Dolne granice

IIUWr. II rok informatyki. *Opracował: Krzysztof Loryś*

3 Wstęp

Dowodzenie dolnych granic na złożoność problemów jest zadaniem znacznie trudniejszym niż dowodzenie ograniczeń górnych. Musimy bowiem pokazać, że każdy algorytm rozwiązujący dany problem musi wykonać nie mniej niż pewną liczbę operacji.

Dopisać:

- Ograniczone modele obliczeń
- Metody dowodzenia

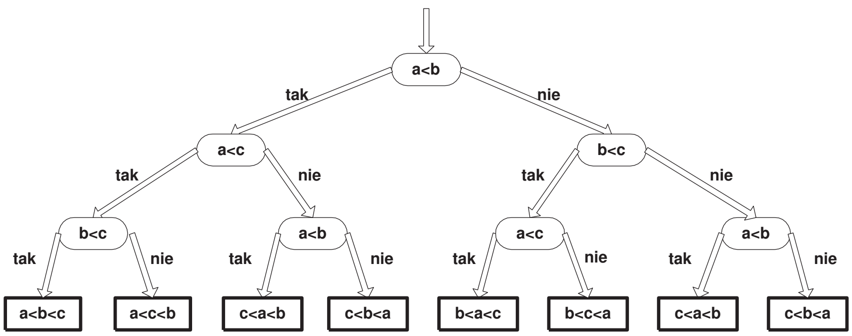
4 Sortowanie - argument teorio-informacyjny w modelu drzew decyzyjnych

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 1 przedstawia drzewo decyzyjne odpowiadające działaniu algorytmu *SelectSort* na ciągach 3-elementowych.

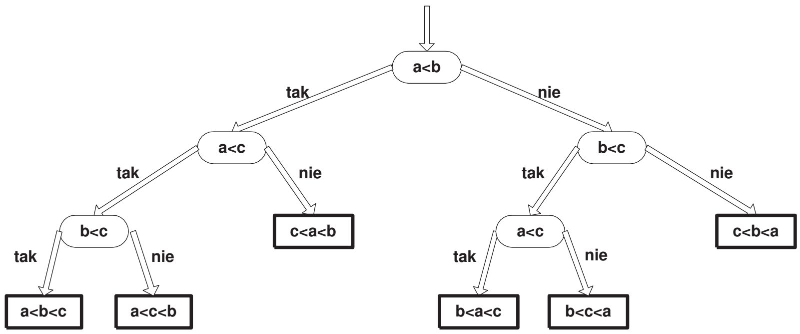


Rysunek 1: *Drzewo D_3 dla algorytmu sortowania przez selekcję.*

□

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 2). Pod względem liczby liści jest ono optymalne.



Rysunek 2: *Optymalne drzewo decyzyjne dla algorytmów sortujących ciągi 3-elementowe.*

Fakt 1 *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.*

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

Wprost z Faktu 1 mamy następujące:

Twierdzenie 1 *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:

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

□

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 $c > 0$.*

4.1 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:

$$\sum_{\sigma \in S_n} P[\sigma] c(\sigma),$$

gdzie S_n jest zbiorem permutacji zbioru {1,2,. . . , n}, $P[\sigma]$ 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.

$$\sum_{v-\text{ liść } T} p_v d_v,$$

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 2 *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 $(1/n!)(n!/2)\log((n/e)^n)$. □

5 Liniowe drzewa decyzyjne

Rozważmy następujący problem:

PROBLEM: (RÓŻNOŚĆ ELEMENTÓW)

Dane: Liczby rzeczywiste $x_1, x_2, \ldots, x_n$.

Zadanie: Sprawdzić, czy $\forall_{1 \leq i \neq j \leq n} x_i \neq x_j$.

Ponieważ zbiór odpowiedzi jest dwuelementowy, metody z poprzedniego rozdziału nie dadzą w tym przypadku sensownego ograniczenia. Metoda, którą tu przedstawimy jest rozwinięciem poprzedniej. Pokażemy, że całą przestrzeń danych można rozbić na wiele rozłącznych podzbiorów i że liści w drzewie decyzyjnym musi być co najmniej tyle, ile jest tych rozłącznych podzbiorów.

Nasza argumentacja będzie prawdziwa także dla

Definicja 1 *Niech $T_n = (V, E)$ będzie liniowym drzewem decyzyjnym dla danych z R^n. Dla każdego wierzchołka $v \in V$ przez $F(v)$ oznaczmy zbiór tych punktów z R^n, które osiągają v.*

Fakt 2 *Dla każdego $v \in V$ zbiór $F(v)$ jest wielościanem wypukłym.*

Pokażemy, że całą dziedzinę danych można rozbić na wiele rozłącznych spójnych zbiorów i że liści w drzewie decyzyjnym musi być więcej niż tych zbiorów. Nasze rozumowanie będzie poprawne także w nieco szerszym modelu obliczeń, w którym możemy nie tylko porównywać dane ze sobą, ale także porównywać kombinacje liniowe tych danych.

Takim obliczeniom odpowiadają tzw. liniowe drzewa decyzyjne. Wierzchołki

Definicja 2 Liniowym drzewem decyzyjnym *dla decyzyjnego problemu P nazywamy drzewo 3-arne, którego wierzchołki wewnętrzne poetykietowane są kombinacjami liniowymi danych, krawędzie wychodzące*

Wygodnie jest patrzeć na ten problem jak na problem określony na przestrzeni R^n: każdy ciąg $x_1, \ldots, x_n$ utożsamiamy z punktem $\bar{x}$ tej przestrzeni, którego współrzędnymi są $x_1, \ldots, x_n$. Problem polega na sprawdzeniu, czy wszystkie współrzędne $\bar{x}$-a mają różne wartości.

Niech T_n będzie liniowym drzewem decyzyjnym dla problemu P ograniczonego do danych rozmiaru n. Każdy wierzchołek v drzewa T_n definiuje pewien podzbiór $S(v)$ punktów R^n, a mianowicie zbiór tych punktów, z którymi można dojść do v z korzenia. Oczywiście dla korzenia zbiór ten zawiera wszystkie punkty z R^n. Kombinacja liniowa $L_v\bar{x}$ etykietująca v rozbija $S(v)$ na trzy rozłączne podzbiory: tych punktów z $S(v)$, dla których wartość L_v jest odpowiednio mniejsza od zera, równa zero i większa od zera. Przenosząc intuicje z R^3 na wyższe wymiary mo

Niech $P_1, \ldots, P_{n!}$ będą różnymi punktami z R^n otrzymanym przez wszystkie permutacje współrzędnych punktu $\langle 1, 2, \ldots, \rangle$.

Fakt 3 *Niech v_i będzie liściem drzewa T_n takim, że $P_i \in S(v_i)$ dla $i = 1, \ldots, n$. Wówczas dla każdego $\forall_{i \neq j} v_i \neq v_j$.*

Dowód. Załóżmy nie wprost, że istnieją różne punkty P_i oraz P_j należące do $S(v)$ dla pewnego liścia v. Niech k będzie najmniejszą liczbą, która występuje na innej pozycji we współrzędnych punktu P_i oraz punktu P_j. Niech to będą odpowiednio pozycje k_i oraz k_j. Bez zmniejszenia ogólności załóżmy, że $k_i < k_j$. Jeśli przez $F(\bar{x})$ oznaczymy różnicę wartości współrzędnych $x_{k_i} - x_{k_j}$ punktu $\bar{x}$, to widzimy, że $F(P_i) < 0$ a $F(P_j) > 0$. Ponieważ F jest funkcją ciągłą, więc na odcinku łączącym P_i i P_j istnieje punkt Y, w którym funkcja F przyjmuje wartość 0, a więc taki punkt, który ma tę samą wartość różnych współrzędnych. Ale ze względu na wypukłość zbioru $S(v)$ punkt Y należy do tego zbioru. Stąd algorytm odpowie błędnie, gdy jak dane podamy współrzędne punktu Y. $\square$

6 Redukcje

Pokażemy, że w modelu RAM problem znajdowania otoczki wypukłej nie jest prostszy niż problem sortowania liczb.

Rozważmy następującą procedurę:

Krok 1. Dane dla problemu sortowania $S = \langle x_1, x_2, \ldots, x_n \rangle$ przekształcamy w $S' = \langle (x_1, x_1^2), \ldots, (x_n, x_n^2) \rangle$. Krok 2. Na danych S' Uruchamiamy algorytm A znajdujący otoczke wypukłą. Krok 3. Na podstawie wyniku $\langle (x_{i_1}, x_{i_1}^2), \ldots, (x_{i_n}, x_{i_n}^2) \rangle$

Dokończyć!!!!

7 Gra z adwersarzem

Algorytm *MinMax1*, który poznaliśmy w trakcie omawiania metody Dziel i Zwyciężaj, wyznacza górną granicę na złożoność problemu jednoczesnego znajdowania minimum i maksimum. Teraz pokażemy, że ta granica jest także granicą dolną, a więc dokładnie wyznaczymy złożoność problemu.

Twierdzenie 3 *Każdy algorytm rozwiązujący powyższy problem (i używający elementów zbioru S jedynie w porównaniach) wykonuje co najmniej $\lceil \frac{3}{2}n - 2 \rceil$ porównań.*

Dowód: Rozważmy następującą grę między algorytmem a złośliwym adwersarzem:

- Sytuacja początkowa: adwersarz twierdzi, że zna trudny dla algorytmu zbiór S, tj. taki, dla którego wskazanie przez algorytm minimum i maksimum będzie wymagało wykonania co najmniej $\lceil \frac{3}{2}n - 2 \rceil$ porównań. Algorytm nie zna S; wie tylko, że liczy on n elementów.
- Cel gry
  - algorytmu: wskazanie indeksów elementów minimalnego i maksymalnego w zbiorze S przy użyciu mniej niż $\lceil \frac{3}{2}n - 2 \rceil$ porównań;
  - adwersarza: zmuszenie algorytmu do zadania co najmniej $\lceil \frac{3}{2}n - 2 \rceil$ porównań.
- Ruchy
  - algorytmu: pytanie o porównanie dwóch elementów ze zbioru S;
  - adwersarza: odpowiedź na to pytanie.
- Koniec gry następuje, gdy algorytm wskaże minimum i maksimum w S. Wówczas adwersarz ujawnia zbiór S.

Tezę twierdzenia udowodnimy, jeśli pokażemy, że adwersarz zawsze, niezależnie od algorytmu, posiada strategię wygrywającą.

Strategia dla adwersarza:

- W trakcie gry adwersarz dzieli S na 4 rozłączne zbiory:

  $A = \{i \mid a_i$ jeszcze nie był porównywany $\}$,
  $B = \{i \mid a_i$ wygrał już jakieś porównanie i nie przegrał żadnego $\}$,
  $C = \{i \mid a_i$ przegrał juz jakieś porównanie i nie wygrał żadnego$\}$,
  $D = \{i \mid a_i$ wygrał już jakieś porównanie i jakieś już przegrał $\}$.

  Początkowo oczywiście $|A| = n$ oraz $|B| = |C| = |D| = 0$.

- Adwersarz rozpoczyna grę z dowolnymi kandydatami na wartości elementów a_i. W trakcie gry będzie, w razie konieczności, modyfikował te wartości, tak by spełniony był warunek

$$(*) \qquad \forall_{a\in A}\forall_{b\in B}\forall_{c\in C}\forall_{d\in D}\ \ b > d > c \text{ oraz } b > a > c.$$

  Zauważ, że wystarczy w tym celu zwiększać wartości elementów o indeksach z B i zmniejszać wartości elementów o indeksach z C. Takie zmiany są bezpieczne dla adwersarza, ponieważ pozostawiają prawdziwymi jego odpowiedzi na dotychczasowe pytania.

Fakt 4 *Powyższa strategia adwesarza jest zawsze wygrywająca.*

Dowód faktu: W trakcie gry wszystkie elementy przechodzą ze zbioru A do B lub C, a dopiero stąd do zbioru D. Ponadto dla danych spełnających (*):

- jedno porównanie może usunąć co najwyżej dwa elementy ze zbioru A,
- dodanie jednego elementu do zbioru D wymaga jednego porównania,
- porównania, w których bierze udział element z A, nie zwiększają mocy zbioru D.

Dopóki A jest niepusty lub któryś ze zbiorów B lub C zawiera więcej niż jeden element, algorytm nie może udzielić poprawnej odpowiedzi. Na opróżnienie zbioru A algorytm potrzebuje co najmniej $\lceil n/2 \rceil$ porównań. Następnych $n-2$ porównań potrzebnych jest na przesłanie wszystkich, poza dwoma, elementów do zbioru D. □ (faktu i twierdzenia)

7.1 Dodatek

Poniższa tabela pokazuje w jaki sposób zmieniają się liczności zbiorów po wykonaniu różnych typów porównań (przy założeniu warunku (*)). Typ XY oznacza, iż porównywany jest element zbioru X z elementem zbioru Y. Mała litera x oznacza element zbioru X.

Typ porównania	$\|A\|$	$\|B\|$	$\|C\|$	$\|D\|$	warunek
AA	-2	+1	+1	bz	
AB	-1	bz	+1	bz	$a < b$
AC	-1	+1	bz	bz	$a > c$
AD	-1	+1	bz	bz	$a > d$
	-1	bz	+1	bz	$a < d$
BB	bz	-1	bz	+1	
BC	bz	bz	bz	bz	$b > c$
BD	bz	bz	bz	bz	$b > d$
CC	bz	bz	-1	+1	
CD	bz	bz	bz	bz	$c < d$
DD	bz	bz	bz	bz	