Planar point location using persistent search trees

Paweł Kacprzak

Instytut Informatyki Uniwersytetu Wrocławskiego

19 grudnia 2011

Krótki plan

1. Problem Planar Point Location
2. Redukcja problemu
3. Struktury danych z historią na przykładzie BST
4. Uogólnienia
5. Krótko o Fully Persistent
6. Podsumowanie

Definicja problemu

Klasyczny problem w geometrii obliczeniowej

Dane:

- n odcinków (czasami też półprostych), które tworzą podział płaszczyzny na wielokąty
- ciąg punktów, które dostajemy **on-line**

Czego szukamy:

- dla każdego punktu, chcemy dowiedzieć się, który wielokąt go zawiera

Interesuje nas:

- czas preprocesingu (najmniej ważny)
- czas zapytania
- pamięć

Wyniki

Dolna granice

- W modelu, w którym pozwalamy tylko na porównywanie, dolna granica na czas zapytania wynosi $\Omega(logn)$
- Jeśli podział płaszczyzny ma być przechowywany, dolna granica na pamięć to $\Omega(n)$

Algorytm, który się wymyśli po chwili zastanowienia

Złożoność czasowa

Dwa wyszukiwania binarne - $O(logn)$

Dlaczego to nas nie zadowala?

Struktura danych może mieć **kwadratowy rozmiar**

Algorytm, który się wymyśli po chwili zastanowienia

Złożoność czasowa

Dwa wyszukiwania binarne - $O(logn)$

Dlaczego to nas nie zadowala?

Struktura danych może mieć **kwadratowy rozmiar**

Algorytm, który się wymyśli po chwili zastanowienia

Złożoność czasowa

Dwa wyszukiwania binarne - $O(logn)$

Dlaczego to nas nie zadowala?

Struktura danych może mieć **kwadratowy rozmiar**

Algorytm, który się wymyśli po chwili zastanowienia

Złożoność czasowa

Dwa wyszukiwania binarne - $O(logn)$

Dlaczego to nas nie zadowala?

Struktura danych może mieć **kwadratowy rozmiar**

Efektywniejsze rozwiązania

- Znane są metody, które osiągają czas zapytania $O(logn)$ i działają w pamięci $O(n)$ ale są skomplikowane i mają zastosowanie tylko w tym jednym problemie.
- Istnieje prosta metoda rozwiązania problemu przy użyciu struktur danych z historią.
- Trzeba się bliżej przyjrzeć instacji problemu.

Efektywniejsze rozwiązania

- Znane są metody, które osiągają czas zapytania $O(logn)$ i działają w pamięci $O(n)$ ale są skomplikowane i mają zastosowanie tylko w tym jednym problemie.
- Istnieje prosta metoda rozwiązania problemu przy użyciu struktur danych z historią.
- Trzeba się bliżej przyjrzeć instacji problemu.

Efektywniejsze rozwiązania

- Znane są metody, które osiągają czas zapytania $O(logn)$ i działają w pamięci $O(n)$ ale są skomplikowane i mają zastosowanie tylko w tym jednym problemie.
- Istnieje prosta metoda rozwiązania problemu przy użyciu struktur danych z historią.
- Trzeba się bliżej przyjrzeć instacji problemu.

Efektywniejsze rozwiązania

- Znane są metody, które osiągają czas zapytania $O(logn)$ i działają w pamięci $O(n)$ ale są skomplikowane i mają zastosowanie tylko w tym jednym problemie.
- Istnieje prosta metoda rozwiązania problemu przy użyciu struktur danych z historią.
- Trzeba się bliżej przyjrzeć instacji problemu.

Redukcja Planar Point Location do ...

Zredukowaliśmy problem Planar Point Location do problemu BST, które umożliwia wyszukiwanie w starszych wersja struktury.

Redukcja Planar Point Location do ...

Zredukowaliśmy problem Planar Point Location do problemu BST, które umożliwia wyszukiwanie w starszych wersja struktury.

Dwa warianty

- Partial Persistent
- Fully Persistent

Dwa warianty

- **Partial Persistent**
- Fully Persistent

Dwa warianty

- **Partial Persistent**
- Fully Persistent

Chcemy wykonywać następujące operacje

- access(x, t)
- insert(x)
- delete(x)

Chcemy wykonywać następujące operacje

- access(x, t)
- insert(x)
- delete(x)

Chcemy wykonywać następujące operacje

- access(x, t)
- insert(x)
- delete(x)

Chcemy wykonywać następujące operacje

- access(x, t)
- insert(x)
- delete(x)

Zalety RB-tree

- podczas wstawiania lub usuwania elementów, tylko stała liczba zmian wskaźników (rotacje)
- co to daje?

Dla uproszczenia będę rysował zwykłe BST, ale myślimy o tym jak o RB-tree

Zalety RB-tree

- podczas wstawiania lub usuwania elementów, tylko stała liczba zmian wskaźników (rotacje)
- co to daje?

Dla uproszczenia będę rysował zwykłe BST, ale myślimy o tym jak o RB-tree

Zalety RB-tree

- podczas wstawiania lub usuwania elementów, tylko stała liczba zmian wskaźników (rotacje)
- co to daje?

Dla uproszczenia będę rysował zwykłe BST, ale myślimy o tym jak o RB-tree

Zalety RB-tree

- podczas wstawiania lub usuwania elementów, tylko stała liczba zmian wskaźników (rotacje)
- co to daje?

Dla uproszczenia będę rysował zwykłe BST, ale myślimy o tym jak o RB-tree

Przyjrzymy się następującym metodom

- Path Copying
- Fat Node
- Node Copying

Przyjrzymy się następującym metodom

- Path Copying
- Fat Node
- Node Copying

Przyjrzymy się następującym metodom

- Path Copying
- Fat Node
- Node Copying

Przyjrzymy się następującym metodom

- Path Copying
- Fat Node
- Node Copying

Path Copying

Idea

Skopiować węzły, w których coś zmieniliśmy (pointer). Kopiujemy też każdy węzeł, który zawiera wskaźnik do właśnie skopiowanego węzła. W drzewie BST węzły zawierają wskaźniki tylko do dzieci, więc kopiowanie prowadzi do utworzenia kopii ścieżki od węzła do korzenia, stąd nazwa metody.

Główna wada

Pamięć **nie** jest liniowa.

Path Copying

Idea

Skopiować węzły, w których coś zmieniliśmy (pointer). Kopiujemy też każdy węzeł, który zawiera wskaźnik do właśnie skopiowanego węzła. W drzewie BST węzły zawierają wskaźniki tylko do dzieci, więc kopiowanie prowadzi do utworzenia kopii ścieżki od węzła do korzenia, stąd nazwa metody.

Główna wada

Pamięć **nie** jest liniowa.

Path Copying

Idea

Skopiować węzły, w których coś zmieniliśmy (pointer). Kopiujemy też każdy węzeł, który zawiera wskaźnik do właśnie skopiowanego węzła. W drzewie BST węzły zawierają wskaźniki tylko do dzieci, więc kopiowanie prowadzi do utworzenia kopii ścieżki od węzła do korzenia, stąd nazwa metody.

Główna wada

Pamięć **nie** jest liniowa.

Co można poprawić?

Idea

- W starszych wersjach struktury nie musimy pamiętać informacji o zbalansowaniu drzewa, a tymbardziej ich uaktualniać.
- W RB-tree, $O(1)$ zmian wskaźników (rotacje), $O(logn)$ zmian kolorów.
- Można wykorzystać każde inne zbalansowane BST, które zmienia tylko $O(1)$ wskaźników podczas operacji typu update

Co można poprawić?

Idea

- W starszych wersjach struktury nie musimy pamiętać informacji o zbalansowaniu drzewa, a tymbardziej ich uaktualniać.
- W RB-tree, $O(1)$ zmian wskaźników (rotacje), $O(logn)$ zmian kolorów.
- Można wykorzystać każde inne zbalansowane BST, które zmienia tylko $O(1)$ wskaźników podczas operacji typu update

Co można poprawić?

Idea

- W starszych wersjach struktury nie musimy pamiętać informacji o zbalansowaniu drzewa, a tymbardziej ich uaktualniać.
- W RB-tree, $O(1)$ zmian wskaźników (rotacje), $O(logn)$ zmian kolorów.
- Można wykorzystać każde inne zbalansowane BST, które zmienia tylko $O(1)$ wskaźników podczas operacji typu update

Fat node

- Pozwalamy węzłom zawierać dowolnie dużo informacji (wskaźników)
- Jak to działa? Zmiane wskaźnika symulujemy przez utworzenie nowego z przypisaną wersją.
- Jak działają operacje? (kolory są nadpisywane na aktualne)
- Pamięć i czas. Insert i delete w RB-tree zmieniają tylko $O(1)$ wskaźników, więc tutaj tworzonych jest O(1) nowych wskaźników. Wyszukiwanie $O((logn) * (logm))$

Fat node

- Pozwalamy węzłom zawierać dowolnie dużo informacji (wskaźników)
- Jak to działa? Zmiane wskaźnika symulujemy przez utworzenie nowego z przypisaną wersją.
- Jak działają operacje? (kolory są nadpisywane na aktualne)
- Pamięć i czas. Insert i delete w RB-tree zmieniają tylko $O(1)$ wskaźników, więc tutaj tworzonych jest $O(1)$ nowych wskaźników. Wyszukiwanie $O((logn) * (logm))$

Fat node

- Pozwalamy węzłom zawierać dowolnie dużo informacji (wskaźników)
- Jak to działa? Zmiane wskaźnika symulujemy przez utworzenie nowego z przypisaną wersją.
- Jak działają operacje? (kolory są nadpisywane na aktualne)
- Pamięć i czas. Insert i delete w RB-tree zmieniają tylko $O(1)$ wskaźników, więc tutaj tworzonych jest O(1) nowych wskaźników. Wyszukiwanie $O((logn) * (logm))$

Fat node

- Pozwalamy węzłom zawierać dowolnie dużo informacji (wskaźników)
- Jak to działa? Zmiane wskaźnika symulujemy przez utworzenie nowego z przypisaną wersją.
- Jak działają operacje? (kolory są nadpisywane na aktualne)
- Pamięć i czas. Insert i delete w RB-tree zmieniają tylko $O(1)$ wskaźników, więc tutaj tworzonych jest O(1) nowych wskaźników. Wyszukiwanie $O((logn) * (logm))$

Node Copying

Idea - Modyfikacja "Fat Node"

- Chcemy pozbyć się narzutu czasowego.
- Pozwalamy, aby węzeł mógł mieć dodatkowo **k** wskaźników poza oryginalnymi dwoma.
- Co jeśli węzeł się "przepełni"?
- Tworzymy jego kopie ustawiając oryginalne wskaźniki na najnowsze wersje z kopiowanego węzła (mamy k wolnych miejsc). Trzeba dodać jeszcze wskaźnik na nową kopie u ojca kopiowanego węzła.

Node Copying

Idea - Modyfikacja "Fat Node"

- Chcemy pozbyć się narzutu czasowego.
- Pozwalamy, aby węzeł mógł mieć dodatkowo **k** wskaźników poza oryginalnymi dwoma.
- Co jeśli węzeł się "przepełni"?
- Tworzymy jego kopie ustawiając oryginalne wskaźniki na najnowsze wersje z kopiowanego węzła (mamy k wolnych miejsc). Trzeba dodać jeszcze wskaźnik na nową kopie u ojca kopiowanego węzła.

Node Copying

Idea - Modyfikacja "Fat Node"

- Chcemy pozbyć się narzutu czasowego.
- Pozwalamy, aby węzeł mógł mieć dodatkowo **k** wskaźników poza oryginalnymi dwoma.
- Co jeśli węzeł się "przepełni"?
- Tworzymy jego kopie ustawiając oryginalne wskaźniki na najnowsze wersje z kopiowanego węzła (mamy k wolnych miejsc). Trzeba dodać jeszcze wskaźnik na nową kopie u ojca kopiowanego węzła.

Node Copying

Idea - Modyfikacja "Fat Node"

- Chcemy pozbyć się narzutu czasowego.
- Pozwalamy, aby węzeł mógł mieć dodatkowo **k** wskaźników poza oryginalnymi dwoma.
- Co jeśli węzeł się "przepełni"?
- Tworzymy jego kopie ustawiając oryginalne wskaźniki na najnowsze wersje z kopiowanego węzła (mamy k wolnych miejsc). Trzeba dodać jeszcze wskaźnik na nową kopie u ojca kopiowanego węzła.

Jak wyglądają operacje?

Analogicznie

- access - $O(logm)$
- update - $O(logn)$

Ale co z pamięcią?

Jak wyglądają operacje?

Analogicznie

- access - $O(logm)$
- update - $O(logn)$

Ale co z pamięcią?

Jak wyglądają operacje?

Analogicznie

- access - $O(logm)$
- update - $O(logn)$

Ale co z pamięcią?

Jak wyglądają operacje?

Analogicznie

- access - $O(logm)$
- update - $O(logn)$

Ale co z pamięcią?

Metoda potencjału

- Dzielimy węzły w drzewie na **żywe** i **martwe**
- Definiujemy potencjał i-tej wersji drzewa jako: Φ_i = liczba żywych węzłów $-1/k*$ liczba pustych miejsc na wskaźniki w żywych węzłach
- Rozpatrujemy i-tą operację. Jej koszt zamortyzowany: $c'_i = c_i + \Phi_i - \Phi_{i-1}$ (faktyczny koszt powiększony o różnice potencjałów)
- Rozważmy każdą operację osobno

Metoda potencjału

- Dzielimy węzły w drzewie na **żywe** i **martwe**
- Definiujemy potencjał i-tej wersji drzewa jako: Φ_i = liczba żywych węzłów $-1/k*$ liczba pustych miejsc na wskaźniki w żywych węzłach
- Rozpatrujemy i-tą operację. Jej koszt zamortyzowany: $c'_i = c_i + \Phi_i - \Phi_{i-1}$ (faktyczny koszt powiększony o różnice potencjałów)
- Rozważmy każdą operację osobno

Metoda potencjału

- Dzielimy węzły w drzewie na **żywe** i **martwe**
- Definiujemy potencjał i-tej wersji drzewa jako: Φ_i = liczba żywych węzłów $-1/k*$ liczba pustych miejsc na wskaźniki w żywych węzłach
- Rozpatrujemy i-tą operację. Jej koszt zamortyzowany: $c_i' = c_i + \Phi_i - \Phi_{i-1}$ (faktyczny koszt powiększony o różnice potencjałów)
- Rozważmy każdą operację osobno

Metoda potencjału

- Dzielimy węzły w drzewie na **żywe** i **martwe**
- Definiujemy potencjał i-tej wersji drzewa jako: Φ_i = liczba żywych węzłów $-1/k*$ liczba pustych miejsc na wskaźniki w żywych węzłach
- Rozpatrujemy i-tą operację. Jej koszt zamortyzowany: $c'_i = c_i + \Phi_i - \Phi_{i-1}$ (faktyczny koszt powiększony o różnice potencjałów)
- Rozważmy każdą operację osobno

Metoda potencjału

- Dzielimy węzły w drzewie na **żywe** i **martwe**
- Definiujemy potencjał i-tej wersji drzewa jako: Φ_i = liczba żywych węzłów $-1/k*$ liczba pustych miejsc na wskaźniki w żywych węzłach
- Rozpatrujemy i-tą operację. Jej koszt zamortyzowany: $c'_i = c_i + \Phi_i - \Phi_{i-1}$ (faktyczny koszt powiększony o różnice potencjałów)
- Rozważmy każdą operację osobno

Skopiowanie węzła

$$c_i' = 1 + ((zw + 1 - 1) - 1/k * (pw + k)) - (zw - 1/k * pw)$$
$$= 1 + zw - 1 - 1/k * pw + zw + 1/k * pw = \mathbf{0}$$

Dodanie nowego wskaźnika (tylko do żywych węzłów)

$$\begin{aligned} c_i' &= 0 + ((zw - 1/k * (pw - 1)) - (zw - (1/k * pw)) \\ &= 0 + zw - 1/k * pw + 1/k - zw + 1/k * pw = \mathbf{1/k} \end{aligned}$$

Utworzenie nowego węzła podczas operacji insert

$$c_i' = 1 + (zw + 1 - 1/k * (pw + k)) - (zw - (1/k * pw))$$
$$= 1 + zw + 1 - 1/k * (pw + k)) - (zw - (1/k * pw)) = \mathbf{1}$$

Jakie to daje oszacowanie?

Ponieważ update powoduje $O(1)$ zmian wskaźników, zamortyzowany koszt pamięciowy wynosi $O(1)$

Uwagi implementacyjne

- Najłatwiej wybrać $k = 1$ i też zadziała
- Najlepsze k zależy od architektury sprzętowej

Uwagi implementacyjne

- Najłatwiej wybrać $k = 1$ i też zadziała
- Najlepsze k zależy od architektury sprzętowej

Uwagi implementacyjne

- Najłatwiej wybrać $k = 1$ i też zadziała
- Najlepsze k zależy od architektury sprzętowej

Czy to działa tylko dla BST?

- Podane metody można dosyć łatwo uogólnić na prawie każdą wskaźnikową strukturę danych.
- Co musi taka struktura spełniać i dlaczego, aby modyfikacja była efektywna?
- Liczba wskaźników wskazujących na węzeł musi być ograniczona przez stałą.

Czy to działa tylko dla BST?

- Podane metody można dosyć łatwo uogólnić na prawie każdą wskaźnikową strukturę danych.
- Co musi taka struktura spełniać i dlaczego, aby modyfikacja była efektywna?
- Liczba wskaźników wskazujących na węzeł musi być ograniczona przez stałą.

Czy to działa tylko dla BST?

- Podane metody można dosyć łatwo uogólnić na prawie każdą wskaźnikową strukturę danych.
- Co musi taka struktura spełniać i dlaczego, aby modyfikacja była efektywna?
- Liczba wskaźników wskazujących na węzeł musi być ograniczona przez stałą.

Czy to działa tylko dla BST?

- Podane metody można dosyć łatwo uogólnić na prawie każdą wskaźnikową strukturę danych.
- Co musi taka struktura spełniać i dlaczego, aby modyfikacja była efektywna?
- Liczba wskaźników wskazujących na węzeł musi być ograniczona przez stałą.

Fully Persistent

- Można też rozważać wariant struktury z dopuszczeniem uaktualnień względem wersji z przeszłości
- Jaki pojawia się problem?
- Nie mamy porządku liniowego na wersjach struktury. Nie można łatwo w węźle zdecydować, którym wskaźnikiem podążać
- Jak temu zaradzić?

Fully Persistent

- Można też rozważać wariant struktury z dopuszczeniem uaktualnień względem wersji z przeszłości
- Jaki pojawia się problem?
- Nie mamy porządku liniowego na wersjach struktury. Nie można łatwo w węźle zdecydować, którym wskaźnikiem podążać
- Jak temu zaradzić?

Fully Persistent

- Można też rozważać wariant struktury z dopuszczeniem uaktualnień względem wersji z przeszłości
- Jaki pojawia się problem?
- Nie mamy porządku liniowego na wersjach struktury. Nie można łatwo w węźle zdecydować, którym wskaźnikiem podążać
- Jak temu zaradzić?

Fully Persistent

- Można też rozważać wariant struktury z dopuszczeniem uaktualnień względem wersji z przeszłości
- Jaki pojawia się problem?
- Nie mamy porządku liniowego na wersjach struktury. Nie można łatwo w węźle zdecydować, którym wskaźnikiem podążać
- Jak temu zaradzić?

Fully Persistent

- Można też rozważać wariant struktury z dopuszczeniem uaktualnień względem wersji z przeszłości
- Jaki pojawia się problem?
- Nie mamy porządku liniowego na wersjach struktury. Nie można łatwo w węźle zdecydować, którym wskaźnikiem podążać
- Jak temu zaradzić?

jak wygląda porządek częściowy

- korzeniem jest wersja 0
- *i* jest ojcem *j* jeśli j powstał bezpośrednio z i

jak wygląda porządek częściowy

- korzeniem jest wersja 0
- *i* jest ojcem *j* jeśli j powstał bezpośrednio z i

jak wygląda porządek częściowy

- korzeniem jest wersja 0
- *i* jest ojcem *j* jeśli j powstał bezpośrednio z i

Porządek liniowy

- Narzucimy następujący porządek liniowy - stworzymy listę wersji.
- Kiedy i-ta wersja struktury jest tworzona, umieszczamy ją na liście tuż za jej ojcem w drzewie wersji
- Taka lista odpowiada przejściu pre-order drzewa wersji

Porządek liniowy

- Narzucimy następujący porządek liniowy - stworzymy listę wersji.
- Kiedy i-ta wersja struktury jest tworzona, umieszczamy ją na liście tuż za jej ojcem w drzewie wersji
- Taka lista odpowiada przejściu pre-order drzewa wersji

Porządek liniowy

- Narzucimy następujący porządek liniowy - stworzymy listę wersji.
- Kiedy i-ta wersja struktury jest tworzona, umieszczamy ją na liście tuż za jej ojcem w drzewie wersji
- Taka lista odpowiada przejściu pre-order drzewa wersji

Porządek liniowy

- Narzucimy następujący porządek liniowy - stworzymy listę wersji.
- Kiedy i-ta wersja struktury jest tworzona, umieszczamy ją na liście tuż za jej ojcem w drzewie wersji
- Taka lista odpowiada przejściu pre-order drzewa wersji

O czym było

- Rozwiązanie problemu Planar Point Location w czasie zapytania $O(logn)$ oraz zamortyzowanej pamięci $O(n)$. To rozwiązanie jest **optymalne**, jeśli podział płaszczyzny musi być przechowywany. Dodatkowo jest proste i eleganckie.
- Przykład struktury z historią - BST (RB-tree)
- Przedstawienie kilku ciekawych metod, które można uogólnić na prawie dowolne struktury wskaźnikowe.
- Dodatkowe rozważania

O czym było

- Rozwiązanie problemu Planar Point Location w czasie zapytania $O(logn)$ oraz zamortyzowanej pamięci $O(n)$. To rozwiązanie jest **optymalne**, jeśli podział płaszczyzny musi być przechowywany. Dodatkowo jest proste i eleganckie.
- Przykład struktury z historią - BST (RB-tree)
- Przedstawienie kilku ciekawych metod, które można uogólnić na prawie dowolne struktury wskaźnikowe.
- Dodatkowe rozważania

O czym było

- Rozwiązanie problemu Planar Point Location w czasie zapytania $O(logn)$ oraz zamortyzowanej pamięci $O(n)$. To rozwiązanie jest **optymalne**, jeśli podział płaszczyzny musi być przechowywany. Dodatkowo jest proste i eleganckie.
- Przykład struktury z historią - BST (RB-tree)
- Przedstawienie kilku ciekawych metod, które można uogólnić na prawie dowolne struktury wskaźnikowe.
- Dodatkowe rozważania

O czym było

- Rozwiązanie problemu Planar Point Location w czasie zapytania $O(logn)$ oraz zamortyzowanej pamięci $O(n)$. To rozwiązanie jest **optymalne**, jeśli podział płaszczyzny musi być przechowywany. Dodatkowo jest proste i eleganckie.
- Przykład struktury z historią - BST (RB-tree)
- Przedstawienie kilku ciekawych metod, które można uogólnić na prawie dowolne struktury wskaźnikowe.
- Dodatkowe rozważania

O czym było

- Rozwiązanie problemu Planar Point Location w czasie zapytania $O(logn)$ oraz zamortyzowanej pamięci $O(n)$. To rozwiązanie jest **optymalne**, jeśli podział płaszczyzny musi być przechowywany. Dodatkowo jest proste i eleganckie.
- Przykład struktury z historią - BST (RB-tree)
- Przedstawienie kilku ciekawych metod, które można uogólnić na prawie dowolne struktury wskaźnikowe.
- Dodatkowe rozważania

Dziękuję za uwagę.