ORT

Adam Szady

4 grudnia 2016

Streszczenie

Drzewa obszarów[1] pozwalają zadawać zapytania o obszary ortogonalne w przestrzeni n-wymiarowej. Niniejszy dokument prezentuje ich strukturę, sposób tworzenia i przeszukiwania. Porównuje z alternatywnymi sposobami realizacji tego zagadnienia. Dostarcza też podstawową dokumentację programisty i użytkownika dla zaimplementowanej aplikacji.

1 Cel pracy

Projekt jest przedsięwzięciem realizowanym na potrzeby przedmiotu *Geometria obliczeniowa*. Celem projektu było zaimplementowanie odpowiedniej struktury danych dla punktów na płaszczyźnie - drzewa obszarów, która pozwala szybko odpowiadać na zapytania o wszystkie punkty q leżące w obszarze zadanym przez x_1, x_2, y_1, y_2. Ściślej rzecz ujmując, dla danego zbioru $P \subset \mathbb{R}^2$ wyznaczyć zbiór Q taki, że $q \in Q \iff q \in P \ \wedge \ x_1 \leqslant q_x \leqslant x_2 \ \wedge \ y_1 \leqslant q_y \leqslant y_2$.

Rozwiązanie powinno służyć jako narzędzie dydaktyczne do objaśnienia tworzenia struktury i realizacji zapytań.

2 Teoria

2.1 Problem

Praca uogólnia zadany problem. To ze względu na to, że nie sprawia większej trudności rozwiązanie powyższego zagadnienia dla dowolnej n-wymiarowej przestrzeni, ani uogólnienie go na dowolną funkcję $\boxplus$ (opisaną poniżej).

Ogólna definicja

Dane są:

- przestrzenie X, V, takie że $X \subseteq V$;
- n porządków liniowych $\prec_0$, $\prec_1$ ..., $\prec_{n-1}$ na X;
- funkcja $\boxplus : V^2 \to V$ (łączna, przemienna[2]);

[1] (ang.) range tree

[2] Przemienność nie jest wymagana w przypadku jednowymiarowym.

- (multi)zbiór $S \subset X$;
- $A,\ B \in X$.

Zadanie: znaleźć $p_1 \boxplus p_2 \boxplus p_3 \boxplus ... \boxplus p_k$, gdzie p_1, p_2, p_3, ..., p_k to wszystkie punkty p spełniające zależność $p \in S \ \wedge \ A \prec p \prec B$. Przy czym $x \prec y \iff x \prec_0 y \ \wedge \ x \prec_1 y \ \wedge \ ... \ \wedge \ x \prec_{n-1} y$.

Możliwe przypadki szczególne

Łatwo zauważyć, że problem odnalezienia punktów dla obszaru ortogonalnego w przestrzeni dwuwymiarowej jest szczególnym przypadkiem tego problemu, dla:

- $V = 2^{R^2}$, X - zbiór singletonów R^2;
- dwóch naturalnych porządków względem kolejnych współrzędnych;
- $\boxplus(a, b) = a \cup b$;
- S - zbioru singletonów rozważanych punktów;
- zaś $A = \{(x_1, y_1)\}$, $B = \{(x_2, y_2)\}$.

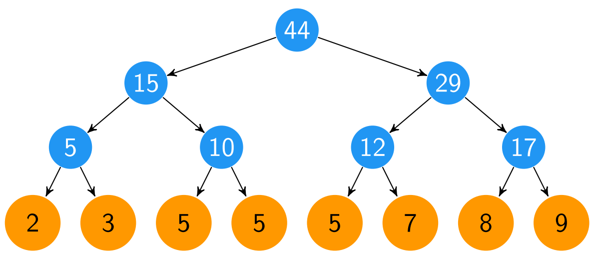
Przy użyciu tego mechanizmu można również rozwiązać chociażby problem znalezienia punktu o największej wadze. Wystarczy, że, mając zdefiniowaną dodatkowy porządek $\prec_w$, można przyjąć: $\boxplus(a, b) = \{a \text{ jeśli } b \prec_w a,\ b \text{ wpp. } \}$.

2.2 Rozwiązanie - drzewa obszarów

2.2.1 Przypadek jednowymiarowy - ORT_0

Wyobrażanie sobie n-wymiarowego rozwiązania najlepiej rozpocząć od rozwiązania jednowymiarowego.

Rysunek 1 przedstawia drzewo przedziałowe dla zestawu danych [2, 3, 5, 5, 5, 7, 8, 9]. Elementy są sortowane według zadanego porządku i umieszczane w liściach. Następnie drzewo tworzone jest w górę, by utworzyć pełne drzewo binarne. Do węzłów niebędących liśćmi przypisywana jest wartość funkcji $\boxplus$ dla dwóch synów. W ten sposób w dowolnym węźle tego drzewa mamy intuicyjnie rozumianą wartość funkcji $\boxplus$ dla całego poddrzewa - w tym przypadku sumy elementów.



Rysunek 1: Jednowymiarowe drzewo przedziałowe z operacją dodawania

Lemat 1 *Przy założeniach, że zbiór V ma skończoną moc*[3]*, zaś operacja $\boxplus(a, b)$ ma złożoność*

[3] Ergo jego elementy mają złożoność pamięciową $\mathcal{O}(1)$

czasową $\mathcal{O}(|a| + |b|)$*, to dla* $n = |S|$ *w drzewie przedziałowym:*

- *Czas konstrukcji tego drzewa to* $\mathcal{O}(n)^4$,
- *zajętość pamięci –* $\mathcal{O}(n)$,
- *zapytanie o przedział odbywa się w czasie* $\mathcal{O}(\log n)$.

Nazwijmy tę strukturę ORT_0.

Co jednak, jeśli chcielibyśmy ograniczyć rozmiar elementów w węzłach przez rozmiar poddrzewa? Taka sytuacja następuje dla definicji $\boxplus(a, b) = a \cup b$.

Lemat 2 *Zakładając, że wielkość elementu drzewa, którego poddrzewo ma m liści jest* $\mathcal{O}(mf(m))$*, zaś operacja* $\boxplus(a, b)$ *ma złożoność czasową* $\mathcal{O}(|a|f(|a|))$*, to dla* $n = |S|$ *w dziele przedziałowym:*

- *Czas konstrukcji tego drzewa to* $\mathcal{O}(nf(n) \log n)$,
- *zajętość pamięci –* $\mathcal{O}(nf(n) \log n)$.

Zatem dla przypadku $\boxplus(a, b) = a \cup b$ mamy $f(m) = 1$, zatem złożoność czasowa i pamięciowa konstrukcji takiego drzewa to $\mathcal{O}(n \log n)$. Oczywiście, w praktycznych zastosowaniach konstrukcja jednowymiarowego drzewa przedziałowego, obsługującego teoriomnogościową sumę jest straszliwym marnowaniem zasobów.

2.2.2 Przypadek dwuwymiarowy - ORT_1

Mając narzędzie do rozwiązania problemu w przestrzeni jednowymiarowej, rozważmy dodanie drugiego wymiaru (porządku).

Strukturę obsługującą dwa wymiary nazwiemy ORT_1, zaś jej elementami będą drzewa jednowymiarowe - ORT_0. Konstrukcję tego drzewa rozpoczynamy jak w przypadku ORT_0, zatem od posortowania elementów i umieszczenia ich w liściach.

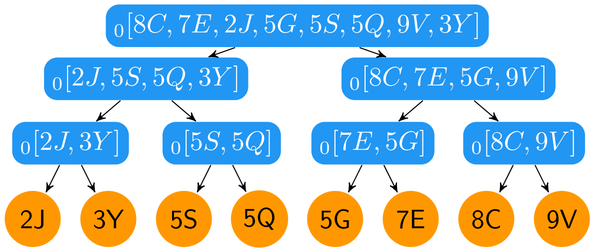
Drzewo ORT_1 ma jednak w specjalny sposób zdefiniowaną funkcję $\boxplus$, która polega na utworzenia drzewa ORT_0 z wszystkich elementów danego poddrzewa. Funkcja ta będzie wykorzystywana jedynie podczas konstrukcji. Te drzewa niższego rzędu będą korzystały z innego porządku niż względem którego tworzone było ORT_1.

Korzystając z lematu 2 i zakładając, że czas konstrukcji i złożoność pamięciowa drzewa ORT_0 jest $\mathcal{O}(n \log n)$, otrzymujemy czas konstrukcji i złożoność pamięciową drzewa ORT_1 klasy $\mathcal{O}(n \log^2 n)$.

2.2.3 Przypadek d-wymiarowy - ORT_{d-1}

Powyższe rozumowanie można łatwo uogólnić do dowolnej wymiarowości drzewa. Drzewo ORT_{d-1} dla $d > 1$ będzie miało elementy w postaci drzew ORT_{d-2}. Stosując rozumowanie indukcyjne łatwo na podstawie lematu 2 wykazać, że złożoność czasowa i pamięciowa konstrukcji drzewa ORD_{d-1} wynosi $\mathcal{O}(n \log^d n)$ jeśli drzewo ORT_0 było klasy $\mathcal{O}(n \log n)$.

[4] Zakładając, że elementy te otrzymujemy w postaci posortowanej



Rysunek 2: Dwuwymiarowe drzewo przedziałowe. $_0[x_1, ..., x_n]$ oznacza drzewo ORT_0 zbudowane z elementami $x_1, ..., x_n$. Zdefiniowane są dwa porządki na elementach - jeden względem cyfr, drugi względem liter.

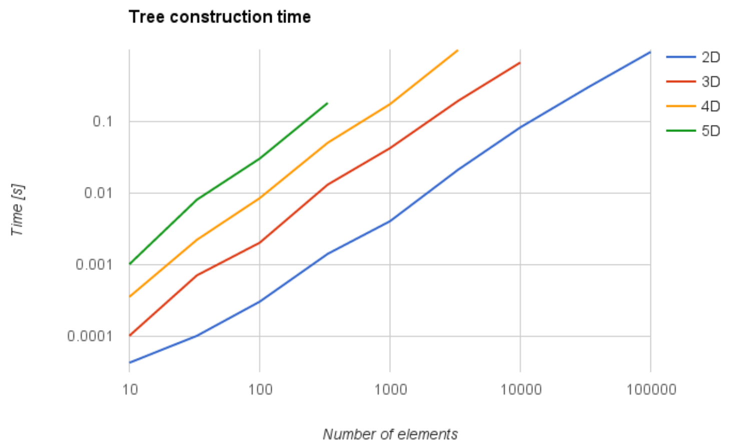
Jeśli jednak założyć, że $\boxplus$ jest $\mathcal{O}(1)$, to otrzymujemy złożoności dla ORT_0 – $\mathcal{O}(n)$, a w konsekwencji dla ORT_{d-1} – $\mathcal{O}(n \log^{d-1} n)$.

Pozostaje jedynie kwestia rozstrzygnięcia, w jaki sposób odpowiadać na zapytania. W drzewie najwyższego rzędu odnajdujemy wszystkie przedziały bazowe odpowiadające żądanemu zakresowi. Ich liczba w każdym wymiarze jest ograniczona przez $\mathcal{O}(\log n)$, zaś koszt ich odnalezienia to także $\mathcal{O}(\log n)$. Następnie, rekurencyjnie dla każdego z tych przedziałów bazowych, odpytujemy odpowiadające im drzewa niższego rzędu. Ostatecznie, ta rekurencyjna procedura zakończy się po $\mathcal{O}(\log^d n)$ krokach. W tym momencie już wystarczy tylko obliczyć funkcję $\boxplus$ na $\mathcal{O}(\log^d n)$ elementach. Ostateczna złożoność zapytania zależy zatem od jej charakteru. Dla zapytań o punkt o największej wadze otrzymamy $\mathcal{O}(\log^d n)$, zaś dla zapytań o wszystkie punkty w obszarze mamy $\mathcal{O}(\log^d n + k)$, gdzie k jest rozmiarem wyniku.

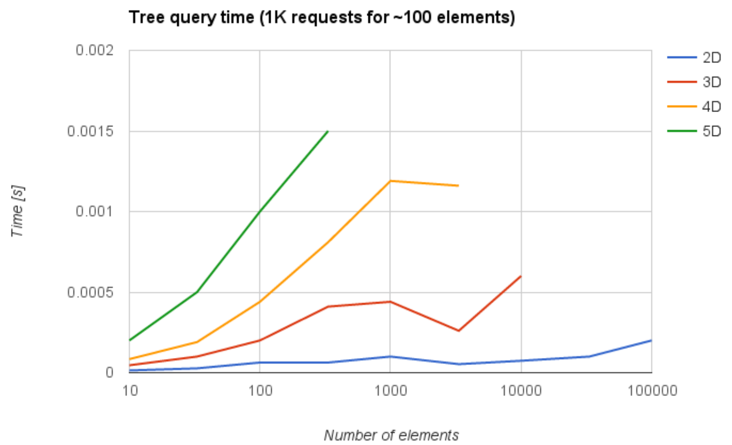
3 Aplikacja

Rozwiązanie zostało utworzone w języku C++11. Dostarcza ono implementację ORT oraz przykładowe użycie.

3.1 Testy wydajnościowe



Rysunek 3: Czasy konstrukcji ORT



Rysunek 4: Czasy zapytań ORT

Rysunki 3 i 4 przedstawiają odpowiednio czasy tworzenia drzewa ORT i wykonywania zapytań - w zależności od liczby elementów oraz wymiarów. Założono rozkład równomierny we wszystkich wymiarach, w zakresie $[0; 1)$. Zapytania o 100 elementów ograniczane były przez hipersześcian o boku $\sqrt[d]{100/N}$. Wykresy zdają się potwierdzać wyznaczone wcześniej teoretyczne złożoności obliczeniowe.

3.2 Dokumentacja programisty

Rozwiązanie dostarcza programiście klasę ORT.

Poniższy listing pokazuje przykładowy sposób konstrukcji drzewa dla przestrzeni 3D, w której punktom przypisano dodatkowy atrybut, względem którego będziemy chcieli odnajdywać maksimum.

```
template<size_t Dim>
using NDPoint = std::array<double, Dim>;

template<size_t Dim>
struct DimCmpSingle {
  template<size_t IthDim>
  static bool precedes(
      const NDPoint<Dim>& p1,
      const NDPoint<Dim>& p2) {
    return p1[IthDim] < p2[IthDim];
  }
};

template<size_t Dim>
struct MaxFn {
  NDPoint<Dim+1> operator()(
      const NDPoint<Dim+1>& p1,
      const NDPoint<Dim+1>& p2) const {
    return p1[Dim] > p2[Dim] ? p1 : p2;
  }
};

auto createMax3DTree(const std::vector<NDPoint<4>>& data) {
  return ORT<3, NDPoint<4>, DimCmpSingle<3>> tree(data, MaxFn<3>{});
}
```

W powyższym kodzie:

- **NDPoint<Dim>** reprezentuje *Dim*-wymiarowy punkt
- **DimCmpSingle<Dim>::precedes<IthDim>** – *IthDim*-ty porządek w *Dim*-wymiarowej przestrzeni.
- **MaxFn<Dim>** – funkcję $\boxplus$
- **ORT<Dim, V, DimCmp>(initial, mix)** – konstruuje ORT o liczbie wymiarów *Dim*, zbiorze $V = V$, porządkach *DimCmp* na danych *initial*, wraz z funktorem *mix* jako $\boxplus$.

3.3 Dokumentacja użytkownika

Dostarczona aplikacja demonstruje działanie struktury ORT w sposób dwojaki. Po pierwsze – demonstruje wyszukiwanie punktów w obszarze ortogonalnym na płaszczyźnie (por. rysunek 5). Po drugie – testuje wydajność struktur względem liczby wymiarów i liczności zbioru punktów (opisanych w 3.1 Testy wydajnościowe).

W pierwszej kolejności należy pobrać bibliotekę *gogui* niezbędną do wizualizacji działania algorytmów. Przykładowo, może być to:

```
$ git clone https://bitbucket.org/mizmuda/gogui.git
$ cd gogui
```

Następnie należy pobrać właściwą aplikację:

```
$ git clone https://github.com/aszady/ort.git
$ cd ort
```

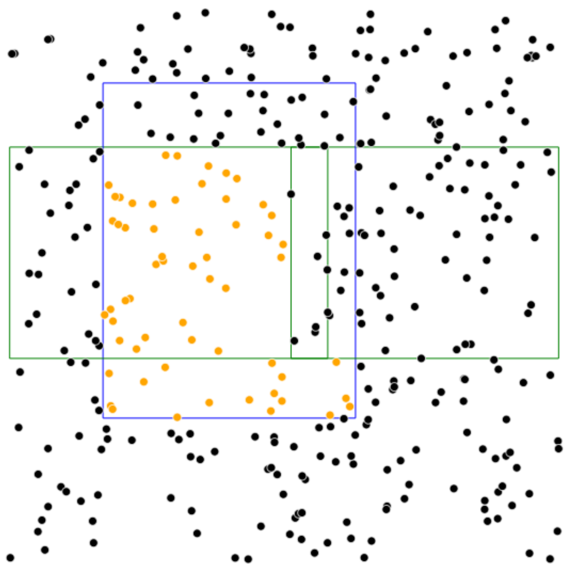
Oraz skompilować rozwiązanie przy użyciu *cmake*:

```
$ cmake .
$ make
```

Następnie aplikację można uruchomić poprzez polecenie:

```
$ ./ort
```

Po uruchomieniu aplikacja wypisze na standardowe wyjście wynik testów wydajnościowych, zaś do pliku `demoj.json` zapisany zostanie wynik wyszukiwania punktów na płaszczyźnie. Plik ten można przekazać do wizualizatora dostarczonego przez bibliotekę *gogui* – na przykład `gogui_vizualization/index.html` uruchomionego w przeglądarce z dostępem do odczytu plików lokalnych.



Rysunek 5: Fragment wizualizacji poszukiwania punktów na płaszczyźnie

4 Materiały

Literatura

1. De Berg, Mark, et al. „Computational geometry.” Computational geometry. Springer Berlin Heidelberg, 2000. 5.3-5.4.

Odnośniki

- Prezentacja: `goo.gl/uoXWdn`
- Kod: `github.com/aszady/ort`
- Ten dokument: `https://github.com/aszady/ort/tree/master/doc/main.pdf`