int tab[20];
    srand(time(NULL));
    for(int i=0; i<20; i++) {
        tab[i] = rand()%100;
        cout << tab[i] << " ";
    }
    cout << endl;
    selection_sort(20, tab);
}
```

```
    for(int i=0; i<20; i++) cout << tab[i] << " ";
    cout << endl;
    return 0;
}

void selection_sort(int n, int t[])
{
    int i, j, k;
    for(i=0; i<n; i++) {
        k=i;
        for(j=i+1; j<n; j++) if(t[j]<t[k]) k=j;
        swap(t[k], t[i]);
    }
}
```

Sortowanie przez wybieranie w ruby

```
#!/usr/bin/ruby

# sortowanie przez wybor

def wsort(list)
    for i in 0...(list.size - 1)
        min = i
        for j in (i+1)...(list.size)
            if list[j] <= list[min]
                min = j
            end
        list[i], list[min] = list[min], list[i]
        end
    end
    return list
end

list = []
puts "podaj dane do posortowania CTRL-D - koniec"
while line = $stdin.gets
    list << line.to_i
end
puts "Dane posortowane"
puts wsort(list)
```

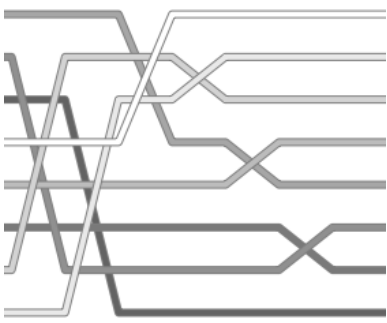
Linki zewnętrzne

- Algorytm przedstawiony z wykorzystaniem cygańskiego tańca ^[1]

Przypisy

[1] <http://www.youtube.com/watch?v=Ns4TPTC8whw>

Sortowanie Shella

Sortowanie Shella	
	
Przykład działania sortowania Shella z odstępami 5, 3, 1	
Rodzaj	Sortowanie
Struktura danych	Tablica
Złożoność	
Czasowa	zależy od ciągu odstępów
Pamięciowa	$O(1)$

Sortowanie Shella (ang. *Shellsort*) – algorytm sortowania działający w miejscu i korzystający z porównań elementów. Stanowi uogólnienie sortowania przez wstawianie, dopuszczające porównania i zamiany elementów położonych daleko od siebie. Jego pierwszą wersję opublikował w 1959 roku Donald Shell^[1].

Złożoność czasowa sortowania Shella w dużej mierze zależy od użytego w nim ciągu odstępów. Wyznaczenie jej dla wielu stosowanych w praktyce wariantów tego algorytmu pozostaje problemem otwartym.

Opis algorytmu

Sortowanie Shella to algorytm wieloprzebiegowy. Kolejne przebiegi polegają na sortowaniu przez proste wstawianie elementów oddalonych o ustaloną liczbę miejsc h , czyli tak zwanym h -sortowaniu.

Poniżej zilustrowano sortowanie przykładowej tablicy metodą Shella z odstępami 5, 3, 1.

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}
dane wejściowe:	62	83	18	53	07	17	95	86	47	69	25	28
po 5-sortowaniu:	17	28	18	47	07	25	83	86	53	69	62	95
po 3-sortowaniu:	17	07	18	47	28	25	69	62	53	83	86	95
po 1-sortowaniu:	07	17	18	25	28	47	53	62	69	83	86	95

Pierwszy przebieg, czyli 5-sortowanie, sortuje osobno przez wstawianie zawartość każdego z fragmentów (a_1, a_6, a_{11}) , (a_2, a_7, a_{12}) , (a_3, a_8) , (a_4, a_9) , (a_5, a_{10}) . Na przykład fragment (a_1, a_6, a_{11}) zmienia z (62, 17, 25) na (17, 25, 62).

Następny przebieg, czyli 3-sortowanie, sortuje przez wstawianie zawartość fragmentów (a_1, a_4, a_7, a_{10}) , (a_2, a_5, a_8, a_{11}) , (a_3, a_6, a_9, a_{12}) .

Ostatni przebieg, czyli 1-sortowanie, to zwykle sortowanie przez wstawianie całej tablicy (a_1, \dots, a_{12}) .

Jak widać, fragmenty tablicy, na których operuje algorytm Shella, są z początku krótkie, a pod koniec dłuższe, ale prawie uporządkowane. W obu tych przypadkach sortowanie przez proste wstawianie działa wydajnie.

Sortowanie Shella nie jest stabilne, czyli może nie zachowywać wejściowej kolejności elementów o równych kluczach. Wykazuje ono zachowanie naturalne, czyli krótszy czas sortowania dla częściowo uporządkowanych danych wejściowych.

Ciągi odstępów

Każdy ciąg odstępów zakończony jedynką prowadzi do poprawnie sortującego algorytmu. Własności tak otrzymanych wersji algorytmu mogą być jednak bardzo różne.

W poniższej tabeli zestawiono większość dotychczas opublikowanych propozycji ciągów odstępów. Niektóre z tych ciągów mają malejące wyrazy zależne od N , czyli rozmiaru sortowanej tablicy. Inne to rosnące ciągi nieskończone, z których należy użyć w odwrotnej kolejności wyrazów mniejszych od N .

Wyraz ogólny ciągu ($k \geq 1$)	Konkretne odstęp	Rząd złożoności pesymistycznej	Autor i rok publikacji
$\lfloor N/2^k \rfloor$	$\lfloor \frac{N}{2} \rfloor, \lfloor \frac{N}{4} \rfloor, \dots, 1$	$\Theta(N^2)$ [gdy $N=2^p$]	Shell, 1959 ^[1]
$2\lfloor N/2^{k+1} \rfloor + 1$	$2\lfloor \frac{N}{4} \rfloor + 1, \dots, 3, 1$	$\Theta(N^{3/2})$	Frank, Lazarus, 1960 ^[2]
$2^k - 1$	$1, 3, 7, 15, 31, 63, \dots$	$\Theta(N^{3/2})$	Hibbard, 1963 ^[3]
$2^k + 1$, na początku 1	$1, 3, 5, 9, 17, 33, 65, \dots$	$\Theta(N^{3/2})$	Papiernow, Stasiewicz, 1965 ^[4]
kolejne liczby postaci $2^p 3^q$	$1, 2, 3, 4, 6, 8, 9, 12, \dots$	$\Theta(N \log^2 N)$	Pratt, 1971 ^[5]
$(3^k - 1)/2$, nie większe niż $\lceil N/3 \rceil$	$1, 4, 13, 40, 121, \dots$	$\Theta(N^{3/2})$	Knuth, 1973 ^[6]
$\prod_{\substack{0 \leq q < r \\ q \neq (r^2+r)/2-k}} a_q$, gdzie $r = \lfloor \sqrt{2k} + \sqrt{2k} \rfloor$, $a_q = \min\{n \in \mathbb{N} : n \geq (5/2)^{q+1},$ $\forall p: 0 \leq p < q \Rightarrow \gcd(a_p, n) = 1\}$	$1, 3, 7, 21, 48, 112, \dots$	$O(e^{\sqrt{8 \ln(5/2) \ln N}})$	Incerpi, Sedgewick, 1985 ^[7]
$4^k + 3 \cdot 2^{k-1} + 1$, na początku 1	$1, 8, 23, 77, 281, \dots$	$O(N^{4/3})$	Sedgewick, 1986 ^[8]
$9(4^{k-1} - 2^{k-1}) + 1, 4^{k+1} - 6 \cdot 2^k + 1$	$1, 5, 19, 41, 109, \dots$	$O(N^{4/3})$	Sedgewick, 1986 ^[8]
$h_k = \max\{\lfloor 5h_{k-1}/11 \rfloor, 1\}, h_0 = N$	$\lfloor \frac{5N}{11} \rfloor, \lfloor \frac{5}{11} \lfloor \frac{5N}{11} \rfloor \rfloor, \dots, 1$?	Gonnet, Baeza-Yates, 1991 ^[9]
$\left\lceil \frac{9^k - 4^k}{5 \cdot 4^{k-1}} \right\rceil$	$1, 4, 9, 20, 46, 103, \dots$?	Tokuda, 1992 ^[10]
nieznany	$1, 4, 10, 23, 57, 132, 301, 701$?	Ciura, 2001 ^[11]

Jeśli N jest potęgą dwójki, to sortowanie z oryginalnym ciągiem odstępów zaproponowanym przez Shella wykonuje w najgorszym przypadku $\Theta(N^2)$ porównań. Przypadek ten zachodzi na przykład wtedy, gdy elementy większe i mniejsze od mediany zajmują odpowiednio parzyste i nieparzyste pozycje tablicy, ponieważ są one porównywane dopiero w ostatnim przebiegu.

Wersja zaproponowana przez Pratta ma wprawdzie wyższą złożoność niż optymalne dla algorytmów sortowania opartych na porównaniach $O(M \log N)$, ale za to prowadzi do sieci sortującej o liczbie komparatorów tego samego rzędu, co sieć Batchera.

Zauważono, że średnio najmniej porównań elementów potrzeba, gdy ilorazy kolejnych odstępów leżą mniej więcej pomiędzy 2,2 a 2,3. Dlatego ciągi Gonnetta i Baezy-Yatesa o ilorazie 2,2 i Tokudy o ilorazie 2,25 sprawdzają się w

praktyce. Nie wiadomo jednak, dlaczego minimum przypada właśnie w tym miejscu. Zalecane jest też stosowanie odstępów o niskich największych wspólnych dzielnikach lub zgoła parami względnie pierwszych.

Pod względem średniej liczby porównań elementów najlepsze znane ciągi odstępów to ciąg (1, 4, 10, 23, 57, 132, 301, 701) i podobne, o wyrazach znalezionych doświadczalnie. Dalsze wyrazy optymalnych ciągów pozostają nieznane. Do dobrych wyników prowadzi przedłużenie ich zgodnie ze wzorem rekurencyjnym $h_k = \lfloor 2,25h_{k-1} \rfloor$.

Do zastosowań praktycznych można też polecić ciąg Tokudy, określony prostymi wzorami $h_k = \lceil h'_k \rceil$, gdzie $h'_k = 2,25h'_{k-1} + 1$, $h'_1 = 1$.

Złożoność obliczeniowa

Zachodzi intrygująca własność: po h_2 -sortowaniu dowolnej h_1 -posortowanej tablicy pozostaje ona nadal h_1 -posortowana^[12]. Każda h_1 -posortowana i h_2 -posortowana tablica jest też $(a_1h_1+a_2h_2)$ -posortowana dla wszystkich całkowitych nieujemnych a_1 i a_2 . Zatem złożoność pesymistyczna sortowania Shella wiąże się z problemem Frobeniusa: dla danych całkowitych h_1, \dots, h_n o nwd = 1 liczba Frobeniusa $g(h_1, \dots, h_n)$ to największa liczba całkowita, której nie da się przedstawić w postaci $a_1h_1 + \dots + a_nh_n$ przy a_1, \dots, a_n całkowitych nieujemnych. Korzystając ze wzorów na liczby Frobeniusa, potrafimy wyznaczać złożoność pesymistyczną dla kilku klas ciągów odstępów. Dowiedzione przypadki zamieszczono w powyższej tabeli.

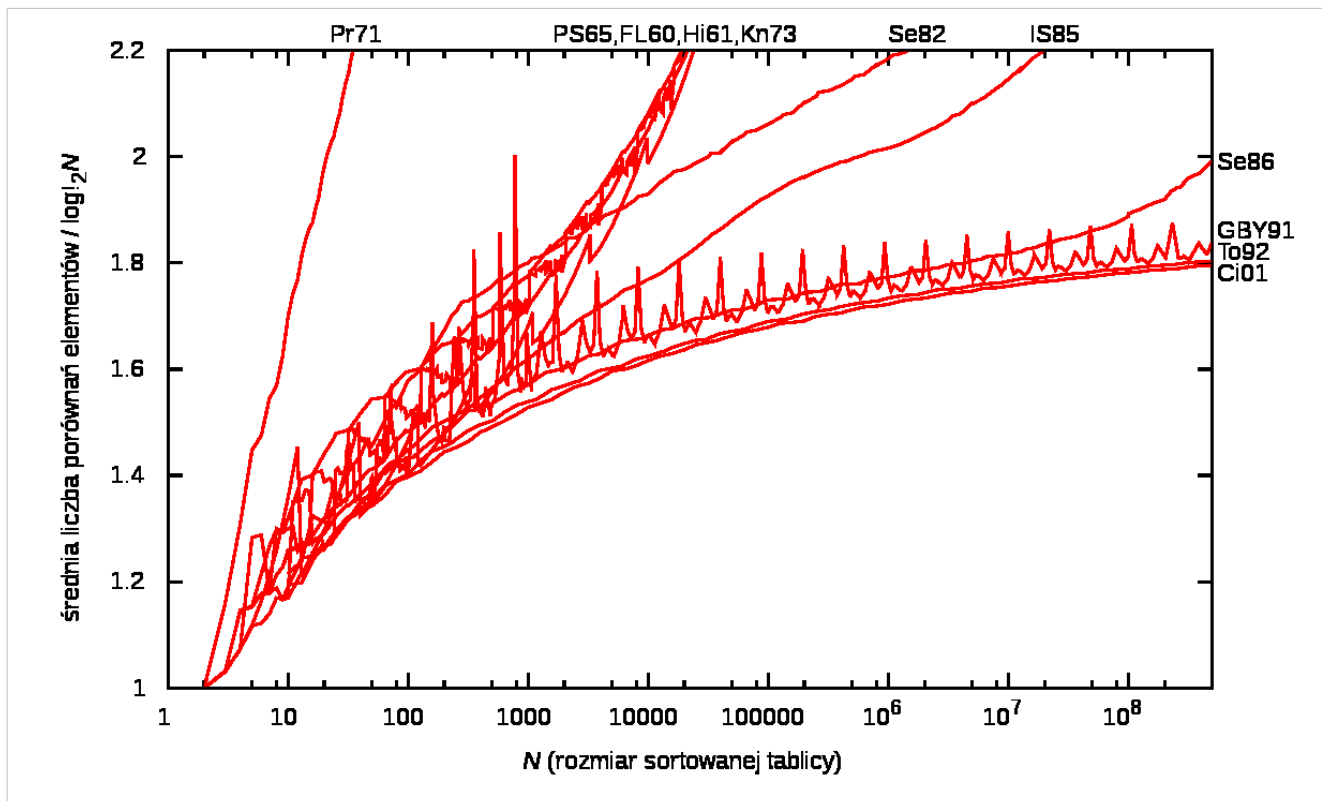
Żaden dowiedziony wynik na temat średniej liczby operacji nie dotyczy praktycznego ciągu odstępów. Espelid wyliczył ją dla odstępów będących potęgami dwójki jako $0,5349N\sqrt{N} - 0,4387N - 0,097\sqrt{N} + O(1)$ ^[13]. Knuth wyznaczył średnią złożoność sortowania N -elementowej tablicy z dwoma przebiegami ($h, 1$) jako $2N^2/h + \sqrt{\pi N^3 h}$ ^[6]. Wynika stąd, że dwuprzebiegowe sortowanie Shella z $h = \Theta(N^{1/3})$ wykonuje średnio $O(N^{5/3})$ porównań. Yao znalazł średnią złożoność sortowania z trzema przebiegami^[14]. Jego wynik uściślili potem Janson i Knuth^[15]. Średnia liczba porównań wykonywanych podczas sortowania z trzema przebiegami ($ch, cg, 1$), gdzie h i g są względnie pierwsze wynosi $\frac{N^2}{4ch} + O(N)$ w pierwszym przebiegu,

$\frac{1}{8g}\sqrt{\frac{\pi}{ch}}(h-1)N^{3/2} + O(hN)$ w drugim przebiegu i $\psi(h, g)N + \frac{1}{8}\sqrt{\frac{\pi}{c}}(c-1)N^{3/2} + O((c-1)gh^{1/2}N) + O(c^2g^3h^2)$ w trzecim przebiegu.

Skomplikowana funkcja $\psi(h, g)$ z ostatniego wzoru jest asymptotycznie równa $\sqrt{\frac{\pi h}{128}}g + O(g^{-1/2}h^{1/2}) + O(gh^{-1/2})$. W szczególności, gdy $h = \Theta(N^{7/15})$, a $g = \Theta(h^{1/5})$, średni czas sortowania jest rzędu $O(N^{23/15})$.

Na podstawie doświadczeń odgadnięto, że z ciągami Hibbarda i Knutha algorytm działa w średnim czasie rzędu $O(N^{5/4})$ ^[6], a z ciągiem Gonneta i Baezy-Yatesa wykonuje średnio $0,41N \ln N (\ln \ln N + 1/6)$ przesunąć elementów^[9]. Aproksymacje średniej liczby operacji czynione kiedyś dla innych ciągów zawodzą, gdy sortowane tablice liczą miliony elementów.

Poniższy wykres przedstawia średnią liczbę porównań elementów w różnych wariantach sortowania Shella, dzieloną przez teoretyczne minimum, czyli $\log_2 N!$, przy czym do ciągu 1, 4, 10, 23, 57, 132, 301, 701 dodano dalsze wyrazy zgodnie ze wzorem $h_k = \lfloor 2,25h_{k-1} \rfloor$.



Korzystając z teorii złożoności Kołmogorowa, Jiang, Li i Vitányi udowodnili następujące dolne ograniczenia na rząd średniej liczby operacji w m -przebiegowym sortowaniu Shella: $\Omega(mN^{1+1/m})$ przy $m \leq \log_2 N$ i $\Omega(mN)$ przy $m > \log_2 N$ ^[16]. Zatem algorytm ten ma szanse działać w średnim czasie rosnącym asymptotycznie jak $M \log N$ tylko z ciągami o liczbie odstępów rosnącej proporcjonalnie do logarytmu długości sortowanych tablic. Nie wiadomo jednak, czy sortowanie Shella może osiągnąć taki asymptotyczny rząd złożoności średniej, optymalny dla sortowań opartych na porównaniach.

Złożoność pesymistyczna dowolnej wersji sortowania Shella jest wyższego rzędu: Plaxton, Poonen i Suel wykazali, że rośnie ona co najmniej jak $\Omega(N(\log N / \log \log N)^2)$ ^[17].

Zastosowania

Z sortowania Shella rzadko się obecnie korzysta w poważnych zastosowaniach. Wykonuje ono więcej działań niż sortowanie szybkie, ponadto częściej od niego nie trafia w pamięć podręczną procesora przy odczytach z pamięci.

Ze względu na stosunkowo krótki kod i nieużywanie stosu bywa ono stosowane zamiast sortowania szybkiego w implementacjach funkcji `qsort` z biblioteki standardowej języka C przeznaczonych dla systemów wbudowanych. Używa go na przykład biblioteka `uClibc`^[18]. Z podobnych przyczyn implementacja sortowania Shella znalazła się w jądrze systemu operacyjnego Linux^[19].

Można też stosować sortowanie Shella jako podalgorytm sortowania introspektywnego używany do sortowania krótkich podtablic, a także gdy głębokość rekurencji przekroczy zadany limit, aby zapobiec patologicznemu spowolnieniu sortowania. Działa tak na przykład `bzip2`, jeden z programów do kompresji danych^[20].

Przypisy

- [1] D.L. Shell. *A High-Speed Sorting Procedure* (<http://penguin.ewu.edu/cscd300/Topic/AdvSorting/p30-shell.pdf>). „Communications of the ACM”. 2 (7), ss. 30–32, 1959. doi:10.1145/368370.368387 (<http://dx.doi.org/10.1145/368370.368387>).
- [2] R.M. Frank, R.B. Lazarus. *A High-Speed Sorting Procedure*. „Communications of the ACM”. 3 (1), ss. 20–22, 1960. doi:10.1145/366947.366957 (<http://dx.doi.org/10.1145/366947.366957>).
- [3] Thomas N. Hibbard. *An Empirical Study of Minimal Storage Sorting*. „Communications of the ACM”. 6 (5), ss. 206–213, 1963. doi:10.1145/366552.366557 (<http://dx.doi.org/10.1145/366552.366557>).
- [4] A. A. Папернов, Г. В. Стасевич. *Об одном методе упорядочивания информации в запоминающих устройствах цифровых машин* (<http://www.mathnet.ru/links/83f0a81df1ec06f76d3683c6cab7d143/ppi751.pdf>). „Проблемы передачи информации”. 1 (3), ss. 81–98, 1965 (ros.).
- [5] Vaughan Ronald Pratt: *Shellsort and Sorting Networks (Outstanding Dissertations in the Computer Sciences)*. Garland, 1979. ISBN 0-824-04406-1.
- [6] Metoda Shella. W: Donald E. Knuth: *Sztuka programowania*. Wyd. 1. T. 3: Sortowanie i wyszukiwanie. WNT, 2002, ss. 87–99. ISBN 83-204-2554-9.
- [7] Janet Incerpi, Robert Sedgewick. *Improved Upper Bounds on Shellsort*. „Journal of Computer and System Sciences”. 31 (2), ss. 210–224, 1985.
- [8] Robert Sedgewick. *A New Upper Bound for Shellsort*. „Journal of Algorithms”. 7 (2), ss. 159–173, 1986. doi:10.1016/0196-6774(86)90001-5 ([http://dx.doi.org/10.1016/0196-6774\(86\)90001-5](http://dx.doi.org/10.1016/0196-6774(86)90001-5)).
- [9] Shellsort. W: Gaston H. Gonnet, Ricardo Baeza-Yates: *Handbook of Algorithms and Data Structures: In Pascal and C*. Wyd. 2. Reading, Massachusetts: Addison-Wesley, 1991, ss. 161–163. ISBN 0-201-41607-7.
- [10] Naoyuki Tokuda: An Improved Shellsort. W: Jan van Leeuwen: *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture*. Amsterdam: North-Holland Publishing Co., 1992, ss. 449–457. ISBN 0-444-89747-X.
- [11] Marcin Ciura: Best Increments for the Average Case of Shellsort. W: Rusins Freiwalds: *Proceedings of the 13th International Symposium on Fundamentals of Computation Theory* (<http://sun.aei.polsl.pl/~mciura/publikacje/shellsort.pdf>). London: Springer-Verlag, 2001, ss. 106–117. ISBN 3-540-42487-3.
- [12] David Gale, Richard M. Karp. *A Phenomenon in the Theory of Sorting*. „Journal of Computer and System Sciences”. 6 (2), ss. 103–115, 1972. doi:10.1016/S0022-0000(72)80016-3 ([http://dx.doi.org/10.1016/S0022-0000\(72\)80016-3](http://dx.doi.org/10.1016/S0022-0000(72)80016-3)).
- [13] Terje O. Espelid. *Analysis of a Shellsort Algorithm*. „BIT Numerical Mathematics”. 13 (4), ss. 394–400, 1973. doi:10.1007/BF01933401 (<http://dx.doi.org/10.1007/BF01933401>).
- [14] Andrew Chi-Chih Yao. *An Analysis of $(h, k, 1)$ -Shellsort*. „Journal of Algorithms”. 1 (1), ss. 14–50, 1980. doi:10.1016/0196-6774(80)90003-6 ([http://dx.doi.org/10.1016/0196-6774\(80\)90003-6](http://dx.doi.org/10.1016/0196-6774(80)90003-6)).
- [15] Svante Janson, Donald E. Knuth. *Shellsort with Three Increments* (<http://arxiv.org/abs/cs/9608105>). „Random Structures and Algorithms”. 10 (1–2), ss. 125–142, 1997. doi:10.1.1.54.9911 (<http://dx.doi.org/10.1.1.54.9911>).
- [16] Tao Jiang, Ming Li, Paul Vitányi. *A Lower Bound on the Average-Case Complexity of Shellsort* (<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.6.6508>). „Journal of the ACM”. 47 (5), ss. 905–911, 2000. doi:10.1.1.6.6508 (<http://dx.doi.org/10.1.1.6.6508>).
- [17] C. Greg Plaxton, Bjarne Poonen, Torsten Suel. *Improved Lower Bounds for Shellsort* (<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.43.1393>). „Annual Symposium on Foundations of Computer Science”. 33, ss. 226–235, 1992. doi:10.1.1.43.1393 (<http://dx.doi.org/10.1.1.43.1393>).
- [18] Manuel Novoa III: `libc/stdlib/stdlib.c` (<http://codesearch.google.com/codesearch/p?hl=en#bLhHCxq0Bgw/downloads/uClibc-snapshot.tar.bz2/7hKHs61H6WA/uClibc/libc/stdlib/stdlib.c&l=787>). [dostęp 2011-03-30].
- [19] `kernel/groups.c` (<http://codesearch.google.com/codesearch/p?hl=en#d7MjWbXYfN0/kernel/groups.c&sa=N&cd=8&ct=ri&l=105>). [dostęp 2011-03-30].
- [20] Julian Seward: `bzip2/blocksort.c` (http://codesearch.google.com/codesearch/p?hl=en#hfE6470xZHk/third_party/bzip2/blocksort.c&l=472). [dostęp 2011-03-30].

Bibliografia

- Metoda Shella. W: Donald E. Knuth: *Sztuka programowania*. Wyd. 1. T. 3: Sortowanie i wyszukiwanie. WNT, 2002, ss. 87–99. ISBN 83-204-2554-9.
- Analysis of Shellsort and Related Algorithms (<http://www.cs.princeton.edu/~rs/shell/>), Robert Sedgewick, Fourth European Symposium on Algorithms, Barcelona, wrzesień 1996.
- O sortowaniu Shella (<http://sun.aei.polsl.pl/~mciura/publikacje/shellsort-delta.pdf>), Marcin Ciura, *Delta*, listopad 2008.

Linki zewnętrzne

- Sortowanie Shella z odstępami 5, 3, 1 przedstawione jako węgierski taniec (<http://www.youtube.com/watch?v=CmPA7zE8mx0>)

Sortowanie grzebieniowe

Sortowanie grzebieniowe (ang. *combsort*) – wynaleziona w 1980 przez Włodzimierza Dobosiewicza Wikipedia: Weryfikowalność, odkryta ponownie i opisana w 1991 roku przez Stephena Lacey'a i Richarda Boxa metoda sortowania tablicowego. Jej główne cechy to:

- oparta na metodzie bubblesort (sortowanie bąbelkowe)
- prawdopodobnie złożoność wynosi $O(n \log n)$, statystycznie gorsza niż quicksort (sortowanie szybkie)
- włączono empirię - współczynnik 1.3 wyznaczony doświadczalnie

wariant podstawowy:

- za rozpiętość przyjmuje się długość tablicy, dzieli się rozpiętość przez 1.3, odrzuca część ułamkową
- bada się kolejno wszystkie pary obiektów odległych o rozpiętość (jeśli są ułożone niemonotonicznie - zamienia się je miejscami)
- wykonuje się powyższe w pętli dzieląc rozpiętość przez 1.3 do czasu, gdy rozpiętość osiągnie wartość 1.

Gdy rozpiętość spadnie do 1 metoda zachowuje się tak jak sortowanie bąbelkowe. Tylko wtedy można określić, czy dane są już posortowane czy nie. W tym celu można użyć zmiennej typu bool, która jest ustawiana po zamianie elementów tablicy miejscami. Przerwywane jest wykonywanie algorytmu, gdy podczas przejścia przez całą tablicę nie nastąpiła zamiana.

Wariant Combsort 11: rozpiętość 9 i 10 zastępowane jest 11

Przykład w języku C / C++

- `tab` - tablica elementów (w przykładzie tablica liczb całkowitych)
- `gap` - rozpiętość; w kolejnych iteracjach pętli dzielona jest przez współczynnik 1.3
- `tmp` - zmienna całkowitoliczbowa; do zamiany elementów
- `swapped` - zmienna logiczna; czy dokonano zamiany elementów

```
void combSort(int* tab, int size)
{
    int gap = size, tmp;
    bool swapped = true;
    while (gap > 1 || swapped) { // jeśli gap = 1 i nie dokonano zamiany -
        wyjście z pętli
        gap = gap * 10 / 13;
        if (gap == 0)
            gap = 1;
        swapped = false;
        for (int i = 0; i + gap < size; ++i) { // wykonuj od 0 do ostatniego elementu tablicy
            if (tab[i + gap] < tab[i]) { // porównanie elementów odległych o rozpiętość
                tmp = tab[i]; // zamiana elementów
                tab[i] = tab[i + gap];
                tab[i + gap] = tmp;
                swapped = true;
            }
        }
    }
}
```

```

    }
  }
}

```

Funkcja do wyznaczania współczynnika rozpiętości (Wariant Combsort 11)

```

int newGap(int gap)
{
    gap = gap * 10 / 13;
    if ( gap == 9 || gap == 10 ) gap = 11;
    if(gap==0) gap=1;
    return gap;
}

```

Przykład w języku pascal

```

procedure Combsort(var a:tab; n:integer);
function newGap(gap:integer):integer;
begin
    gap := trunc(gap / 1.3);
    if (gap = 9) OR (gap=10) then
        gap := 11;
    if gap < 1 then
        gap := 1;
    newGap:=gap;
end;
var
    top, gap, i, j : integer;
    x : element;
    swapped : boolean;
begin
    gap := n;
    repeat
        gap := newGap(gap);
        top := n - gap;
        swapped := false;
        for i := 1 to top do
            begin
                j := i + gap;
                if a[i] > a[j] then
                    begin
                        x := a[i];
                        a[i] := a[j];
                        a[j] := x;
                        swapped := true;
                    end;
            end;
        until (gap = 1) and not swapped;

```

```
end;
```

Bibliografia

- Włodzimierz Dobosiewicz. *An Efficient Variation of Bubble Sort*. „Inf. Process. Lett.”. 11(1): 5-6, 1980.

Sortowanie introspektywne

Sortowanie introspektywne (ang. *introspective sort* lub *introsort*) - odmiana sortowania hybrydowego, w której wyeliminowany został problem złożoności $O(n^2)$ występującej w najgorszym przypadku algorytmu sortowania szybkiego.

Sortowanie Hybrydowe

Pomysł Sortowania Hybrydowego opiera się na spostrzeżeniu, że ogromna liczba rekurencyjnych wywołań algorytmu Quick Sort jest realizowana dla małych tablic. W przypadku małych tablic czynności organizacyjne dokonywane przez procedurę Partition, takie jak wyznaczenie mediany z trzech, przygotowanie wskaźników i sprawdzanie warunków na nie nakładanych, zajmują relatywnie dużo czasu w stosunku do rozmiaru samej tablicy. Poza tym, samo wywołanie rekurencyjne jest czasochłonne i zajmuje miejsce na stosie, a dla 5-elementowej tablicy mogą być potrzebne nawet 3 wywołania. Dlatego też w algorytmie Sortowania Hybrydowego przyjęto, że małe tablice będą sortowane jednym z elementarnych algorytmów sortowania, które chociaż mają kwadratową złożoność obliczeniową, dla zbiorów o niewielkim rozmiarze działają relatywnie szybko. Opisana metoda nazywana jest „odcinaniem małych podzbiorów” i realizowana jest w ten sposób, że warunkiem wywołania rekurencyjnego procedury Quick Sort jest rozmiar tablicy większy od pewnej ustalonej wartości, eksperymentalnie wyznaczonej na 9.

Po zakończeniu rekurencyjnych wywołań procedury Quick Sort tablica podzielona jest na szereg małych podzbiorów o rozmiarze nie większym niż 9, poroździelanych elementami, które w rekurencyjnych wywołaniach procedury Quick Sort wykorzystywane były jako elementy osiowe. Dla każdego takiego podzbioru prawdą jest, że klucz żadnego z jego elementów nie jest mniejszy od klucza jakiegokolwiek elementu poprzedzającego zbiór w tablicy ani nie jest większy od klucza jakiegokolwiek elementu po nim następującego. Dla prawie posortowanej tablicy wywołuje się wówczas procedurę Sortowania Przez Wstawianie, której zadaniem jest uporządkowanie zbioru do końca. Sortowanie Przez Wstawianie jest elementarnym algorytmem sortowania, ale posiada liniową złożoność obliczeniową dla uporządkowanych tablic danych, ponieważ każdy kolejny element jest przepychany do przodu tablicy do momentu natrafienia na element od niego mniejszy. Jeśli tablica jest podzielona na podzbiory w taki sposób, jak opisano powyżej, to elementy każdego podzbioru będą porównywane tylko ze sobą.

Sortowanie Introspektywne

Głównym założeniem algorytmu Sortowania Introspektywnego jest obsługa najgorszego przypadku algorytmu Sortowania Szybkiego tak, aby zapewnić logarytmiczno-liniową złożoność obliczeniową. Przypomnijmy, że w najgorszym przypadku podziały wykonywane przez procedurę Partition były zdegenerowane i algorytm Quick Sort wykonywał $O(n^2)$ porównań.

Rozwiązaniem problemu złożoności obliczeniowej $O(n^2)$ w najgorszym przypadku jest badanie głębokości rekurencji. W procedurze głównej Sortowania Introspektywnego: Hybrid Introspective Sort tworzona jest stała M o wartości $2 \cdot \log_2 n$, która określa maksymalną dozwoloną głębokość wywołań rekurencyjnych. Następnie wywoływana jest procedura Intro Sort. Procedura Intro Sort przyjmuje jako dodatkowy parametr wartość M , która określa maksymalną dozwoloną głębokość wywołań rekurencyjnych z poziomu, na którym obecnie się znajdujemy.

Jeżeli wartość parametru M wynosi 0, wywołania rekurencyjne są kończone i dla podproblemu, którym obecnie się zajmujemy, wywoływana jest procedura Sortowania Przez Kopcowanie, które jest traktowane jako sortowanie pomocnicze.

W przypadku gdy $M > 0$ procedura Intro Sort działa podobnie jak procedura Quick Sort. Wywoływana jest procedura Partition, która dzieli tablicę na dwa rozłączne podzbiory, gromadząc w pierwszym elementy posiadające klucze o wartości mniejszej równej wartości klucza elementu rozdzielającego, a w drugim elementy o wartościach kluczy większych równych kluczowi pivota. Następnie dla obu podzbiorów wywołana jest rekurencyjnie procedura Intro Sort z parametrem M pomniejszonym o 1, w związku z czym maksymalna dozwolona głębokość rekurencyjnych wywołań z następnego poziomu jest o 1 mniejsza.

Dzięki zastosowanej procedurze otrzymujemy gwarancję, że w najgorszym przypadku algorytmu Sortowania Szybkiego (sekwencja Median-Of-Three killer) po zrealizowaniu $2 \cdot \log_2 n$ zdegenerowanych podziałów rekurencja zostanie zatrzymana i wywołana zostanie procedura sortowania pomocniczego Heap Sort, którego złożoność obliczeniowa wynosi $O(n \cdot \log_2 n)$. Dla losowych danych Sortowanie Przez Kopcowanie działa około 3 razy dłużej niż Sortowanie Szybkie, zauważmy jednak, że dozwolona głębokość wywołań rekurencyjnych $2 \cdot \log_2 n$ przekracza głębokość rekurencji w średnim przypadku, czyli około $1,39 \cdot \log_2 n$. Dzięki temu dla danych losowych Sortowanie Introspektywne będzie działało tak samo jak Sortowanie Szybkie z odcinaniem małych podzbiorów, a Sortowanie Przez Kopcowanie jako sortowanie pomocnicze wywoływane będzie tylko wówczas, gdy przynajmniej początkowe podziały będą zdegenerowane.

Listing procedury Hybrydowego Sortowania Introspektywnego w języku C++

```
template <class Item>
void Hybrid_Introspective_Sort (Item *Array, long N)
{
    IntroSort (Array, N, (int) floor (2*log (N) / M_LN2) );
    Insertion_Sort (Array, N);
}

template <class Item>
void IntroSort (Item *Array, long N, int M)
{
    long i;
    if (M<=0)
    {
        Heap_Sort (Array, N);
        return;
    }
    i=Partition (Array, 0, N);
    if (i>9)
        IntroSort (Array, i, M-1);
    if (N-1-i>9)
        IntroSort (Array+i+1, N-1-i, M-1);
}
```

Kod procedury Partition jest taki sam, jak w przypadku sortowania szybkiego:

```
template <class Item>
long Partition (Item *Array, long L, long R)
```



```

{
    long i, j;
    if (R>=3)
        MedianOfThree(Array,L,R);
    for (i=L, j=R-2; ; )
    {
        for ( ; Array[i]<Array[R-1]; ++i);
        for ( ; j>=L && Array[j]>Array[R-1]; --j);
        if (i<j)
            Exchange(Array,i++,j--);
        else break;
    }
    Exchange(Array,i,R-1);
    return i;
}

template <class Item>
void MedianOfThree (Item *Array, long &L, long &R)
{
    if (Array[++L-1]>Array[--R])
        Exchange(Array,L-1,R);
    if (Array[L-1]>Array[R/2])
        Exchange(Array,L-1,R/2);
    if (Array[R/2]>Array[R])
        Exchange(Array,R/2,R);
    Exchange(Array,R/2,R-1);
}

template <class Item>
void Exchange (Item *Array, long i, long j)
{
    Item temp;
    temp=Array[i];
    Array[i]=Array[j];
    Array[j]=temp;
}

```

Kod procedury Heap Sort to kod sortowania przez kopcowanie:

```

template <class Item>
void Heap_Sort (Item *Array, long N)
{
    long i;
    for (i=N/2; i>0; --i)
        Heapify(Array-1,i,N);
    for (i=N-1; i>0; --i)
    {
        Exchange(Array,0,i);
    }
}

```

```

        Heapify(Array-1,1,i);
    }
}

template <class Item>
void Heapify (Item *Array, long i, long N)
{
    long j;
    while (i<=N/2)
    {
        j=2*i;
        if (j+1<=N && Array[j+1]>Array[j])
            j=j+1;
        if (Array[i]<Array[j])
            Exchange(Array,i,j);
        else break;
        i=j;
    }
}

```

Procedura Insertion Sort to procedura sortowania przez wstawianie:

```

template <class Item>
void Insertion_Sort (Item *Array, long N)
{
    long i, j;
    Item temp;
    for (i=1; i<N; ++i)
    {
        temp=Array[i];
        for (j=i; j>0 && temp<Array[j-1]; --j)
            Array[j]=Array[j-1];
        Array[j]=temp;
    }
}

```

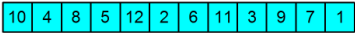
Złożoność obliczeniowa, najlepszy i najgorszy przypadek algorytmu

W przypadku ogólnym, a więc również w najgorszym, algorytm Sortowania Introspektywnego posiada złożoność obliczeniową $O(n \cdot \log_2 n)$. W najgorszym przypadku algorytm wykonuje najpierw $M=2 \cdot \log_2 n$ rekurencyjnych wywołań, takich jak w Sortowaniu Szybkim, a następnie dla pozostałego podzbioru wywołuje procedurę Heap Sort. Tak więc w najgorszym przypadku dane na pozycjach o indeksach: 0, od $N/2-M$ do $N/2$ i od $N/2-2 \cdot M$ do N muszą być ustawione w sekwencji Median-Of-Three killer, a dane na pozostałych pozycjach muszą stanowić zły przypadek dla algorytmu Heap Sort, na przykład być posortowane. W takiej sytuacji algorytm wykona około $4 \cdot n \cdot \log_2 n + n - 6 \cdot (\log_2 n)^2$ operacji porównywania danych.

W najlepszym i pośrednim przypadku jednak algorytm Sortowania Introspektywnego działa tak samo, jak algorytm Sortowania Szybkiego, stąd można wywnioskować, że w najlepszym przypadku dla danych posortowanych wykonywanych będzie około $n \cdot \log_2 n$ porównań, a dla danych losowych około $1,39 \cdot n \cdot \log_2 n$ porównań.

Algorytm Sortowania Introspektywnego potrzebuje $O(\log_2 n)$ pamięci na stos w każdym przypadku i jest algorytmem sortującym w miejscu.

Sortowanie przez kopcowanie

Sortowanie przez kopcowanie	
	
Animacja przedstawiająca działanie algorytmu Heap sort	
Rodzaj	Sortowanie
Struktura danych	Tablica
Złożoność	
Czasowa	$O(n \log(n))$
Pamięciowa	$O(1)$

Sortowanie kopcem (ang. *heapsort*) - zwane też inaczej sortowaniem przez kopcowanie. Algorytm ten jest jedną z ciekawszych metod sortowania z racji faktu, iż jest on szybki oraz nie pochłania zbyt wiele zasobów pamięci. Jego złożoność czasowa to $O(n \log n)$, a pamięciowa $O(1)$. Algorytm ten jest w praktyce z reguły nieco wolniejszy od sortowania szybkiego, lecz ma lepszą pesymistyczną złożoność czasową (przez co jest odporny np. na atak za pomocą celowo spreparowanych danych, które spowodowałyby jego znacznie wolniejsze działanie). Sortowanie przez kopcowanie jest niestabilne, co może być czasami uznawane za wadę.

Opis algorytmu

Podstawą algorytmu jest użycie kolejki priorytetowej zaimplementowanej w postaci **binarnego kopca zupełnego**. Szczegóły implementacji kopca wyjaśnione są w artykułach kopiec i kopiec binarny. Zaletą kopca jest to, że oprócz stałego czasu dostępu do elementu maksymalnego (lub minimalnego) oferuje on logarytmiczny czas wstawiania i usuwania elementów. Ponadto kopiec można łatwo implementować w postaci tablicy.

Algorytm sortowania przez kopcowanie składa się z dwóch faz. W pierwszej sortowane elementy reorganizowane są w celu utworzenia kopca. W drugiej zaś dokonywane jest właściwe sortowanie.

Tworzenie kopca

Podstawową zaletą algorytmu jest to, że do stworzenia kopca wykorzystać można tę samą tablicę, w której początkowo znajdują się nieposortowane elementy. Dzięki temu uzyskuje się stałą złożoność pamięciową.

Początkowo do kopca należy tylko pierwszy element w tablicy. Następnie kopiec rozszerzany jest o drugą, trzecią i kolejne pozycje tablicy, przy czym przy każdym rozszerzeniu, nowy element jest przemieszczany w górę kopca, tak aby spełnione były relacje pomiędzy węzłami. Schematycznie wygląd sortowanej tablicy można przedstawić w następujący sposób:

kopiec	reszta nieposortowanych elementów
--------	-----------------------------------

a kopiec rozrasta się, aż do wyczerpania nieposortowanej części tablicy.

Dzięki logarytmicznej złożoności pojedynczych operacji wstawiania (rozszerzania kopca), całkowita złożoność tego etapu to $O(n \log n)$.

Można też ten krok wykonać szybciej - w czasie $O(n)$. W tym celu należy budować kopiec w następujący sposób:

reszta nieposortowanych elementów	małe kopce
-----------------------------------	------------

Aby osiągnąć taką strukturę, wystarczy pierwsze $n \div 2$ elementów tablicy (zakładając, że kopiec implementujemy w tablicy) przesunąć w dół kopca procedurą *shift-down*:

```

shift-down (T[1..n], i)
  k ← i
  repeat
    j ← k
    if 2j ≤ n and T[2j] > T[k]
      k ← 2j
    if 2j+1 ≤ n and T[2j+1] > T[k] and T[2j+1] > T[2j]
      k ← 2j+1
    swap (T[j], T[k])
  until j = k

```

Zatem procedura budująca kopiec wyglądałaby następująco:

```

build-heap (T[1..n])
  for i ← n div 2 downto 1
    shift-down (T, i)

```

Procedura build-heap buduje kopiec w czasie $O(n)$.

Sortowanie

Po utworzeniu kopca następuje właściwe sortowanie. Polega ono na usunięciu wierzchołka kopca, zawierającego element maksymalny (minimalny), a następnie wstawieniu w jego miejsce elementu z końca kopca i odtworzenie porządku kopcowego. W zwolnione w ten sposób miejsce, zaraz za końcem zmniejszonego kopca wstawia się usunięty element maksymalny. Operacje te powtarza się aż do wyczerpania elementów w kopcu. Wygląd tablicy można tu schematycznie przedstawić następująco:

kopiec elementów do posortowania	posortowana tablica
----------------------------------	---------------------

Tym razem kopiec kurczy się, a tablica przyrasta (od elementu ostatniego do pierwszego).

W tej fazie wykonuje się, jak w poprzedniej, n kroków (usuwanie elementu połączone z odtwarzaniem porządku kopcowego), każdy o koszcie logarytmicznym, zatem złożoność tej fazy to także $O(n \log n)$.

Prezentację takiego sortowania można zobaczyć na stronie Sortowanie przez kopcowanie ^[1] Ww strona zawiera applet prezentujący działanie procedury budującej kopiec z danych zawartych w tablicy oraz umożliwia podgląd zawartości tablicy w postaci drzewa.

Porównanie z innymi algorytmami sortowania

Algorytm sortowania przez kopcowanie jest na ogół nieco wolniejszy niż sortowanie szybkie. Jego przewagą natomiast jest lepsza złożoność pesymistyczna wynosząca $O(n \log n)$, podczas gdy dla quicksort jest to $O(n^2)$, co jest nie do przyjęcia dla dużych zbiorów danych. Także złożoność pamięciowa $O(1)$ jest lepsza niż $\Omega(\log n)$ algorytmu quicksort.

Heapsort jest nieco wolniejszy od algorytmu sortowania przez scalanie (mergesort), mającego taką samą asymptotyczną złożoność czasową. Mergesort wymaga jednak $\Omega(n)$ dodatkowej pamięci. Zaletą mergesort jest prostsza definicja i lepsze zachowanie w przypadkach, gdy dane do sortowania pobierane są z wolnej pamięci masowej; jest też łatwiejszy do zrównoleglenia.

Przypisy

[1] <http://operacjenakopcach.strefa.pl>

Sortowanie koktajlowe

Sortowanie koktajlowe, znane także jako **dwukierunkowe sortowanie bąbelkowe**, **sortowanie przez wstrząsanie** (które również odwołuje się do odmiany sortowania przez wybieranie), jest odmianą sortowania bąbelkowego, które jest stabilnym algorytmem sortującym za pomocą porównań. Algorytm w przeciwieństwie do sortowania bąbelkowego sortuje liczby w zbiorze w dwóch kierunkach.

Opis algorytmu

Sortowanie koktajlowe oparte jest na spostrzeżeniu, iż każdy obieg wewnętrznej pętli sortującej umieszcza na właściwym miejscu element najstarszy, a elementy młodsze przesuwają o 1 pozycję w kierunku początku zbioru. Jeśli pętla ta zostanie wykonana w kierunku odwrotnym, to wtedy najmłodszy element znajdzie się na swoim właściwym miejscu, a elementy starsze przesuną się o jedną pozycję w kierunku końca zbioru. Łącząc te dwie pętle sortując wewnętrznie naprzemiennie w kierunku normalnym i odwrotnym, otrzymujemy algorytm sortowania koktajlowego.

Kod źródłowy

Przykład implementacji algorytmu w pseudokodzie:

```
function cocktail_sort(list, list_length) // pierwszy element zbioru ma index 0
{
    bottom = 0;
    top = list_length - 1;
    zamiana = true;
    while(zamiana == true) // jeżeli żaden element nie został zamieniony, lista jest posortowana
    {
        zamiana = false;
        for(i = bottom; i < top; i = i + 1)
```

```
{
    if(list[i] > list[i + 1]) // sprawdzamy czy elementy są na właściwych miejscach
    {
        zamien(list[i], list[i + 1]); // zamieniamy elementy miejscami
        zamiana = true;
    }
}
// zmniejszamy wartość `top` ponieważ element o największej wartości jest nie posortowany
top = top - 1;
for(i = top; i > bottom; i = i - 1)
{
    if(list[i] < list[i - 1])
    {
        zamien(list[i], list[i - 1]);
        zamiana = true;
    }
}
// zwiększamy wartość `bottom` ponieważ element o najmniejszej wartości jest nieposortowany
bottom = bottom + 1;
}
```

Optymalizacja

Jedynym możliwym sposobem optymalizacji jest dodanie instrukcji warunkowej, która sprawdza, czy nastąpiła zamiana elementów po pierwszym przejściu pętli, jeżeli nie – lista jest posortowana.

Różnice w stosunku do sortowania bąbelkowego

Sortowanie koktajlowe jest odmianą sortowania bąbelkowego. Różni się od niego tym, że zamiast wielokrotnie przechodzić przez listę od dołu do góry przechodzi na przemian z dołu do góry i następnie z góry do dołu. Dzięki temu ma trochę lepsze osiągi niż standardowe sortowanie bąbelkowe. Wynika to z tego że sortowanie bąbelkowe przechodzi przez listę tylko w jednym kierunku, zatem może cofać się jedynie o jedną pozycję bez możliwości powtarzania.

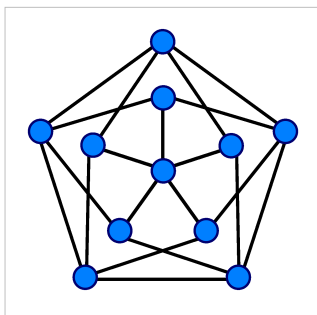
Np. posortowanie elementów (2,3,4,5,1) metoda koktajlową wymaga tylko jednego przejścia aby elementy zostały posortowane, zaś jeśli użyjemy metody bąbelkowej, będziemy potrzebowali czterech przejść.

Złożoność

Typowa czasowa złożoność obliczeniowa sortowania koktajlowego jest klasy $O(n^2)$, jednak przy sortowaniu zbiorów w znacznym stopniu posortowanych klasa złożoności obliczeniowej redukuje się do $O(n)$

Graf (matematyka)

Niniejszy artykuł jest częścią cyklu **teoria grafów**.



Najważniejsze pojęcia

graf
drzewo
podgraf
cykl
klika
stopień wierzchołka
stopień grafu
dopełnienie grafu
obwód grafu
pokrycie wierzchołkowe
liczba chromatyczna
indeks chromatyczny
izomorfizm grafów
homeomorfizm grafów

[więcej...](#)

Wybrane klasy grafów

graf pełny
graf spójny
drzewo
graf dwudzielny
graf regularny
graf eulerowski
graf hamiltonowski
graf planarny

[więcej...](#)

Algorytmy grafowe

A*
Bellmana-Forda
Dijkstry
Fleury'ego
Floyda-Warshalla
Johnsona
Kruskala
Prima
przeszukiwanie grafu
– wszerz
– w głąb
najbliższego sąsiada

Zagadnienia przedstawiane jako problemy grafowe

problem komiwojażera
problem chińskiego listonosza
problem marszrutyżacji
problem kojarzenia małżeństw

Inne zagadnienia

kod Graya
diagram Hassego
kod Prüfera

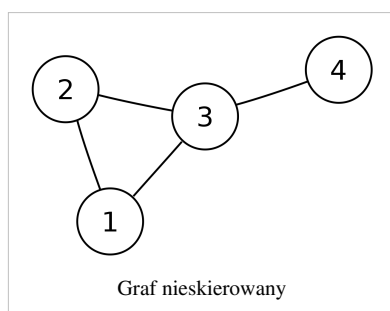
Graf to – w uproszczeniu – zbiór *wierzchołków*, które mogą być połączone *krawędziami*, w taki sposób, że każda krawędź kończy się i zaczyna w którymś z wierzchołków (ilustracja po prawej stronie). Grafy to podstawowy obiekt rozważań teorii grafów. Za pierwszego teoretyka i badacza grafów uważa się^[1] Leonarda Eulera, który rozstrzygnął zagadnienie mostów królewieckich.

Wierzchołki grafu zwykle są numerowane i czasem stanowią reprezentację jakichś obiektów, natomiast krawędzie mogą wówczas obrazować relacje między takimi obiektami. Krawędzie mogą mieć wyznaczony kierunek, a graf zawierający takie krawędzie jest grafem skierowanym. Krawędź może posiadać także wagę, to znaczy przypisaną liczbę, która określa na przykład odległość między wierzchołkami (jeśli na przykład graf jest reprezentacją połączeń między miastami). W grafie skierowanym wagi mogą być zależne od kierunku przechodzenia przez krawędź (np. jeśli graf reprezentuje trud poruszania się po jakimś terenie, to droga pod górkę będzie miała przypisaną większą wagę niż z górki).

Definicje

Różni autorzy stosują bardzo odmienne sposoby definiowania i oznaczania elementów grafu.

Graf



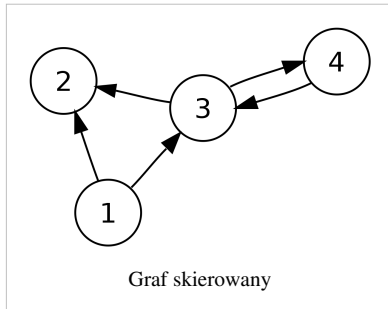
Graf, graf prosty lub **graf nieskierowany** to uporządkowana para $G := (V, E)$ gdzie:

- V jest niepustym zbiorem. Elementy tego zbioru nazywamy *wierzchołkami*,
- E jest rodziną dwuelementowych podzbiorów zbioru wierzchołków V , zwanych **krawędziami**:

$$E \subseteq \{\{u, v\} : u, v \in V, u \neq v\}.$$

Wierzchołki należące do krawędzi nazywane są jej *końcami*. Zazwyczaj V (i co za tym idzie E) są określane jako zbiory skończone. Jednak powyższa definicja tego nie wymaga i w praktyce rozważa się też czasami grafy o nieskończonej liczbie wierzchołków (wtedy liczba krawędzi może być skończona, lub nieskończona).

Graf skierowany



Graf skierowany lub inaczej **digraf** to uporządkowana para $G := (V, A)$ gdzie:

- V jest zbiorem *wierzchołków*,
- A jest zbiorem uporządkowanych par różnych wierzchołków ze zbioru V , zwanych **krawędziami skierowanymi**, lub **łukami**: $A \subseteq V \times V$.

Przyjmuje się, że krawędź $e := (x, y)$ jest skierowana z x do y , czyli wychodzi z x , a wchodzi do y .

Graf mieszany

Graf mieszany to uporządkowana trójka $G := (V, E, A)$ gdzie zbiory V, E, A są zdefiniowane jak wyżej, czyli może zawierać jednocześnie krawędzie skierowane i nieskierowane.

Warianty definicji

W wielu zastosowaniach tak zdefiniowane grafy nie są wystarczające i wprowadza się pewne modyfikacje.

Na przykład aby wprowadzić **pętlę** czyli krawędź, której oba końce są tym samym wierzchołkiem, w definicji grafu nieskierowanego należy dopuścić zbiory jednoelementowe $\{v\}$ albo użyć dwuelementowego multizbioru $\{v, v\}$. W grafie skierowanym pętla jest naturalnie reprezentowana przez parę (v, v) .

Czasami potrzebna jest możliwość połączenia dwóch wierzchołków przy pomocy więcej niż jednej krawędzi (w przypadku grafu skierowanego chodzi o łuki o takim samym zwrocie). Graf, który na to pozwala, nazywany jest **multigrafem**. Uzyskuje się go np. przez zdefiniowanie E , lub A jako multizbioru.

Przez zdefiniowanie funkcji z V, E , lub A w pewien zbiór X , można przypisać krawędziom lub wierzchołkom **etykiety**, służące do przechowywania dodatkowych informacji. Etykiety liczbowe są często nazywane **wagami**. Dla grafów z wagami zbiór tworzący graf jest rozszerzony o funkcję $\delta : E \rightarrow K$ taką, że dla każdej krawędzi $e \in E$, $\delta(e)$ jest wagą danej krawędzi

Przykłady odmiennych sposobów definiowania grafu:

- Graf może być też określony jako niepusty zbiór wierzchołków i dana na nim relacja binarna R taka, że dla dowolnych wierzchołków v i u vRu zachodzi wtedy i tylko wtedy, gdy istnieje krawędź łącząca v i u . Dla grafów nieskierowanych relacja ta jest symetryczna (zob. też "macierz sąsiedztwa" poniżej).
- Graf *nieskierowany* można też definiować jako trójkę $G = (V, E, \gamma)$, gdzie

- V jest zbiorem wierzchołków
- E zbiorem krawędzi
- γ funkcją ze zbioru krawędzi w rodzinę jednoelementowych lub dwuelementowych podzbiorów zbioru wierzchołków – $\gamma : E \rightarrow (\{\{v, u\} : v, u \in V\} \cup \{\{w\} : w \in V\})$. Wówczas jeżeli e jest krawędzią grafu to:
 - kończy się ona wierzchołkami $u, v \in V$, gdy $\gamma(e) = \{u, v\}$
 - jest ona pętlą wierzchołka v , gdy $\gamma(e) = \{v\}$
- Graf *skierowany* określa się też jako trójkę $G = (V, E, \gamma)$, gdzie zbiory V i E są zdefiniowane analogicznie do grafów nieskierowanych a γ jest funkcją ze zbioru krawędzi w zbiór uporządkowanych par (kwadrat kartezjański, czyli iloczyn kartezjański zbioru ze sobą) wierzchołków – $\gamma : E \rightarrow V \times V$. Wówczas, jeżeli e jest krawędzią grafu G to istnieją takie wierzchołki $u, v \in V$, że $\gamma(e) = (u, v)$. W takim przypadku krawędź e biegnie z u do v .

Graf geometryczny

Dla każdego grafu istnieje nieskończenie wiele przedstawiających go *rysunków*, czasami jednak rozważane są w przypadku grafów własności *stricte* geometryczne (współrzędne geometryczne wierzchołków, tylko proste krawędzie, "zmieszczenie się" w pewnej przestrzeni itp.). Grafy rozpatrywane jako figury w przestrzeni (w której są one "zanurzone" i która nadaje im cechy charakterystyczne dla danej przestrzeni) nazywa się *grafami geometrycznymi*.

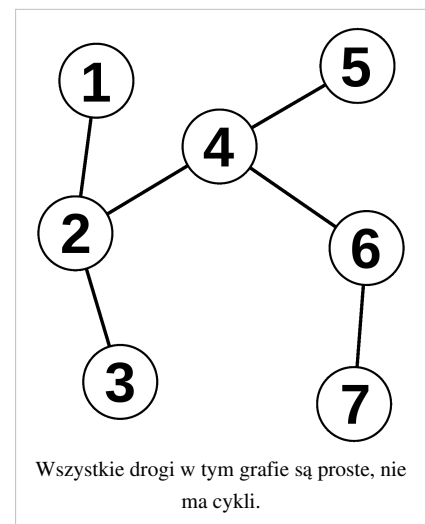
Pojęcia służące do opisu grafów

Alfabetyczna lista definicji

- *Acentryczność wierzchołka grafu*

To maksymalna odległości wierzchołka do innych wierzchołków grafu, lub inaczej długość najdłuższej ścieżki prostej zaczynającej się w danym wierzchołku.

- *Cykl*



Zamknięta droga prosta e_a, e_b, \dots, e_z , taka, że krawędź e_z kończy się w początkowym wierzchołku drogi

- *Droga*

Wyznaczona przez krawędzie *trasa* polegająca na podróżowaniu od wierzchołka do wierzchołka po łączących je krawędziach. Jeżeli przez e_i oznaczy się i -tą krawędź grafu, to droga może być jednoznacznie zapisana jako e_a, e_b, \dots, e_z

- *Droga prosta*

Droga nie zawierająca dwóch tych samych krawędzi

- *Długość drogi/ścieżki*

To liczba krawędzi/wierzchołków tworzących daną drogę/ścieżkę

- *Droga acykliczna*

Droga nie zawierająca cyklu

- *Gęstość grafu*

Stosunek liczby krawędzi do największej możliwej liczby krawędzi: $\frac{2|E|}{|V|(|V| - 1)}$.

Używa się również określeń: *graf gęsty*, jeżeli ma on *dużo* krawędzi w stosunku do liczby wierzchołków i podobnie *graf rzadki*, jeżeli ma on *mało* krawędzi w stosunku do liczby wierzchołków. Przy czym znaczenie słów mało i dużo może zależeć od kontekstu.

- *Klika*

Podzbiór wierzchołków danego grafu, z których każdy jest sąsiadem każdego innego (czyli podgraf pełny).

- *Kolorowanie grafu*

To nadanie każdemu wierzchołkowi *koloru*, tak by żadne sąsiadujące ze sobą wierzchołki nie były *pokolorowane* tym samym kolorem.

- *Krawędzie sąsiednie*

Krawędzie kończące się w jednym wierzchołku. W przypadku grafów skierowanych zazwyczaj wymagana jest "zgodność kierunków" krawędzi, tj. dwie krawędzie są sąsiednie, jeżeli odpowiednio kończą się i zaczynają w tym samym wierzchołku.

- *Krawędź/wierzchołek krytyczny*

Krawędź/wierzchołek, po usunięciu której/którego ze zbioru pokrywającego zmniejsza się indeks pokrycia krawędziowego/wierzchołkowego.

- *Liczba chromatyczna*

Najmniejsza liczba kolorów potrzebna do prawidłowego pokolorowania grafu.

- *Nadgraf grafu H*

Taki graf, że H jest jego podgrafem.

- *Pętla*

Krawędź zaczynająca i kończąca się w tym samym wierzchołku

- *Podgraf grafu H*

Graf G uzyskany poprzez usunięcie części wierzchołków z H , wraz z kończącymi się w nich krawędziami

- *Graf r -regularny:*

Graf, w którym każdy wierzchołek grafu jest stopnia r .

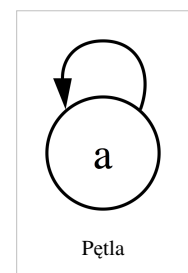
- *Sąsiad:*

Dwa wierzchołki są sąsiadami, jeśli istnieje krawędź pomiędzy nimi.

- *Spójna składowa grafu G*

Spójna składowa grafu to możliwie największy *spójny* podgraf grafu G . Graf spójny ma jedną spójną składową.

- *Stopień wierzchołka v*



Liczba kończących się w nim krawędzi. Oznaczenie: $\deg(v)$. W przypadku grafów skierowanych mówi się o stopniach wejściowym i wyjściowym – $\deg_{In}(v)$, $\deg_{Out}(v)$

- *Ściana*

Obszar zamknięty wyznaczony przez krawędzie grafu (tzw. krawędzie tworzące ścianę). Z pojęciem ściany ściśle powiązane jest twierdzenie Eulera.

Uwaga! Za ścianę uważa się też nieskończony obszar znajdujący się "na zewnątrz" grafu (a więc **każdy** graf ma co najmniej jedną ścianę)!

- *Ściany sąsiadujące*

Ściany są *sąsiadujące*, jeżeli mają co najmniej jedną wspólną krawędź tworzącą.

- *Ścieżka*

Intuicyjnie jest bardzo podobna do *drogi*, z tym, że jest wyznaczona przez *wierzchołki*, tj. można ją opisać poprzez ciąg wierzchołków v_a, v_b, \dots, v_z

- *Ścieżka prosta*

Ścieżka wyznaczona tak, by żaden wierzchołek na *trasie* nie powtarzał się

- *Ścieżka zamknięta*

Ścieżka $v_a, v_b, \dots, v_z, v_a$, czyli kończąca się w początkowym wierzchołku

- *Usunięcie wierzchołka*

Przez usunięcie wierzchołka rozumie się *wymazanie* go, oraz wszystkich kończących się w nim krawędzi z danego grafu

- *Waga krawędzi*

Często od grafu reprezentującego np. sieć połączeń komunikacyjnych oczekuje się nie tylko informacji o istniejącym połączeniu (krawędzi lub ścieżki), ale też o np. długości połączenia. Wprowadza się wtedy *wagi*, wartość przypisaną każdej krawędzi. Graf taki można wykorzystać np. do wyznaczenia optymalnej, w sensie przejechanych kilometrów trasy, lub, ogólniej rozwiązanie problemu komiwojagera, wyznaczenia optymalnego rozłożenia kabli w sieci, koordynowania wysyłania plików metodą peer to peer itp.

- *Wierzchołek izolowany*

Wierzchołek o stopniu 0, czyli nie będący końcem żadnej krawędzi.

- *Wierzchołek pokrywający krawędź*

Wierzchołek v *pokrywa* krawędź e , jeżeli e kończy się w v . W analogiczny sposób definiuje się krawędź pokrywającą dany wierzchołek – krawędź e kryje wierzchołek v , gdy się w nim kończy.

- *Minimalny pokrywający podzbiór krawędzi/wierzchołków*

To możliwie najmniejszy podzbiór krawędzi/wierzchołków grafu, taki, że pokrywają one wszystkie wierzchołki/krawędzie danego grafu.

Liczność minimalnego zbioru pokrywającego krawędzi/wierzchołków nazywa się *indeksem pokrycia wierzchołkowego/krawędziowego*. Wszystkie podzbiory o tej liczności i własności nazywa się *pokryciem minimalnym*.

- *Wierzchołek rozspajający*

Wierzchołek, po usunięciu którego zwiększa się liczba spójnych składowych grafu. Nazywany przegubem tworzy "*wąskie gardło*" grafu – tj. istnieją w grafie dwa wierzchołki takie, że każda łącząca je droga musi przejść przez wierzchołek rozspajający.

- *Most*

Krawędziowy "odpowiednik" wierzchołka rozpadającego – krawędź, po usunięciu której wzrasta liczba spójnych składowych grafu.

Oznaczenia formalne

Często dla danego grafu G stosuje się skrócone oznaczenia oparte na alfabecie greckim oraz łacińskim:

$n = |V(G)|$ liczba wierzchołków G

$m = |E(G)|$ liczba krawędzi G

$\delta(G)$ najmniejszy stopień wierzchołka w G

$\Delta(G)$ największy stopień wierzchołka w G

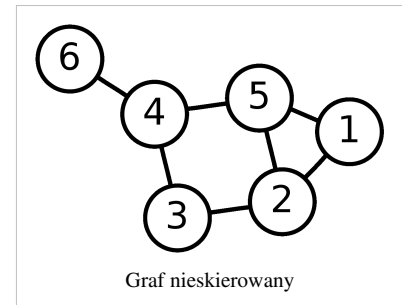
$\chi(G)$ liczba chromatyczna G

$\chi'(G)$ indeks chromatyczny G

$\omega(G)$ liczba spójnych składowych G

Przykład dla konkretnego grafu

To przykład grafu nieskierowanego G wraz z jego ilustracją:



$$V(G) = \{v_1, v_2, v_3, v_4, v_5, v_6\}$$

$$E(G) = \{\{v_1, v_2\}, \{v_1, v_5\}, \{v_5, v_4\}, \{v_4, v_6\}, \\ \{v_4, v_3\}, \{v_3, v_2\}, \{v_2, v_5\}\}$$

Jego własności:

- Przykładową ścieżką prostą może być v_6, v_4, v_5, v_1 a cyklem v_5, v_4, v_3, v_2, v_5
- Stopnie wierzchołków:

$$\deg(v_1) = 2$$

$$\deg(v_2) = 3$$

$$\deg(v_3) = 2$$

$$\deg(v_4) = 3$$

$$\deg(v_5) = 3$$

$$\deg(v_6) = 1$$

- Krawędź $\{v_1, v_5\}$ jest sąsiednia z $\{v_5, v_2\}$, ale nie jest z $\{v_2, v_3\}$
- Graf G ma trzy ściany – zewnętrzną oraz dwie wyznaczone odpowiednio przez ścieżki np. v_2, v_3, v_4, v_5 i v_1, v_5, v_2
- Graf G jest spójny, czyli ma jedną spójną składową. Natomiast podgraf grafu G , składający się z wierzchołków v_1, v_2, v_5, v_6 i incydentnych z nimi krawędziami, ma dwie spójne składowe – cykl v_1, v_2, v_5, v_1 i wierzchołek izolowany v_6 .

Izomorfizm i homeomorfizm grafów

Graficzna reprezentacja grafów (w postaci kropek i łączących je krzywych) jest tylko sposobem przedstawienia relacji zachodzącej między wierzchołkami. Dla każdego grafu istnieje nieskończenie wiele przedstawiających go jednoznacznie wykresów, *rysunków*. Co więcej, właściwości grafów (takie jak większość podanych w następnej sekcji) są niezależne od sposobu numerowania wierzchołków, kolejności ich rysowania itp. Grafy różniące się tylko sposobem ich przedstawienia, lub indeksami nadanymi wierzchołkom, nazywamy *izomorficznymi*.

Dwa grafy są *homeomorficzne*, jeśli z jednego grafu można otrzymać drugi zastępując wybrane krawędzie łańcuchami prostymi lub łańcuchy proste pojedynczymi krawędziami. Mówiąc obrazowo, chodzi o *dorysowywanie* na krawędziach dowolnej liczby wierzchołków, bądź *wymazywanie* ich.

Klasy grafów

Grafy można podzielić ze względu na różne własności, zazwyczaj zachowane w obrębie izomorfizmów danego grafu. Najczęściej dotyczą one tylko grafów prostych (nie zawierających pętli i krawędzi wielokrotnych), część z tych własności można rozszerzyć na multigrafy. Najczęściej spotykane klasy grafów to:

- graf prosty (właściwy)

Graf nie zawierający pętli ani krawędzi wielokrotnych. Graf nieprosty nazywany jest *multigrafem*. Z reguły zdanie *G jest grafem* oznacza w domyśle, że *G* jest grafem prostym
- graf pełny

Graf, którego każdy wierzchołek jest połączony bezpośrednio krawędzią z każdym innym. Graf pełny o n wierzchołkach oznacza się K_n .
- graf regularny stopnia k

Graf, którego każdy wierzchołek jest stopnia k
- graf kubiczny

Specjalne określenie dla grafów regularnych stopnia 3
- graf acykliczny

Graf nie zawierający żadnej *drogi zamkniętej*
- graf spójny

Graf, w którym dla każdego wierzchołka istnieje *droga* do każdego innego wierzchołka
- graf k -spójny

Graf posiadający k spójnych składowych
- drzewo

Spójny graf acykliczny
- las

Graf, którego wszystkie spójne składowe są drzewami
- graf dwudzielny

Graf, którego wierzchołki mogą być podzielone na dwa zbiory, tak by w obrębie jednego zbioru żaden wierzchołek nie był połączony z innym
- *graf dwudzielny pełny*

Graf dwudzielny taki, że każdy wierzchołek z jednego zbioru jest połączony krawędzią z każdym wierzchołkiem ze zbioru drugiego. Pełny graf dwudzielny o $n_1 + n_2$ wierzchołkach oznacza się K_{n_1, n_2}
- graf k -dzielny

To naturalne rozszerzenie klasy grafów dwudzielnych – jest to graf, którego zbiór wierzchołków można podzielić na k parami rozłącznych podzbiorów takich, że żadne dwa węzły należące do tego samego zbioru nie są połączone krawędzią

- *pełny graf k -dzielny*

Jeżeli zbiór wierzchołków dzieli się na k nie połączonych między sobą podzbiorów wierzchołków, to jeżeli dla każdego wierzchołka v_i z j -tego przedziału v_i jest połączony z każdym wierzchołkiem z każdego z przedziałów poza j , to jest to pełny graf k -dzielny

- graf eulerowski

Graf posiadający *drogę prostą* przechodzą przez każdą *krawędź*.

- graf hamiltonowski

Graf posiadający *ścieżkę prostą* przechodzą przez każdy *wierzchołek*.

- graf planarny

Graf, dla którego istnieje *graf izomorficzny*, który można przedstawić na płaszczyźnie tak, by żadne krawędzie się nie przecinały (oczywiście, nie w sensie "spotkania się" w jednym wierzchołku).

Kazimierz Kuratowski udowodnił, że grafy pełne K_5 i $K_{3,3}$ są nieplanarne, oraz że każdy inny graf nieplanarny musi posiadać podgraf homeomorficzny z którymś z tych grafów.

- graf płaski

To izomorficzne przedstawienie grafu takie, że żadne dwie krawędzie się nie przecinają

- graf platoński

Graf, którego przedstawienie tworzy siatkę wielościanu foremego

- graf komórkowy

Graf płaski, którego wszystkie ściany są utworzone przez drogi zamknięte tej samej długości

- graf krytyczny

Graf, którego każdy wierzchołek/krawędź jest krytyczny/krytyczna

- graf symetryczny

Graf skierowany taki, że jeżeli istnieje krawędź (u, v) to istnieje też krawędź (v, u) . Graf asymetryczny ma własność: jeżeli istnieje krawędź (u, v) to nie istnieje krawędź (v, u)

- graf podstawowy grafu skierowanego

To niemal ten sam, ale nieskierowany, bo bez zwrotów na krawędziach

- turniej

To graf pełny, w którym zorientowano krawędzie, lub inaczej, graf skierowany którego graf podstawowy jest grafem pełnym

Operacje na grafach

Operacje binarne

Suma grafów

Definicja: $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$

Jeżeli dane są dwa grafy $G_1 = (V_1, E_1)$ oraz $G_2 = (V_2, E_2)$, to ich sumą jest graf, którego zbiór wierzchołków i krawędzi tworzą wszystkie wierzchołki i krawędzie tych grafów.

Przecięcie grafów

Definicja: $G_1 \cap G_2 = (V_1 \cap V_2, E_1 \cap E_2)$

Jest definiowane analogicznie do sumy. Jeżeli dane są dwa grafy $G_1 = (V_1, E_1)$ i $G_2 = (V_2, E_2)$, to ich przecięciem jest graf, którego wierzchołki i krawędzie wchodzą w skład obu tych grafów.

Zespolenie grafów

Definicja: $G_1 + G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup \{\{v_1, v_2\} : v_1 \in V_1, v_2 \in V_2\})$

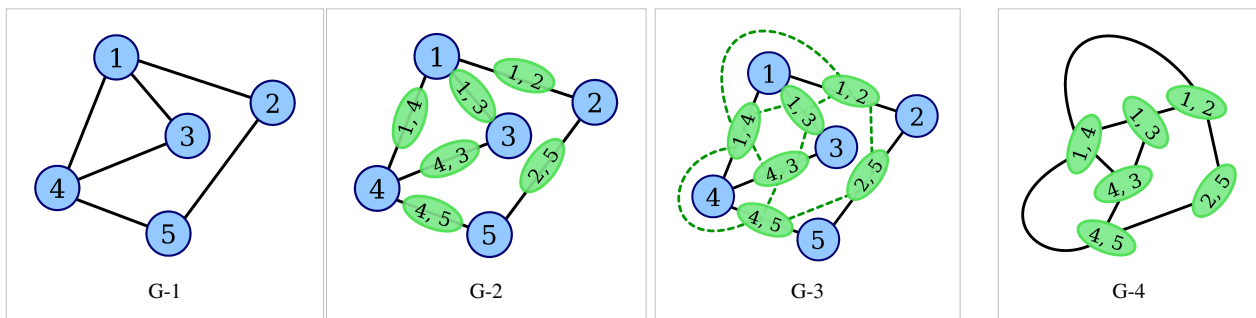
Zespoleniem grafów $G_1 = (V_1, E_1)$ i $G_2 = (V_2, E_2)$ nazywamy graf w którym z każdego wierzchołka G_1 poprowadzono krawędzie do każdego wierzchołka G_2 .

Operacje unarne

- graf krawędziowy grafu prostego G

Graf który dla każdej krawędzi z G ma wierzchołek połączony z wierzchołkami reprezentującymi pozostałe krawędzie sąsiadujące ze sobą w G .

Etapy konstrukcji grafu krawędziowego:



- dopełnienie grafu

Dopełnieniem grafu G nazywamy graf \overline{G} w którym dwa wierzchołki są sąsiednie wtedy i tylko wtedy, gdy nie były sąsiednie w G . Inaczej mówiąc w dopełnieniu dwa wierzchołki są połączone krawędzią wtedy, gdy nie były połączone w grafie wyjściowym.

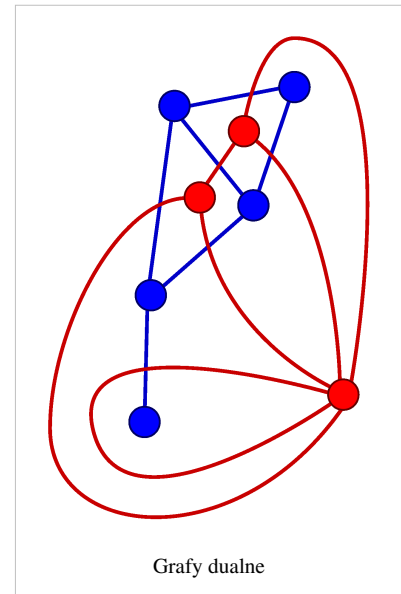
- graf dualny grafu G

Graf, którego wierzchołki odpowiadają ścianom w G . Wierzchołki te są połączone, jeżeli odpowiednie ściany w G są sąsiednie.

Dopisek: rysunek tłumaczy doskonale jak zrobić graf dualny do grafu planarnego, dla grafu nieplanarnego musimy znaleźć dwuwymiarową przestrzeń (osadzoną w wielowymiarze) w której ten graf jest "planarny" – na przykład K_5 nie można bez przecięć narysować na kuli, ale da się na torusie i tam możemy znaleźć jego graf dualny

- domknięcie przechodnie grafu G

Graf posiadający te same wierzchołki co G ; dowolne dwa wierzchołki są w nim połączone wtedy i tylko wtedy, gdy w G istnieje między nimi droga.



Sposoby reprezentacji grafów

Każdy graf może być jednoznacznie reprezentowany na wiele sposobów. Dla człowieka w przypadku grafów o "rozsądnej" liczbie wierzchołków i krawędzi najwygodniejszy jest rysunek grafu. Pozostałe sposoby reprezentacji wykorzystywane są w komputerach. Każda z tych reprezentacji ma swoje wady i zalety, generalnie ograniczające są dwa warunki – ilość pamięci przeznaczona na reprezentację i jej możliwości szybkiego odpowiadania na pytania typu *czy między wierzchołkami v i u jest krawędź?*. W przypadku grafów rzadkich listy sąsiedztwa okazują się wystarczająco szybkie by zrezygnować z pamięciożernych tablic.

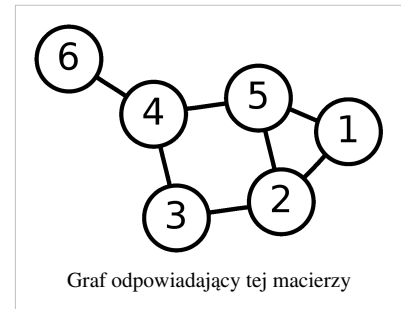
Najwygodniejszy dla człowieka jest rysunek grafu, reprezentujący wierzchołki i łączące je krawędzie (rys. obok). Wierzchołki oznaczane są zazwyczaj kropkami lub kołami, niekiedy zawierającymi indeksy bądź inne dodatkowe informacje. Krawędzie reprezentowane są krzywymi bądź prostymi, w przypadku krawędzi ważonych, waga umieszczona jest bezpośrednio nad krawędzią.

Innymi najczęściej stosowanymi metodami reprezentacji grafów są macierze sąsiedztwa, listy sąsiedztwa i macierze incydencji. W przypadku implementacji algorytmów grafowych wybiera się tę metodę, za pomocą której dla danego problemu uzyska się program działający z mniejszą złożonością obliczeniową.

Macierz sąsiedztwa

Najprostszą ze struktur danych umożliwiającą przedstawienie skomplikowanego grafu lub jego przechowywanie w pamięci komputera jest **macierz sąsiedztwa**, zawierająca dane na temat połączeń między wierzchołkami. Macierz jest rozmiaru $|V(G)|$ na $|V(G)|$, wyraz leżący z i -tego wiersza i j -tej kolumny zawiera wartość będącą liczbą krawędzi łączących i -ty i j -ty wierzchołek. Sposób ten pozwala na reprezentację zarówno grafów prostych, jak i grafów zawierających krawędzie wielokrotne oraz pętle własne. W przypadku grafów prostych wyrazami w macierzy będą wartości boole'owskie – *jest krawędź*, bądź *nie ma krawędzi*).

Macierz sąsiedztwa dla rozważanego wcześniej grafu nieskierowanego:



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Aby dowiedzieć się, ile krawędzi łączy wierzchołki v_i i v_j , wystarczy sprawdzić wartość komórki $A[i, j]$.

Tak zaimplementowana komputerowa struktura danych gwarantuje, że operacje sprawdzenia, czy $(v_i, v_j) \in E(G)$, dodania oraz usunięcia krawędzi odbywają się w stałym czasie. Do jej wad należy duża ilość potrzebnej pamięci – $O(n^2)$, oraz fakt, że czas potrzebny do przejrzenia zbioru krawędzi jest proporcjonalny do kwadratu liczby wierzchołków (złożoność obliczeniowa wynosi $O(n^2)$, zamiast do liczby krawędzi).

Lista sąsiedztwa

Drugą popularną reprezentacją grafu są tzw. listy sąsiedztwa – dla każdego wierzchołka zapamiętywana jest lista sąsiadujących z nim wierzchołków, np.:

$$v_1 \rightarrow v_2, v_5$$

$$v_2 \rightarrow v_1, v_5, v_3$$

$$v_3 \rightarrow v_2, v_4$$

$$v_4 \rightarrow v_3, v_5, v_6$$

$$v_5 \rightarrow v_1, v_2, v_4$$

$$v_6 \rightarrow v_4$$

W implementacji tej metody stosuje się listy jednokierunkowe oraz jednowymiarową tablicę wskaźników o rozmiarze $|V(G)|$, gdzie i -ty element tablicy jest wskaźnikiem do początku listy przechowującej sąsiadów i -tego wierzchołka.

W odróżnieniu od macierzy sąsiedztwa, lista sąsiedztwa wymaga ilości pamięci proporcjonalnej do liczby krawędzi, także przejrzenie całego zbioru krawędzi jest proporcjonalne do jego rozmiaru. W stosunku do macierzy sąsiedztwa większą złożoność mają jednak operacje elementarne – sprawdzenie, czy $\{v_i, v_j\} \in E(G)$ wymaga czasu proporcjonalnego do mniejszego ze stopni wierzchołków, a np. usunięcie krawędzi – do większego z nich.

Macierz incydencji

Macierz incydencji M wymiaru $|V(G)|$ na $|E(G)|$ zawiera informacje takie, że $M_{i,j} = 1$ tylko, gdy j -ta krawędź kończy się w i -tym wierzchołku (czyli jest z nim incydentna). W przeciwnym wypadku $M_{i,j} = 0$. Niech

$$e_1 = \{1, 2\}$$

$$e_2 = \{1, 5\}$$

$$e_3 = \{5, 4\}$$

$$e_4 = \{4, 6\}$$

$$e_5 = \{4, 3\}$$

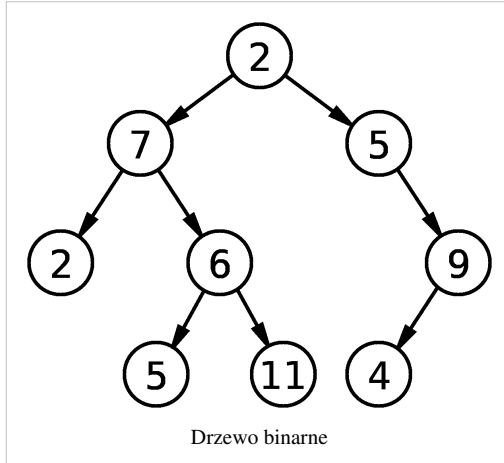
$$e_6 = \{3, 2\}$$

$$e_7 = \{5, 2\}$$

oznaczają wszystkie krawędzie grafu z przykładu. Macierz incydencji o kolumnach e_i i wierszach v_i może wyglądać tak:

$$M = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Zastosowania



Kiedy rozwój informatyki pozwolił na reprezentowanie grafów za pomocą komputera, okazało się, że algorytmy na nich oparte znajdują wiele praktycznych zastosowań. Szczególny rodzaj grafów zwanych drzewami okazał się przydatny do reprezentacji hierarchii. Przedstawione na rysunku obok drzewo binarne może opisywać np. mistrzostwa sportowe czy drzewo genealogiczne, a po dodaniu etykiet może służyć np. do tworzenia kodów Huffmana, do opisu rozwoju populacji bakterii w laboratorium albo niedeterministycznego automatu skończonego.

Kiedy komputery stały się powszechne okazało się, że grafy można zastosować w wielu problemach. Jako graf przedstawiono sieć dróg. Skrzyżowania stały się wierzchołkami grafu, a ulice

jego krawędziami. Potem w podobny sposób przedstawiono sieci pomieszczeń i korytarzy w budynkach. Taka reprezentacja pozwoliła komputerom na poszukiwanie najlepszej drogi ze swojego obecnego położenia do pożądanego celu. Oprogramowanie oparte na algorytmach analizujących grafy znalazło zastosowanie w przenośnych urządzeniach PDA wyposażonych w GPS, które potrafią wskazać kierowcy trasę w nieznanym mieście.

Innym przykładem wykorzystania grafów stały się gry komputerowe, gdzie system sztucznej inteligencji musiał odszukać najlepszą drogę dla postaci sterowanych przez program, która pozwoli zaatakować ludzkiego przeciwnika. Sztuczna inteligencja mogła rozwiązać to zagadnienie tylko dzięki odpowiedniej reprezentacji mapy wirtualnego otoczenia jako grafu.

Projektanci robotów mobilnych również skorzystali z podobnych algorytmów, aby ich maszyny mogły bez udziału człowieka odnaleźć trasę w trudnym terenie. Przedstawienie sieci komputerowych w postaci grafów pozwoliło na

stworzenie oprogramowania usprawniającego trasowanie w Internecie.

Aby zwiększyć wydajność pracy w dużych organizacjach, realizację zleczanych przez klientów zadań przedstawiono w postaci grafów. Pracownikom odpowiadać mogą wierzchołki, a przepływ zadań między nimi opisać można za pomocą krawędzi. Zaprojektowano oprogramowanie pozwalające automatycznie śledzić pracę tak opisaną organizacji, co miało służyć wzrostowi wydajności. Rozwiązania tego typu znalazły zastosowanie w działach wsparcia technicznego klientów dużych korporacji.

Uogólnienia

W hipergrafach krawędź może łączyć więcej niż dwa wierzchołki.

Geometryczny graf nieskierowany może być traktowany jako kompleks symplecjalny złożony z 1-sympleksów (krawędzi) i 0-sympleksów (wierzchołków). Tym samym kompleksy symplecjalne są uogólnieniem grafów, gdyż pozwalają też na większą liczbę wymiarów.

W teorii modeli graf jest szczególnym przypadkiem struktury matematycznej. W tym przypadku jednak nie ma ograniczenia na liczbę krawędzi: może to być dowolna liczba kardynalna.

Przypisy

[1] patrz Wright i Ross w bibliografii

Bibliografia

- Kenneth A. Ross, Charles R.B. Wright: *Matematyka dyskretna*. z ang. przeł. E. Sepko-Guzicka, W. Guzicki, P. Zakrzewski. Warszawa: Wydaw. Naukowe PWN, 2005. ISBN 83-0114-380-0. (pol.)
- Robin J. Wilson: *Wprowadzenie do teorii grafów*. z ang. tł. Wojciech Guzicki. Warszawa: Wydaw. Naukowe PWN, 1998. ISBN 83-0112-641-8. (pol.)
- Witold Lipski, Tomasz Czajka: *Kombinatoryka dla programistów*. Warszawa: Wydawnictwa Naukowo-Techniczne, 2004. ISBN 83-204-2968-4. (pol.)
- Juliusz Lech Kulikowski: *Zarys teorii grafów : zastosowanie w technice*. Warszawa: Państw. Wydaw. Naukowe, 1986. ISBN 83-0105-277-5. (pol.)
- Marek Kubale: (red.) *Optymalizacja dyskretna. Modele i metody kolorowania grafów*. Wydawnictwa Naukowo-Techniczne, 2002. ISBN 83-204-2747-9. (pol.)

Linki zewnętrzne

- Reinhard Diestel: Graph Theory (<http://www.math.uni-hamburg.de/home/diestel/books/graph.theory/>) (ang.). [dostęp 24 stycznia 2009]. – Książka dostępna w internecie na temat teorii grafów
- Chris K. Caldwell: Graph theory tutorial (<http://www.utm.edu/departments/math/graph/>) (ang.). [dostęp 24 stycznia 2009].
- Algorithms. Tutoring web page (<http://students.ceid.upatras.gr/~papagel/project/contents.htm>) (ang.). [dostęp 24 stycznia 2009]. – Aplety obrazujące niektóre algorytmy teorii grafów
- Example graphs (<http://www.aisee.com/graphs/>) (ang.). [dostęp 24 stycznia 2009]. – Galeria grafów reprezentujących rzeczywiste problemy

Algorytm Grahama

Algorytm Grahama to efektywny algorytm wyszukiwania otoczki wypukłej skończonego zbioru punktów płaszczyzny; nie istnieją warianty dla przestrzeni o wyższych wymiarach. Pomysłodawcą algorytmu jest Ronald Graham^[1].

Czasowa złożoność obliczeniowa wynosi $O(n \log n)$.

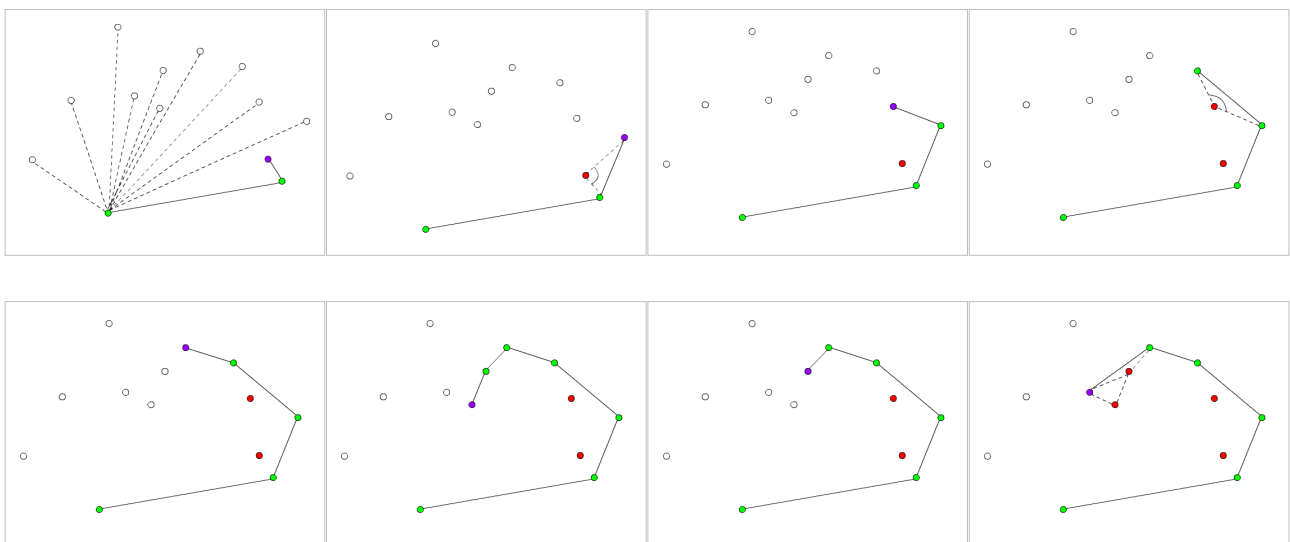
Algorytm przebiega następująco:

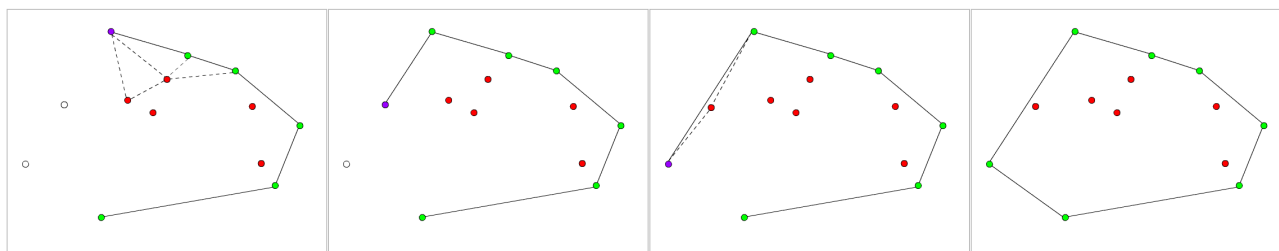
1. Wybierz dowolny punkt (ozn. O) należący do otoczki wypukłej punktów.
2. Przesuń wszystkie punkty tak, by punkt O pokrył się z początkiem układu współrzędnych.
3. Posortuj punkty leksykograficznie względem:
 - kąta pomiędzy wektorem OP_i a dodatnią osią układu współrzędnych,
 - odległości punktu P_i od początku układu współrzędnych.
4. Wybierz punkt (ozn. S) o najmniejszej współrzędnej y ; jeśli kilka punktów ma tę samą współrzędną y , wybierz spośród nich ten o najmniejszej współrzędnej x .
5. Przeglądaj listę posortowanych punktów poczynając od punktu S :
 - Od bieżącej pozycji weź trzy kolejne punkty (ozn. A, B, C).
 - Jeśli punkt B leży na zewnątrz trójkąta AOC , to należy do otoczki wypukłej. Przejdź do następnego punktu na liście.
 - Jeśli punkt B leży wewnątrz trójkąta AOC , to znaczy, że nie należy do otoczki. Usuń punkt B z listy i cofnij się o jedną pozycję (o ile bieżąca pozycja jest różna od początkowej).

Ze względu na wcześniejsze kroki algorytmu (sortowanie i sposób wybierania analizowanych trójek punktów) dwa z trzech warunków należenia punktu B do trójkąta AOC są spełnione, tj. B leży po "wewnętrznej" względem trójkąta stronie prostych OA i OC . Zatem do stwierdzenia przynależności do trójkąta wystarczy zbadać, po której stronie odcinka AC znajduje się punkt B , w tym celu wykorzystuje się znak iloczynu wektorowego $|C - A| \times |B - A|$.

W praktyce można również utożsamić krok 4. z 1., tzn. przyjąć, że $O = S$.

Przykładowy przebieg algorytmu





Przypisy

- [1] Ronald Graham. *An efficient algorithm for determining the convex hull of a finite planar set*. „Information Processing Letters”. 1, ss. 132-133, 1972 (ang.).

Bibliografia

- Mark de Berg, Mirosław Kowaluk: *Geometria obliczeniowa : algorytmy i zastosowania*. Warszawa: Wydawnictwa Naukowo-Techniczne, 2007. ISBN 978-83-204-3244-2.
- Lech Banachowski, Krzysztof Diks, Wojciech Rytter: *Algorytmy i struktury danych*. Warszawa: Wydawnictwa Naukowo-Techniczne, 2003, ss. 263-267. ISBN 83-204-2796-7.

AJAX

AJAX (ang. *Asynchronous JavaScript and XML*, asynchroniczny JavaScript i XML) – technologia tworzenia aplikacji internetowych, w której interakcja użytkownika z serwerem odbywa się bez przeładowywania całego dokumentu, w sposób asynchroniczny. Ma to umożliwiać bardziej dynamiczną interakcję z użytkownikiem niż w tradycyjnym modelu, w którym każde żądanie nowych danych wiąże się z przesłaniem całej strony HTML.

Podstawowe elementy AJAX

Na technologię tę składa się parę elementów:

- XMLHttpRequest - klasa umożliwiająca asynchroniczne przesyłanie danych; dzięki asynchroniczności w trakcie pobierania danych użytkownik może wykonywać inne czynności, można także pobierać dane jednocześnie z wielu miejsc.
- JavaScript - mimo użycia w nazwie, może to być *de facto* dowolny język skryptowy funkcjonujący po stronie użytkownika (np. JScript czy VBScript).
- XML - język znaczników, poprzez który miałyby być opisane odbierane informacje. W praktyce jednak dane często przekazywane są w innym formacie, przy czym odbierane są wtedy jako tekst. Mogą to być zarówno gotowe fragmenty HTML, jak i fragmenty kodu JavaScript (zob. JSON), może to być też format specyficzny dla danego zastosowania.

Teoretycznie są to wszystkie wymagane elementy, jednak w praktyce używane są jeszcze odpowiednie skrypty funkcjonujące po stronie serwera i współpracujące z bazą danych. Można sobie jednak bez nich poradzić, jeśli wszystkie potrzebne dane zostały już wcześniej wygenerowane (np. zawartość poszczególnych stron prostego serwisu).

Wady i ograniczenia

- Udostępnianie treści strony poprzez język skryptowy ogranicza dostęp do niej dla części użytkowników. Dotyczy to zarówno osób celowo blokujących sobie skrypty (np. ze względu na wysokie obciążenie komputera), jak i używających czytników ekranowych (w których obsługa skryptów może być mocno ograniczona). Problem ten można obejść udostępniając alternatywne, bardziej tradycyjne rozwiązania przynajmniej dla podstawowych funkcji serwisu internetowego. To jednak znacząco zwiększa koszty wprowadzania nowych rozwiązań i np. portale posiadające obsługę kont pocztowych udostępniają czasem starsze wersje interfejsu.
- Utrudnione jest automatyczne pobieranie stron, gdyż programy takie nie interpretują zwykle języków skryptowych. Możliwość dowolnego pobierania zawartości serwisu nie musi być jednak korzystna z punktu widzenia właścicieli serwisu.
- Bezpośrednie indeksowanie przez serwisy wyszukiwujące może być utrudnione, jednak wystarczy zadbać o dostarczanie linków wyświetlających całą treść strony lub stworzyć mapę witryny.
- Część starych skryptów do analizy ruchu na stronie oparta jest o klasyczny model udostępniania całych stron (konieczność odświeżenia całości). Nowsze skrypty potrafią jednak uwzględnić właściwy pomiar oglądalności stron. Można również bez przeszkód analizować logi żądań wysyłanych do serwera WWW.
- Wadą rozwiązań w znaczącym stopniu opartych na AJAX jest fakt, że przestaje funkcjonować tradycyjny schemat przeglądania stron umożliwiający swobodne poruszanie się w przód i w tył. Na przykład jeśli kliknięcie w link powoduje wywołanie skryptu zmieniającego wnętrze strony (menu pozostaje bez zmian), to użytkownik nie będzie mógł się cofnąć korzystając z przycisku "Wstecz" przeglądarki. Twórcy serwisu WWW mogą jednak zbudować analogiczny mechanizm rozwijając go nawet do wycofywania zmian w konkretnym fragmencie strony.

Przykład utworzenia nowego obiektu XMLHttpRequest

```
function ajaxFunction() {  
    var xmlhttp;  
    try{  
        // Utworzenie obiektu XMLHttpRequest (silnik Gecko, WebKit, Presto,  
        Trident w IE>6)  
        xmlhttp=new XMLHttpRequest();  
    } catch(e) {  
        // Wyłapuje błąd jeśli JavaScript nie posiada obiektu  
        XMLHttpRequest  
        try {  
            // Utworzenie obiektu ActiveXObject, który jest zawarty w  
            kontrolce ActiveX IE  
            xmlhttp=new ActiveXObject("Msxml2.XMLHTTP");  
        } catch(e) {  
            try {  
                // Utworzenie obiektu ActiveXObject, dla innych wersji IE  
                xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");  
            } catch(e) {  
                // Wyświetlenie błędu o braku obsługi obiektu XMLHttpRequest  
                alert("Your browser does not support AJAX!");  
                return false;  
            }  
        }  
    }  
}
```

```
// zwrócenie obiektu  
return xmlHttp;  
}
```

Biblioteki AJAX

popularne biblioteki JavaScript powiązane z AJAX-em

- Prototype - biblioteka ułatwiająca korzystanie z możliwości oferowanych przez AJAX
- jQuery - biblioteka ułatwiająca korzystanie z możliwości oferowanych przez AJAX
- Ext - dawniej rozszerzenie Prototype, JQuery oraz YUI obecnie samodzielna biblioteka
- Script.aculo.us - rozszerza Prototype ułatwiając tworzenie animacji i interfejsów
- MooTools - modułowa biblioteka AJAX zawierająca również ułatwienia do tworzenia efektów wizualnych
- Yahoo! UI Library - biblioteka narzędziowa ogólnie dla DHTML
- Dojo Toolkit - biblioteka narzędziowa ogólnie dla DHTML
- AJAX.OOP - biblioteka narzędziowa stworzona dla AJAX
- picoAjax - biblioteka prosta i szybka biblioteka JavaScript ułatwiająca korzystanie z możliwości oferowanych przez technologię AJAX.

popularne frameworki zawierające skrypty działające po stronie serwera

Nazwa	Język skryptów	Uwagi
Echo	Java	
Google Web Toolkit	Java	
IT Mill Toolkit	Java	
ItsNat	Java	
OpenXava	Java	
ZK Framework	Java	
Ajax.NET Professional	ASP.NET	
ASP.NET AJAX	ASP.NET	
Sajax	PHP	
Symfony	PHP	
Tigermouse	PHP	
Xajax	PHP	
Pyjamas	Python	

Linki zewnętrzne

- Jesse James Garrett: Ajax: A New Approach to Web Applications ^[1] ([ang.](#)). Adaptive path.
- AJAX ^[2] w DMoz
- Ajax Tutorial ^[3] ([ang.](#))
- Attacking AJAX Applications ^[4] - prezentacja dotycząca bezpieczeństwa w skryptach Ajax
- [5] - picoAjax

Przypisy

- [1] <http://www.adaptivepath.com/publications/essays/archives/000385.php>
- [2] <http://dmoz.org/Computers/Programming/Languages/JavaScript/Ajax>
- [3] <http://www.xul.fr/en-xml-ajax.html>
- [4] http://www.isecpartners.com/files/iSEC-Attacking_AJAX_Applications.BH2006.pdf
- [5] <http://www.picoajax.pl>

Hypertext Transfer Protocol

World Wide Web
Struktura stron WWW HTML, XHTML, XML, XSL
Generowanie dynamicznych stron WWW Active Server Pages, ASP.NET, JavaServer Pages, PHP
Po stronie użytkownika kaskadowe arkusze stylów, JavaScript, AJAX, kolory w Internecie
Przesyłanie danych Hypertext Transfer Protocol, HTTPS, HTTP referrer, serwer WWW, VoiceXML, XMLHttpRequest
Pojęcia architektura informacji, użyteczność, dostępność

HTTP (ang. *Hypertext Transfer Protocol* – protokół przesyłania dokumentów hipertekstowych) to protokół sieci WWW (ang. *World Wide Web*). Obecną definicję HTTP stanowi RFC 2616. Za pomocą protokołu HTTP przesyła się żądania udostępnienia dokumentów WWW i informacje o kliknięciu odnośnika oraz informacje z formularzy. Zadaniem stron WWW jest publikowanie informacji – natomiast protokół HTTP właśnie to umożliwia.

Protokół HTTP jest użyteczny, ponieważ udostępnia znormalizowany sposób komunikowania się komputerów ze sobą. Określa on formę żądań klienta (tj. np. przeglądarki www) dotyczących danych oraz formę odpowiedzi serwera na te żądania. Jest zaliczany do protokołów bezstanowych (ang. *stateless*) z racji tego, że nie zachowuje żadnych informacji o poprzednich transakcjach z klientem (po zakończeniu transakcji wszystko "przepada"). Pozwala to znacznie zmniejszyć obciążenie serwera, jednak jest kłopotliwe w sytuacji, gdy np. trzeba zapamiętać konkretny stan dla użytkownika, który wcześniej łączył się już z serwerem. Najczęstszym rozwiązaniem tego problemu jest wprowadzenie mechanizmu ciasteczek. Inne podejścia to m.in. sesje po stronie serwera, ukryte parametry (gdy aktualna strona zawiera formularz) oraz parametry umieszczone w URL-u (jak np. `/index.php?userid=3`).

HTTP standardowo korzysta z portu nr 80 (TCP).

Metody HTTP

1. GET – pobranie zasobu wskazanego przez URI, może mieć postać warunkową jeśli w nagłówku występują pola warunkowe takie jak "If-Modified-Since"
2. HEAD – pobiera informacje o zasobie, stosowane do sprawdzania dostępności zasobu
3. PUT – przyjęcie danych w postaci pliku przesyłanych od klienta do serwera
4. POST – przyjęcie danych przesyłanych od klienta do serwera (np. wysyłanie zawartości formularzy)
5. DELETE – żądanie usunięcia zasobu, włączone dla uprawnionych użytkowników
6. OPTIONS – informacje o opcjach i wymaganiach istniejących w kanale komunikacyjnym
7. TRACE – diagnostyka, analiza kanału komunikacyjnego
8. CONNECT – żądanie przeznaczone dla serwerów pośredniczących pełniących funkcje tunelowania

Metoda CONNECT nie jest częścią standardu HTTP/1.1, jednak jest powszechnie implementowana na podstawie dokumentu *internet-draft* wygasłego w 1999 roku^[1].

Typowe zapytanie HTTP

1. **GET / HTTP/1.1** (prośba o zwrócenie dokumentu o URI / zgodnie z protokołem HTTP 1.1)
2. **Host: host.com** (wymagany w HTTP 1.1 nagłówek Host służący do rozpoznania hosta, jeśli serwer na jednym IP obsługuje kilka VirtualHostów)
3. **User-Agent: Mozilla/5.0 (X11; U; Linux i686; pl; rv:1.8.1.7) Gecko/20070914 Firefox/2.0.0.7** (nazwa aplikacji klienckiej)
4. **Accept:**
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8
(akceptowane (bądź nieakceptowane dla q=0) przez klienta typy plików)
5. **Accept-Language: pl,en-us;q=0.7,en;q=0.3** (preferowany język strony – nagłówek przydatny przy Language negotiation)
6. **Accept-Charset: ISO-8859-2,utf-8;q=0.7,*;q=0.7** (preferowane kodowanie znaków, patrz strona kodowa)
7. **Keep-Alive: 300** (czas, jaki klient chce zarezerwować do następnego zapytania w przypadku połączenia *Keep-Alive*)
8. **Connection: keep-alive** (chęć nawiązania połączenia stałego *Keep-Alive* z serwerem HTTP/1.0)
9. **znak powrotu karetki i nowej linii (CRLF)**

HTTP/1.1 dopuszcza wysłanie kilku żądań naraz (pipelining). HTTP/1.0 zakłada jedno żądanie i jedną odpowiedź.

Odpowiedź serwera WWW

1. **HTTP/1.1 200 OK** (kod odpowiedzi HTTP, w tym wypadku zaakceptowanie i zwrócenie zawartości)
2. **Date: Thu, 20 Dec 2001 12:04:30 GMT** (czas serwera)
3. **Server: Apache/2.0.50 (Unix) DAV/2** (opis aplikacji serwera)
4. **Set-Cookie: PSID=d6dd02e9957fb162d2385ca6f2829a73; path=/** (nakazanie klientowi zapisania ciasteczka)
5. **Expires: Thu, 19 Nov 1981 08:52:00 GMT** (czas wygaśnięcia zawartości zwróconego dokumentu. Data w przeszłości zabrania umieszczenia dokumentu w pamięci podręcznej. Jest to stara metoda zastąpiona przez Cache-Control)
6. **Cache-Control: no-store, no-cache, must-revalidate** (no-store zabrania przechowywania dokumentu na dysku, nawet gdy nie jest to pamięć podręczna. must-revalidate nakazuje bezwzględnie stosować się do wytycznych i sprawdzić świeżość dokumentu za każdym razem)

7. **Pragma: no-cache** (informacje dotyczące zapisywania zawartości w pamięci podręcznej. Stara, niestandardowa metoda.)
8. **Keep-Alive: timeout=15, max=100**
9. **Connection: Keep-Alive** (akceptacja połączenia *Keep-Alive* dla klientów HTTP/1.0)
10. **Transfer-Encoding: chunked** (typ kodowania zawartości stosowanej przez serwer)
11. **Content-Type: application/xhtml+xml; charset=utf-8** (typ MIME i strona kodowa zwróconego dokumentu)
12. **znak powrotu karetki i nowej linii (CRLF)**
13. *tutaj zawartość dokumentu*

HTTP do obsługi połączeń *Keep-Alive* wymaga, aby odpowiedź od serwera miała znaną długość (przez podanie *Content-Length* lub użycie *Transfer-Encoding: chunked*). W przeciwnym wypadku koniec odpowiedzi sygnalizuje zerwanie połączenia i *Keep-Alive* nie może działać.

Nagłówek *Keep-Alive* jest rozszerzeniem HTTP/1.0. W HTTP/1.1 ten nagłówek nie jest potrzebny, gdyż połączenia *Keep-Alive* są domyślne (zachowanie zmienia *Connection: close*).

Przypisy

- [1] Ari Luotonen: Tunneling TCP based protocols through Web proxy servers (<http://www.web-cache.com/Writings/Internet-Drafts/draft-luotonen-web-proxy-tunneling-01.txt>). 1998.

Linki zewnętrzne

- Oficjalna specyfikacja najnowszej wersji protokołu HTTP (<http://www.w3.org/Protocols/>) ([ang.](#))
- Krótki opis HTTP (<http://www.jmarshall.com/easy/http/>) ([ang.](#))
- RFC2616 (<http://tools.ietf.org/html/2616>): Hypertext Transfer Protocol

Sesja (informatyka)

Sesja to w informatyce obiekt, zapamiętujący przez pewien czas na serwerze szczegóły dotyczące połączenia z klientem. Cechą charakterystyczną sesji jest to, że przypisane do niej dane mają przeważnie charakter chwilowy, ulotny (w przeciwieństwie np. do preferencji przypisywanych do konta klienta).

Rodzaje sesji

Sesje internetowe

Ze względu na bezstanowość protokołu HTTP, niezbędny jest sposób na każdorazowe przekazywanie informacji pomiędzy przeładowaniami witryny. Przykładem jest tutaj informacja o "byciu zalogowanym", tak aby nie trzeba było podawać za każdym razem hasła. Problem ten można rozwiązać przez rozpoczęcie sesji, która w ujęciu WWW oznacza uniwersalny, spersonalizowany worek do przechowywania danych po stronie serwera. W sesji można przechowywać dowolne wartości, np. adres IP klienta, odwiedzone przez niego podstrony, wybrane produkty w sklepie internetowym, informacja o zalogowaniu, identyfikatory itp. Dla każdego klienta tworzona jest osobna sesja, dzięki temu informacje te nie mieszają się pomiędzy różnymi użytkownikami danej strony. Serwer musi posiadać sposób na rozpoznawanie, która sesja należy do którego klienta. Osiągane jest to poprzez stosowanie identyfikatora sesji, który po stronie klienta jest na ogół przechowywany w ciasteczku lub rzadziej w treści URL-a.

Sesje w obrębie pojedynczej jednostki

Sesja nie musi obejmować zasięgiem wielu komputerów. Dany program komputerowy, czy nawet system operacyjny, może rozpoczynać sesję po uprzednim zalogowaniu się i wówczas podział na serwer i klienta nie jest już tak oczywisty. Można tu mówić o tym, że serwerem jest program (ewentualnie urządzenia wyjściowe – np. monitor), a klientem są urządzenia wejściowe (np. myszka, klawiatura).

Dane sesji w wypadku systemu operacyjnego mogą obejmować otwarte okna (uruchomione programy), ich pozycję itp. W przypadku innych programów komputerowych mogą to być np. ostatnio otwarte strony internetowe, pliki, czy projekty (grupy plików).

Zagrożenia

Przechwytywanie sesji

Przechwytywanie sesji odnosi się do prób przejęcia poprawnej sesji (klucza sesji), aby uzyskać nieautoryzowany dostęp do usług bądź informacji systemu komputerowego. Zwykle odnosi się do nieuprawnionego wejścia w posiadanie klucza sesji, wykorzystywanego do identyfikacji użytkownika podczas jego wizyty w witrynie internetowej. Jest to istotne zagadnienie przy konstruowaniu stron internetowych, gdyż ciasteczka wykorzystywane do utrzymywania sesji mogą być bardzo łatwo ukradzione, na przykład poprzez wykorzystanie technik XSS.

Dane sesji mogą być przechowywane po stronie serwera, jednak pewnego rodzaju identyfikator sesji, czyli identyfikator "bycia zalogowanym", musi być przekazywany do serwera (wysyłany zwykle poza świadomością użytkownika poprzez ciasteczka, bądź metodą GET lub POST). Przechwycenie takiego identyfikatora (klucza sesji), umożliwia osobie postronnej dostęp do danej strony, tak, jakby przeglądał ją pełnoprawny użytkownik.

Linki zewnętrzne

- opis sesji w PHP na pl.php.net ^[1] (pol.)
- (MSDN) Page.Session ^[2] (ang.)

Przypisy

[1] <http://pl.php.net/manual/pl/ref.session.php>

[2] [http://msdn2.microsoft.com/en-us/library/system.web.ui.page.session\(vs.80\).aspx](http://msdn2.microsoft.com/en-us/library/system.web.ui.page.session(vs.80).aspx)

Ciasteczko

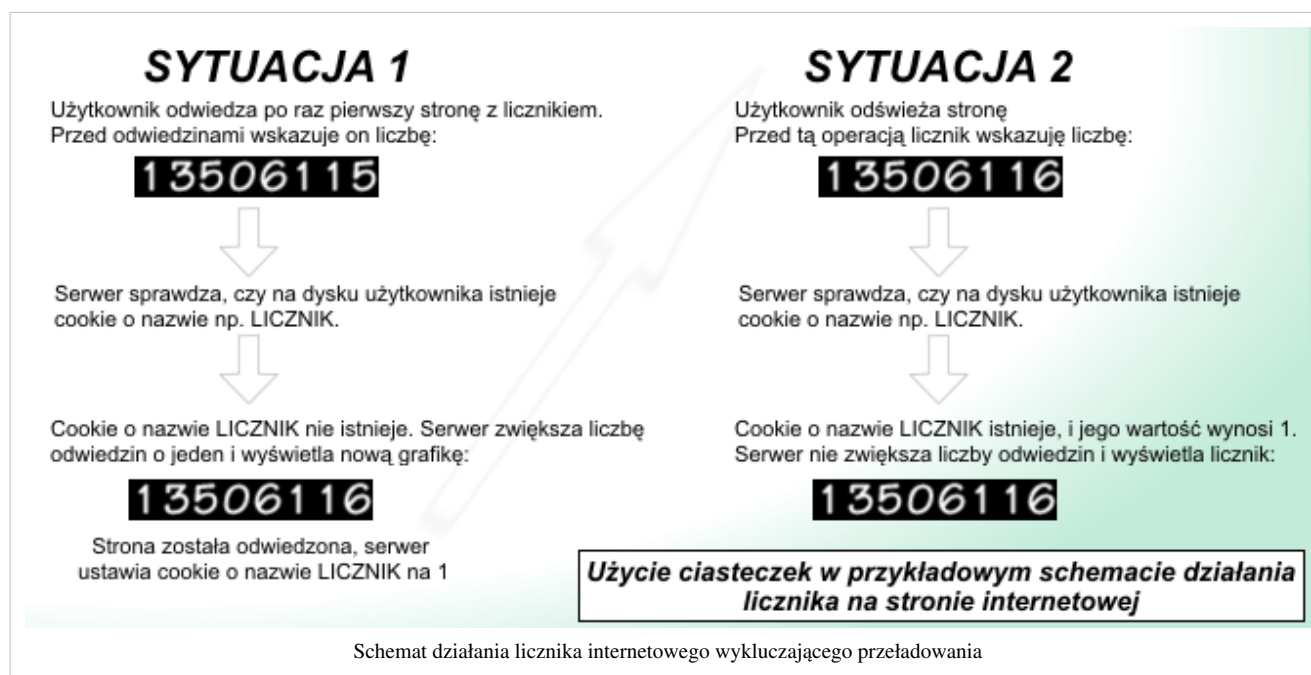
Ciasteczka (ang. *cookies*) to niewielkie informacje tekstowe, wysyłane przez serwer WWW i zapisywane po stronie użytkownika (zazwyczaj na twardym dysku). Domyślne parametry ciasteczek pozwalają na odczytanie informacji w nich zawartych jedynie serwerowi, który je utworzył. Ciasteczka są stosowane najczęściej w przypadku liczników, sond, sklepów internetowych, stron wymagających logowania, reklam i do monitorowania aktywności odwiedzających.

Mechanizm ciasteczek został wymyślony przez byłego pracownika Netscape Communications – Lou Montulliego.

Zastosowanie

Ciasteczka mogą zawierać rozmaite rodzaje informacji o użytkowniku danej strony WWW i "historii" jego łączności z daną stroną (a właściwie serwerem). Zazwyczaj wykorzystywane są do automatycznego rozpoznawania danego użytkownika przez serwer, dzięki czemu może on wygenerować przeznaczoną dla niego stronę. Umożliwia to tworzenie spersonalizowanych serwisów WWW, obsługi logowania, "koszyków zakupowych" w internetowych sklepach itp.

Zastosowanie ciasteczek do sond i liczników internetowych wygląda następująco – serwer może łatwo sprawdzić, czy z danego komputera oddano już głos lub też czy odwiedzono daną stronę, na tej podstawie wykonać odpowiednie operacje i wygenerować dla użytkownika zindywidualizowaną treść strony. Schematyczny sposób wykorzystywania ciasteczek przy obsłudze licznika internetowego, wykluczającego przeładowania (zwiększanie liczby odwiedzin przy odświeżeniu strony) przedstawiony jest poniżej:



Część serwisów korzysta z ciasteczek jako sposobu przenoszenia między stronami loginu i zaszyfrowanego hasła (rozwiązanie mniej bezpieczne), lub też spreparowanej informacji o zalogowaniu (rozwiązanie bezpieczne), dzięki czemu nie jest konieczne logowanie na każdej podstronie. Wyłączenie obsługi ciasteczek uniemożliwia często zalogowanie się, co może być rozwiązane poprzez przechowywanie danych o zalogowaniu po stronie serwera, zawsze jednak użytkownik musi zostać w jakiś sposób zidentyfikowany (np. poprzez identyfikator sesji zawarty w adresie URL).

Specyfika działania

Mechanizm ciasteczek został wprowadzony po to, by w bezstanowym protokole HTTP umożliwić odróżnienie osób odwiedzających dany serwis. Ciasteczka są informacjami zapisywanymi trwale lub tymczasowo na żądanie serwera na dysku użytkownika. Najczęściej przechowywane są w jednym pliku tekstowym lub binarnym.

Dane zapisane w ciasteczkach mają postać naprzemiennych ciągów nazwy i wartości odpowiadającej jej zmiennej. Serwer WWW chcąc wysłać żądanie utworzenia ciasteczka na dysku użytkownika dołącza do nagłówka HTTP polecenie "Set-Cookie", po którym następuje ciąg przekazywanych danych. Zapamiętane ciasteczko może najczęściej odczytać jedynie serwer, który je wysłał. W danych po poleceniu Set-Cookie określone są:

- nazwa i przypisaną jej wartość,
- domena i ścieżka dostępu, które są związane z przekazywanym ciasteczkiem,
- czas ważności danego ciasteczka (po jego upływie przeglądarka usunie je).

Do zapisania ciasteczka wymagana jest jedynie jego nazwa. Jeśli nie zostanie podana domena, do wartości zapisanych w ciasteczku dostęp będzie miał jedynie serwer, z którego wysłano żądanie zapisu. Niepodanie czasu ważności spowoduje usunięcie ciasteczka po zamknięciu przeglądarki. Ciasteczka, które wygasają po zakończonej sesji, zwane są ciasteczkami sesyjnymi. Mają one ustalony okres ważności, którego mechanizm wymusza serwer (zwykle nie można polegać bowiem na prawidłowości ustawienia zegara na komputerze z przeglądarką).

Działanie mechanizmu ciasteczek po stronie użytkownika zależy od konfiguracji jego przeglądarki. Niektóre z nich umożliwiają odmowę zapisu, inne pozwalają na ustawienie daty wygaśnięcia innej od tej deklarowanej w nagłówku HTTP. Zaawansowaną kontrolę nad zachowaniem ciasteczek posiadają m.in. Firefox, Opera i inne nowoczesne przeglądarki.

Składnia nagłówka HTTP

Nagłówek wysłany przez serwer ma następującą postać:

```
Set-Cookie: nazwa=wartość; expires=DATA; path=ŚCIEŻKA; domain=DOMENA; secure
```

`nazwa=wartość`

Wartość ta jest jedynym wymaganym atrybutem przy wysyłaniu ciasteczka. Składa się z dowolnych znaków z wyjątkiem średników, przecinków, białych spacji i slashów (/). Jeśli zajdzie potrzeba ich użycia, najczęściej koduje się je w formacie odpowiednim dla URL (%XX), gdzie XX to kod ASCII znaku (np. %2F to zakodowana postać slashu, a %20 – spacji).

`expires=data`

Atrybut `expires` informuje przeglądarkę o dacie wygaśnięcia danego ciasteczka. Zostanie ono usunięte z dysku, gdy jego data ważności zostanie przekroczona. Jeśli nie podano daty wygaśnięcia, to ciasteczko zostanie usunięte po zakończeniu sesji.

Data musi być podana w następującym formacie (przykład): "Tuesday, 05-Nov-2004 08:30:09 GMT"

Format ten oparty jest na RFC 822, RFC 850, RFC 1036, i RFC 1123 z drobną zmianą odnośnie separatora daty – tu występuje kreska, podana jest również strefa czasowa GMT^[1].

`domain=domena`

Ten parametr określa widoczność ciasteczka. W trakcie sprawdzania pliku na komputerze klienta zawierającego ciasteczka, przeglądarka porównuje zapisaną domenę z domeną serwera, do którego wysyła nagłówki. Przeglądarka wysyła wszystkie nie przeterminowane ciasteczka, których domena jest zawarta w domenie serwera (dodatkowo może być sprawdzana ścieżka wywoływanego pliku i typ połączenia).

W specyfikacji Netscape'a^[1] wprowadzone jest w tym zakresie dodatkowe ograniczenie. To znaczy domena zostanie dopasowana, jeśli zawiera minimum dwie kropki, albo minimum trzy – jeśli domena główna serwera nie jest jedną z domen specjalnych, czyli: "COM", "EDU", "NET", "ORG", "GOV", "MIL", "INT". Ma to zapobiegać ustawianiu domen typu ".com", ".edu", czy ".va.us". Może to jednak powodować nieoczekiwane rezultaty, ponieważ ustawienie dla ciasteczka domeny w formacie "domena.org" spowoduje, że ciasteczka będą widoczne tylko dla danej domeny, ale nie będą wysyłane do poddomen, czyli np. "forum.domena.org". Problem ten omija się ustawiając domenę ".domena.org"^[2].

Domyślnie `domain` przyjmuje wartość domeny strony, z której wysłano żądanie zapisu ciasteczka.

`path=ścieżka`

Atrybut `path` jest podawany w celu ograniczenia widoczności ciasteczka do danej ścieżki dostępu do katalogu (liczy się ścieżka widoczna w URL-u pliku, a nie rzeczywiste położenie na dysku serwera). Wszystkie strony umieszczone w tym katalogu i jego podkatalogach będą mogły je wykorzystać. Należy zauważyć, że podanie parametru `path` w postaci "/wiki" pozwoli na odczytanie danych z ciasteczek plikom w katalogach "/wikipedia", "/wiki/Cookie" itp.

Widoczność ciasteczka będzie niezależna od położenia pliku, jeśli podana została ścieżka "/". Natomiast domyślnie `path` przyjmuje wartość ścieżki do strony, z której wysłano żądanie zapisu ciasteczka.

`secure`

Ten parametr nie posiada wartości. Jeśli zostanie podany, to ciasteczko będzie widoczne (wysłane) tylko wtedy gdy połączenie będzie szyfrowane (obecnie możliwe przy użyciu protokołu HTTPS).

Przy pobieraniu zawartości strony z serwera, przeglądarka sprawdzi (jak podano powyżej) zapamiętane ciasteczka, w których parametry domeny i ścieżki zgadzają się z adresem URL strony. Jeśli je znajdzie, dołącza je do nagłówka HTTP w postaci:

```
Cookie: nazwa_ciasteczka_1=wartosc_ciasteczka_1; nazwa_ciasteczka_2=wartosc_ciasteczka_2; ...
```

Właściwości

- Ciasteczka o tej samej nazwie ale o innych ścieżkach będą nadpisywane.
- W celu skasowania należy wysłać ciasteczko o takiej samej nazwie i czasie wygaśnięcia z minioną datą.
- Możliwe jest wysyłanie kilku ciasteczek w jednym nagłówku (poprzez kilka atrybutów Set-Cookie).
- Istnieją limity przy zapisywaniu ciasteczek na dysku (po ich przekroczeniu przeglądarka usuwa starsze ciasteczka).
 - maksymalna liczba ciasteczek: 300.
 - maksymalna wielkość ciasteczka: 4 kilobajty.
 - maksymalna liczba ciasteczek z jednego serwera lub z jednej ścieżki: 20.
- Gdy jest zainstalowany serwer Proxy nagłówki Set-Cookie nie powinny być przechowywane w pamięci proxy.
- Jeżeli serwer Proxy dostanie odpowiedź z nagłówkiem zawierającym Set-Cookie powinien go przekazać do klienta bez względu na rodzaj odpowiedzi np. 304 (nagłówek niezmieniony) czy 200 (nagłówek inny niż zapisany w cache'u).

Argumenty przeciw

Niektórzy są wrogami mechanizmu ciasteczek. Niechęć ta wynika z następujących przesłanek:

- Mechanizm ten jest bardzo niedoskonały. Weryfikowalność. Jeśli nie skonfiguruje się poprawnie kont użytkowników w systemie, to tak naprawdę nie rozpoznaje on użytkownika, tylko jego przeglądarkę internetową. Powoduje to, że z jednej strony osoba korzystająca z kilku takich komputerów lub kilku przeglądarek na jednym komputerze nie jest rozpoznawana poprawnie; z drugiej strony jeśli kilka osób korzysta z tego samego komputera i przeglądarki, mechanizm nie może ich rozróżnić.
- Do dzisiaj krążą plotki, jakoby ciasteczka były źródłem rozprzestrzeniania się wirusów. W rzeczywistości jest to niemożliwe – nie można zainfekować komputera wirusem jedynie z powodu włączonej obsługi ciasteczek.
- Niechęć do ciasteczek bierze się również z niedostatecznych metod zarządzania nimi. Nowoczesne przeglądarki posiadają zaawansowane narzędzia, które w intuicyjny sposób pozwalają zarządzać informacjami zawartymi w ciasteczkach, lecz znaczna część użytkowników korzysta z Internet Explorera, w którym nie ma takich opcji (aczkolwiek istnieją programy firm trzecich, które to umożliwiają)
- Mechanizm ciasteczek działa bez świadomości i wiedzy użytkownika, naruszając jego prywatność. Często stosowany jest w monitorowaniu zachowań i aktywności użytkowników, np. portale i sklepy internetowe mogą gromadzić w ten sposób informacje o zainteresowaniach użytkowników i wyświetlać reklamy o treści nawiązującej do odwiedzanych stron. Co prawda większość przeglądarek ma możliwość całkowitego blokowania ciasteczek lub włączenia opcji ostrzegającej każdorazowo o ich przesyłaniu, lecz ogromna większość użytkowników albo w ogóle nie wie, co to jest ciasteczko, albo po krótkim czasie odblokowuje ten mechanizm, gdyż bez niego nie można skorzystać z wielu serwisów.

Alternatywa dla ciasteczek – dane w adresie URL

Gdy użytkownik ma wyłączoną obsługę ciasteczek, wówczas dane należy przesłać w inny sposób. W ramach protokołu HTTP jest to możliwe przy użyciu metody GET bądź POST. W praktyce jednak używa się jedynie metody GET — ze względu na łatwość jej użycia oraz na to, że metoda POST jest związana głównie z formularzami.

Zastosowanie metody GET wiąże się jednak z koniecznością podania danych w adresie URL. Jest to jednak zadaniem kłopotliwym i niebezpiecznym, ponieważ sprowadza się do konieczności dodawania odpowiednich parametrów do wszystkich wewnętrznych linków zawartych na stronach serwisu^[3]. Jest to kłopotliwe ze względu na potencjalną ilość takich danych, a niebezpieczne ze względu na to, że użytkownik może np. chcieć zachować taką stronę i nie będąc świadomy zawartych w niej poufnych danych, wysłać komuś mailem.

 Ta sekcja jest załącznikiem. Jeśli możesz, .

Blokowanie ciasteczek

Wszystkie nowoczesne przeglądarki pozwalają na włączenie bądź wyłączenie mechanizmu ciasteczek (domyślnie zazwyczaj jest on włączony).

Przypisy

- [1] Specyfikacja ciasteczek w Netscape (http://wp.netscape.com/newsref/std/cookie_spec.html)
- [2] phpBB FAQ (<http://www.phpbb.com/support/documents.php?mode=faq>) – patrz pytanie 23: "I (or my users) cannot stay logged in to the forum!"
- [3] Sesje PHP — Przekazywanie identyfikatora sesji (<http://pl.php.net/manual/pl/ref.session.php#session.idpassing>)

Kaskadowe arkusze stylów

World Wide Web
Struktura stron WWW HTML, XHTML, XML, XSL
Generowanie dynamicznych stron WWW Active Server Pages, ASP.NET, JavaServer Pages, PHP
Po stronie użytkownika kaskadowe arkusze stylów, JavaScript, AJAX, kolory w Internecie
Przesyłanie danych Hypertext Transfer Protocol, HTTPS, HTTP referrer, serwer WWW, VoiceXML, XMLHttpRequest
Pojęcia architektura informacji, użyteczność, dostępność

Kaskadowe arkusze stylów (ang. *Cascading Style Sheets*, *CSS*) to język służący do opisu formy prezentacji (wyświetlania) stron WWW. CSS został opracowany przez organizację W3C w 1996 r. jako potomek języka DSSSL przeznaczony do używania w połączeniu z SGML-em. Pierwszy szkic CSS zaproponował w 1994 r. Håkon Wium Lie^[1].

Arkusze stylów CSS to lista dyrektyw (tzw. reguł) ustalających w jaki sposób ma zostać wyświetlana przez przeglądarkę internetową zawartość wybranego elementu (lub elementów) (X)HTML lub XML. Można w ten sposób opisać wszystkie pojęcia odpowiedzialne za prezentację elementów dokumentów internetowych, takie jak rodzina czcionek, kolor tekstu, marginesy, odstęp międzywierszowy lub nawet pozycja danego elementu względem innych elementów bądź okna przeglądarki. Wykorzystanie arkuszy stylów daje znacznie większe możliwości pozycjonowania elementów na stronie, niż oferuje sam (X)HTML.

CSS został stworzony w celu odseparowania struktury dokumentu od formy jego prezentacji. Separacja ta zwiększa zakres dostępności witryny, zmniejsza złożoność dokumentu, ułatwia wprowadzanie zmian w strukturze dokumentu. CSS ułatwia także zmiany w renderowaniu strony w zależności od obsługiwanego medium (ekran, palmtop, dokument w druku, czytnik ekranowy). Stosowanie zewnętrznych arkuszy CSS daje możliwość zmiany wyglądu wielu stron naraz bez ingerowania w sam kod (X)HTML, ponieważ arkusze mogą być wspólne dla wielu dokumentów.



Historia

Pierwotnie HTML był językiem wyłącznie do opisu struktury dokumentu. Jednak z czasem zrodziła się potrzeba ożywienia wyglądu takich dokumentów. Powoli dodawano nowe znaczniki do HTML pozwalające kontrolować kolory, typografię, dodawać nowe media (np. obrazki). Te niestandardowe rozszerzenia realizowane były przez najpopularniejszych producentów przeglądarek bez porozumienia z drugim. Doprowadziło to do zaimplementowania nowych znaczników działających w konkretnej grupie przeglądarek i nie działających w innych przeglądarkach. Projektanci zostali zmuszeni do wysyłania do klienta różnych wersji tej samej witryny w zależności od użytej przeglądarki, uzyskanie identycznego wyglądu w różnych przeglądarkach było praktycznie niemożliwe. Håkon Wium Lie jako pierwszy zaproponował CHSS (Cascading HTML Style Sheets) w październiku 1994 roku. Później Lie i Bert Bos pracowali wspólnie nad standardem CSS (literka H została usunięta ze względu na możliwość stosowania stylów do innych podobnych do HTML języków).

W tym czasie została utworzona organizacja World Wide Web Consortium, która z Lie'em i Bossem na czele przejęła prace nad CSS. Pod koniec 1996 roku wydano oficjalną dokumentację CSS, Kaskadowe arkusze stylów, poziom 1.

W3C zatwierdziło dwa oficjalne standardy CSS: CSS 1 i CSS 2, a także dnia 7 czerwca 2011 roku standard CSS 2.1^[2].

Trwają również prace nad CSS3. Wersja ta w stosunku do poprzedników wzbogaci się o wiele selektorów oraz właściwości, nowością jest także modułowy charakter języka – nie będzie to już jednolita rekomendacja, lecz kilkadziesiąt osobnych dokumentów, co pozwoli na włączanie lub wyłączanie odpowiednich modułów w przeglądarkach w zależności od chwilowych potrzeb^[3].

Obsługa przez przeglądarki

Początki implementacji CSS1

Specyfikacja CSS1 została opublikowana pod koniec 1996. Kilka miesięcy później pojawiła się przeglądarka Internet Explorer 3 zapewniająca podstawową obsługę CSS1. Była to ważna cecha, która w czasach dominacji Netscape Navigatora, pozwalała przeglądarce Microsoftu wysunąć się na prowadzenie. Obsługa CSS1 była na tyle dobra, że można było porzucić niestandardowy znacznik `` i rozpocząć eksperymentowanie z marginesami i innymi elementami układu strony. W praktyce projektanci napotkali liczne problemy związane z niekompletną i pełną błędów implementacją CSS1. Dopiero począwszy od IE4 który ukazał się pod koniec 1997 roku CSS1 działało prawidłowo. Netscape w wersji czwartej zaimplementował CSS1 lecz, jak się okazało, z licznymi błędami. Powszechnie uważano że sam CSS jest wadliwy a to skłoniło wielu projektantów do jego zarzucenia. W efekcie powszechne uznanie CSS1 za standard, bardzo się opóźniło. Z dzisiejszej perspektywy jest to język dość prosty a zarazem dający projektantowi wiele możliwości. Pozwala przede wszystkim dokładnie rozmieścić poszczególne elementy strony oraz stosować warstwy. Jedną z podstawowych właściwości CSS 1 jest kaskadowość. Pliki stylów dołączone przez autora dokumentu, mogą zostać podmienione przez odbiorcę w celu dopasowania prezentacji do indywidualnych potrzeb. Najważniejsze że wszystko może być zmodyfikowane w jednym pliku a tym samym odpadło monotonne "gmeranie" oddzielnie, w każdym z plików witryny

CSS 1 we współczesnych przeglądarkach

CSS 1 jest w pełni obsługiwany przez współczesne, popularne przeglądarki tj. oparte na następujących silnikach renderujących stronę:

- Gecko np. Firefox, Camino, SeaMonkey.
- Trident głównie Internet Explorer.
- Presto (Opera).
- WebKit/ np. Safari, Google Chrome.
- KHTML m.in. Konqueror.

Jednym z testów sprawdzających CSS 1 jest Acid1.

CSS 2

W CSS 2 zmodyfikowanym do stabilnej wersji 2.1 wprowadzone zostały nowe selektory i właściwości. W nowej wersji właściwościami stylu objęto strukturę dokumentu, oddzielając styl prezentacji dokumentów od ich zawartości. CSS2.1 upraszcza autorskie opracowanie w sieci i konserwację strony. Teoretycznie stało się możliwe wybranie np. elementu HTML, który jest bezpośrednio pod innym elementem (jest dzieckiem danego elementu). W praktyce użycie wielu z nowych elementów języka przez parę lat uniemożliwiała dominacja IE 6 i późniejszego IE 7, którego wsparcie CSS 2.1 jest słabe^[4]. Nowsza wersja IE dołączyła jednak do pozostałych przeglądarek i w pełni przechodzi m.in. test Acid2, w którym testowano elementy standardu CSS 2.1. CSS 2.1 opiera się na CSS 1 i z nielicznymi wyjątkami, wszystkie aktualne style wersji pierwszej są też obecne w wersji drugiej. CSS 2.1 obsługuje specyficzne medialne arkusze stylu tak, że autorzy mogą dostosować prezentację swoich dokumentów do wizualnych przeglądarek, urządzeń słuchowych, drukarek, urządzeń Braille'a, urządzeń ręcznych, itd. CSS 2.1 wspomaga ustawienie (pozycjonowanie) treści, obsługuje ściągane czcionki, wspiera układ graficzny tabeli, internalizację, automatyczne liczniki, numerację i niektóre właściwości dotyczące interfejsu użytkownika.

CSS 3

CSS 3 jest obecnie szczątkowo obsługiwany przez większość najnowszych przeglądarek (np. opacity w Gecko). Właściwości CSS 3 o implementacji znajdującej się w fazie eksperymentalnej opatrzone są prefiksem odpowiednim dla przeglądarki (np. -moz-border-radius dla Gecko, -webkit-border-radius dla WebKit).

Selektory CSS3 są obsługiwane przez wszystkie główne przeglądarki (silniki).

Składnia arkuszy

Arkusz stylów składa się z reguł określających styl dla wybranych elementów dokumentu (HTML, SVG i innych). Reguła składa się z selektora oraz deklaracji. Selektor określa grupę elementów (rzadziej pojedynczy element), którego ma dotyczyć deklaracja. Deklaracja określa formatowanie i składa się z nazwy jednej z właściwości i jej wartości napisanej po dwukropku. Deklaracja musi być otoczona nawiasami klamrowymi.

```
selektor { właściwość: wartość }
```

Dodatkowo możliwe jest grupowanie zarówno selektorów jak i deklaracji. Zgrupowane selektory rozdziela się przecinkami, a deklaracje średnikami:

```
selektor1, selektor2 { właściwość1: wartość1; właściwość2: wartość2; }
```

Dozwolone jest stosowanie średnika po wszystkich deklaracjach, nie jest jednak dozwolone stosowanie przecinka po ostatnim selektorze. Ponadto niektóre wartości mogą być zgrupowane i podane w ramach jednej deklaracji. W takim wypadku składnia zależy od definicji składni zbiorczej właściwości^[5].

Poniżej podana jest przykładowa reguła dla języka (X)HTML, w której przypisujemy wszystkim akapitom niebieski kolor tekstu:

```
p { color: blue; }
```

Selektorem jest tutaj p, właściwością color, a wartością blue.

Selektory zawarte w pierwszej specyfikacji CSS zapewniają możliwość opisanie docelowej grupy elementów przez:

- nazwę elementu (np. „h1”)
- klasę elementu (np. „elementy_menu”), także w połączeniu z nazwą elementu (np. „img.wyrownane_do_prawej”)
- id elementu (np. „#menu_lewe”)
- przodków danego elementu (np. „div#menu_lewe a” zostanie zastosowane do linków zawartych w elemencie div o id „menu_lewe”)
- stan linków określany przez pseudoklasy (:visited, :link, :active)
- inne pseudoklasy typograficzne (:first-line, :first-letter)

Dodawanie stylów do dokumentu

Dokument można powiązać z arkuszem określając relację tego pierwszego z osobnym dokumentem CSS za pomocą elementu link:

```
<link rel="stylesheet" href="arkusz.css" /> <!-- wersja uniwersalna dla (X)HTML -->
```

W przypadku dokumentu XML (w tym także XHTML serwowanego z XML-owym typem zawartości) można użyć specyficznej dla XML-a instrukcji przetwarzania:

```
<?xml-stylesheet type="text/css" href="arkusz.css"?>
```

Reguły CSS można też umieszczać wewnątrz nagłówka dokumentu (X)HTML oraz w niektórych dokumentach opartych na XML dzięki elementowi style:

```
<style type="text/css">p { color: red }</style>
```

Można również dodawać deklaracje bezpośrednio do danego elementu dokumentu za pomocą atrybutu style:

```
<p style="color: red">Lorem ipsum</p>
```

Ta ostatnia metoda nie jest jednak zalecana, ponieważ utrudnia zachowanie spójności w wyglądzie.

Model kaskadowy

Nazwa „kaskadowe arkusze stylów” wynika z faktu, iż gdy reguły CSS wykluczają się wzajemnie w arkuszu zewnętrznym, arkuszu wewnętrznym oraz na poziomie elementów HTML, priorytet stylów ustalany jest hierarchicznie. Przyjęto, że oddziaływanie stylów z arkuszy zewnętrznych może być modyfikowane przez style zdefiniowane w nagłówku dokumentu, te zaś mogą być modyfikowane przez reguły zdefiniowane bezpośrednio w ciele dokumentu. Pierwszeństwo mają zatem style zdefiniowane „bliżej” formatowanego elementu. Kolejność interpretacji reguł formatujących dany element przez przeglądarkę przedstawia się następująco:

1. Domyślny arkusz przeglądarki WWW (niezależny od autora strony)
2. Domyślny arkusz użytkownika przeglądarki (jak wyżej)
3. Zewnętrzne arkusze stylów i definicje stylów w nagłówku dokumentu
4. Definicje stylów w atrybucie style elementu

Ten model działania pokazuje, w jaki sposób działa kaskada stylów. Między stylami z różnych źródeł nie muszą zresztą wcale występować żadne konflikty – wszystkie style uzupełnią się, tworząc jeden wielki „wirtualny” styl.

Przypisy

- [1] Håkon Wium Lie: Cascading HTML Style Sheets – A Proposal (<http://www.w3.org/People/howcome/p/cascade.html>) (ang.). 1994-10-10. [dostęp 2010-06-26].
- [2] Cascading Style Sheets Standard Boasts Unprecedented Interoperability (<http://www.w3.org/2011/05/css-pr.html>) (ang.). 2011-06-07. [dostęp 2011-06-09].
- [3] Module Overview (<http://www.w3.org/TR/css3-roadmap/#modlist>) (ang.). W: *Introduction to CSS3* [on-line]. W3C, 2001-05-23. [dostęp 2010-06-26].
- [4] Peter-Paul Koch: CSS contents and browser compatibility (<http://www.quirksmode.org/css/contents.html>) (ang.). 2009-03-28. [dostęp 2010-06-26].
- [5] Håkon Wium Lie, Bert Bos: Cascading Style Sheets, level 1 (<http://www.w3.org/TR/CSS1/#grouping>) (ang.). W3C, 2008-04-11. [dostęp 2009-08-16].

Bibliografia

- Jeffrey Zeldman: *Projektowanie serwisów WWW. Standardy sieciowe. Wydanie II*. Helion, 2007. ISBN 83-246-0774-9.

Linki zewnętrzne

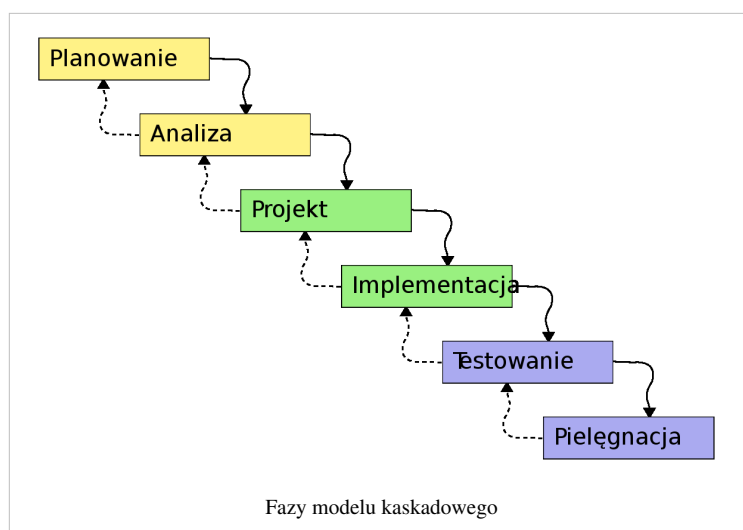
- Kaskadowe arkusze stylów, poziom 1 (<http://www.w3.org/TR/REC-CSS1>)
- Kaskadowe arkusze stylów, poziom 2 (<http://www.w3.org/TR/REC-CSS2>)
- Kaskadowe arkusze stylów, poziom 2 korekta 1 (<http://www.w3.org/TR/CSS21>)
- Kaskadowe arkusze stylów, poziom 3 (<http://www.w3.org/Style/CSS/current-work>)
- Walidator CSS (<http://jigsaw.w3.org/css-validator/>)

Model kaskadowy

Model kaskadowy (ang. *waterfall model*) – jeden z kilku rodzajów procesów tworzenia oprogramowania zdefiniowany w inżynierii oprogramowania. Jego nazwa wprowadzona została przez Winstona W. Royce w roku 1970, w artykule "Managing the Development of Large Software Systems" (zarządzanie tworzeniem dużych systemów informatycznych).

Polega on na wykonywaniu podstawowych czynności jako odrębnych faz projektowych, w porządku jeden po drugim. Każda czynność to kolejny schodek (kaskada):

1. Planowanie systemu (w tym Specyfikacja wymagań)
2. Analiza systemu (w tym Analiza wymagań i studium wykonalności)
3. Projekt systemu (poszczególnych struktur itp.)
4. Implementacja (wytworzenie kodu)
5. Testowanie (poszczególnych elementów systemu oraz elementów połączonych w całość)
6. Wdrożenie i pielęgnacja powstałego systemu.



Jeśli któraś z faz zwróci niesatysfakcjonujący produkt cofamy się wykonując kolejne iteracje aż do momentu kiedy otrzymamy satysfakcjonujący produkt na końcu schodków.

Zastosowanie

Model kaskadowy jest rzadko używany z następujących powodów:

- Nie można przejść do następnej fazy przed zakończeniem poprzedniej
- Model ten posiada bardzo nieelastyczny podział na kolejne fazy
- Iteracje są bardzo kosztowne - powtarzamy wiele czynności

Tego typu modelu należy używać wyłącznie w przypadku gdy wymagania są zrozumiałe i przejrzyste, ponieważ każda iteracja jest czasochłonna i wymaga dużych wydatków na ulepszanie.

Linki zewnętrzne

- Royce, W.W., "Managing the Development of Large Software Systems" (PDF) ^[1] Artykuł z 1970 roku, w którym W.W. Royce zaproponował pojęcie "modelu kaskadowego".

Przypisy

[1] <http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf>

Dziedziczenie (programowanie)

Dziedziczenie (ang. *inheritance*) to w programowaniu obiektowym operacja polegająca na stworzeniu nowej klasy na bazie klasy już istniejącej.

Na przykład, jeśli mamy klasę (w C++):

```
class Punkt
{
public:
    float x,y;
    Punkt(float _x, float _y);
    virtual void wypisz();
    virtual void przesun(float przesuniecie_x, float przesuniecie_y);
};
```

Załóżmy, że w naszym programie wynikła potrzeba użycia dodatkowej klasy, która różni się od tej jedynie w kilku szczegółach funkcjonalnych. Dzięki dziedziczeniu nie musimy tworzyć takiej klasy od zera, a możemy zamiast tego wprowadzić jedynie konieczne modyfikacje do klasy już istniejącej.

Przykładowo chcemy, by nowa klasa miała dodatkowy składnik:

```
string nazwa;
```

a także odpowiednio zmieniona metodę wypisz.

Dziedziczona klasa może zostać zdefiniowana w następujący sposób:

```
class NazwanyPunkt: public Punkt
{
public:
```

```
string nazwa;  
NazwanyPunkt(float _x=0, float _y=0, string _nazwa=NULL);  
virtual void wypisz();  
};
```

W ten sposób powstała nowa klasa o nazwie NazwanyPunkt (klasa pochodna, podklasa, potomek) wywodząca się od klasy Punkt (klasa podstawowa, nadklasa, rodzic).

Bardzo ważna w naszej klasie pochodnej jest pierwsza linijka:

```
class NazwanyPunkt: public Punkt
```

Wyrażenie znajdujące się po dwukropku nazywa się listą pochodzenia. W składni języka C++ informuje ona od czego wywodzi się klasa pochodna.

W C++ klasie pochodnej możemy zdefiniować:

- dodatkowe dane składowe (w naszym przykładzie: string nazwa;)
- dodatkowe funkcje składowe (w naszym przykładzie zmieniliśmy funkcję wypisz())
- nową treść funkcji wirtualnej

W innych językach szczegóły dziedziczenia mogą wyglądać odmiennie, np. w CLOS klasa pochodna może wpływać na metody odziedziczone po klasie podstawowej, ogólna zasada dziedziczenia pozostaje jednak taka sama.

Ten sam przykład dziedziczenia zapisany w Javie:

```
class NazwanyPunkt extends Punkt { }
```

Dziedziczenie wielokrotne

Dziedziczenie wielokrotne (ang. *multiple inheritance*) nazywane także **dziedziczeniem wielobazowym** to operacja polegająca na dziedziczeniu po więcej niż jednej klasie bazowej. Dziedziczenie wielokrotne stosowane jest na przykład w języku C++. W innych językach programowania (np. w Javie) dopuszczalne jest wyłącznie dziedziczenie jednokrotne, zaś do uzyskania efektu, który w C++ osiąga się poprzez dziedziczenie wielokrotne używa się interfejsów.

Przeanalizujmy przykład (w C++):

```
class Samochod {  
    public:  
        int iloscKol;  
        void jedz() { /* ciało metody */ }  
};  
  
class Lodz {  
    public:  
        float wypornosc;  
        void plyn() { /* ciało metody */ }  
};  
  
class Amfibia : public Samochod , public Lodz {  
    public:  
        unsigned int kolor;  
};
```

W efekcie wielokrotnego dziedziczenia Klasa Amfibia (amfibia to pojazd lądowo-wodny) posiada wszystkie pola i metody swoich klas bazowych.

Dzięki zastosowaniu wielokrotnego dziedziczenia, stworzony obiekt danej klasy jest wielotypowy. W odniesieniu do powyższego przykładu możliwe są następujące operacje:

```
...  
void napompujKolo( Samochod& samochod ) { /* ciało metody */ }  
void naprawKadlub( Lodz& lodz ) { /* ciało metody */ }  
  
int main() {  
    Amfibia amfibia;  
    napompujKolo( amfibia );  
    naprawKadlub( amfibia );  
    return 0;  
}
```

Widzimy, że obiekt amfibia jest jednocześnie typu Samochod i Lodz.

Wielokrotne dziedziczenie a interfejsy

Zarówno dziedziczenie wielokrotne, jak i interfejsy pozwalają na uzyskanie równoważnego efektu -- możliwości traktowania obiektu polimorficznie ze względu na wiele, niespokrewnionych ze sobą typów. Wielodziedziczenie jednakże jest techniką znacznie bardziej niebezpieczną, gdyż w przeciwieństwie do interfejsów, łączy w sobie środki do współdzielenia implementacji ze środkami współdzielenia zewnętrznego kontraktu klasy, a zatem dwie funkcje o radykalnie różnych zastosowaniach. Dlatego też użycie wielokrotnego dziedziczenia wymaga znacznej wiedzy o mechanizmach języka i ścisłej dyscypliny od stosującego je programisty, w przeciwnym wypadku bowiem istnieje niebezpieczeństwo stworzenia hierarchii klas, w której zmiana szczegółów implementacyjnych może pociągnąć za sobą konieczność zmiany kontraktu lub też sytuacji, w której nie będzie możliwe stworzenie hierarchii o pożądanym kształcie bez wprowadzania nieprawidłowego zachowania obiektów.

Dla kontrastu, w przypadku użycia interfejsów, czynności dziedziczenia (współdzielenia implementacji) i dzielenia interfejsu (czyli zewnętrznego kontraktu) są celowo rozdzielone. W ten sposób nie jest możliwe przypadkowe pomylenie tych dwóch pojęć, co miałyby opłakane skutki.

Argumentem podnoszonym na rzecz wielokrotnego dziedziczenia bywa fakt, że umożliwia ono proste wykorzystanie istniejącej implementacji z więcej niż jednej klasy bazowej. Jest to prawda, jednak w rzeczywistości bardzo rzadko ten właśnie efekt jest tym co *naprawdę* ma zastosowanie w danej sytuacji, często zaś istnieje fałszywe wrażenie, iż jest to potrzebne. Jeśli istotnie zachodzi potrzeba wykorzystania więcej niż jednej implementacji, w przypadku użycia interfejsów można wykorzystać techniki osadzania i delegacji; powoduje to konieczność większej pracy ze strony programisty, jednak zazwyczaj jest to *pożądane*, gdyż zmusza do głębszego zastanowienia się nad pomysłem łączenia niespokrewnionych klas, a dodatkowo powoduje, że nowa klasa zachowuje się dokładnie tak jak oczekuje tego programista, a nie tak jak stanowi definicja języka (która rzadko pokrywa się z intuicją w bardziej skomplikowanych obszarach, a wielodziedziczenie należy do najbardziej skomplikowanych).

Należy również pamiętać, że powyższe argumenty odnoszą się głównie do "tradycyjnych" języków o statycznym systemie typów, takich jak C++. W innych językach, takich jak Python, również istnieje mechanizm wielodziedziczenia, jednakże konstrukcja systemu typów i mechanizmu klas jest radykalnie odmienna, co powoduje że powyższa dyskusja traci swoją aktualność.

Źródła

- Scott Meyers, *More Effective C++*

ltg:Bērnaklase

IPv4

IPv4 (ang. *Internet Protocol version 4*) – czwarta wersja protokołu komunikacyjnego IP przeznaczonego dla Internetu. Identyfikacja hostów w IPv4 opiera się na adresach IP. Dane przesyłane są w postaci standardowych datagramów. Wykorzystanie IPv4 jest możliwe niezależnie od technologii łączącej urządzenia sieciowe – sieć telefoniczna, kablowa, radiowa, itp. IPv4 znajduje się obecnie w powszechnym użyciu. Dostępna jest również nowsza wersja – IPv6. Dokładny opis czwartej wersji protokołu IP znajduje się w RFC 791. W modelu TCP/IP protokół IPv4 znajduje się w warstwie sieciowej.

Budowa pakietu IP

+	Bity 0 - 3	4 - 7	8 - 15	16 - 18	19 - 31		
0	Wersja	Długość nagłówka	Typ usługi	Całkowita długość			
32	Numer identyfikacyjny			Flagi	Kontrola przesunięcia		
64	Czas życia pakietu		Protokół warstwy wyższej	Suma kontrolna nagłówka			
96	Adres źródłowy IP						
128	Adres docelowy IP						
160	Opcje IP				Uzupełnienie		
192	Dane						

- Pierwsze, 4-bitowe pole zawiera numer wersji protokołu IP (dla IPv4 jest to 4).
- Kolejne 4-bitowe pole zawiera długość samego nagłówka protokołu (bez danych).
- Następne 8 bitów prezentuje tzw. "typ usługi" (ang. Type of Service). Jest to najbardziej podstawowy sposób wyznaczania priorytetu danego datagramu. Na podstawie ToS routery mogą szybciej (np. dla sesji SSH), lub wolniej (np. dla przesyłania danych) przepuszczać przez siebie dane datagramy, zwiększając bądź też zmniejszając w ten sposób interaktywność transmisji.
- Kolejnym 16-bitowym polem jest całkowita długość pakietu (razem z danymi). Jego długość (wynosząca 2^{16}) umożliwia ustawienie rozmiaru pakietu na 65536 bajtów. Warto dodać, że minimalny rozmiar pakietu to 20 bajtów.
- Kolejne 16-bitowe pole to numer identyfikacyjny, potrzebny między innymi do fragmentacji i defragmentacji pakietów.
- Kolejnym 3-bitowym polem są flagi, które są używane przy fragmentacji pakietów.
- Następne 13-bitowe pole służy do odpowiedniego "poukładania" pofragmentowanych pakietów w taki sposób, aby dane zawarte w tych pakietach miały taki sam układ, jak w pakiecie przed fragmentacją.
- Pole TTL (8 bitów) to czas życia pakietów. Jest to liczba z zakresu 0-255. Przy przechodzeniu pakietu przez router jest ona zmniejszana o jeden. W momencie osiągnięcia przez TTL zera, router nie przekazuje pakietu do kolejnego urządzenia sieciowego.
- Kolejne, 8-bitowe pole to numer protokołu warstwy wyższej, takimi jak ICMP (1), TCP (6) czy UDP (17).
- Następnym polem jest suma kontrolna nagłówka pakietu. Służy ona kontroli, czy wszystkie dane zostały przetransmitowane. Przy każdej zmianie zawartości pakietu, router oblicza sumę kontrolną dla pakietu i zapisuje

ją w odpowiednim polu.

- Dalsze pola zawierają adres źródłowy i docelowy. To właśnie na podstawie nich można określić pochodzenie i miejsce docelowe pakietu w sieci.
- Ostatnim, 32-bitowym polem są opcje, które w normalnej transmisji zwykle nie są używane.
- Pole Padding (wypełnienie) jest opcjonalne i jego zawartością są zera dopełniające długość nagłówka do wielokrotności 32 bitów.

Adres IP

Aby możliwa była komunikacja w protokole IP konieczne jest nadanie każdemu hostowi adresu IP czyli unikalnego identyfikatora, który pozwoli na wzajemne rozpoznawanie się poszczególnych uczestników komunikacji. Użytkownicy Internetu nie muszą znać adresów IP. Nazwa *www.wikipedia.org* jest tłumaczona na adres IP dzięki wykorzystaniu protokołu DNS. Adres IP jest dostarczany każdemu użytkownikowi przez dostawcę internetu (ISP). Może być przydzielany statycznie lub dynamicznie. Zapotrzebowanie na adresy IP jest tak duże, że pula nieprzydzielonych adresów zaczyna się wyczerpywać.

Adresy i maski

W IPv4, czyli obecnym standardzie adresowania internetu, adres IP to liczba 32-bitowa (od 0 do 4294967295), zapisywana w porządku big endian. Liczby w adresie IP nazywają się oktetami, ponieważ w postaci binarnej mają one osiem bitów. Te osiem bitów daje w sumie 256 kombinacji, więc każdy oktet przedstawia liczbę od 0 do 255.

Najpopularniejszy sposób zapisu adresów IP, to przedstawianie ich jako 4 dziesiętnych liczb od 0 do 255 oddzielonych kropkami. W rzeczywistości komputery traktują adres Wikipedii jako liczbę 32-bitową:

```
3482223596
```

Taki zapis jest mało czytelny, wobec czego stosuje się podział adresu IP na cztery oktety. Adres Wikipedii zapisujemy binarnie:

```
11001111 10001110 10000011 11101100
```

po czym każdą grupę 8-bitów przekształcamy do postaci dziesiętnej:

```
207      142      131      236
```

Z adresowaniem IP wiąże się pojęcie maski sieciowej. Wyobraźmy sobie sieć złożoną z 3 komputerów o adresach:

```
Komputer 1: 192.168.1.1
```

```
Komputer 2: 192.168.1.2
```

```
Komputer 3: 192.168.1.3
```

Początek adresu dla wszystkich z nich jest ten sam, a końcówka się zmienia. Aby ściśle zdefiniować adresy przynależne do danej sieci wymyślono pojęcie maski podsieci. Umówiono się, że określona liczba pierwszych bitów adresu IP ma być taka sama, a pozostałe bity w sieci mogą się różnić. W ten sposób powstaje proste kryterium, pozwalające komputerom na określenie swojego położenia na podstawie adresu. Maskę sieci zapisuje się podobnie jak adres IP. Dla przykładu

```
255.255.255.0
```

co binarne daje:

```
11111111 11111111 11111111 00000000
 255      255      255      0
```

Jeżeli komputery oprócz komunikacji w swojej sieci lokalnej mają łączyć się z internetem, to maska sieciowa staje się bardzo ważna. Gdy urządzenie sieciowe stwierdzi, że adres docelowy, z którym chce wymieniać dane nie pasuje do maski, to próbuje się z nim łączyć przez bramę sieciową. Porównywanie opiera się na zerowaniu w adresie bitów równych zeru w masce (logiczny AND bitów maski i adresu IP). Jeżeli komputer 3 łączy się komputerem 2, to wykonuje następujące operacje:

Maska	11111111	11111111	11111111	00000000
	255	255	255	0
Mój IP	11000000	10101000	00000001	00000011
	192	168	1	3
Wynik a	11000000	10101000	00000001	00000000
	192	168	1	0
Maska	11111111	11111111	11111111	00000000
	255	255	255	0
Docelowy IP	11000000	10101000	00000001	00000010
	192	168	1	2
Wynik b	11000000	10101000	00000001	00000000
	192	168	1	0

Wynik a oraz *Wynik b* są równe wobec czego komputer 3 wie, że komputer 2 jest w tej samej podsieci. Jeżeli komputer 3 będzie chciał pobrać stronę z serwera Wikipedii to operacja porównania będzie następująca:

Maska	11111111	11111111	11111111	00000000
	255	255	255	0
Mój IP	11000000	10101000	00000001	00000011
	192	168	1	3
Wynik a	11000000	10101000	00000001	00000000
	192	168	1	0
Maska	11111111	11111111	11111111	00000000
	255	255	255	0
IP Wikipedii	11001111	10001110	10000011	11101100
	207	142	131	236
Wynik b	11001111	10001110	10000011	00000000
	207	142	131	0

Wynik a, oraz wynik b są różne. W takiej sytuacji komputer 3 będzie się próbował połączyć z Wikipedią przez skonfigurowaną w nim bramę sieciową.

Rozdzielanie adresów

Nazwa	Pierwszy adres IP	Ostatni adres IP	Klasa	Największy ciągły blok
Blok 24-bitowy	1.0.0.0	126.0.0.0	pojedyncza sieć klasy A	10.0.0.0/8
Blok 20-bitowy	128.1.0.0	191.254.0.0	16 kolejnych sieci klasy B	172.16.0.0/16
Blok 16-bitowy	192.0.1.0	223.255.254.0	256 kolejnych sieci klasy C	192.168.0.0/24
Blok 12-bitowy	224.0.0.0	239.255.255.254	Klasa D	224.0.0.0/20
Blok 8-bitowy	240.0.0.0	255.255.255.254	Klasa E	240.0.0.0/24

W adresach klasy A tylko pierwszy oktet wskazuje adres sieci; pozostałe trzy oktety opisują unikatowy adres węzła w sieci. Choć jest tylko 126 adresów sieci klasy A, każdy taki adres może obejmować w przybliżeniu 17 milionów węzłów. Adresy klasy A zostały przyznane organizacjom rządowym i wielkim instytucjom.

Adresy klasy B używają pierwszych dwóch oktetów do wskazania adresu sieci i ostatnich dwóch jako unikatowego węzła sieci. Z uwagi na większą długość, adresów klasy B jest więcej, ale w ramach każdego można unikatowo opisać tylko około 65 000 węzłów.

W adresach klasy C używa się pierwszych trzech oktetów jako adresu sieciowego i tylko ostatniego oktetu jako adresu węzła. Stąd istnieje wiele dostępnych adresów klasy C, ale każdy z nich może być użyty tylko do 254 węzłów.

Ze względu na skończoną ilość adresów oraz konieczność ich agregacji dla celów uproszczenia trasowania powstały Regionalne Rejestry Internetowe (ang. *RIR*) – organizacje zajmujące się przydzielaniem puli adresów dla poszczególnych dostawców Internetu (ang. *ISP*). Organizacją nadrzędną jest Agencja Zarządzania Numeracją Internetową (ang. *IANA*), która zajmuje się dystrybucją poszczególnych klas A. Do organizacji regionalnych należą:

- APNIC (ang. *Asia Pacific Network Information Centre*) – dla rejonu Azji i Pacyfiku,
- ARIN (ang. *American Registry for Internet Numbers*) – dla rejonu Ameryki Północnej,
- LACNIC (ang. *Regional Latin-American and Caribbean IP Address Registry*) – dla rejonu Ameryki Łacińskiej i wysp Karaibskich,
- RIPE (fr. *Réseaux IP Européens*) – dla rejonu Europy, Bliskiego Wschodu i centralnej Azji,
- AfriNIC – dla rejonu Afryki (Rozpoczęła działanie 22 lutego 2005, wcześniej dystrybucją zajmowały się RIPE NCC, APNIC i ARIN).

Jeżeli ISP potrzebuje więcej adresów zwraca się do właściwej organizacji regionalnej i otrzymuje kolejny zakres numerów IP. Dla przykładu ARIN przydzielił adresy od 64.78.200.0 do 64.78.207.255 firmie Verado, Inc, która przekazała pulę od 64.78.205.0 do 64.78.205.15 firmie Bomis. Bomis adres 64.78.205.6 udostępnił Wikipedii.

Powszechnie panuje pogląd, że pula dostępnych adresów jest na wyczerpaniu, jednak w oficjalnym zestawieniu zajętości adresacji IP jest jeszcze wiele bloków zarezerwowanych przez IANA ([1]).

Adresy należące do puli 127.0.0.0/8 (127.x.x.x) są przypisane do urządzenia loopback i zawsze odnoszą się do komputera lokalnego. Adres 0.0.0.0 to adres domyślny (ang. *default*).

Prywatne adresy IPv4

Istnieje pula prywatnych adresów IP. Mogą być one wykorzystane tylko w sieciach lokalnych. Infrastruktura Internetu ignoruje te adresy IP. IANA (*Internet Assigned Numbers Authority*) zarezerwował następujące trzy bloki przestrzeni adresów IP dla prywatnych sieci:

- 10.0.0.0 - 10.255.255.255 – dla sieci prywatnych klasy A (maska: 255.0.0.0/8)
- 172.16.0.0 - 172.31.255.255 – dla sieci prywatnych klasy B (maska: 255.240.0.0/12)
- 192.168.0.0 - 192.168.255.255 – dla sieci prywatnych klasy C (maska: 255.255.0.0/16)

Adresy prywatne można wykorzystywać za pomocą lokalnych routerów w sieciach lokalnych, ale nie działają one w publicznej części internetu. Jeżeli administrator sieci lokalnej przydzieli swoim komputerom adresy IP z puli prywatnej, to routery mogą łatwo rozpoznać kiedy komputery chcą się łączyć z internetem. W takiej sytuacji brama internetowa wykorzystuje technikę maskowania adresów sieciowych NAT, która pozwala na łączenie się z internetem komputerom nie posiadającym własnych publicznych adresów IP. Komputery z adresami prywatnymi nie mogą pełnić roli serwerów sieciowych w Internecie chyba, że posłużymy się techniką maskowania adresów docelowych (DNAT).

Automatyczne przydzielanie adresów IPv4 może być realizowane poprzez zastosowanie protokołów DHCP, RARP, BOOTP, PPP.

Wykorzystanie adresów IPv4

Początkowo wszystkie adresy IPv4 były zarządzane bezpośrednio przez IANA, która w zależności od wnioskowanych potrzeb przydzielała określoną pulę adresów klasy A, B lub C. Wielkie firmy, jak Xerox, Ford czy IBM automatycznie otrzymywały po ponad 16 mln adresów internetowych, nawet jeżeli tak duża liczba nie była im potrzebna. Jeżeli mała firma z kilkunastoma węzłami chciała podłączyć się do Internetu przyznawano jej adresy z klasy C. To z kolei dawało jej kontrolę nad ponad dwustoma adresami węzłów, z których nikt inny nie mógłby skorzystać. Ze względu na marnotrawstwo oraz niespodziewanie duże zapotrzebowanie na adresację internetową z całego świata zmieniono zasady i powołano do życia organizacje regionalne, których zadaniem stało się nadzorowanie wykorzystania dostępnych adresów. Jednym ze sposobów oszczędzania adresów stało się także ponowne wykorzystanie adresów, które z jakichś powodów zostały zwolnione. To już codzienna praktyka dostawców Internetu. Okazuje się jednak, że w niektórych przypadkach takie działanie jest bardzo niebezpieczne i powoduje wiele różnego rodzaju problemów – z praktycznym unieruchomieniem dostępu do Internetu włącznie^[2].

Obecnie klasy A przydzielane są organizacjom regionalnym, te dalej rozdzielają je do ISP w blokach po 4 klasy C (1024 adresy), a następnie ISP przydzielają adresy swoim klientom. Duży nacisk kładzie się na wykorzystywanie mechanizmów NAT, umożliwiających korzystanie z jednego adresu zewnętrznego przez wiele urządzeń posiadających adresy lokalne. W ten sposób ogranicza się przydzielanie adresów urządzeniom (tj. drukarki, punkty dostępowe, itp.) działającym jedynie w obrębie zamkniętych sieci. Wciąż można dostać przypisanie do klasy C dla swojej organizacji, ale staje się to coraz trudniejsze. Trzeba wykazać rzeczywistą potrzebę dysponowania taką liczbą adresów.

Istnieją koncepcje, według których każde urządzenie elektroniczne ma zostać podłączone do Internetu. W takiej sytuacji pula adresów IPv4 będzie stanowczo za mała. Z tego powodu nastąpi prawdopodobnie przejście z protokołu IPv4 na IPv6, który zwiększy o cztery rzędy wielkości pulę dostępnych adresów.

W ramach klasy C adresów istnieje podział na tzw. podsieci (subnets). Rozmiar podsieci wyznaczany jest przez jej maskę. Najmniejszą podsiecią jest sieć składająca się z 4 adresów, największą ze 128. Dla sieci 4 komputerowej maska wynosi: $256 - 4 = 252$ (NETMASK = 255.255.255.252). Dla tak wyznaczonej podsieci można określić następujące parametry:

NETWORK = 195.205.36.32 (Adres IP – przykładowa podsieć sieci klasy C przyznana przez dostawcę)

NETMASK = 255.255.255.252 (maska podsieci)

adresy komputerów = 195.205.36.33 i 195.205.36.34

BROADCAST = 195.205.36.35 (adres rozgłoszeniowy)

W praktyce maska 255.255.255.252 oznacza, iż do sieci tej można podłączyć 2 komputery i używana jest przez administratorów sieci komputerowych do spinania poszczególnych segmentów sieci.

Aby znaleźć adres rozgłoszeniowy musimy przekształcić Adres IP oraz maskę podsieci na system binarny:

Adres IP:	11000011	11001101	00100100	00100000
	195	205	36	32
Maska podsieci:	11111111	11111111	11111111	11111100
	255	255	255	252

Patrzemy na maskę i wpisujemy jedynki na tych pozycjach adresu IP, na których w masce są zera. To jest nasz adres rozgłoszeniowy (broadcast):

Broadcast	11000011	11001101	00100100	00100011
	195	205	36	35

Mimo optymalizacji systemu przydzielania adresów, pula wolnych adresów topnieje w tempie szybszym niż przypuszczano. W styczniu 2011 roku japoński instytut INTEC Systems Institute, Inc. ogłosił, że pula adresowa wyczerpie się 12 lutego 2011 roku^[3]. Na stronie instytutu znajduje się licznik^[4] wolnych zasobów puli adresowej IPv4.

W dniu 3 lutego 2011 organizacja IANA przydzieliła regionalnym rejestratorom 5 ostatnich wolnych bloków klasy A.^[5] Oznacza to, że pula adresów IPv4, zarządzana przez IANA, została w całości wyczerpana. Operatorzy i przedsiębiorstwa w dalszym ciągu mogą otrzymywać adresy IP od regionalnych rejestratorów, aż do wyczerpania puli przez nich zarządzanej. Dalszy rozwój internetu zależy od przejścia na protokół IPv6.

Przypisy

- [1] <http://www.iana.net/assignments/ipv4-address-space>
- [2] Tomasz Grabowski: Adresik z odzysku - problem kurczących się zasobów IPv4 (http://obfusc.at/ed/ipv4_recycling_pl.html). 2004. [dostęp 2009-02-02].
- [3] Krzysztof Paślawski: Za 20 dni zabraknie adresów IP (<http://www.crn.pl/news/wydarzenia/e-biznes/2011/01/za-20-dni-zabraknie-adresow-ip>). 2011. [dostęp 2011-01-18].
- [4] http://inetcore.com/project/ipv4ec/index_en.html
- [5] The Internet Corporation for Assigned Names and Numbers: ICANN Press Release (<http://www.icann.org/en/news/releases/release-03feb11-en.pdf>). 2011. [dostęp 2011-02-04].

Linki zewnętrzne

- RFC 791: Internet Protocol (<http://tools.ietf.org/html/rfc791>)
- RFC 1918: Address Allocation for Private Internets (<http://tools.ietf.org/html/rfc1918>)
- adres-ip.eu (<http://adres-ip.eu>) - Sprawdź dowolny adres IP (lokalizacja, geolokalizacja, typ)

IPv6

IPv6 (ang. *Internet Protocol version 6*) – protokół komunikacyjny, będący następcą protokołu IPv4, do którego opracowania przyczynił się w głównej mierze problem malejącej, kończącej się ilości adresów IPv4. Podstawowymi zadaniami nowej wersji protokołu było zwiększenie długości adresu z 32-bitów do 128-bitów, uproszczenie nagłówka protokołu oraz zapewnienie jego elastyczności poprzez wprowadzenie rozszerzeń a także wprowadzenie wsparcia dla klas usług. Protokół jest znany także jako *IP Next Generation* oraz *IPng*^[1]. Głównymi dokumentami opisującymi protokół są RFC2460 oraz RFC4291.

Wdrażanie IPv6

Pierwsze dokumenty RFC opisujące protokół IPv6 powstały w 1995 roku. W latach 1996-2006 w infrastrukturę Internetu wdrażany był projekt 6BONE w formie eksperymentalnej sieci działającej w oparciu o IPv6. Po zamknięciu tego projektu niektórzy dostawcy usług internetowych (ISP) rozpoczęli produkcyjne dostarczanie IPv6 tak samo jak obecnie IPv4; spora część użytkowników IPv6 korzysta jednak z tego protokołu za pomocą tuneli wykorzystujących poprzednią wersję protokołu (tzw. tunelowanie IPv6-in-IPv4). Najprostszą metodą zestawienia takiego tunelu jest obecnie mechanizm 6to4.

Powody powstania IPv6 i brak kompatybilności z IPv4

Powszechnie stosowany obecnie protokół IPv4 ma pojemność około 4 miliardów adresów (2^{32}). W czasach gdy powstawał protokół IP (lata siedemdziesiąte), wydawało się to wystarczające - wtedy nikt nie przewidywał takiej popularności komputerów i Internetu. Jednakże już w pierwszych latach użytkowania IP podjęto prace mające na celu zaoszczędzenie adresów - wprowadzono wtedy adresowanie bezklasowe. Na początku lat dziewięćdziesiątych było jednak już jasne, że adresy IP wkrótce ulegną wyczerpaniu, więc w 1992 rozpoczęto prace nad stworzeniem IPNG - protokołu internetowego nowej generacji, co doprowadziło do pełnej definicji nowego protokołu w roku 1996. Nowy protokół nie mógł nosić numeru wersji 5, ponieważ numer ten został już wcześniej użyty dla eksperymentalnego protokołu Internet Streaming Protocol (w zamierzeniu mającego przenosić treści audio i video), dlatego też użyto kolejnego numeru - 6.

Przy tworzeniu nowych technologii i protokołów projektanci często stają przed dylematem stworzenia całkiem nowej jakości, czy też utrzymywania zgodności wstecz, często kosztem pewnych ograniczeń lub wprowadzenia znacznej komplikacji. W czasach tworzenia IPv6 Internet nie był tak popularny jak teraz i szacowano, że zamiana protokołu na IPv6 będzie możliwa bez utrzymania zgodności między nowszą i starszą wersją protokołu IP, dlatego też zaprojektowano całkowicie nowy protokół, bez obciążenia balastem koniecznej zgodności z IPv4. Prace nad projektowaniem IPv6 trwały dość długo, a po ich zakończeniu wizja końca adresacji IPv4 była wciąż dość odległa w czasie, dlatego też nowy protokół nie został wdrożony produkcyjnie. W tej chwili jednak ilość komputerów w Internecie i jego zastosowania uniemożliwiają zamianę protokołu IPv4 na IPv6 i jednocześnie oba protokoły muszą być używane przez urządzenia sieciowe, które przez to będą mieć faktycznie połączenie do dwóch rozłącznych sieci.

Różnice między protokołem IPv6 a IPv4

	IPv4	IPv6
adresy	32 bity	128 bitów
wsparcie dla IPsec	opcjonalne	wymagane
identyfikacja ruchu dla QoS	brak	przy użyciu pola Flow Label
fragmentacja	przez nadającego hosta i routery	jedynie przez nadającego hosta
suma kontrolna w nagłówku	obecna	brak
opcje	w nagłówku	przeniesione do nagłówków dodatkowych
ramki zgłoszeń	ARP	wielopoziomowe wiadomości typu Neighbor Solicitation
zarządzania grupami multicastowymi	IGMP	MLD (ang. <i>Multicast Listener Discovery Protocol</i>)
protokół komunikatów kontrolnych	ICMP	ICMPv6, wymagany
adresy transmisji	do wysyłania danych do wszystkich węzłów w podsieci	zastąpione przez grupowy adres typu link-local
przydzielanie adresu	wymagana konfiguracja ręczna, przez DHCP lub APIPA	nie wymaga konfiguracji ręcznej ani DHCP
mapowanie nazw hostów w DNS	używa rekordów <i>A</i> oraz <i>PTR</i> w domenie IN-ADDR.ARPA	używa rekordów <i>AAAA</i> oraz <i>PTR</i> w domenie IP6.ARPA

Adresacja

W protokole IPv6 adres zapisany jest w 128-bitowej liczbie i może identyfikować jeden bądź wiele interfejsów. W przypadku tego protokołu adres jest bardziej przejrzysty niż adres w poprzedniej wersji protokołu. W przeciwieństwie do poprzedniej wersji protokołu, zakres adresu, czyli obszar jego widoczności, jest ograniczony przez odpowiedni prefiks.

Zazwyczaj adres składa się z ośmiu 16-bitowych bloków złożonych z cyfr szesnastkowych i oddzielonych dwukropkiem. Dozwolone jest pomijanie początkowych zer w bloku, a także pominięcie ciągu bloków składających się wyłącznie z zer. Pomijając bloki zer dubluje się separator bloków (*dwukropek*). Poniższe adresy są równoznaczne:

- 2001:0db8:0000:0000:0000:0000:1428:57ab
- 2001:0db8:0:0:0:0:1428:57ab
- 2001:0db8:0:0::1428:57ab
- 2001:0db8::1428:57ab
- 2001:db8::1428:57ab

Typy adresów

W adresacji wykorzystywanej w protokole IPv6 wykorzystywane są trzy typy adresów:

- adresy unicast - identyfikujący pojedynczy interfejs; pakiety, które są kierowane na ten typ adresu dostarczane są tylko do odbiorcy
- adresy multicast - identyfikujący grupę interfejsów (mogą one należeć do różnych węzłów), pakiety wysyłane na ten adres dostarczane są do wszystkich członków grupy
- adresy anycast - podobnie jak adresy multicast, identyfikują one grupę interfejsów, jednak pakiet wysyłany na ten adres dostarczany jest tylko do najbliższego węzła (węzeł ten jest wyznaczany przez protokół routingu)

W przeciwieństwie do poprzedniej wersji protokołu, IPv6 nie definiuje adresów typu broadcast. Jednym z powodów jest fakt, że pakiety wysyłane na ten adres odbierane były przez wszystkie węzły w sieci, nawet takie, które nie potrafiły danych pakietów zinterpretować.

Zakresy adresów

Charakterystyczną cechą protokołu jest fakt, że zostały zdefiniowane zakresy adresów. W przypadku adresów unicastowych wyróżniane są następujące zakresy:

- adresy lokalne dla łącza (link-local address) - są to adresy wykorzystywane tylko do komunikacji w jednym segmencie sieci lokalnej lub przy połączeniu typu point-to-point. Routery nie przekazują pakietów z tego rodzaju adresem. Z puli pozostałych adresów wyróżniane są przez prefiks FE80::/10. Każdy interfejs musi mieć przydzielony co najmniej jeden adres lokalny dla łącza, nawet jeżeli posiada adres globalny lub unikalny adres lokalny^[2].
- unikalne adresy lokalne (unique local address) - są to adresy będące odpowiednikami adresów prywatnych wykorzystywanych w protokole IPv4. Z puli pozostałych adresów wyróżniane są przez prefiks FC00::/7. Od adresów lokalnych łącza odróżnia je także prefiks routingu.^[3]
- adresy globalne (global unicast address) - widoczne w całym internecie, są odpowiednikami adresów publicznych stosowanych w IPv4; do adresów tego typu należą adresy nie wymienione w pozostałych punktach^[2].

Przez pewien czas, protokół definiował adresy *site-local address*, identyfikowane przez prefiks FEC0/10, jednak, ze względu na wiele kontrowersji zostały uznane za przestarzałe^[3]

Mapowanie adresów IPv4 na IPv6

Możliwe jest reprezentowanie adresów protokołu IPv4 jako adresów IPv6. Jedną z możliwości jest stworzenie adresu IPv6, którego młodsze 32 bity zawierają adres IPv4, natomiast starsze 96 bitów jest wypełniona specjalnym wzorcem bitów (::ffff). Tak skonstruowany adres ma postać ::ffff:127.0.0.1 (za 127.0.0.1 można podstawić dowolny adres IP) i umożliwia normalną komunikację w sieci.

Adresy specjalne

Następujące adresy i grupy adresów posiadają specjalne, zarezerwowane znaczenie:

- ::/128 - adres nieokreślony (zawierający same zera).
- ::1/128 - loopback, adres wskazujący na host lokalny.
- ::/96 - pula zarezerwowana dla zachowania kompatybilności z protokołem IPv4 (pierwsze 96 bitów stanowią 0, pozostają 32 bity na adresy w formacie IPv4).
- ::ffff:0:0/64 - jw., ale pozwala wykorzystywać komunikację według protokołu IPv6 w sieci IPv4.
- 2001:7f8::/32 - pula zarezerwowana dla punktów wymiany ruchu, każdy z nich dostaje jedną podsieć /48.
- 2001:db8::/32 - pula wykorzystywana w przykładach i dokumentacji - nigdy nie będzie wykorzystywana produkcyjnie.
- 2002::/24 - adresy typu 6to4. Są to adresy wygenerowane na podstawie istniejących, publicznych adresów IPv4, dostępne dla każdego użytkownika.
- 3ffe::/16 - adresy testowej sieci 6BONE (adresy zostały wycofane 6 czerwca 2006 w związku z zakończeniem działania 6BONE).
- fc00::/7 - pula lokalnych unikalnych adresów IPv6 typu unicast (RFC 4193), będąca odpowiednikiem adresów prywatnych IPv4, choć, zgodnie z nazwą, powinny być unikalne na świecie.
- fe80::/10 - pula link-local określa adresy w obrębie jednego łącza fizycznego (np. segmentu sieci Ethernet). Pakiety z tej puli nie są przekazywane poza podsieć, jej działanie jest analogiczne do automatycznie konfigurowanych adresów z puli 169.254.0.0/16 w IPv4.

- `fec0::/10` - pula site-local określa adresy w obrębie jednej lokalnej organizacji. Obecnie nie zaleca się wykorzystywania tej puli; przyszłe implementacje IPv6 nie będą musiały obsługiwać tej puli.
- `ff00::/8` - pula multicastowa używana do komunikacji multicast.

W protokole IPv6 nie występuje pojęcie komunikacji broadcastowej (dane rozsyłane do wszystkich węzłów w danej podsieci). Aby wysłać dane do wielu odbiorców jednocześnie, należy korzystać z komunikacji multicastowej.

Nagłówek i jego rozszerzenie

W przeciwieństwie do protokołu IPv4, którego długość nagłówka wynosi od 20 do 60 bajtów, długość nagłówka protokołu IPv6 jest stała i wynosi 40 bajtów. Jego znaczną część zajmują adresy źródłowy oraz docelowy - 32 bajty, łatwo więc obliczyć, że na pozostałe dane pozostaje tylko 8 bajtów. Dzięki stałej długości nagłówek IPv6 jest dużo prostszy niż nagłówek poprzedniej wersji protokołu a zarazem dużo łatwiejszy w przetwarzaniu. Jest także dużo bardziej elastyczny - dodatkowe opcje protokołu mogą być umieszczane w opcjonalnych nagłówkach rozszerzających (ang. extension headers), następujących po nagłówku głównym IPv6. Takie rozwiązanie umożliwia zwiększenie możliwości protokołu bez wprowadzania zmian do podstawowego nagłówka^[4].

Opis pól nagłówka protokołu IPv6^[4]

Bity	0-3	4-7	8-11	12-15	16-19	20-23	24-27	28-31
0	Wersja	Priorytet	Etykieta przepływu					
32	Długość danych				Następny nagłówek		Limit przeskoków	
64	Adres źródłowy (128 bitów)							
96								
128								
160								
192	Adres docelowy (128 bitów)							
224								
256								
288								

Podstawowy nagłówek protokołu składa się z następujących pól^[4]:

- Wersja (4 bity) - definiująca wersję protokołu, w przypadku IPv6 pole te zawiera wartość 6 (bitowo 0110)
- Klasa ruchu (8 bitów) - określa sposób w jaki ma zostać potraktowany pakiet danych. W poprzedniej wersji protokołu pole te nazywało się *Type of Service*, jednak ze względu na to, że w IPv6 stosowane są inne mechanizmy priorytetowania danych, nazwę tego pola zmieniono
- Etykieta przepływu (20 bitów) - pomagające odróżnić pakiety, które wymagają takiego samego traktowania (ich pole klasy ruchu ma tę samą wartość)
- Długość danych (16 bitów) - wielkość pakietu, nie wliczając długości podstawowego nagłówka (wliczając jednak nagłówki rozszerzające)
- Następny nagłówek (8 bitów) - identyfikuje typ następnego nagłówka, pozwalając określić czy jest to nagłówek rozszerzający czy nagłówek warstwy wyższej. W przypadku tego drugiego, wartość pola jest identyczna z wartością pola w protokole IPv4
- Limit przeskoków (8 bitów) - określa ilość węzłów po odwiedzeniu których pakiet zostaje porzucony. W poprzedniej wersji protokołu pole te nosiło nazwę *time to live* i zawierało liczbę skoków, która była zmniejszana przez każdy odwiedzony węzeł
- Adres źródłowy (128 bitów) - adres węzła, który wysłał pakiet

- Adres docelowy (128 bitów)- adres węzła do którego adresowany jest pakiet

Nagłówki rozszerzające

Nagłówki rozszerzające służą do zwiększania możliwości protokołu IPv6. Na chwilę obecną zdefiniowanych jest sześć rozszerzeń protokołu:

- Hop-by-Hop Options
- nagłówek routingu
- nagłówek fragmentacji
- nagłówek opcji docelowych
- nagłówek uwierzytelniania
- Encrypted Security Payload

Autokonfiguracja

Dla podsieci będących LAN-em przydzielana jest pula adresów z maską /64 co umożliwia tworzenie unikalnych numerów IP w oparciu o (niepowtarzalne) numery sprzętowe MAC; adres taki (dla adresu MAC 11:22:33:44:55:66) będzie miał postać: 64bitowy_prefiks_sieci:1322:33FF:FE44:5566 (pierwszy bajt adresu MAC zwiększany jest o 2, w środku wstawiane jest FFFE). 64-bitowy prefiks sieci jest informacją rozgłaszaną przy pomocy ICMPv6 przez routery; natomiast jeżeli host nie uzyskał wspomnianego prefiksu w jego miejsce wstawiane jest fe80:: (czyli fe80:0000:0000:0000) - taki adres nazywa się "link-local" (nie jest on routowany do sieci zewnętrznych, jednak zawsze (także gdy prefiks został uzyskany) może być używany wewnątrz sieci lokalnej). Oczywiście nadal możemy korzystać z przydziału IP przez DHCP oraz ręcznych ustawień IP.

DNS

Obsługa adresacji IPv6 w systemie DNS została zaprojektowana jako rozszerzenie systemu DNS, które jest całkowicie zgodne wstecz z IPv4, co nie wprowadza żadnych problemów implementacyjnych w tym zakresie. Nazwy hostów w DNS zawierają adres IPv4 w rekordzie 'A', adresy IPv6 umieszcza się w rekordzie 'AAAA' - taka konstrukcja powoduje, że dany host może mieć jednocześnie adres IPv4 i IPv6 w DNS. Odwrotny DNS wprowadza kilka różnic: adres IPv4 postaci 123.45.67.89 jest mapowany na rekord 89.67.45.123.in-addr.arpa, natomiast adres IPv6 postaci 2001:db8::1428:57ab mapowany jest na rekord b.a.7.5.8.2.4.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa (kiedyś w tym celu wykorzystywano domenę ip6.int). W przypadku obu rodzajów adresacji wykorzystuje się ten sam typ rekordu - 'PTR'.

Przypisy

- [1] Internet Protocol, Version 6 (IPv6) Specification (<http://tools.ietf.org/html/rfc2460>).
- [2] Internet Protocol Version 6 (IPv6) Addressing Architecture (<http://tools.ietf.org/html/rfc3513>) ([ang.](#)). [dostęp 2011-02-10].
- [3] Unique Local IPv6 Unicast Addresses (<http://tools.ietf.org/html/rfc4193>) ([ang.](#)). [dostęp 2011-02-10].
- [4] Silvia Hagen: *"IPv6 Essentials, Second Editin"*. O'Reilly, 2006. ISBN 0-596-10058-2.

Linki zewnętrzne

- Strona grupy roboczej (<http://www.ipv6tf.org/>)
- Polska Grupa Robocza IPv6 (Polish IPv6 Task Force) (<http://www.pl.ipv6tf.org/>)

Klasa (programowanie obiektowe)

W programowaniu obiektowym **klasa** jest częściową lub całkowitą definicją dla obiektów. Definicja obejmuje dopuszczalny stan obiektów oraz ich zachowania. Obiekt, który został stworzony na podstawie danej klasy nazywany jest jej *instancją*. Klasy mogą być typami języka programowania - przykładowo, instancja klasy *Owoc* będzie mieć typ *Owoc*. Klasy posiadają zarówno interfejs, jak i strukturę. Interfejs opisuje, jak komunikować się z jej instancjami za pośrednictwem metod, zaś struktura definiuje sposób mapowania stanu obiektu na elementarne atrybuty.

Zależnie od implementacji, klasy mogą istnieć w programie tylko na etapie jego kompilacji, w wyniku której są tłumaczone na kod strukturalny lub też posiadać swoją reprezentację w kodzie wynikowym w postaci metaobektów. Za ich pomocą program może odczytywać i manipulować informacjami o klasie podczas wykonywania.

Języki programowania implementujące klasy różnią się pod względem oferowanej funkcjonalności. Większość z nich wspiera różne formy dziedziczenia oraz hermetyzacji.

Zagadnienia

Z pojęciem klasy związanych jest wiele dodatkowych zagadnień oraz funkcjonalności, które są implementowane w językach programowania.

Instancjonowanie

Klasa nie jest samodzielnym bytem, lecz szablonem do tworzenia nowych obiektów określonego typu i posiadających określone zachowanie. Obiekt utworzony na podstawie danej klasy nazywany jest jej *instancją*, a proces jego tworzenia - *instancjonowaniem*.

Poszczególne instancje klasy posiadają ten sam zbiór zachowań i atrybutów, lecz różnią się przechowywanymi w nich wartościami. Przykładowo, klasa *Samochód* opisuje pojęcie „samochodu” poprzez wymienienie charakteryzujących go atrybutów: prędkości maksymalnej, mocy silnika, producenta czy modelu. Jednak dwie różne instancje tej klasy będą różnić się od siebie wartościami tych atrybutów: jeden samochód może mieć prędkość maksymalną 220 km/h, zaś drugi - 240 km/h.

Interfejs

Uwaga: w tej sekcji termin „interfejs” nie odnosi się do pojęcia interfejsu funkcjonującego w języku Java, aczkolwiek są one ze sobą powiązane.

Obiekty wchodzą w interakcje ze światem zewnętrznym poprzez metody. Określają one możliwe zachowania, jakie na obiekcie można wykonać, a ich definicje znajdują się w klasie danego obiektu. Metoda jest rodzajem podprogramu języka programowania z dodatkową właściwością, tj. dostępem do atrybutów obiektu, na którym została wywołana. Metody mogą zarówno odczytywać, jak i modyfikować atrybuty obiektu, dlatego określane są także mianem „zachowań”. Zbiór metod, którymi dysponuje obiekt, nazywany jest jego interfejsem.

W przykładzie z samochodem klasa może definiować następujące metody: jedź, hamuj, skręć lub aktualnaPrędkość. Część z nich wpływa na aktualny stan obiektu, powodując np. zatrzymanie pojazdu, zaś inne służą wyłącznie do uzyskiwania dodatkowych informacji.

W programowaniu obiektowym zazwyczaj metody interfejsu są tak dobierane, aby były niezależne od siebie nawzajem, tj. nie ma żadnych ograniczeń na kolejność ich wywoływania.

Struktura

Oprócz interfejsu, klasa definiuje także strukturę danych przechowywanych w obiektach. Są one dzielone na elementarne atrybuty, zwane także *polami* lub *właściwościami*. Zbiór wartości wszystkich atrybutów tworzy stan konkretnego obiektu, który przechowywany jest w pamięci lub innym nośniku danych pod określonym adresem, dzięki czemu możliwy jest dostęp do niego za pośrednictwem referencji.

W większości języków programowania struktura pamięci wykorzystywanej do przechowywania stanu obiektu jest ustalana w momencie kompilacji programu na podstawie definicji klasy oraz właściwości architektury sprzętowej. Alternatywnym podejściem jest model języka Python, w którym wartości atrybutów obiektu są zapisane jako tablica asocjacyjna par klucz-wartość. Dzięki temu poszczególne obiekty tej samej klasy mogą różnić się ilością i znaczeniem atrybutów, które mogą być do nich dynamicznie dodawane.

Klasy definiują również zbiór niezmienników, które są zachowywane przez wszystkie metody klasy. Niezmiennik jest pewnym wyrażeniem odnoszącym się do atrybutu, które musi być zawsze spełnione aby stan obiektu był prawidłowy, niezależnie od tego, jakie operacje na nim wykonamy. Jeśli przy pomocy obiektów reprezentujemy różne samochody, możemy zdefiniować niezmiennik definiujący minimalny promień skrętu. Programista nie może utworzyć obiektu samochodu, którego wartość promienia skrętu będzie mniejsza, niż minimalna, a gwarantuje nam to obecność niezmiennika. Niezmienniki mogą być implementowane poprzez zabronienie bezpośredniego dostępu do atrybutów obiektu i utworzenie dodatkowych metod dostępowych, które oprócz ich ustawiania, sprawdzają także niezmienniki. W niektórych językach niezmienniki można definiować bezpośrednio jako część specyfikacji klasy.

Elementy statyczne

Część języków dopuszcza tworzenie tzw. metod oraz atrybutów statycznych. Nie są one związane z żadnym konkretnym obiektem klasy, lecz tworzą globalny stan oraz globalnie dostępne operacje, które można wywoływać nawet wtedy, gdy nie posiadamy żadnej instancji klasy.

Rodzaje klas

Istnieje wiele rodzajów klas różniących się właściwościami i zastosowaniami. Należy zauważyć, że poniższe rodzaje nie są rozłączne. Przykładowo, ponieważ nie może istnieć klasa, która jest jednocześnie finalna i abstrakcyjna, można wnioskować, że klasy finalne są szczególnym przypadkiem klas właściwych.

Klasy właściwe

Klasą właściwą nazywamy każdą klasę, która może być instancjonowana.

Klasy abstrakcyjne

Klasa abstrakcyjna to przeciwieństwo klasy właściwej - nie można utworzyć obiektu takiej klasy. Ma ona zastosowanie jedynie wtedy, gdy język programowania obsługuje dziedziczenie. Klasa abstrakcyjna stanowi wtedy wzorzec do dalszego rozszerzenia, który sam w sobie nie może być pełnoprawnym, poprawnym obiektem.

Klasy abstrakcyjne najczęściej posiadają przynajmniej jedną metodę abstrakcyjną. Jest to rodzaj metody, dla którego zdefiniowana jest wyłącznie lista argumentów, nazwa oraz zwracane wartości, natomiast nie jest ona zaimplementowana. Implementacją podanego interfejsu muszą zająć się klasy potomne.

Niektóre języki takie, jak Java czy C# obsługują ponadto szczególny rodzaj klas abstrakcyjnych zwany interfejsem. Interfejs nie może definiować żadnych atrybutów, a wszystkie jego metody są abstrakcyjne. Ponadto najczęściej nie dotyczą ich ograniczenia pojedynczego dziedziczenia.

Klasy finalne

Klasa finalna ma sens jedynie w przypadku dziedziczenie - nazywamy tak klasę, której nie można rozszerzyć.

Klasy lokalne i wewnętrzne

Niektóre języki programowania umożliwiają tworzenie klas w innych przestrzeniach, niż globalna. Istnieje kilka typów takich klas.

Klasa wewnętrzna lub zagnieżdżona to klasa, która jest zdefiniowana wewnątrz innej klasy. Klasy wewnętrzne posiadają dostęp do metod statycznych klasy głównej, lecz nie muszą być instancjonowane wraz z nią. Zależnie od języka, mogą, ale nie muszą być dostępne spoza klasy głównej.

Klasy lokalne mogą być tworzone wewnątrz funkcji i metod. Obiekty tych klas nie mogą istnieć poza tymi funkcjami i są niszczone wraz z zakończeniem ich wykonywania. Język programowania może ponadto narzucać dodatkowe ograniczenia na takie klasy.

Klasy anonimowe

W większości języków programowania każda klasa posiada swą unikalną nazwę, dzięki czemu możliwe jest odwoływanie się do niej w różnych sytuacjach. Niektóre języki wspierają także tworzenie klas anonimowych, które nie posiadają nazwy.

Metaklasy

Metaklasa to klasa, której instancjami są inne klasy. Ta specjalna konstrukcja umożliwia dostęp do właściwości oraz definicji klasy w trakcie wykonywania programu.

Klasy częściowe

W przeciwieństwie do zwykłych klas, definicja klasy częściowej może być rozszerzona na więcej, niż jeden plik. Umożliwia to lepsze radzenie sobie z dużymi ilościami kodu. Podczas kompilacji wszystkie częściowe definicje łączone są w kompletną klasę, dzięki czemu nie ma żadnych funkcjonalnych różnic między nimi, a zwykłymi klasami.

Przykład

Przykład klasy w języku Java:

```
public class Samochod
{
    private int predkoscMax;
    private String producent;
    private String model;

    private int predkosc;

    public Samochod(String producent, String model, int predkoscMax)
    {
        this.predkoscMax = predkoscMax;
        this.producent = producent;
        this.model = model;
        this.predkosc = 0;
    } // end Samochod();
}
```

```
public void przyspiesz(int i)
{
    if(this.predkosc + i < this.predkoscMax)
    {
        this.predkosc += i;
    }
    else
    {
        this.predkosc = this.predkoscMax;
    }
} // end przyspiesz();

public void zwolnij(int i)
{
    if(this.predkosc - i > 0)
    {
        this.predkosc -= i;
    }
    else
    {
        this.predkosc = 0;
    }
} // end zwolnij();

public int aktualnaPredkosc()
{
    return this.predkosc;
} // end aktualnaPredkosc();
} // end Samochod;
```

Obiekt (programowanie obiektowe)

Obiekt to podstawowe pojęcie wchodzące w skład paradygmatu programowania obiektowego w analizie i projektowaniu oprogramowania oraz w programowaniu.

Jest to struktura zawierająca:

- dane
- metody, czyli funkcje służące do wykonywania na tych danych określonych zadań.

Z reguły obiekty (a właściwie klasy, do których te obiekty należą) są konstruowane tak, aby dane przez nie przenoszone były dostępne wyłącznie przez odpowiednie metody, co zabezpiecza je przed niechcianymi modyfikacjami. Takie zamknięcie danych nazywa się enkapsulacją czyli jakby zamknięcie ich w *kapsule*.

W istocie obiekty są rozwinięciem koncepcji programowania z kontrolą typów zmiennych. W programowaniu obiektowym obiekty tworzone są dynamicznie jako podstawowy element konstrukcji programu. Podobnie jak dla typu liczb naturalnych czy typu zmiennych znakowych, dla których zdefiniowane są pewne operacje jak np. dodawanie czy konkatencja, a nie są zdefiniowane inne, jak np. operacje logiczne, tak dla obiektów programista decyduje o wykonalności pewnych operacji oraz definiuje ich funkcyjną postać. Użycie obiektów polega na ich zainicjalizowaniu (np. na nadaniu zmiennej całkowitej pewnej wartości np. 7) oraz na wykonywaniu na nich operacji zgodnie z definicją typu - obiektu.

Każdy obiekt ma trzy cechy:

- tożsamość, czyli cechę umożliwiającą jego identyfikację i odróżnienie od innych obiektów;
- stan, czyli aktualny stan danych składowych;
- zachowanie (ang. *behaviour*), czyli zestaw metod wykonujących operacje na tych danych.

Zobacz też:

- Programowanie obiektowe
 - Klasa (programowanie obiektowe)
 - Metoda (programowanie obiektowe)
 - Dziedziczenie (programowanie obiektowe)
-


Klasa abstrakcyjna

Klasa abstrakcyjna w programowaniu obiektowym jest to klasa, która nie może mieć swoich reprezentantów pod postacią obiektów.^[1] Stosuje się ją zazwyczaj do zdefiniowania interfejsów. Zależnie od użytego języka programowania klasy abstrakcyjne tworzy się na różne sposoby.

Idea klasy abstrakcyjnej

Klasa abstrakcyjna jest pewnym uogólnieniem innych klas (na przykład dla występujących w rzeczywistości obiektów), lecz sama jako taka nie istnieje. Ustalmy, że przez "figurę" będziemy rozumieć "koło", "kwadrat" lub "trójkąt". Te obiekty matematyczne mogą być reprezentowane przez pewne klasy. Obiekty te posiadają już konkretne właściwości takie jak promień (dla konkretnego koła) czy długość boku (dla konkretnego kwadratu). Klasy tych obiektów wywodzą się z pewnej uogólnionej klasy określanej jako po prostu *figura*. Jednak nie jesteśmy w stanie określić jaką konstrukcję miałby obiekt klasy *figura*, ponieważ figura geometryczna jako taka nie istnieje. Istnieją natomiast wywodzące się od niej klasy koło czy kwadrat. Dodatkowo oczywistym jest, że *figura* nie posiada konkretnej wartości pola czy obwodu, jednak już na tym etapie wiemy, że *każda* figura tak zdefiniowana (koło, kwadrat czy trójkąt) posiada pole i obwód, które będzie różnie obliczane dla różnych figur. Dzięki temu *figura* definiuje pewien interfejs dla klas wywodzących się od niej.

Klasy abstrakcyjne w różnych językach

 Zobacz przykłady klas abstrakcyjnych na stronie Wikiźródeł

C++

W C++ klasą abstrakcyjną jest klasa, która posiada zadeklarowaną co najmniej jedną metodę czysto wirtualną. Każda klasa, która dziedziczy po klasie abstrakcyjnej i sama nie chce być abstrakcyjną, musi implementować wszystkie odziedziczone metody wirtualne.

Java

W Javie klasę abstrakcyjną możemy stworzyć na dwa sposoby: za pomocą słowa kluczowego **abstract**, które tworzy klasyczną klasę abstrakcyjną lub za pomocą słowa kluczowego **interface** tworzącego abstrakcyjny interfejs

C#

W C# klasą abstrakcyjną jest klasa, która została zadeklarowana jako abstrakcyjna za pomocą słowa kluczowego *abstract*.

Właściwości

- Klasa abstrakcyjna w Javie nie musi posiadać metod czysto wirtualnych aby być abstrakcyjną, jednak takie użycie klasy abstrakcyjnej określa klasę jako *nieinstancjowalną* i jest rzadziej spotykane.
- Jeśli co najmniej jedna metoda w klasie zostanie zadeklarowana jako czysto wirtualna (abstrakcyjna), to ta klasa musi zostać zadeklarowana jako abstrakcyjna.
- Jeżeli wszystkie metody klasy są czysto wirtualne, zaleca się, aby taką klasę zadeklarować jako interfejs klasy.

Przypisy

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Inżynieria oprogramowania: wzorce projektowe*. Warszawa: WNT, 2008, s. 455. ISBN 978-83-204-3472-9.

Interfejs (programowanie obiektowe)

W programowaniu obiektowym **interfejs** jest abstrakcyjną reprezentacją klasy. Interfejs umożliwia korzystanie z danej klasy, niezależnie od faktycznej implementacji. Interfejs pozwala na hermetyzację obiektów, utworzonych w oparciu o klasy zawierające definicję (implementację) wspólnego interfejsu. Tak rozumiany interfejs, w ramach programowania obiektowego, określany jest też mianem *interfejs klasy*.

W języku C++ interfejs może być zdefiniowany jako klasa abstrakcyjna, natomiast w Javie, C#, Object Pascalu oraz PHP stosuje się w tym celu specjalną deklarację ze słowem *interface*.

Java

Java	
Pojawienie się	1995
Paradygmat	Wieloparadygmatowy (obiektyowy, strukturalny, imperatywny)
Implementacje	OpenJDK, HotSpot
Aktualna wersja stabilna	6 Update 26 (1.60.0_26), 7 czerwca 2011; 28 dni temu
Twórca	James Gosling
Licencja	GNU General Public License
Platforma sprzętowa	wieloplatformowy
Platforma systemowa	Windows x86/x64/IA64, Linux x86/x64/IA64, Solaris ^[1] [2]
http://www.java.com	

Definicja intuicyjna:

Java (wym. "dżawa") to język programowania. Programy napisane w Javie można uruchamiać na wielu urządzeniach, takich jak telefony komórkowe lub komputery oraz pod różnymi systemami operacyjnymi, przy użyciu Wirtualnej maszyny Javy, która musi być w danym środowisku zainstalowana.

Java to obiektowy język programowania stworzony przez grupę roboczą pod kierunkiem Jamesa Goslinga z firmy Sun Microsystems. Java jest językiem tworzenia programów źródłowych kompilowanych do kodu bajtowego, czyli postaci wykonywanej przez maszynę wirtualną. Język cechuje się silnym typowaniem. Jego podstawowe koncepcje zostały przejęte z języka Smalltalk (maszyna wirtualna, zarządzanie pamięcią) oraz z języka C++ (duża część składni i słów kluczowych).

Javy nie należy mylić ze skryptowym językiem JavaScript, z którym wspólną ma jedynie składnię podstawowych instrukcji.

Obecnym właścicielem tej technologii jest Oracle Corporation.

Główne koncepcje

Autorzy języka Java określili kilkanaście kluczowych koncepcji swojego języka. Najważniejsze z nich to:

Obiektość

W przeciwieństwie do proceduralno-obiekowego języka C++, Java jest silnie ukierunkowana na obiektość. Wszelkie dane i akcje na nich podejmowane są pogrupowane w **klasy obiektów**. O **obiekcie** można myśleć jako o samoistnej części programu, która może przyjmować określone *stany* i ma określone *zachowania*, które mogą zmieniać te stany bądź przysyłać dane do innych obiektów. Wyjątkiem od całkowitej obiektości (jak np. w Smalltalku) są typy proste (int, float itp.).

```
// oznacza komentarz
// w tej postaci to obiekt reprezentujący kolorowy punkt
public class Figura {

    // właściwości (atrybuty/pola)
    private float środekX;
    private float środekY;
```

```
private int kolor; //tak naprawdę do przechowywania (tworzenia)
                    //koloru używa się zazwyczaj obiektu
java.awt.Color

// operacje (metody)
public float obliczPole() {
    return 0;
}
public float obliczObwód() {
    return 0;
}
public void wyświetl() {...}
...
}
```

Jak widać z powyższego, silne typowanie oznacza, że każda wprowadzana zmienna czy pole musi mieć przypisany typ przechowywanych w niej danych (float oznacza typ zmiennoprzecinkowy), a każda metoda musi deklarować, jakiego typu dane zwraca (lub void, jeśli nic nie zwraca). Z przykładu widać też, że w nazwach zmiennych i metod można używać polskich liter – to zasługa wbudowanej obsługi kodowania Unicode. Pokazany w nazwach zmiennych i metod standard kodowania (polegający na pisaniu słów bez spacji, a z kapitalizowaniem drugiego i następnych słów składowych) jest nieobowiązkowy, ale jest jedną z uznanych za dobre praktyk programowania w Javie

Dziedziczenie

W Javie wszystkie obiekty są pochodną obiektu nadrzędnego (jego klasa nazywa się po prostu Object), z którego dziedziczą podstawowe zachowania i właściwości. Dzięki temu wszystkie mają wspólny podzbiór podstawowych możliwości, takich jak ich: identyfikacja, porównywanie, kopiowanie, niszczenie czy wsparcie dla programowania współbieżnego.

```
// "extends" oznacza dziedziczenie po klasie Figura pól: środekX,
środekY i kolor
// oraz metod: obliczPole, obliczObwód i wyświetl
public class Kwadrat extends Figura {
    // dodatkowe atrybuty
    private float wierzchołekX;
    private float wierzchołekY;
    private float długośćBoku;

    // "przesłaniamy" operacje rodzica, dzięki czemu dla każdej
zdefiniowanej Figury
    // można policzyć pole czy obwód
    @Override
    public float obliczPole() {
        return długośćBoku*długośćBoku;
    }
    ...
}
```

Choć C++ udostępniał dziedziczenie wielobazowe, projektanci Javy odeszli od tego pomysłu. Java umożliwia jedynie dziedziczenie jednobazowe, a więc wyłącznie jedna klasa może przekazać swoje właściwości i operacje jako podstawę do rozszerzania ich o dodatkowe możliwości. Dzięki temu wyeliminowano możliwość konfliktów między właściwościami przekazywanymi przez klasy nadrzędne.

By zrekompensować spadek elastyczności wynikający z pojedynczego dziedziczenia wprowadzono interfejsy oraz podklasy umożliwiające wielokrotne dziedziczenie implementacji. Pozwalają one nazwać pewien określony zbiór operacji, dzięki czemu można określić, że dany obiekt, któremu przypisano dany interfejs (**implementujący go**), umożliwia wykonanie owego zestawu operacji.

Niezależność od architektury

Tę właściwość Java ma dzięki temu, że kod źródłowy programów pisanych w Javie kompiluje się do kodu pośredniego. Powstały kod jest niezależny od systemu operacyjnego i procesora, a wykonuje go tzw. *wirtualna maszyna Javy*, która (między innymi) tłumaczy kod uniwersalny na kod dostosowany do specyfiki konkretnego systemu operacyjnego i procesora. W tej chwili wirtualna maszyna Javy jest już dostępna dla większości systemów operacyjnych i procesorów.

Jednak z uwagi na to, że kod pośredni jest interpretowany, taki program jest wolniejszy niż kompilowany do kodu maszynowego. Z tego względu maszynę wirtualną często uzupełnia się o kompilator JIT. Istnieją również niezależne od Suna kompilatory Javy – przykładem podprojekt GCC o nazwie GCJ. W rezultacie powstaje szybszy kod, ale można go uruchamiać na jednej tylko platformie, a więc nie jest przenośny.

Sieciowość i obsługa programowania rozproszonego

Dzięki wykorzystaniu reguł obiektowości, Java nie widzi różnicy między danymi płynącymi z pliku lokalnego a danymi z pliku dostępnego przez HTTP czy FTP.

Biblioteki Javy udostępniają wyspecjalizowane funkcje umożliwiające programowanie rozproszone – zarówno między aplikacjami Javy (RMI) jak i między aplikacją Javy a aplikacjami napisanymi w innych językach (CORBA, usługi sieciowe). Inne biblioteki udostępniają możliwość pisania aplikacji uruchamianych w przeglądarkach internetowych (aplety Javy) oraz aplikacji działających ciągle po stronie serwera (serwlety).

Niezawodność i bezpieczeństwo

W zamierzeniu Java miała zastąpić C++ – obiektowego następcę języka C. Jej projektanci zaczęli od rozpoznania cech języka C++, które są przyczyną największej liczby błędów programistycznych, by stworzyć język prosty w użyciu, bezpieczny i niezawodny.

O ile po pięciu odsłonach Javy jej prostota jest dyskusyjna, o tyle język faktycznie robi dużo, by utrudnić programiście popełnienie błędu. Przede wszystkim Java ma system wyjątków czyli sytuacji, gdy kod programu natrafia na nieprzewidywane trudności, takie jak np.:

- operacje na elemencie poza zadeklarowaną granicą tablicy lub elemencie pustym
- czytanie z niedostępnego pliku lub nieprawidłowego adresu URL
- podanie nieprawidłowych danych przez użytkownika

W innych językach programowania programista oczywiście może wprowadzić wewnętrzne testy sprawdzające poprawność danych, pozycję indeksu tablicy, inicjalizację zmiennych itd., ale jest to jego dobra wola i nie jest to jakoś szczególnie wspierane przez dany język. W Javie jest inaczej – obsługa wyjątków jest obowiązkowa, bez tego program się nie skompiluje. Przy tym obiekty wchodzące w skład pakietu standardowego Javy (i gros obiektów z pakietów pochodzących od poważnych programistów niezależnych) implementują wyjątki w każdym miejscu kodu, którego wykonanie jest niepewne ze względu na okoliczności zewnętrzne.

Sama obsługa wyjątków polega na napisaniu kodu, który wykona się w odpowiedzi na taką sytuację nadzwyczajną. Może to być np. podstawienie wartości domyślnej przy natrafieniu na nieprawidłową wartość parametru, zaniechanie danej akcji i powrót do stanu stabilnego czy choćby zapisanie pracy przed wyjściem. W sytuacji wyjątkowej program przerywa normalne wykonanie i tworzy specjalny obiekt wyjątku odpowiedniej klasy, który "wyrzuca" z normalnego biegu programu. Następnie zdefiniowany przez użytkownika kod "łapie" ten obiekt wyjątku i podejmuje odpowiednie działanie. Działanie może być dwojakiego typu: wspomniane wyżej środki zaradcze lub odrzucenie takiego "śmierdzącego jaja" dalej, do bloku programu, który nakazał wykonanie wadliwej operacji. Takie podawanie sobie wyjątku może być wieloetapowe i jeśli skończy się w bloku głównym programu powoduje jego przerwanie i ogłoszenie błędu krytycznego.

Oprócz systemu wyjątków Java od wersji 1.4 ma dwa inne systemy wspomagające pisanie niezawodnych programów: logowanie i asercje. Pierwsze pozwalają na zapisanie w plikach dziennika przebiegu działania programu, z dodatkową możliwością filtrowania zawartości, określenia poziomu logowanych błędów itp. Drugie rozwiązanie pozwala na upewnienie się, że pewne założenia co do określonych wyrażeń (np. że liczba, z której wyciągamy pierwiastek jest nieujemna) są prawdziwe. Asercje są o tyle ciekawe, że działają tylko z odpowiednią opcją wykonania programu, dzięki czemu programista może sprawdzić działanie programu, a później bez wysiłku spowodować pominięcie testowej części kodu po prostu przez ominięcie tej opcji.

Krytyka i kontrowersje

Język *Java* pomimo swoich wielu zalet, ma wiele wad w tym takie, które wzbudzają liczne kontrowersje:

- Najczęściej wymienianą wadą języka *Java* jest to, że programy pisane w *Javie* wykonują się wolniej niż programy pisane w językach natywnie kompilowanych (np. C++). Zarzut ten odnosi się szczególnie do starych wersji *Javy*, kiedy zaawansowane mechanizmy takie jak JIT albo współbieżny odśmiecacz nie były dostępne. Obecnie zdania są mocno podzielone. Można podać przykłady programów zarówno takich, które w *Javie* będą wykonywały się wolniej niż w C++, jak i takich, które będą wykonywały się szybciej.
- *Javie* zarzuca się, że niezbyt dobrze nadaje się do zastosowań czasu rzeczywistego. Głównym problemem jest brak przewidywalności wydajności oraz nieoczekiwane przestoje powodowane działaniem odśmiecacza. W nowych wersjach *Javy* ten drugi problem został radykalnie ograniczony, jednak nadal do zastosowań czasu rzeczywistego lepiej stosować języki natywnie kompilowane.
- Często *Javie* zarzucane jest to, że ma mniejszą funkcjonalność niż np. C++, co ogranicza programistę. Jako przykład przywołuje się czasami fakt, że aby uruchomić niewielki program trzeba napisać dłuższy kod programu. Niektórzy jednak zwracają uwagę, że również język C++ nie ma wielu elementów, które można znaleźć w *Javie* jak np. klasy anonimowe, odśmiecacz pamięci, system pakietów, dynamiczne ładowanie klas czy reflection API.
- Java wymusza konwersję typów prostych (np. `int`) na odpowiadający typ referencyjny (np. `java.lang.Integer`) przy pracy z kolekcjami, co może mieć negatywny wpływ na wydajność w specyficznych sytuacjach i zmniejsza czytelność programu. Typy generyczne, wprowadzone w wersji 5 oraz tzw. autoboxing, czyli automatyczna konwersja pomiędzy typami prostymi i ich obiektowymi odpowiednikami usunęły problem czytelności. Programista może już także wybrać, jakim algorytmem zostanie zaimplementowana dana kolekcja (np. lista może być zaimplementowana jako tablica o zmiennym rozmiarze, drzewo, albo lista jednokierunkowa), co pozwala na poprawienie wydajności.

Dystrybucje języka Java

Pakiety

Java nie jest monolitem, lecz składa się z szeregu klas definiujących obiekty różnego typu. Dla przejrzystości klasy te pogrupowane są w hierarchicznie ułożone **pakiety**. Każdy pakiet grupuje klasy związane z pewnym szerokim zakresem zastosowań języka np. *java.io* (klasy wejścia-wyjścia), *java.util.prefs* (klasy użytkowe do obsługi preferencji) czy *java.awt* (system obsługi trybu graficznego). Hierarchię klas oddają nazwy pakietów, które skonstruowane są podobnie jak ścieżki dostępu do plików. Na przykład klasa *Preferences* znajdująca się w pakiecie *java.util.prefs* ma pełną nazwę: *java.util.prefs.Preferences*, co oznacza:

- *java* – pakiet należy do zestawu standardowych pakietów Javy,
- *util* – to różnego typu klasy użytkowe (pomocnicze) głównie organizujące obsługę różnego typu struktur danych,
- *prefs* – system obsługi preferencji w sposób niezależny od platformy, w którym preferencje systemowe i użytkownika są składowane w postaci hierarchicznego rejestru,
- *Preferences* – konkretna nazwa klasy.

Dzięki takiemu systemowi nazwy klas są niepowtarzalne, co pozwala uniknąć niejednoznaczności (np. czy chodzi o klasę *List* implementującą strukturę listy danych czy o *List* implementującą graficzną listę wyświetlaną w okienku).

Wszystkie klasy pisane przez programistów niezależnych powinny być umieszczane w innych hierarchiach. Firma Sun często zaleca, by w nazewnictwie klas niestandardowych przed właściwą nazwą pakietu stosować odwróconą nazwę domeny internetowej autora pakietu. Na przykład narzędzie *Ant* znajduje się w pakiecie *org.apache.ant*, co zapobiega konfliktom nazw z pakietami innych autorów, którzy również chcieliby nazwać swój pakiet *Ant*.

Domyślnie klasy pakietu nie są możliwe do użycia poza nim. Stąd nie występują konflikty nazw klas przy imporcie różnych pakietów. Klasa pakietu staje się publiczną przy deklaracji `public class Foo`.

JRE a JDK

Pakiety z hierarchii *java* i *javax* (dodatki wprowadzone w późniejszych wersjach) należą do podstawowego zestawu klas rozprowadzanych jako Java. Zestaw ten jest dostępny w dwóch wersjach: JRE (Java Runtime Environment) – udostępnia kod bajtowy wszystkich klas standardowych i wirtualną maszynę do ich uruchamiania, zaś JDK (Java Development Kit) dodatkowo udostępnia źródła tych klas oraz dodatkowe narzędzia takie jak kompilator, paker czy debugger. Podział ten wprowadzono dlatego, że użytkownik Javy do uruchamiania programów potrzebuje tylko JRE, natomiast do programowania działających aplikacji potrzeba już JDK.

Implementacje Javy

Potocznie pod nazwą **Java** rozumie się nie tylko język programowania, ale także całe środowisko (JDK) tworzone przez firmę Sun. Z tego uogólnienia wynikają pewne nieścisłości, jak np. to, że Java jest niezależna od architektury – nie jest to jednak cecha samego języka, a mechanizmu wirtualnej maszyny, wykorzystywanego w standardowej implementacji Suna.

Swoją własną implementację JDK, certyfikowaną w ramach Java Community Process, tworzy na przykład IBM, a na bazie kodu oryginalnej implementacji powstaje przeznaczona dla Linuksa Blackdown Java.

Istnieją też projekty odtworzenia poszczególnych elementów środowiska. Wśród nich są wirtualne maszyny Javy tworzone przez społeczność FLOSS SableVM i Kaffe, doświadczalny IBM-owski kompilator Jikes, czy optymalizowany pod względem szybkości dla architektur Intela JRockit, autorstwa firmy BEA. Najczęściej wykorzystują one bibliotekę standardowych klas rozwijaną w ramach projektu GNU Classpath.

Inne podejście prezentuje projekt GCJ, który pozwala kompilować programy w Javie bezpośrednio do kodu maszynowego.

JavaFX

W 2007 roku firma Sun ogłosiła swe plany wprowadzenia nowego języka skryptowego o nazwie *JavaFX Script*. Cele przyświecające temu językowi są podobne do tych, które zawsze towarzyszyły Javie:

- prostota tworzenia aplikacji
- niezależność od architektury
- platformą uruchomieniową mają być komputery PC oraz urządzenia przenośne (np. telefony komórkowe)

Język *JavaFX Script* nie jest zupełnie nowym produktem, lecz rozszerzeniem wcześniejszej Javy. Nowe aplikacje będą funkcjonować bez żadnych modyfikacji na każdej maszynie wirtualnej Javy.

Tym posunięciem firma Sun stara się dorównać konkurencji, która promuje takie rozwiązania jak:

- technologia Flex (Adobe)
- Silverlight (Microsoft)

Przypisy

- [1] <http://www.oracle.com/technetwork/java/javase/system-configurations-135212.html> Supported System Configurations for non-Itanium platforms → Lista implementacji poza architekturą Itanium
- [2] <http://www.oracle.com/technetwork/java/javase/downloads/default-177153.html> About 1.6.0_22 (6u22) for the Itanium® architecture → Uwagi do wydania 1.6.0_22 (6u22) dla architektury Itanium

Linki zewnętrzne

- Oficjalna strona języka Java prowadzona przez Oracle (<http://www.oracle.com/technetwork/java/index.html>) (ang.)
- Oficjalna strona języka Java dla użytkowników indywidualnych (<http://java.com/pl/>) (pol.)
 - Sprawdzanie wersji Javy w przeglądarce internetowej (<http://www.java.com/pl/download/help/testvm.xml>) (pol.)
- JDK firmy IBM (<http://www-106.ibm.com/developerworks/java/jdk/>) (ang.)
- Zbiór specyfikacji języka Java (<http://www.jcp.org/en/jsr/all>) (ang.)

Tutoriale

- Kurs dla początkujących (<http://javaprogramming.awardspace.com/>) (pol.)
- Dość rozbudowany kurs języka Java (http://www.webdeveloper.pl/manual_java/) (pol.)
- Oficjalny tutorial firmy Sun Microsystems (<http://java.sun.com/docs/books/tutorial/>) (ang.)
- WikiJava – The wiki with tutorials and examples in Java (<http://www.wikijava.org/>) (ang.)

Network Address Translation

NAT (skr. od ang. *Network Address Translation*, tłumaczenie adresów sieciowych; czasem *Native Address Translation*, tłumaczenie adresów rodzimych), znane również jako *maskarada sieci* lub *maskarada IP* (od ang. *network/IP masquerading*) – technika przesyłania ruchu sieciowego poprzez router, która wiąże się ze zmianą źródłowych lub docelowych adresów IP, zwykle również numerów portów TCP/UDP pakietów IP podczas ich przepływu. Zmieniane są także sumy kontrolne (tak IP jak i TCP/UDP), aby potwierdzić wprowadzone zmiany.

Większość systemów korzystających z NAT ma na celu umożliwienie dostępu wielu hostom w sieci prywatnej do internetu przy wykorzystaniu pojedynczego publicznego adresu IP (zob. brama sieciowa). Niemniej NAT może spowodować komplikacje w komunikacji między hostami i może mieć pewien wpływ na osiągi.

Zastosowanie NAT

Wraz ze wzrostem liczby komputerów w Internecie, zaczęła zbliżać się groźba wyczerpania puli dostępnych adresów internetowych IPv4. Aby temu zaradzić, lokalne sieci komputerowe, korzystające z tzw. *adresów prywatnych* (specjalna pula adresów tylko dla sieci lokalnych), mogą zostać podłączone do Internetu przez jeden komputer (lub router), posiadający mniej adresów internetowych niż komputerów w tej sieci.

Router ten, gdy komputery z sieci lokalnej komunikują się ze światem, dynamicznie tłumaczy *adresy prywatne* na adresy zewnętrzne, umożliwiając użytkowanie Internetu przez większą liczbę komputerów niż posiadana liczba adresów zewnętrznych.

NAT jest często stosowany w sieciach korporacyjnych (w połączeniu z proxy) oraz sieciach osiedlowych.

Wady i zalety NAT

Korzystanie z Internetu poprzez NAT ma pewne wady:

- nie można na własnym komputerze uruchomić serwera dostępnego w Internecie bez zmian wymagających interwencji administratora,
- utrudnione korzystanie z sieci P2P i bezpośrednie wysyłanie plików,
- utrudnione korzystanie z gier sieciowych z osobami spoza sieci, należy skorzystać z aplikacji VPN, np. Hamachi, można też przekierować porty, ale wtedy z gry może skorzystać tylko jeden komputer.

Do zalet należą:

- większa anonimowość, gdyż serwery, z którymi nastąpiło połączenie nie mogą zidentyfikować konkretnego hosta po samym adresie IP,
- możliwość dostępu do Internetu dla większej ilości komputerów niż ilość dostępnych publicznych adresów IP.

Rodzaje NAT

Można wyróżnić 2 podstawowe typy NAT:

- **SNAT** (*Source Network Address Translation*) to technika polegająca na zmianie adresu źródłowego pakietu IP na jakiś inny. Stosowana często w przypadku podłączenia sieci dysponującej adresami prywatnymi do sieci Internet. Wtedy router, przez który podłączono sieć, podmienia adres źródłowy prywatny na adres publiczny (najczęściej swój własny).
- **DNAT** (*Destination Network Address Translation*) to technika polegająca na zmianie adresu docelowego pakietu IP na jakiś inny. Stosowana często w przypadku, gdy serwer, który ma być dostępny z Internetu ma tylko adres prywatny. W tym przypadku router dokonuje translacji adresu docelowego pakietów IP z Internetu na adres tego serwera.

Szczególnym przypadkiem SNAT jest maskarada, czyli sytuacja, gdy router ma zmienny adres IP (np. otrzymuje go w przypadku połączenia modemowego dodzwanianego). Wtedy router zmienia adres źródłowy na taki, jak adres interfejsu, przez który pakiet opuszcza router.

Oprogramowanie

W przypadku systemu operacyjnego Linux funkcje NAT definiowane są za pomocą programów iptables lub ipchains, a w przypadku FreeBSD ipfw (IP firewall), ipf (IP filter) lub pf (OpenBSD Packet Filter).

Linki zewnętrzne

- Zasada działania NAT ^[1]

Przypisy

[1] <http://www.tech-portal.pl/content/view/29/38/>

Programowanie strukturalne

Programowanie strukturalne to paradygmat programowania zalecający hierarchiczne dzielenie kodu na bloki, z jednym punktem wejścia i jednym lub wieloma punktami wyjścia. Chodzi przede wszystkim o nieużywanie (lub ograniczenie) instrukcji skoku (*goto*). Dobrymi strukturami są np. instrukcja warunkowa (*if*, *if...else*), pętle (*while*, *repeat*), wyboru (*case*, ale nie *switch* z C i potomnych). Strukturalność zakłócają instrukcje typu: *break*, *continue*, *switch* (w C itp.), które jednak w niektórych przypadkach znacząco podnoszą czytelność kodu.

Popularne języki programowania

Praktycznie w każdym języku można programować strukturalnie, jednakże w niektórych jest to styl naturalny (np. w Pascalu), a w niektórych łatwo wypaść z tego stylu (np. w BASIC-u).

Referencja (informatyka)

Referencja w informatyce to wartość, która zawiera informacje o położeniu innej wartości w pamięci lub na innym nośniku danych. W odróżnieniu od wskaźników, zarządzanie referencjami realizowane jest wyłącznie przez kompilator lub interpreter, a programista nie posiada żadnych informacji o konkretnym sposobie implementacji referencji.

Mechanizm referencji jest powszechnie wykorzystywany w językach programowania, gdyż idea jego działania polega m.in. na uniemożliwieniu wykonywania operacji uznawanych za potencjalnie niebezpieczne, które w wypadku błędu programistycznego mogłyby doprowadzić do awarii, zwiększając w ten sposób niezawodność oprogramowania^[1]. Referencje nie ograniczają możliwości programisty – wszystkie operacje, które nie bazują na jawnej znajomości organizacji pamięci, mogą być zaimplementowane wyłącznie za ich pomocą. Z tego powodu w wielu językach programowania wysokiego poziomu jest to jedyny mechanizm obsługi pamięci (np. Java^[1]). W innych referencje współistnieją ze wskaźnikami (np. C++^[1]).

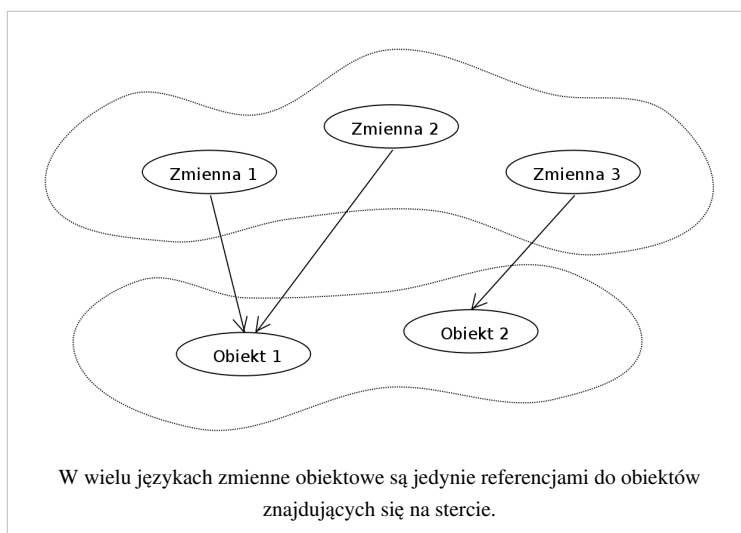
Zastosowania

Dopełnienie mechanizmu wskaźników

W niskopoziomowych językach takich, jak C++, referencje są implementowane jako oferująca mniej możliwości, ale bezpieczniejsza odmiana wskaźników. Podobnie jak i one, referencje wskazują tutaj na pewien obszar pamięci z tą różnicą, że nie mogą być modyfikowane. Innymi słowy, już podczas tworzenia zmiennej będącej referencją musimy określić, na co będzie ona wskazywać, a ponadto referencji nie wolno później modyfikować.

Referencje do obiektów

W wielu obiektowych językach wysokiego poziomu (np. Java) programista nie może samodzielnie zarządzać pamięcią. Sposób jej organizacji zależy od implementacji konkretnej maszyny wirtualnej. Wyróżniony jest w niej obszar zwany stertą, w której przechowywane są wszystkie utworzone przez aplikację obiekty. Zmienne obiektowe nie przechowują bezpośrednio obiektów, a jedynie referencje do sterty. Przypisując wartość jednej zmiennej obiektowej do drugiej, lub przekazując ją przez wartość, przekazujemy jedynie referencję, co oznacza, że od tego momentu obie zmienne będą odnosić się do tego samego obiektu, a zmiany stanu wprowadzone za pośrednictwem jednej z nich będą natychmiast widoczne przez drugą.



Dodatkowo, dopuszcza się istnienie pustych referencji (*null*), które aktualnie nie wskazują na żaden obiekt.

Odśmiecanie pamięci

Referencje mają kluczowe znaczenie dla algorytmów odśmiecania pamięci. Umożliwiają śledzenie, które obiekty i wartości są wciąż używane przez program, a które mogą być bezpiecznie usunięte. Najprostszą techniką kontroli czasu życia danych na sterze jest zliczanie referencji^[2]. Aplikacja utrzymuje specjalny licznik dla każdego obiektu, który przechowuje informacje o liczbie referencji aktualnie na niego wskazujących. Jeśli spadnie on do zera, oznacza to, że nie jest już używany, a przydzielona mu pamięć powinna być zwolniona. Technika ta nie potrafi jednak wykrywać cykli (obiekty A i B posiadają referencje do siebie nawzajem, lecz reszta programu już nie).

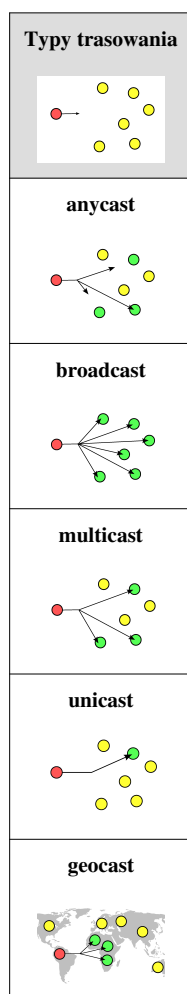
Zobacz także

- Zmienna wskaźnikowa
- Odśmiecanie pamięci
- Słaba referencja

Przypisy

- [1] Harry Henderson: *Encyclopedia of computer science and technology*. Facts on File, Inc, 2008, ss. 376 - 377. ISBN 9780816063826. [\(ang.\)](#)
- [2] Recycling techniques (<http://www.memorymanagement.org/articles/recycle.html>) [\(ang.\)](#). The Memory Management Reference (<http://www.memorymanagement.org/>). [dostęp 2010-12-08].

Trasowanie (telekomunikacja)



Trasowanie (ang. *routing*, pol. ruting, routowanie) – w informatyce wyznaczanie trasy i wysłanie nią pakietu danych w sieci komputerowej. Urządzenie węzłowe, w którym kształtowany jest ruch sieciowy, nazywane jest routerem, może to być np. komputer stacjonarny lub dedykowane urządzenie (również nazywane *routerem*).

Pakiety przesyłane przez sieć opatrzone są adresem nadawcy i odbiorcy. Zadaniem routerów jako węzłów pośrednich między nadawcą a odbiorcą jest przesłanie pakietów do celu po jak najlepszej ścieżce. Typowy router bierze pod uwagę tylko informacje z nagłówka IP, czyli sprawdza tylko informacje z warstwy sieci (trzeciej) modelu OSI. Obowiązkiem routera IP przy przekazywaniu pakietu dalej do celu jest obniżenie o jeden wartości TTL (ang. *Time To Live*, czas życia). Datagram IP, który trafia do routera z wartością 1 (a zostanie ona zmniejszona na tym routerze do 0) w polu TTL zostanie utracony, a do źródła router odsyła datagram ICMP z kodem TTL Exceeded.

Routery utrzymują tablice trasowania, na podstawie których kierują pakiety od określonych nadawców do odbiorców, bądź kolejnych routerów. Tablica może być budowana statycznie (trasowanie statyczne) lub dynamicznie (protokoły trasowania dynamicznego, takie jak RIP, IGRP, EIGRP, OSPF, BGP, IS-IS).

Trasowanie ma na celu możliwie najlepiej (optymalnie) dostarczyć pakiet do celu. Pierwotnie jedynym kryterium wyboru było posiadanie jak najdokładniejszej trasy do celu, ale obecnie protokoły trasowania mogą uwzględniać podczas wyboru trasy również takie parametry jak priorytet pakietu (standardy ToS/DSCP), natężenie ruchu w poszczególnych segmentach sieci itp. W przypadku trasowania brzegowego (wykorzystującego BGP) w Internecie wybór trasy jest silnie związany z polityką poszczególnych dostawców (i zawartymi między nimi umowami o wymianie ruchu) i bywa daleki od optymalnego.

Popularnym algorytmem służącym do wyznaczania tras w sieciach wewnętrznych jest algorytm Dijkstry wyznaczania najkrótszej ścieżki w grafie (np. OSPF).

Przeciążanie funkcji

Przeciążanie funkcji (ang. *overloading*) - skrótowa nazwa na **przeciążanie nazwy funkcji**; w programowaniu występowanie pod taką samą nazwą wielu funkcji różniących się zestawem argumentów. W trakcie kompilacji bądź parsowania program znajduje właściwą funkcję po liczbie oraz typach argumentów. Możliwe jest więc współistnienie kilku funkcji o tej samej nazwie, lecz różniących się typami argumentów.

Przykłady

Przykład w języku C++

```
void funkcja (int);  
  
void funkcja (int, char);  
  
void funkcja (float, int);
```

Przykład w języku Object Pascal

```
procedure Foo(f: integer); overload;  
  
procedure Foo(s: string; f: integer); overload;  
  
procedure Foo(d: double; f: integer); overload;
```

Przeciążanie funkcji w języku Object Pascal dostępne jest od środowiska Delphi w wersji 4.

Przykład w języku PL/1

Nieco inaczej przeciążanie nazw realizowano w historycznych już dzisiaj językach. Jako przykład posłużyć może deklaracja rodziny procedur w języku PL/1, w którym definiowano różne procedury, a następnie nazwę nowej procedury i listę procedur wywoływanych przez tę nazwę dla konkretnych przypadków listy argumentów^[1].

Przykład w PL/1:

```
DCL A GENERIC (PR1 WHEN (FLOAT) ,
                PR2 WHEN (CHAR) ,
                PR3 WHEN (FLOAT, CHAR) ,
                PR4 WHEN (LABEL) ) ;
```

Przykład w języku Forth

Jeszcze inaczej przeciążanie funkcji (lub ogólniej identyfikatorów) traktowane jest w języku Forth, co wynika ze specyfiki tego języka. W języku Forth definiujemy słowa (które mogą oznaczać podprogram, stałą, zmienną, słownik, kompilator (*w sensie języka Forth*)). Język umożliwia definiowanie słów w określonych słownikach. Dzięki temu można przeciążać nazwy lecz nie w obrębie pojedynczego słownika, ale w obrębie całej (drzewiastej) struktury słowników, umieszczając słowa o tych samych nazwach w różnych słownikach. Zmieniając kolejność przeszukiwania słowników, wybierane będą różne podprogramy (z różnych słowników o tej samej nazwie)^{[2] [3]}.

Przykład:

```
VOC1 DEFINITIONS
: OP-D DUP ROT DUP + ;
VOC2 DEFINITIONS
: OP-D DUP ROT DUP * ;
VOC3 DEFINITIONS
: OP-D DUP ROT DUP - ;
VOC4 DEFINITIONS
: OP-D DUP ROT DUP / ;

...

3 4
VOC1
OP-D .
VOC2
OP-D .
```

Powyższy przykład definiuje podprogram OP-D, który w zależności od tego, który ze słowników VOC1, VOC2, VOC3, VOC4 będzie przeszukiwany w pierwszej kolejności wykona dodawanie, mnożenie, odejmowanie lub dzielenie, z zachowaniem argumentów operacji. Działanie powyższych instrukcji spowoduje wyprowadzenie liczby 7 (3+4), a następnie 12 (3*4).

Sytuacje niejednoznaczne i błędy przeciążania

Nie zawsze kompilator jest w stanie odpowiednio odróżnić funkcje o tej samej nazwie, muszą się one od siebie wyraźnie różnić. Czasem zdarzają się też sytuacje, w której dwie funkcje o tej samej nazwie mogą współistnieć, ale dane wywołanie byłoby niejednoznaczne. Problemy są następujące:

- Jeśli funkcje (jedna lub obie) przyjmują argumenty domyślne, to mogą współistnieć jeśli te "niedomyślne" argumenty mają te same typy, tyle że wywołanie z podaniem wyłącznie wymaganych argumentów będzie wtedy niejednoznaczne (i odrzucone). Najlepiej nie mieszać ze sobą przeciążania i argumentów domyślnych.
- Podczas wybierania funkcji do wywołania uwzględnia się też domyślne konwersje i trzeba znać ich priorytety; np. jeśli funkcję przeciążymy na typy **short** i **long**, to wywołanie z typem **float** jest niejednoznaczne.
- W zależności od języka przeciążać można tylko na podstawie typów parametrów, lub na podstawie typów parametrów i wartości zwracanej (np. w języku Ada).
- Uzyskanie wskaźnika do przeciążonej funkcji jest niemożliwe. Udaje się to tylko w przypadku wyrażeń, w których oczekuje się konkretnego typu od takiego wyrażenia, np. przypisując do zmiennej typu wskaźnika do funkcji - wtedy zostanie wybrana ta funkcja, która pasuje do sygnatury funkcji, do której jest ten wskaźnik. Nie można jednak takiej funkcji używać jako argumentu do szablonu funkcji - jest to znany problem z używaniem funkcji `std::transform` wraz z `std::tolower`.

Przypisy

- [1] Jan Bielecki: *Rozszerzony PL/I i JCL w systemie OS/RIAD*. Warszawa: Państwowe Wydawnictwo Naukowe, 1986, seria: Biblioteka Informatyki. ISBN 83-01-06146-4. (pol.)
- [2] Jan Bielecki: *Język FORTH*. Warszawa: Wydawnictwa Naukowo-Techniczne, 1988, seria: Mikrokomputery. ISBN 83-204-0930-6. (pol.)
- [3] Jan Ruszczyk: *Poznajemy FORTH*. Warszawa: SOETO, 1987, seria: Informatyka mikrokomputerowa. (pol.)

POST (metoda)

POST - metoda przesyłania danych w sieci internet. Istnieje w ramach protokołu HTTP i wykorzystywana jest najczęściej do wysłania informacji z formularza znajdującego się na stronie internetowej.

Metoda POST powinna być stosowana tam, gdzie dane odwołanie jest formą interakcji z użytkownikiem, która nie może być utrwalona w formie adresu. Powinna być także stosowana tam, gdzie dana operacja może wywołać wiążące dla użytkownika skutki - np. zapisanie na listę dyskusyjną^[1].

Przykładowy pakiet z informacjami POST wygląda tak:

```
POST /login.jsp HTTP/1.1
Host: www.mysite.com
User-Agent: Mozilla/4.0
Content-Length: 29
Content-Type: application/x-www-form-urlencoded

login=julius&password=zgadnij
```

Przypisy

[1] URIs, Addressability, and the use of HTTP GET and POST (<http://www.w3.org/2001/tag/doc/whenToUseGet.html>). W3C, 2003.

Linki zewnętrzne

- RFC 2616

GET (metoda)

Metoda GET – sposób przekazywania danych pomiędzy kolejnymi odsłonami dokumentów sieciowych w protokole HTTP. Polega na umieszczeniu par *parametr=wartość* w adresie URI strony, np. `index.php?lang=pl&cat=2&sidebar=yes`.

Metoda GET powinna być stosowana tam, gdzie dane odwołanie powinno być adresowalne, to znaczy gdy w URI może być przechowywany pewien stan aplikacji, stały w czasie, do którego użytkownik mógłby chcieć wrócić (np. pozycja na mapie). Strony dostępne przez metodę GET powinny być bezpieczne dla osoby odwiedzającej (ang. *safe interactions*), tzn. nie powinny wywoływać wiążących dla niej skutków (np. zapisanie na listę dyskusyjną)^[1].

Ciąg po znaku zapytania może być wykorzystywany przez skrypty serwerowe do dostosowania generowanego kodu HTML do preferencji klienta. Najczęściej służy to do podania podstrony serwisu, którą chcemy obejrzeć, wersji językowej lub specjalnej wersji do wydruku. Metoda GET jest też często używana do przekazywania identyfikatora sesji. Powyższy przykład przekazuje informacje serwerowi, że:

- zmienna *lang* jest równa "pl"
- zmienna *cat* jest równa "2"
- zmienna *sidebar* jest równa "yes"

Zwrócenie wartości zmiennych z przykładu w języku programowania PHP

```
<?php
//wyświetlenie całej zawartości tablicy $_GET
echo "<pre>";
print_r($_GET);
echo "</pre>";

// przed pobraniem parametrów warto sprawdzić czy index zmiennej
został ustawiony
// czyli użyć funkcji isset($_GET['lang'])

//pobranie danych z tablicy $_GET, która służy do obsługi metody GET
$lang = $_GET['lang'];
$cat = $_GET['cat'];
$sidebar = $_GET['sidebar'];

echo "Wartość parametru lang: $lang<br />\n"; //zwrócenie wartości
parametru lang w osobnej linii
echo "Wartość parametru cat: $cat<br />\n"; //zwrócenie wartości parametru
cat w osobnej linii
echo "Wartość parametru sidebar: $sidebar<br />\n"; //zwrócenie wartości
```



```
parametru sidebar w osobnej linii
?>
```

Wynikiem powinna być strona (dla ułatwienia w wyniku podano treść, która będzie widoczna w przeglądarce po wykonaniu zapytania, tj. usunięto `
` itp.):

```
Array
(
    [lang] => pl
    [cat] => 2
    [sidebar] => yes
)
Wartość parametru lang: pl
Wartość parametru cat: 2
Wartość parametru sidebar: yes
```

Korzystając z tej metody można tworzyć podstrony w jednym pliku:

```
<?php
    echo "Strona główna<br /><br />\n";
    echo '<a href="?zmienna=ja">Podstrona</a>\n';

    if ($_GET['zmienna'] == 'ja') { //jeśli parametr 'zmienna' przyjmie
watość 'ja'
        echo 'Podstrona';
    }
?>
```

Wówczas gdy uruchomimy powyższy przykład w przeglądarce z adresem uzupełnionym o `?zmienna=ja` powinniśmy otrzymać:

```
Strona Główna
Podstrona
```

Ponieważ spełniliśmy warunek w instrukcji `if`.

Przypisy

[1] URIs, Addressability, and the use of HTTP GET and POST (<http://www.w3.org/2001/tag/doc/whenToUseGet.html>). W3C, 2003.

Linki zewnętrzne

- RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1 - specyfikacja protokołu HTTP/1.1 ([ang.](#))

Przeszukiwanie liniowe

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

Liczba koniecznych porównań zależy wprost od położenia szukanego elementu w sekwencji danych – wynosi od 1 do n , gdzie n to całkowita liczba elementów. Algorytm ma złożoność $O(n)$.

Psuedokod:

```
szukany_klucz := określona wartość;
znaleziony_indeks := ?;
for index := 1 to n do
    if tablica[index].klucz = szukany_klucz then
        begin
            {znaleziono element o podanym kluczu, zapamiętujemy pozycję w
tablicy}
            znaleziony_indeks := index;
            break; { koniec iterowania}
        end;
```

Wyszukiwanie liniowe może być jedynym sposobem wyszukiwania, gdy nie wiadomo niczego na temat kolejności kluczy.

Dla dużej liczby danych algorytm jest bardzo nieefektywny, jednak gdy danych jest względnie mało, jest z powodzeniem stosowany (np. w tablicach mieszających, w których problem kolizji rozwiązuje się metodą łańcuchową).

Dodatkowo jeśli wiadomo, że pewne klucze mogą być wyszukiwane częściej niż inne, można modyfikować kolejność danych, tak aby ponowne wyszukiwanie tego samego klucza kończyło się powodzeniem szybciej. Metoda ta nosi nazwę *move-to-front*, Donald Knuth wymienia w swojej pracy *Sztuka programowania* dwie strategie:

1. natychmiastowe przesunięcie znalezionego elementu na początek sekwencji,
2. przesunięcie tylko o jedną pozycję w stronę początku sekwencji.

Wyszukiwanie binarne

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 przejrzania wszystkich elementów tablicy.

Zasada działania algorytmu

Uporządkowana tablica jest dzielona na coraz mniejsze przedziały do momentu, gdy szukany element zostanie znaleziony, bądź przedział osiągnie długość zero, co oznacza brak elementu.

W pojedynczym kroku rozważa się jeden przedział charakteryzowany dwoma indeksami: początkowym a i końcowym b . Algorytm rozpoczyna wyszukiwanie od całej tablicy.

Następnie wyznaczany jest środek tego przedziału $c = \left\lfloor \frac{a+b}{2} \right\rfloor$. Wówczas testowane jest, czy element zapisany

pod indeksem c jest tym poszukiwanym — jeśli tak, to algorytm w tym miejscu kończy działanie.

W przeciwnym razie przedział jest zawężany - dzięki uporządkowaniu danych wiadomo, że albo poszukiwany element może znajdować się gdzieś przed indeksem c albo za nim. Innymi słowy wybór ogranicza się do przedziału $[a, c-1]$, gdy poszukiwany element jest mniejszy od zapisanego pod indeksem c , albo $[c+1, b]$ w przeciwnym razie.

Algorytm kończy się niepowodzeniem, jeśli przedział będzie pusty, tzn. $b < a$ (lewy koniec przedziału "znajdzie się" za prawym końcem).

Pseudokod

```
A := [...] { n-elementowa tablica uporządkowana }
lewo := 0      { indeks początku przedziału }
prawo := n-1   { indeks końca przedziału - początkowo cała tablica A }

y := poszukiwana wartość

while lewo < prawo do
  begin
    srodek := (lewo + prawo)/2; { dzielenie całkowitoliczbowe }
    x := A[srodek];
    if x < y then
      lewo := srodek + 1;
    else
      prawo := srodek;
      { Zauważmy, że szukana wartość nadal znajduje się w przedziale
        A[lewo]..A[prawo] (a przedział się zmniejszył). }
    end;
  end;

wartość_znaleziona := A[lewo]; { O ile w tablicy A był element o
wartości y, nadal jest on w przedziale A[lewo..prawo], a lewo = prawo }
```

Wyszukiwanie interpolacyjne

Wariant wyszukiwania binarnego, w którym punkt podziału (indeks c) jest wyznaczany metodą interpolacji liniowej.

Jeśli wartości kluczy na krańcach przedziału wynoszą X_a i X_b i poszukiwana wartość $X_a \leq X \leq X_b$, wówczas indeks można wyznaczyć jako $c = \lfloor a + t \cdot (b - a) \rfloor$, gdzie parametr wynika z wartości kluczy:

$$t = \frac{X - X_a}{X_b - X_a}.$$

Algorytm charakteryzuje o wiele lepsza średnia złożoność obliczeniowa niż zwykłego wyszukiwania binarnego, wynosi bowiem $O(\log \log n)$, a nie $O(\log n)$. Złożoność w przypadku pesymistycznym jest jednak liniowa.

Jak podaje Knuth, testy empiryczne wykazują, że podejście to dobrze sprawdza się dla bardzo dużych rozmiarów tablic, dla niewielkich nie widać wyraźnej przewagi ze względu na bardziej złożone wyliczanie indeksu c .

Pomysłodawcą metody był W.W. Peterson; została ona opracowana ok. 1957 roku.

Bibliografia

- Donald Knuth, *Sztuka programowania. Tom III: Sortowanie i wyszukiwanie*, WNT 2002

Źródła i autorzy artykułu

Lista *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26979396> *Autorzy:* AdSR, Azureus, Birczanin, Derbeth, Joi, Jozef-k, Kazimierz drobicki, Kocio, Kuszi, Mestiv, Michalgarbowski, Omega933, Paweł Ł Zawada, Pleple2000, Plushy, Rzukow, Severson, Sid, Szczepan1990, Taw, Tilia, Trzmiel, Visor, Wojciech mula, WojciechSwiderski, A, 18 anonimowych edycji

Kolejka (informatyka) *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26459756> *Autorzy:* Azureus, CD, Gluth, Julo, Margos, Mciura, Nedops, Rzukow, Shaqspeare, Sobi3ch, Tojot, Trzmiel, Wojciech mula, Xywek, 5 anonimowych edycji

Stos (informatyka) *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26935229> *Autorzy:* 2coma7, Azureus, Derbeth, Gang65, Grotesque, Julo, Kbsc, Kocio, Kocur, Lampak, Leinad, Maikking, Mat86, Merdis, Olaf, Piomar, Pivosz, Rklisowski, Rnm, Sobi3ch, Stok, Trzmiel, Wojciech mula, Xywek, Yusek, 56 anonimowych edycji

Drzewo (informatyka) *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26148069> *Autorzy:* A., Bluszczokrzew, Derbeth, Exe, Intol, Kocio, Lcamtuf, Mathel, Mbiskup, Norbert.wolniewicz, Pz, Qu3a, SpiderMum, Sulikk, Superborsuk, Taw, ToSter, Tomko222, Wojciech mula, 21 anonimowych edycji

Binarne drzewo poszukiwań *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=27019449> *Autorzy:* Anatol, Beno, CiaPan, CudPotwórca, Derbeth, Ejcum, Joi, Kamulek, Krzysztof.math, Matekm, Mathel, Matiz87, Michalgarbowski, Movax, Nux, Pur, Rolnikov, Royas, Saf, Severson, Spykaj, ToSter, Wojciech mula, Zyx, 43 anonimowych edycji

Drzewo AVL *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=25155655> *Autorzy:* Anatol, Bukaj, Buyek, CiaPan, Derbeth, Foobar, Joi, Kocur, Krzymar, Kyokpae, MTM, Masur, Mathel, Merlinthe, Michalgarbowski, Mlepicki, Nux, Pooteek, Royas, Saf, Spook, Sunridin, VanDut, Wojciech mula, 15 anonimowych edycji

Drzewo czerwono-czarne *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26591150> *Autorzy:* Anatol, AndrzejHelu, BeŻet, CutterTheCat, Derbeth, Glonoad, Kocur, Malarz pl, Matekm, Mathel, Michalgarbowski, Micpol, Pamelus, Royas, Saf, Stok, Template namespace initialisation script, Wojciech mula, Zyx, 14 anonimowych edycji

Drzewo RST *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=24596832> *Autorzy:* Kamil9874, Marekos, Qu3a, 1 anonimowych edycji

Funkcja skrótu *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=27014374> *Autorzy:* 4C, ABach, AI, Ananael, Beno, Borszczuk, Cyrylm, Dodek, Filemon, Googl, Izaak, Kbsc, Kimbar, Kocio, Kravietz, Kuszi, Lcamtuf, Mieszau, Mikolajpodbielski, Mtrojnar, Olaf, Pkuczynski, Qu3a, Robsuper, Roo72, Selena von Eichendorf, Sq7obj, Step, Superborsuk, Taw, Tomta1, Wazow, Zjem ci chleb, Zzzziomexxx, 19 anonimowych edycji

Algorytm Knutha-Morrisa-Pratta *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=21639734> *Autorzy:* A., BartekChom, Byczek1, Iceberg, Kapitanzbik, Kbsc, Kuszi, Markotek, Przykuta, RafałRawicki, Redruid, Sentiel, Serpens, Sheriff, Siedlaro, Wojtekwas, Wyksztalcioch, 16 anonimowych edycji

Algorytm Karpa-Rabina *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=21747752> *Autorzy:* A., Goralesco1, Karmelki90, Kuszi, Mario58, Olaf, Qu3a, Slaweks, Wojciech mula, 8 anonimowych edycji

Algorytm Boyera i Moore'a *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26279871> *Autorzy:* A., Googl, Kotek1986, Logolego, Matekm, Matma Rex, NH2501, Ohny, 3 anonimowych edycji

Sortowanie bąbelkowe *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26637155> *Autorzy:* A., Adonis, Ajsmen91, Beau, Brd, Cathy Richards, Dodek, Faramir, Farmer Jan, Googl, Hulek, Jacek kl, Jakozaur, Jakubhal, Jano, Joi, Jotempe, Kocio, Kuszi, LukKot, ML4332, Macar, Mariuszli, Matekm, Matma Rex, Matt Z, MonteChristof, Mpfiz, Mr. Hania, Niki K, Olaf, Paelius, PawełMM, Piecu, Refycul, Rnm, RomanK, Różowy Floyd, Sfu, SpiderMum, Stiepan Pietrov, Stok, Stotr, Stv, Taw, Volfen, Wojciech mula, Wojtalik, Yusek, conversion script, pw201.warszawa.sdi.tpnet.pl, Zangle, 108 anonimowych edycji

Sortowanie przez wstawianie *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26566994> *Autorzy:* A., Aptu, Beau, C4, Cathy Richards, Cubiszon, Derbeth, Eldevarth, Googl, Havelock V., Jakozaur, Jano, Jerry, Kocio, Kpjas, Lord t, Mariuszli, Markotek, Matekm, Mateusz Rosiek, Matusz, McMonster, Mciura, Mik, Olaf, Omikronsc, Pippinbb, Ple91, Refycul, Roffik, RomanK, Różowy Floyd, Staszek99, Stiepan Pietrov, Stotr, Stv, Wojciech mula, Xaweryz, conversion script, pj211.sosnowiec.sdi.tpnet.pl, 64 anonimowych edycji

Sortowanie przez scalanie *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26651034> *Autorzy:* A., Beau, CiaPan, Dawidholewa, Derbeth, DrJolo, Eldevarth, Farmer Jan, Fullofstars, Ghost, Googl, Katafrakt, Kocio, Kpjas, Krzacz, Kuszi, LEW21, Malarz pl, Masur, Mat86, Mik, Olaf, Rnm, Stiepan Pietrov, Wiklef, WojciechSwiderski, Wpedzich, 38 anonimowych edycji

Sortowanie przez zliczanie *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26514055> *Autorzy:* 4C, A., Ajank, Beau, Derbeth, Farmer Jan, GiM, Jakozaur, Joymaster, Kaszkawal, Kimbar, Kixner, Kocio, Markotek, Mik, Selena von Eichendorf, Stawel, Szymon p, Tomekohio, 14 anonimowych edycji

Sortowanie kubełkowe *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=21421806> *Autorzy:* A., Beau, BeŻet, CiaPan, Czupirek, Lolek01, Louve, MatthiasGor, Nux, Rentier, Sfu, Wpedzich, 18 anonimowych edycji

Sortowanie pozycyjne *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26589379> *Autorzy:* A., Beau, Bulwersator, Dplaneta, Ewkaa, Masur, Mik, Myki, Olaf, Rnm, Slaweks, 11 anonimowych edycji

Sortowanie biblioteczne *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26642116> *Autorzy:* Jotempe, Lampak, Lord Barman, Mik, Mix321, Qu3a, Tdc6502, XAVeRY, 3 anonimowych edycji

Sortowanie szybkie *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26998006> *Autorzy:* A., Akurat, Aleks80, Alus, BeŻet, CiaPan, Derbeth, Ejdzaj, Farary, Farmer Jan, Firkraag, GiM, Googl, Gryyf, Hipertracker, Jacek kl, Jan Winnicki, Jano, Jersz, Joi, Kbsc, Kisztof, Kixner, Kochas315, Kocio, Koshmaar, Kpjas, Lampak, Macper90, Matekm, Olaf, Piastu, Picus viridis, Qysiu, Rnm, Rzukow, Skalee, Skylark4ks, Spook, Step, Stiepan Pietrov, Stok, Stv, Tadam, Taw, Tewux, VanDut, Wazow, Wiklef, Wojciech mula, Xpicto, Youandme, 67 anonimowych edycji

Sortowanie przez wybieranie *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26650350> *Autorzy:* A., ArturM, Bulwersator, Derbeth, Enkidu666, Esio, Gilrandir, Hipertracker, Jakozaur, Kocio, Kuszi, MK wars, Macar, Malarz pl, Mariuszli, Matusz, Mciura, Mik, Mik01aj, Mpfiz, MrNeo, Nux, Olaf, Piastu, Rdroz, RomanK, Stiepan Pietrov, Tomkiewicz, Wojciech mula, Xion AS, 39 anonimowych edycji

Sortowanie Shella *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26794543> *Autorzy:* A., Blade BMRQ, Bulwersator, Calasilyar, Conish, Derbeth, Farary, Farmer Jan, Googl, Kirq, Lcamtuf, Leopold, Maikking, MariuszR, Mciura, Olaf, RomanK, Roo72, Step, Stiepan Pietrov, Stok, Teukros, Wojciech mula, Xpicto, Zbiczek, 29 anonimowych edycji

Sortowanie grzebieniowe *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26674294> *Autorzy:* A., Bartos34, BeŻet, Bulwersator, CiaPan, Farmer Jan, Kisztof, Kocio, Olaf, Selena von Eichendorf, Sławomir Pryczek, Wimmer, 7 anonimowych edycji

Sortowanie introspektywne *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=23844597> *Autorzy:* A., Benedict, Derbeth, Farmer Jan, Jakozaur, Vindicator, 4 anonimowych edycji

Sortowanie przez kopcowanie *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26591813> *Autorzy:* A., Beno, Brzozza93, Bukaj, CiaPan, DamSob, Derbeth, Eldevarth, Farmer Jan, Filu, Jakozaur, Jano, Kamilmara, Kbsc, Kielek, Kocio, Kuszi, Leafnode, Malarz pl, Mik, Morte, Oort, Pleple2000, RedRad, Schizofrenikh, Scobac2, Stotr, Wojciech mula, 34 anonimowych edycji

Sortowanie koktajlowe *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=24404805> *Autorzy:* A., BennyBlanco, DrJolo, Piastu, Telpeloth

Graf (matematyka) *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26898513> *Autorzy:* A., ABX, ASorEX, Adoomer, Argothiel, Baartosz, Batonikus, Beau, Beaumont, Beno, Bocianski, Brohacz, Buldożer, CD, CiaPan, Delimata, Derbeth, Farmer Jan, Faxe, Galileo01, Genesis, Googl, Ingen, Jan Winnicki, Jersz, Jpiw, Kbsc, Komar00727, Kuki, Kuszi, Lagesag, LeetBaal, LukKot, Lukas Canonius, Maire, McCartney, Milek80, Mlepicki, MonteChristof, Nazin, Nux, Odojl, Olaf, Pamelus, Petryk, Pioziom, Pleple2000, Polimerex, Rnm, Robert Borkowski, Rosomak, Royas, Rzukow, Selena von Eichendorf, Ser spodlaski, Sialababamak, Sigvatr, Steal, Step, Stotr, Superborsuk, Szemek, Tanja5, Taw, ToSter, Witek52, Wojciech mula, WojciechSwiderski, Wojtekwas, Yaevin, Zero, 57 anonimowych edycji

Algorytm Grahama *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26460680> *Autorzy:* CiaPan, Dome, Googl, Mario58, Masur, MatFizka, Step, Wojciech mula, 4 anonimowych edycji

AJAX *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=25891221> *Autorzy:* Airwolf, Andrzej19, Areli, Astromp, Beau, BeŻet, Catz, Cydor, Darekm, Derbeth, Dkrysiak, Draakhan, Eteru, Filemon, Good(k)night, Grotesque, Grypc net, Holek, Jarekadam, KamStak23, Kangel, Kocio, Konradr, MOP, Maikking, Makawity, Malin, Mathel, Mith05, Moarc, Nux, PMG, Pawelkg, Piotrek290, Pnti, Pojdulos, Ptak82, Qu3a, Rosamarcin, Royas, Step, Stimoroll, Szypec, TIM, Tanoo, Taw, Tomfab, Vinyanov, Wariat, Wpedzich, XDaniX, Xywek, Zergu, 75 anonimowych edycji

edycji

Hypertext Transfer Protocol *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26264252> *Autorzy:* Aajanek, Adzinok, Alfons6669, Arek1979, Bahador, BartłomiejB, Beau, Blueshade, Bolkow86, CiaPan, Ciacho5, Darekm, Dashmen, Derbeth, Dziabong, Faxe, Felix, Filemon, FxJ, Gang65, Hashar, Herr Kriss, Ignasiak, JDavid, Kb, Klapi, Klemen Kocjancic, Kocio, Kravietz, Kroczu, Leafnode, Ludo Ag.Ent, LukKot, Lzur, Macar, MarGr, Marcimon, Margoz, Matusz, Mg, Mik, MiloszD, Nightman, Nux, Patern, Pawelkg, Paweł ze Szczecina, Pawmak, Pimke, Polimerek, Random, Roman 92, Soltys0, Stv, Supergames, Szuflad, Tholrin, Tscsa, Ulrich17, Usher, Viatoro, Zero, Zeroos, conversion script, Łukasz M. Krupiński (tekkdevil), 91 anonimowych edycji

Sesja (informatyka) *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=24350175> *Autorzy:* Adzinok, BeŻet, Cathy Richards, Darktemplar, Delta 51, Derbeth, Dobromila, Filemon, Frees, Jagger, Kuszi, MPK100, Nux, PMG, Pjahr, Qu3a, Rafostry, Roman 92, Roo72, Sir Lothar, Wames, Wojciech mula, Zero, 12 anonimowych edycji

Ciasteczko *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26858350> *Autorzy:* AI, Awersowy, BaQu, Blueshade, C00lfon, Cathy Richards, Derbeth, Entereczek, Faxe, Grotesque, Gurthg Shae, Herr Kriss, Ignasiak, Julio, Kb, Kocio, Konradk, Kpjas, Malarz pl, Marcin Otorowski, MichałG, Micpol, Misza13, Nowak2000, Nux, Omega933, Piastu, Polimerek, Poopy, Qu3a, Rogra, Roo72, SelenavonEichendorf, Shaqspeare, Siedlaro, Szwedzki, Tilia, ToAr, Viatoro, Volrath, WTM, Wiki Kedar, Wpedzich, 52 anonimowych edycji

Kaskadowe arkusze stylów *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26838552> *Autorzy:* A.Kucala, ABach, Ajsmen91, Aojnews, Beau, Birczanin, Chrumps, Clazus, Darekm, Derbeth, E2rd, EMeczKa, Ejdzey, Faxe, Filemon, Gandalf, Gudyś, Illuminatus, Jakozaur, KamStak23, Kicekpiczek, Klapaucjusz, Kocio, Kret 18, Larin, Leopold, Lianmei, Mathel, Matrix0123456789, Michalgarbowski, Michał Sobkowski, Mikolajpodbielski, MwGamera, NeferKaRe, Nimdil, Nmiran, Nux, Oksi, Pawelkg, Piastu, Qion, RobertBlaut, Roo72, Rozek19, SebaTriv, SelenavonEichendorf, Serdelll, Seveneven, St.Mons, Sweater, Szoferka, Szwedzki, T ziel, TOR, Teceha, Timido, ToAr, Tomtal, Verwolf, Vinyanov, Wave, Wimmer, Witia, Zaufany, 78 anonimowych edycji

Model kaskadowy *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26050942> *Autorzy:* Adren, Artizm, Beau, Beno, Harmer, Hunaghe, Kaźmierczyk Krzysztof, Kocio, Mmiszka, Nux, Qu3a, 10 anonimowych edycji

Dziedziczenie (programowanie) *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26521417> *Autorzy:* Awersowy, Beau, CiaPan, DaKo, Dodek, EMeczKa, Jersz, KrzysM99, Mathrick, Robert Borkowski, Royas, Slaweks, Tdc6502, Teodozjan, Xpicto, Zolv, 17 anonimowych edycji

IPv4 *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26646745> *Autorzy:* Adzinok, Arek1979, Beau, Blizinsk, Blueshade, BroviPL, Byczysz, Chepyr, Culmensis, D.biesiada, Darekm, DeeL, Devil33, Eraz, Faxe, Gang65, Grzexs, Huskibuskihuski, Ilario, Kauczuk, Kb, KJ, Kocio, Konradk, Lampak, Lord Ag.Ent, Lupus, Marcin Maziarz, Matusz, OTB, PawełMM, Pducnc, Pewu, Psu, Radekidz, Rafostry, S99, Saibamen, Sobi3ch, Stok, Superborsuk, Szeled, Szumyk, TOR, Taw, Tdc6502, Topory, Uzytkownik, Wpedzich, Wykształcioch, Zero, conversion script, 60 anonimowych edycji

IPv6 *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26755060> *Autorzy:* Adzinok, Aegis Maelstrom, Akira, Anpe2007, Artur Perwenis, Balcer, Bartos34, Bercik, Birczanin, BroviPL, Chrumps, DeeL, Derbeth, Ed88, Emergie, Faxe, Grzegorz-Janoszka, Gtworek, Ireknowak, Jacek Synowiec, Jarekadam, Jersz, Kb, Kbsc, KJ, Kmichalak8, Kocio, Konradk, Kornasz, Kigajowniczek, Lampak, Lmbanach, Lofix, Lzur, MTM, Macar, Marciooo, Marek RS, Mario58, Matekm, McMonster, Mulat, Musp, Nikolaus, OTB, Patrol110, Pio, Psz, Rafal wa, Rafostry, Rol, Saper, Slaweks, Sobi3ch, Steelman, Tiamak, Trivelt, Tscsa, Vaxquis, Vmario, WRIM, Wojkier, Yarek, Youandme, conversion script, Łukasz M. Krupiński (tekkdevil), 81 anonimowych edycji

Klasa (programowanie obiektowe) *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26928668> *Autorzy:* Cardel, Kocio, Kuszi, Liseeeek, Maikking, Oczykota, Robsuper, Sippel2707, Taw, Trzmiel, Turkusowy smok, Zolv, Zyx, 6 anonimowych edycji

Obiekt (programowanie obiektowe) *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26480701> *Autorzy:* Kakaz, Lampak, Mulat, Ptak82, RaSta86, RomanK, Trzmiel, Zolv, Zyx, 1 anonimowych edycji

Klasa abstrakcyjna *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26942110> *Autorzy:* Czupirek, Filu, Googl, Januszkaja, Kb, Matekm, Michalgarbowski, Paweł Pieńkowski, Paweł Ł Zawada, Royas, Running turtle, Sq7obj, Tdc6502, Zolv, 3 anonimowych edycji

Interfejs (programowanie obiektowe) *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=27045054> *Autorzy:* Coldpeer, Exe, Gang65, HooH, Lampak, Mulat, Robsuper, Sq7obj, Superborsuk, 2 anonimowych edycji

Java *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=27039257> *Autorzy:* Airip, Absolwent, AdSR, Adrian.s6, Aligatorok, Amateja, Aradek, Arfrever, Azarien, Beau, Beno, Blueshade, C4In, CD, Ceziex, Clazus, Derbeth, Desire, Dudzislav, Ency, Faxe, Forseti, Gang65, Gdarin, Gevara, Gknor, Gorion, GrJ, Grotesque, Grun, Hashar, Jaross, Javaboy, Jersz, Jpiw, Kocio, Konryd, Kpjas, Kuszi, Kutar, Lampak, Lcamtuf, LukKot, Madadam, Marcin Suwalczan, Marcoos, MarcowyGnom, Markotek, Matbi, Mathel, Matusz, Mciura, MiloszD, Mlepicki, Moarc, Morte, Mpenza, Mrug, Olaf, Omega933, Ominous, Pablo000, Pan Camel, Piotr Skrodzewicz, Piotr Zierhoffer, Piotrop, Pleple2000, Pojdulos, Ponton, Przemyslaw.royzcki, Qu3a, Radoslaww, Raq0, Rhkubiak, Roo72, Rzęsor, S99, Samuray-X, SelenavonEichendorf, Serdelll, Sir Lothar, Sobi3ch, Sproject, Ss181292, Stok, Stotr, Stv, Szwedzki, T ziel, Tashi, Taw, Tdc6502, TheAdam0s, Titter, Tom000, Tomq, Topory, Tscsa, Vindicator, Vshader, Warionm, Webkid, Wojciech mula, WolverinePL, Zduniak, Zdzichu, Zergu, Zero, Zipoking, conversion script, Łeba, 116 anonimowych edycji

Network Address Translation *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=27062953> *Autorzy:* Adzinok, Arctgx, Beno, Bercik, Blueshade, Citromaniak, Crabtree, Delimata, Ejdzey, Filip em, Gang65, Jerry, Kauczuk, Kocio, Konradk, Kwiat3k, Lapiex, Lzur, MKF, Macar, Marcimon, Patrol110, Pbnan, Polimerek, Qu3a, Qwertyy, Roo72, Sanurss, Sobi3ch, Szczepan1990, Tdc6502, Thealx, ToSter, Tomko222, Topory, Uzytkownik, Wanted, Youandme, conversion script, 32 anonimowych edycji

Programowanie strukturalne *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=25383541> *Autorzy:* Derbeth, Fraximus, Grotesque, Hulek, JRS, Kocio, Masur, Matusz, Royas, Taw, Tdc6502, Zyx, 20 anonimowych edycji

Referencja (informatyka) *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=25402206> *Autorzy:* AdSR, Adzinok, Beno, BeŻet, Derbeth, Dijkstra, Dlvooy, Doctor, Entereczek, Gang65, Joee, Kocio, Lampak, Leopold, Macar, Matekm, Mciura, Pkierski, PracownikFizyczny, Qu3a, Rebelia, Robaczek, Rozek19, SelenavonEichendorf, Szoferka, Szołtys, Taw, Tom000, Zolv, Zyx, 19 anonimowych edycji

Trasowanie (telekomunikacja) *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26813135> *Autorzy:* AdSR, CiaPan, Derbeth, Furbolg, Gang65, Joi, Kb, Kocio, Konradk, Lolek01, Loraine, Magalia, Mattiks, McMonster, MesserWoland, OTB, Pawmal, Pitazboras, Qu3a, Rafostry, Sanurss, Siedlaro, Youandme, Zero, 20 anonimowych edycji

Przeciążanie funkcji *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26738406> *Autorzy:* Alef, Beau, Darekm, Derbeth, Dodek, Ethouris, Faxe, Joee, Kbsc, Kuszi, Leopold, Lrds, Malrob, Picus viridis, Royas, Severson, 11 anonimowych edycji

POST (metoda) *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=24632478> *Autorzy:* Adzinok, BoJeR, Dodek, Klisek, Kravietz, Mmh, 4 anonimowych edycji

GET (metoda) *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=19749052> *Autorzy:* Asman, Darekm, Entereczek, Fullofstars, Gigante, Kb, Kravietz, MG, Maciejo93, Michalgarbowski, Nux, Piotrala, Sevela.p, Stv, 6 anonimowych edycji

Przeszukiwanie liniowe *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=25521505> *Autorzy:* CiaPan, Derbeth, Farmer Jan, Kocio, Kuszi, Olaf, Taw, Wojciech mula, conversion script, pw201.warszawa.sdi.tpnet.pl, 6 anonimowych edycji

Wyszukiwanie binarne *Źródło:* <http://pl.wikipedia.org/w/index.php?oldid=26682993> *Autorzy:* Bmiq, Crabtree, Dirdival, Fullofstars, Gökhan, Kuszi, LEW21, Nietaki, Olaf, Pimke, Raq0, Wojciech mula, Ymar, Павле, 12 anonimowych edycji

Źródła, licencje i autorzy grafik

Grafika: Singly linked list.png Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Singly_linked_list.png Licencja: Public Domain Autorzy: User:Dcoetzee

grafika: Singly-linked-list.svg Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Singly-linked-list.svg> Licencja: Public Domain Autorzy: Lasindi

grafika: Doubly-linked-list.svg Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Doubly-linked-list.svg> Licencja: Public Domain Autorzy: Lasindi

grafika: Circularly-linked-list.svg Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Circularly-linked-list.svg> Licencja: Public Domain Autorzy: Lasindi

Plik: Data stack.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Data_stack.svg Licencja: Public Domain Autorzy: User:Boivie

Plik: Sorted binary tree.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Sorted_binary_tree.svg Licencja: Public Domain Autorzy: Miles

Plik: Binary search tree.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Binary_search_tree.svg Licencja: Public Domain Autorzy: User:Booyabazooka, User:Dcoetzee

Plik: Binary search tree search 4.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Binary_search_tree_search_4.svg Licencja: Public Domain Autorzy: Binary_search_tree.svg: Booyabazooka derivative work: movax

Plik: Binary search tree delete 13.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Binary_search_tree_delete_13.svg Licencja: Public Domain Autorzy: Binary_search_tree.svg: Booyabazooka derivative work: movax

Plik: Binary search tree delete 14.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Binary_search_tree_delete_14.svg Licencja: Public Domain Autorzy: Binary_search_tree.svg: Booyabazooka Binary_search_tree_delete_13.svg: movax derivative work: movax

Plik: Binary search tree delete 3.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Binary_search_tree_delete_3.svg Licencja: Public Domain Autorzy: Binary_search_tree_delete_14.svg: movax Binary_search_tree.svg: Booyabazooka Binary_search_tree_delete_13.svg: movax derivative work: movax

Plik: Binary search tree delete 8.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Binary_search_tree_delete_8.svg Licencja: Public Domain Autorzy: Binary_search_tree_delete_3.svg: movax Binary_search_tree_delete_14.svg: movax Binary_search_tree.svg: Booyabazooka Binary_search_tree_delete_13.svg: movax derivative work: movax

Plik: Unbalanced binary tree.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Unbalanced_binary_tree.svg Licencja: Public Domain Autorzy: Me (Intgr)

Plik: Wiki letter w.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Wiki_letter_w.svg Licencja: GNU Free Documentation License Autorzy: Jarkko Piironen

Image: Wikisource-logo.svg Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Wikisource-logo.svg> Licencja: logo Autorzy: Nicholas Moreau

Grafika: AVLtreef.svg Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:AVLtreef.svg> Licencja: Public Domain Autorzy: User:Mikm

Grafika: Unbalanced binary tree.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Unbalanced_binary_tree.svg Licencja: Public Domain Autorzy: Me (Intgr)

Plik: Red-black tree example.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Red-black_tree_example.svg Licencja: Creative Commons Attribution-ShareAlike 3.0 Unported Autorzy: en:User:Cburnett

Plik: Bst rotations.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Bst_rotations.svg Licencja: Public Domain Autorzy: Zyxist

Grafika: Bubble sort animation.gif Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Bubble_sort_animation.gif Licencja: Creative Commons Attribution-ShareAlike 2.5 Autorzy: Original uploader was Nmnogueira at en.wikipedia

Grafika: Insertion sort animation.gif Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Insertion_sort_animation.gif Licencja: Creative Commons Attribution-ShareAlike 2.5 Autorzy: Nuno Nogueira (Nmnogueira)

Plik: Merge sort animation2.gif Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Merge_sort_animation2.gif Licencja: Creative Commons Attribution-ShareAlike 2.5 Autorzy: CobaltBlue

Plik: Mergesort algorithm diagram.png Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Mergesort_algorithm_diagram.png Licencja: Public Domain Autorzy: User:Helix84

Grafika: Sorting quicksort anim.gif Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Sorting_quicksort_anim.gif Licencja: Creative Commons Attribution-ShareAlike 3.0 Unported Autorzy: Wikipedia:en:User:RolandH

Grafika: Selection sort animation.gif Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Selection_sort_animation.gif Licencja: Public Domain Autorzy: en:Marco Polo at en.wikipedia.org

Grafika: Shellsort-edited.png Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Shellsort-edited.png> Licencja: Public Domain Autorzy: crashmatrix (talk)

Plik: Shell sort average number of comparisons (Polish).svg Źródło: [http://pl.wikipedia.org/w/index.php?title=Plik:Shell_sort_average_number_of_comparisons_\(Polish\).svg](http://pl.wikipedia.org/w/index.php?title=Plik:Shell_sort_average_number_of_comparisons_(Polish).svg) Licencja: Creative Commons Attribution-ShareAlike 3.0 Autorzy: Mciura

Grafika: Heap sort example.gif Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Heap_sort_example.gif Licencja: Creative Commons Attribution-ShareAlike 3.0 Unported Autorzy: Nagae

Plik: Grötzsch graph.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Grötzsch_graph.svg Licencja: Public Domain Autorzy: User:Bkell, User:Booyabazooka, User:Deadstar

Plik: Undirected graph.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Undirected_graph.svg Licencja: Public Domain Autorzy: Dcoetzee, Luks, 1 anonimowych edycji

Plik: Directed graph.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Directed_graph.svg Licencja: Public Domain Autorzy: Dcoetzee, Grafite, Luks

Plik: Graph theory tree.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Graph_theory_tree.svg Licencja: Public Domain Autorzy: User:ZeroOne

Plik: Self-loop.png Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Self-loop.png> Licencja: Public Domain Autorzy: Dcoetzee, EugeneZelenko, Frank C. Müller, Grafite, Haui, Jodo, Joey-das-WBF, Roomba, Squizzz

Plik: 6n-graf.svg Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:6n-graf.svg> Licencja: Public Domain Autorzy: User:AzaToth

Plik: Line graph construction 1.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Line_graph_construction_1.svg Licencja: GNU Free Documentation License Autorzy: User:Booyabazooka

Plik: Line graph construction 2.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Line_graph_construction_2.svg Licencja: GNU Free Documentation License Autorzy: User:Booyabazooka

Plik: Line graph construction 3.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Line_graph_construction_3.svg Licencja: GNU Free Documentation License Autorzy: User:Booyabazooka

Plik: Line graph construction 4.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Line_graph_construction_4.svg Licencja: GNU Free Documentation License Autorzy: User:Booyabazooka

Plik: Dual graphs.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Dual_graphs.svg Licencja: Public Domain Autorzy: Original uploader was Booyabazooka at en.wikipedia

Plik: Binary tree.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Binary_tree.svg Licencja: Public Domain Autorzy: User:Dcoetzee

Grafika: Grah1.gif Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Grah1.gif> Licencja: Creative Commons Attribution-ShareAlike 3.0 Autorzy: AlexSITMO, WikipediaMaster, 1 anonimowych edycji

Grafika: Grah2.gif Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Grah2.gif> Licencja: Creative Commons Attribution-ShareAlike 3.0 Autorzy: AlexSITMO, WikipediaMaster, 1 anonimowych edycji

Grafika: Grah3.gif Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Grah3.gif> Licencja: Creative Commons Attribution-ShareAlike 3.0 Autorzy: AlexSITMO, WikipediaMaster, 1 anonimowych edycji

Grafika: Grah4.gif Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Grah4.gif> Licencja: Creative Commons Attribution-ShareAlike 3.0 Autorzy: AlexSITMO, WikipediaMaster, 1 anonimowych edycji

Grafika: Grah5.gif Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Grah5.gif> Licencja: Creative Commons Attribution-ShareAlike 3.0 Autorzy: AlexSITMO, WikipediaMaster, 1 anonimowych edycji

Grafika: Grah6.gif Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Grah6.gif> Licencja: Creative Commons Attribution-ShareAlike 3.0 Autorzy: AlexSITMO, WikipediaMaster, 1 anonimowych edycji

Grafika: Grah7.gif Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Grah7.gif> Licencja: Creative Commons Attribution-ShareAlike 3.0 Autorzy: AlexSITMO, WikipediaMaster, 1 anonimowych edycji

Grafika: Grah8.gif Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Grah8.gif> Licencja: Creative Commons Attribution-ShareAlike 3.0 Autorzy: AlexSITMO, WikipediaMaster, 1 anonimowych edycji

Grafika: Grah9.gif Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Grah9.gif> Licencja: Creative Commons Attribution-Sharealike 3.0 Autorzy: AlexSITMO, WikipediaMaster, 1 anonimowych edycji

Grafika: Grah10.gif Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Grah10.gif> Licencja: Creative Commons Attribution-Sharealike 3.0 Autorzy: AlexSITMO, WikipediaMaster, 1 anonimowych edycji

Grafika: Grah11.gif Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Grah11.gif> Licencja: Creative Commons Attribution-Sharealike 3.0 Autorzy: AlexSITMO, WikipediaMaster, 1 anonimowych edycji

Grafika: Grah12.gif Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Grah12.gif> Licencja: Creative Commons Attribution-Sharealike 3.0 Autorzy: AlexSITMO, WikipediaMaster, 1 anonimowych edycji

Plik:UzycieCiasteczek.png Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:UzycieCiasteczek.png> Licencja: nieznany Autorzy: Original uploader was Marcin Otorowski at pl.wikipedia

Plik:Nuvola mimetypes source css.png Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:Nuvola_mimetypes_source_css.png Licencja: GNU Lesser General Public License Autorzy: Alno, Alphax, WikipediaMaster, Ysangkok

Plik:POL model kaskadowy.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:POL_model_kaskadowy.svg Licencja: Creative Commons Attribution-ShareAlike 3.0 Unported Autorzy: Maciej Jaros (commons: Nux, wiki-pl: Nux)

Plik:References pl.svg Źródło: http://pl.wikipedia.org/w/index.php?title=Plik:References_pl.svg Licencja: Public Domain Autorzy: Zyxist

Plik:cast.svg Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Cast.svg> Licencja: Public Domain Autorzy: Easyas12c

Plik:anycast.svg Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Anycast.svg> Licencja: Public Domain Autorzy: Easyas12c, H Padleckas, Jarekt, 1 anonimowych edycji

Plik:broadcast.svg Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Broadcast.svg> Licencja: Public Domain Autorzy: Easyas12c

Plik:multicast.svg Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Multicast.svg> Licencja: Public Domain Autorzy: Easyas12c, Lupo, 1 anonimowych edycji

Plik:unicast.svg Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Unicast.svg> Licencja: Public Domain Autorzy: Easyas12c, Perhelion

Plik:geocast.svg Źródło: <http://pl.wikipedia.org/w/index.php?title=Plik:Geocast.svg> Licencja: Creative Commons Zero Autorzy: Revolus

Licencja

Creative Commons Attribution-Share Alike 3.0 Unported
<http://creativecommons.org/licenses/by-sa/3.0/>
