SORTOWANIE

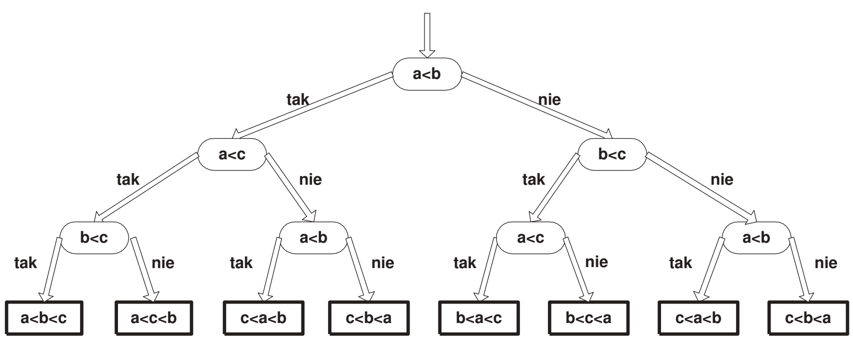
IIUWr. II rok informatyki. Przygotował: Krzysztof Loryś

3 Dolne granice.

Rozważając dolne ograniczenia na złożoność problemu sortowania ograniczymy się, podobnie jak w przypadku problemu jednoczesnego znajdowania minimum i maksimum, do klasy algorytmów, które na elementach ciągu wejściowego wykonują jedynie operacje porównania. Działanie takich algorytmów można w naturalny sposób reprezentować *drzewami decyzyjnymi*. Niezbyt formalnie można je zdefiniować jako skończone drzewa binarne, w których każdy wierzchołek wewnętrzny reprezentuje jakieś porównanie, każdy liść reprezentuje wynik obliczeń a krawędzie odpowiadają obliczeniom wykonywanym przez algorytm pomiędzy kolejnymi porównaniami.

Ponieważ od drzew decyzyjnych wymagamy by były skończone, jedno drzewo nie może reprezentować działania algorytmu dla dowolnych danych. Z reguły przyjmujemy, że algorytm reprezentowany jest przez nieskończoną rodzinę drzew decyzyjnych $\{D_i\}_{i=1}^{\infty}$, gdzie drzewo D_n odpowiada działaniu algorytmu na danych o rozmiarze n.

PRZYKŁAD
Rysunek 7 przedstawia drzewo decyzyjne odpowiadające działaniu algorytmu *SelectSort* na ciągach 3-elementowych.

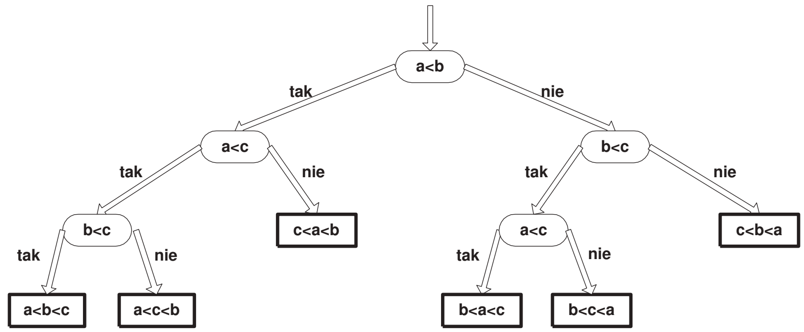


Rysunek 7: *Drzewo D_3 dla algorytmu sortowania przez selekcję.*

□

Jak łatwo zauważyć, algorytm Select Sort wykonuje niektóre porównania niepotrzebnie. Są to porównania "a<b" znajdujące się w odległości 2 od korzenia. Po ich usunięciu otrzymamy inne, mniejsze, drzewo decyzyjne dla sortowania ciągów 3-elementowych (patrz Rysunek 8). Pod względem liczby liści jest ono optymalne.

Fakt 12 *Niech $\mathcal{A}$ będzie algorytmem sortującym, a $\{D_i\}_{i=1}^{\infty}$ – odpowiadającą mu rodziną drzew decyzyjnych. Wówczas drzewo D_n posiada co najmniej $n!$ liści, dla każdego n.*



Rysunek 8: *Optymalne drzewo decyzyjne dla algorytmów sortujących ciągi 3-elementowe.*

UZASADNIENIE: Każda permutacja ciągu wejściowego może być wynikiem, a każdy liść drzewa D_n odpowiada jednemu wynikowi. $\square$

Wprost z Faktu 12 mamy następujące:

Twierdzenie 3 *Niech $\mathcal{A}$ będzie algorytmem sortującym, a $\{D_i\}_{i=1}^{\infty}$ – odpowiadającą mu rodziną drzew decyzyjnych. Wówczas drzewo D_n ma wysokość co najmniej $\Omega(n \log n)$.*

UZASADNIENIE: Drzewo binarne o $n!$ liściach (a takim jest D_n) musi mieć wysokość co najmniej $\log(n!)$. Ze wzoru Stirlinga, $n!$ możemy z dołu oszacować przez $(n/e)^n$, co daje nam:

$$\log n! \geq n\,(\log n - \log e) \geq n \log n - 1.44n$$

$\square$

Ponieważ wysokość drzewa D_n odpowiada liczbie porównań wykonywanych w najgorszym przypadku przez algorytm A dla danych o rozmiarze n, otrzymujemy dolne ograniczenie na złożoność czasową (w najgorszym przypadku) algorytmów sortowania.

Wniosek 1 *Każdy algorytm sortujący za pomocą porównań ciąg n - elementowy wykonuje co najmniej $cn \log n$ porównań dla pewnej stałej $c > 0$.*

3.1 Ograniczenie na średnią złożoność

Działanie algorytmu sortowania, który dane wykorzystuje wyłącznie w porównaniach, zależy jedynie od względnego porządku pomiędzy elementami. W szczególności nie zależy ono od bezwzględnych wartości elementów. Dlatego badając złożoność takich algorytmów możemy ograniczać się do analizy zachowania algorytmu na permutacjach zbioru $\{1, 2, \ldots, n\}$, a średnia złożoność algorytmu na danych rozmiaru n może być policzona jako suma:

$$\sum_{\sigma - permutacja\ zbioru\ \{1,2,\ldots,n\}} P[\sigma] c(\sigma),$$

gdzie $P[\sigma]$ jest prawdopodobieństwem wystąpienia permutacji σ jako danych wejściowych, a $c(\sigma)$ jest równa liczbie porównań wykonywanych na tych danych. W języku drzew decyzyjnych można ją wyrazić jako średnią wysokość drzewa, tj.

$$\sum_{v-\ li\acute{s}\acute{c}\ T} p_v d_v,$$

gdzie p_v oznacza prawdopodobieństwo dojścia do liścia v, a d_v - jego głębokość.

Teraz łatwo widać, że dla wielu rozkładów danych średnia złożoność algorytmu także wynosi $\Omega(n \log n)$. Wystarczy bowiem, by istniały stałe c i d takie, że prawdopodobieństwa dojścia do liści znajdujących się na głębokości nie mniejszej niż $cn \log n$ sumują się do wartości nie mniejszej d. W szczególności otrzymujemy:

Twierdzenie 4 *Jeżeli każda permutacja ciągu n-elementowego jest jednakowo prawdopodobna jako dana wejściowa, to wówczas każde drzewo decyzyjne sortujące ciągi n-elementowe ma średnią głębokość co najmniej* $\log n!$.

UZASADNIENIE: Na głębokości nie większej niż $\log(n/e)^n - 1$ znajduje się mniej niż $n!/2$ liści. Tak więc co najmniej $n!/2$ liści osiągalnych z prawdopodobieństwem $1/n!$ leży na głębokości większej, co implikuje, że średnia wysokość drzewa decyzynego jest większa niż $(1/n!)(n!/2)\log((n/e)^n)$. □

4 Quicksort

O algorytmie *Quicksort* wspomnieliśmy omawiając strategię dziel i zwyciężaj. Podany tam schemat algorytmu można zapisać w następujący sposób:

procedure $quicksort(A[1..n], p, r)$
  if $r - p$ jest małe **then** $insert - sort(A[p..r])$
  else $choosepivot(A, p, r)$
    $q \leftarrow partition(A, p, r)$
    $quicksort(A, p, q)$
    $quicksort(A, q + 1, r)$

Kluczowe znaczenie dla efektywności algorytmu mają wybór *pivota*, tj. elementu dzielącego, dokonywany w procedurze *choosepivot*, oraz implementacja procedury *partition* dokonującej przestawienia elementów tablicy A.

4.1 Implementacja procedury partition

Zakładamy, że w momencie wywołania $partition(A, p, r)$ pivot znajduje się w $A[p]$. Procedura przestawia elementy podtablicy $A[p..r]$ dokonując jej podziału na dwie części: w pierwszej – $A[p..q]$ – znajdą się elementy nie większe od pivota, w drugiej – $A[q + 1, r]$ – elementy nie mniejsze od pivota. Granica tego podziału, wartość q, jest przekazywana jako wynik procedury.