Drzewa Obszarów

Bartosz Rybicki*

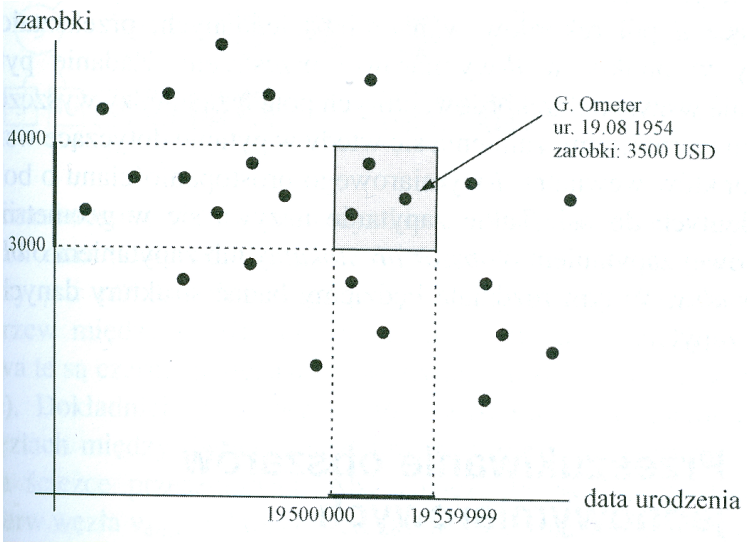
Wrocław, 9 listopada 2010

Streszczenie

Drzewa obszarów to struktura wykorzystywana do zadawania zapytań o punkty w obszarze ortogonalnym. O punktach można myśleć jak o rekordach w bazie danych. Praca zawiera opis struktury oraz algorytmu przy pomocy którego można tą strukturę zbudować. Opisany jest również sposób zadania zapytań o zbiór punktów. Wszystkie opisane algorytmy są analizowane zarówno jeśli chodzi o złożoność czasową jak i o pamięciową.

1 Opis problemu i jego motywacja na przykładzie zadawania zapytań bazie danych

Wyobraźmy sobie, że chcemy uzyskać informacje o pracownikach naszej firmy, którzy zarabiają miedzy 3000 a 4000 tysiące złotych oraz urodzili się miedzy 1950 a 1955 rokiem. Każdy pracownik może zostać opisany jako para informacji (pensja, rok urodzenia). Można o tym myśleć jak o punkcie w przestrzeni R^2. Przykładowy wynik zapytania:



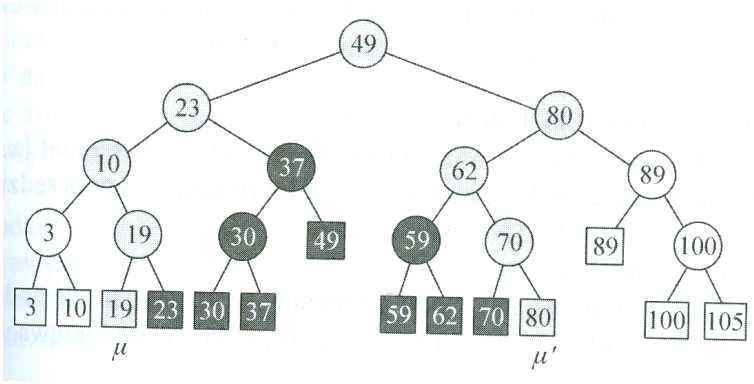
Źródło: Geometria Obliczeniowa. Algorytmy i zastosowania, M. de Berg

Uwaga 1.1 *Zauważmy, że na każdy rekord zawierający d pól możemy patrzeć jak na punkt w przestrzeni R^d. Zatem od teraz nie będziemy myśleć o rekordach w bazie, ale o punktach w przestrzeni R^d.*

*rybicki.bartek(at)gmail.com numer albumu: 209096

2 Przeszukiwanie obszarów jednowymiarowych

Załóżmy, że mamy zbiór punktów na osi OX i interesują nas te należące do przedziału $[x, x']$. Dla zadanego zbioru punktów możemy zbudować zrównoważone drzewo wyszukiwań binarnych i przy jego pomocy odpowiadać na zapytania o punkty z danego odcinka. Załóżmy, że mamy następujący zbiór punktów: $\{3, 10, 19, 23, 30, 37, 49, 59, 62, 70, 80, 100, 105\}$ i chcemy odpowiedzieć na pytanie o punkty leżące w przedziale $[18, 77]$.



Źródło: Geometria Obliczeniowa. Algorytmy i zastosowania, M. de Berg

2.1 Algorytm budowy drzewa

Alg(P)

1: **if** |P| == 1 **then**
2: stwórz liść pamiętający jeden punkt
3: **else**
4: v_{value} := mediana(P_x);
5: podziel P na zbiory: $P_{x \leq mediana}$, $P_{mediana < x}$;
6: v_{lewy} := Alg($P_{\leq x}$);
7: v_{prawy} := Alg($P_{x<}$);
8: **end if**
9: return v;

Twierdzenie 1 *Czas budowy struktury to $O(nlogn)$ a wymagana pamięć to $O(n)$.*

Dowód 1 *Czas: $T(n) = 2 * T(\frac{n}{2}) + O(n) = O(nlogn)$ - szukamy mediany oraz dwukrotnie wywołujemy się dla zbioru o połowę mniejszego.*
Pamięć: $M(n) = 2 * M(\frac{n}{2}) + O(1) = O(n)$ - korzeń zajmuje pamięć rzędu O(1). Inny sposób myślenia o koszcie pamięciowym tej struktury to liczenie kosztu "od dołu" najniższy poziom wymaga n komórek pamięci, kolejny $\frac{n}{2}$ itd. Zatem $M(n) = n + \frac{n}{2} + \frac{n}{4} + ... + \frac{n}{2^{logn}} = O(n)$

Uwaga 2.1 *Zauważmy, że dla posortowanego zbioru czas budowy struktury to $O(n)$. Dzieje się tak ponieważ szukanie mediany odbywa się w czasie stałym. Można też budować drzewo od dołu i nie przejmować się problemem mediany.*

Definicja 1 *Wierzchołek dzielący dla przedziału [x, x'] to taki w którym rozchodzą się ścieżki od korzenia do x oraz x'.*

2.2 Algorytm zapytania o przedział

Alg(T, [x,x'])

1: schodź w dół drzewa do czasu aż znajdziesz wierzchołek dzielący lub dojdziesz do liścia;
2: **if** v_{dziel} jest liściem **then**
3: sprawdź czy v_{dziel} należy do przedziału i ewentualnie dodaj go do wyniku;
4: **else**
5: zejdź ścieżką od x i dodaj do wyniku punkty wiszące w liściach drzew na prawo od ścieżki sprawdź czy punkt pamiętany w liściu (na końcu ścieżki do x) należy do przedziału, jeśli tak to dodaj go do wyniku;
6: zejdź ścieżką od x' i dodaj do wyniku punkty wiszące w liściach drzew na lewo od ścieżki sprawdź czy punkt pamiętany w liściu (na końcu ścieżki do x) należy do przedziału, jeśli tak to dodaj go do wyniku;
7: **end if**
8: return wynik;

Twierdzenie 2 *Czas wykonania zapytania to O(log n + k), gdzie k to rozmiar wyniku, tj. liczba punktów składających się na wynik.*

Dowód 2 *Zejście w drzewie do x oraz x' to O(log n), bo drzewo jest zrównoważone. Koszt przejścia odpowiednich poddrzew jest liniowy od liczby ich liści i wynosi O(k). Zatem całkowity czas potrzebny na wykonanie zapytania oraz obliczenie wyniku to O(log n + k)*

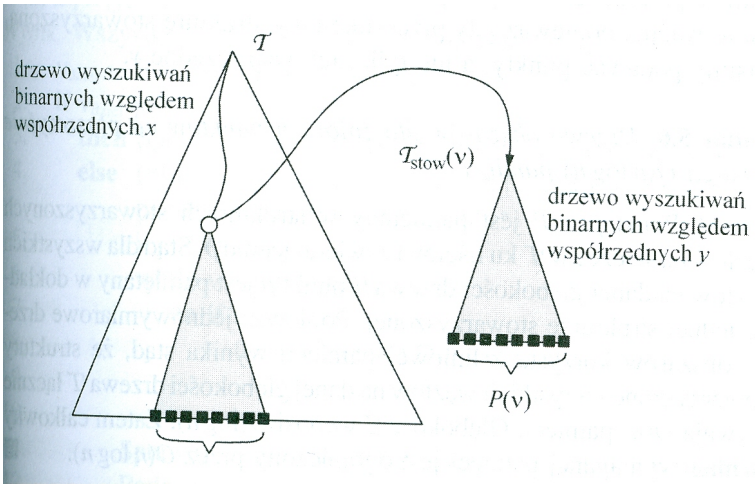
3 Drzewa obszarów (2D)

Uwaga 3.1 *Tymczasowo załóżmy, że żadne dwa punkty nie mają takich samych współrzędnych x oraz y. Potem pokażemy jak pozbyć się tego (nierealnego) założenia.*

Definicja 2 *Podzbiór punktów pamiętanych w liściach poddrzewa zakorzenionego w węźle v nazywamy podzbiorem kanonicznym v oraz oznaczamy jako P(v).*

3.1 Opis struktury

- Główne drzewo T jest zrównoważonym drzem wyszukiwań binarnych (BST) zbudowanym względem współrzędnych x punktów ze zbioru P
- Dla każdego węzła wewnętrznego lub liścia v w drzewie T, podzbiór kanoniczny P(v) jest pamiętany w zrównoważonym drzewie wyszukiwań binarnych $T_{stow}(v)$ względem współrzędnej y punktów. Węzeł v pamięta wskaźnik do drzewa $T_{stow}(v)$, które nazywamy strukturą stowarzyszoną z v.



Źródło: Geometria Obliczeniowa. Algorytmy i zastosowania, M. de Berg

3.2 Algorytm budowy struktury

- P - zbiór punktów
- P_x - zbiór P posortowany po współrzędnej x
- P_y - zbiór P posortowany po współrzędnej y

Alg(P_x, P_y)
1: **if** (|P| == 1) **then**
2: stwórz liść pamiętający jeden punkt i zbuduj T_{stow} dla tego punktu;
3: **else**
4: v_{value} := mediana(P_x);
5: podziel P_x oraz P_y na zbiory: $P_x^{x \leq v_{value}}$, $P_x^{v_{value} < x}$ $P_y^{x \leq v_{value}}$ oraz $P_y^{v_{value} < x}$;
6: v_{lewy} := Alg($P_x^{x \leq v_{value}}$, $P_y^{x \leq v_{value}}$);
7: v_{prawy} := Alg($P_x^{v_{value} < x}$, $P_y^{v_{value} < x}$);
8: v_{stow} := zbuduj $T_{stow}(P_y)$;
9: **end if**
10: return v;

Twierdzenie 3 *Zarówno czas jak i pamięć potrzebna do budowy struktury to $O(nlogn)$.*

Dowód 3 *Czas: $T(n) = 2 * T(\frac{n}{2}) + O(n) = O(nlogn)$ - szukamy mediany oraz dwukrotnie wywołujemy się dla zbioru o połowę mniejszego. Czas potrzebny na zbudowanie struktury stowarzyszonej jet liniowy, bo zbiór P_y jest odpowiednio posortowany.*
Pamięć: Każdy punkt p jest pamiętany w strukturze stowarzyszonej węzłów na ścieżce w T ku liściowi zawierającemu p. Ścieżka ma długość $O(logn)$ a wszystkich punktów jest $O(n)$ zatem cała struktura zajmuje $O(nlogn)$ komórek pamięci.

3.3 Algorytm zapytania o zbiór punktów w obszarze $[x, x'] \times [y, y']$

Alg(T, $[x, x'] \times [y, y']$)
1: v_{dziel} := WierzchołekDzielący(T, [x, x']);
2: **if** v_{dziel} jest liściem **then**
3: sprawdź czy ma znajdować się w wyniku jeśli tak to dodaj go do rozwiązania;
4: **else**
5: idź ścieżką do x i wykonuj zapytania jednowymiarowe dla poddrzew na prawo od ścieżki, na końcu ścieżki sprawdź czy punkt w liściu ma być podany;
6: idź ścieżką do x' i wykonuj zapytania jednowymiarowe dla poddrzew na lewo od ścieżki, na końcu ścieżki sprawdź czy punkt w liściu ma być podany;
7: **end if**
8: return wynik;

Twierdzenie 4 *Czas zapytania o punkty z obszaru $[x, x'] \times [y, y']$ to $O(log^2 n + k)$, gdzie k to rozmiar wyniku.*

Dowód 4 *Każde zapytanie o obszar jednowymiarowy zajmuje czas rzędu $O(logn + k_v)$ takich zapytań będzie $O(logn)$, natomiast liczba zwróconych punktów to $\sum k_v = k$. Zatem czas całego zapytania zamyka się w $O(log^2 n + k)$.*

4 Uogólnienie drzew obszarów na wiele wymiarów

Algorytm budowy struktury jest bardzo podobny do algorytmu dla drzew dwuwymiarowych. Podstawowa różnica w algorytmie to brak sortowania po współrzędnych. Struktury stowarzyszone dla drzew pierwszego poziomu to (d-1)-wymiarowe drzewa obszarów. Analogicznie dla drzew kolejnych poziomów.

Twierdzenie 5 *Niech P będzie zbiorem n punktów w d-wymiarowej przestrzeni, gdzie $d \geq 2$. Drzewo obszarów dla P używa $O(nlog^{d-1}n)$ pamięci i można je zbudować w czasie $O(nlog^{d-1}n)$. Punkty z P leżące w prostokątnym obszarze zapytania, można podać w czasie $O(log^d n + k)$, gdzie k jest rozmiarem wyniku.*

Dowód 5 *Tworzenie d-wymiarowego drzewa obszarów polega na zbudowaniu zrównoważonego BST, co wymaga czasu $O(nlogn)$, i konstruowaniu struktur stowarzyszonych. W węzłach na dowolnej głębokości drzewa pierwszego poziomu każdy punkt jest pamiętany w dokładnie jednej strukturze stowarzyszonej. Czas potrzebny na utworzenie struktur stowarzyszonych na pewnej głębokości można ograniczyć przez $O(T_{d-1}(n))$, jest tak ponieważ $T_{d-1}(n)$ jest conajmniej liniowe. Zatem całkowity czas potrzebny na utworzenie struktury można opisać następującym wzorem*

$$T_d(n) = O(nlogn) + O(logn) * O(T_{d-1}(n)) = O(nlog^{d-1}n)$$

jest tak ponieważ $T_2(n) = O(nlogn)$ - pokazaliśmy to przy okazji analizy drzew dwuwymiarowych. Analizę zużywanej pamięci można przeprowadzić w analogiczny sposób z tą różnicą, że BST wymaga O(n) pamięci, zatem pamięć potrzebna do utworzenia struktury może zostać opisana następującym wzorem:

$$M_d(n) = O(n) + O(logn) * O(M_{d-1}(n)) = O(nlog^{d-1}n)$$

jest tak ponieważ $M_2(n) = O(nlogn)$ - pokazaliśmy to przy okazji analizy drzew dwuwymiarowych. Oznaczmy przez $Q_d(n)$ czas potrzebny na wyznaczenie wszystkich drzew jednowymiarowych, które w swoich liściach przechowują wynik naszego zapytania. Wiemy, że $Q_2(n) = O(log^2 n)$, zatem prawdziwy jest wzór

$$Q_d(n) = O(logn) + O(logn)Q_{d-1}(n) = O(log^d n)$$

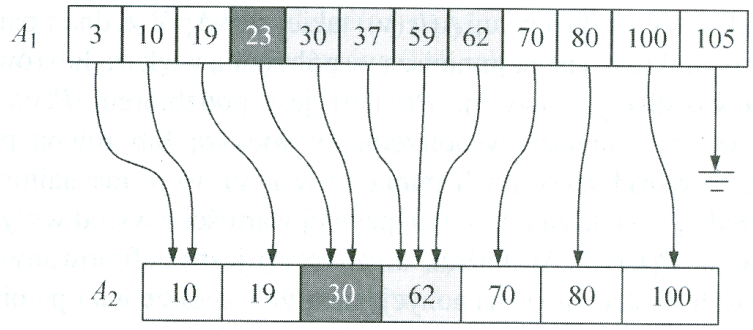
Zatem czas potrzebny na wykonanie zapytania oraz zwrócenie wyniku to $O(log^d n + k)$, gdzie k jest rozmiarem wyniku.

5 Kaskadowanie cząstkowe

Omawialiśmy już strukturę która operowała na elementach z R^2 i odpowiadała na zapytania o obszar $[x, x'] \times [y, y']$ w czasie $O(log^2 n + k)$, teraz pokażemy jak zredukować ten czas do $O(logn + k)$.

5.1 Przykład

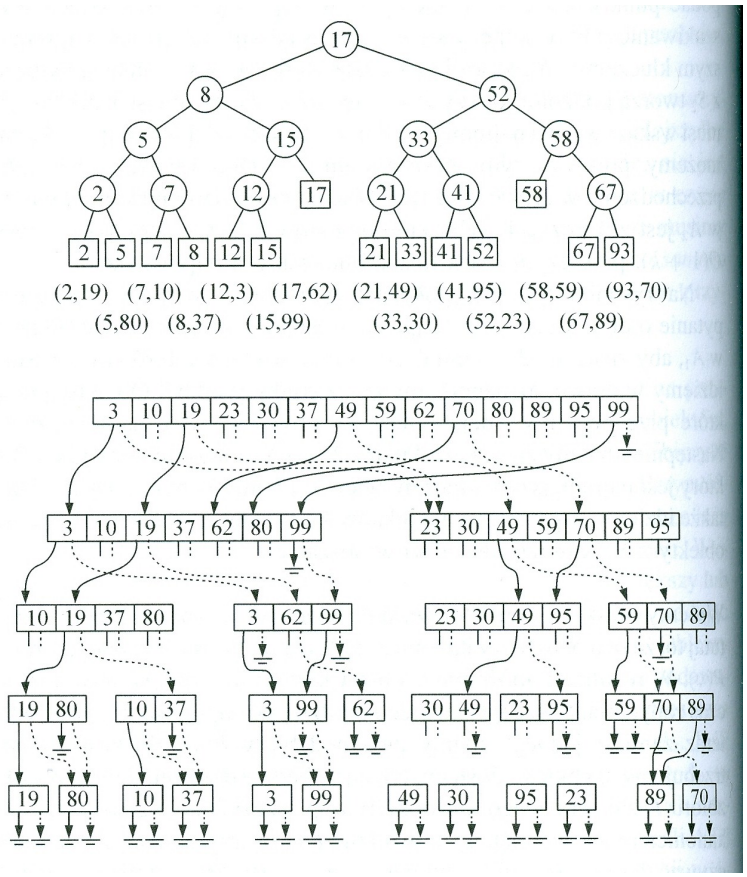
Przedstawmy pomysł kaskadowania cząstkowego na prostym przykładzie. Niech $S_2 \subseteq S_1$ oraz istnieją tablicę A_1, A_2, takie że A_1 przechowuje posortowane elementy z S_1, analogicznie dla A_2 oraz S_2. Jeśli $A_1[i]$ pamięta wartość y_i to przechowujemy wskaźnik z $A_1[i]$ do pozycji w A_2 z najmniejszym kluczem większym lub równym niż y_i. Jeśli nie ma takich kluczy to wskaźnik na wartość NULL. Teraz gdy chcemy wypisać wszystkie elementy z S_1 oraz S_2, które należą do przedziału $[y_1, y_2]$ wystarczy nam jedno przeszukiwanie binarne w tablicy A_1 oraz przejście obu tablic co zajmuje czas $O(logn + k)$, gdzie k to liczba punktów z S_1 oraz S_2, które należą do przedziału $[y, y']$. Przykład zapytania dla przedziału $[20, 65]$



Źródło: Geometria Obliczeniowa. Algorytmy i zastosowania, M. de Berg

5.2 Przeniesienie pomysłu na drzewa obszarów

Struktura stowarzyszona dla konkretnego wierzchołka v w drzewie obszarów przestaje być zbalansowanym BST. Warto zamienić je na tablicę $A(v)$ punktów uporządkowanych względem współrzędnej y. Każdy element tablicy A(v) pamięta dwa wskaźniki do odpowiednich pozycji w $A(v_{lewy})$ oraz $A(v_{prawy})$. Przypuśćmy, że $A(v)[i]$ pamięta punkt $p = (p_x, p_y)$, wtedy wskaźnik do $A(v_{lewy})$ pokazuje na pozycję najmniejszego elementu $p' = (p'_x, p'_y)$ dla którego zachodzi $p_y \leq p'_y$. Analogicznie dla drugiego wskaźnika i tablicy $A(v_{prawy})$.



Źródło: Geometria Obliczeniowa. Algorytmy i zastosowania, M. de Berg

W jaki sposób odpowiadamy na pytanie o obszar $[x, x'] \times [y, y']$? W tablicy stowarzyszonej wierzchołka dzielącego robimy wyszukiwanie binarne w celu znalezienia najmniejszego elementu większego lub równego y. Dalej procedura jest analogiczna jak wcześniej. Schodzimy ścieżką do wierzchołka x i wypisujemy poddrzewa, które leżą na prawo od ścieżki. W jaki sposób wypisujemy poddrzewa? Znamy wskaźnik do najmniejszego (względem współrzędnej y) elementu z tablicy

stowarzyszonej (możemy ustalać odpowiednie pozycje w tablicach stowarzyszonych przechodząc drzewo), który spełnia nasze kryteria, zatem przechodzimy odpowiednią tablicę aż dojdziemy do ostatniego elementu, który jest $\leq y^{'}$.

Twierdzenie 6 *Czas zapytania o obszar* $[x, x'] \times [y, y']$ *wynosi* $O(\log n + k)$

Dowód 6 *Jakie operacje wykonujemy? Wyszukiwanie binarne w tablicy stowarzyszonej z wierzchołkiem dzielącym, przejście od wierzchołka dzielącego do liści przechowujących x oraz x', wypisane odpowiednich elementów z tablic stowarzyszonych z wybranymi wierzchołkami. Czas wszystkich operacji zamyka się w* $O(\log n + k)$.

Uwaga 5.1 *Ta poprawka pozwala nam przyspieszyć wyszukiwanie punktów w przestrzeni* R^d *z czasu* $O(\log^d n + k)$ *do* $O(\log^{d-1} n + k)$ *dla* $d \geq 2$. *W jaki sposób? Struktury jednowymiarowe zamieniamy na tablice stowarzyszone, a zapytanie dla struktury dwuwymiarowej wykonujemy tak jak to zostało opisane wyżej.*

6 Ogólny zbiór punktów

6.1 Pozbycie się niewygodnego założenia

Poczyniliśmy (nierealne) założenie o tym że żadne dwa punkty nie mają takiej samej współrzędnej x lub y. Teraz pokażemy jak pozbyć się tego niewygodnego założenia. Każdy punkt $p = (p_x, p_y)$ będziemy reprezentować jako $\hat{p} = ((p_x|p_y), (p_y|p_x))$. Porządek na parach $(a|b)$ zadajemy w następujący sposób

$$(a|b) < (\acute{a}|\acute{b}) \Leftrightarrow a < \acute{a} \vee (a = \acute{a} \wedge b < \acute{b})$$

Uwaga 6.1 *Jeśli punkty p oraz q są różne to* $\hat{p}$ *i* $\hat{q}$ *są różne na wszystkich współrzędnych.*

6.2 Sposób zadawania zapytania

Załóżmy, że chcemy zapytać o obszar $R = [x, x'] \times [y, y']$. W tym celu definiujemy obszar $\hat{R} = [(x| - \infty), (x'| + \infty)] \times [(y| - \infty), (y'| + \infty)]$ o który będziemy pytać algorytm. Pokażmy, że $p \in R \Leftrightarrow \hat{p} \in \hat{R}$

$$p = (p_x, p_y) \Rightarrow \hat{p} = ((p_x|p_y), (p_y|p_x))$$

$$p \in R \Leftrightarrow x \leq p_x \leq x' \wedge y \leq p_y \leq y'$$

$$\Updownarrow$$

$$(x, -\infty) \leq (p_x, p_y) \leq (x', +\infty) \wedge (y, -\infty) \leq (p_y, p_x) \leq (y', +\infty)$$

$$\Updownarrow$$

$$\hat{p} \in \hat{R}$$

Literatura

[1] Geometria Obliczeniowa. Algorytmy i zastosowania, M. de Berg