

Orthogonal Range Searching in 2D using Ball Inheritance

Mads Ravn

Computer Science, Aarhus University

2015

- Giv en præsentation af den i specialet introducerede simplificerede datastruktur til range searching i 2d. (3)
- Beskriv ball-inheritance problemet og forklar sammenhængen til range searching. (2)
- Beskriv også det klassiske kd-træ (1)
- og fortæl om hvilke eksperimenter du har foretaget for at sammenligne performance af de to strukturer. Forklar hvad du så og om det var som forventet. (4)

Outline

1 Introduction

- Orthogonal Range Searching i 2D
- kd-tree

2 Ball Inheritance Search

- Ball Inheritance Problem
- Ball Inheritance Search Data Structure

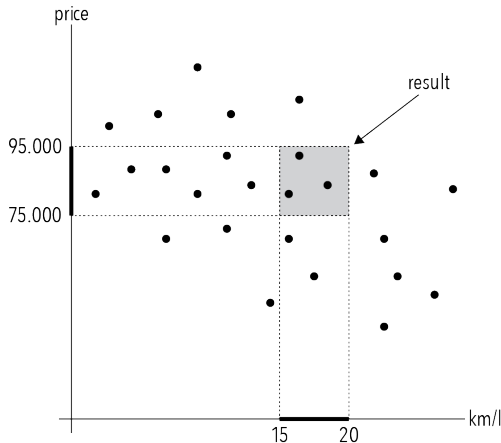
3 Resultater

- Resultater

Outline

- 1 Introduction
 - Orthogonal Range Searching i 2D
 - kd-tree
- 2 Ball Inheritance Search
 - Ball Inheritance Problem
 - Ball Inheritance Search Data Structure
- 3 Resultater
 - Resultater

Orthogonal Range Searching i 2D



Punkter
repræsenterer
objekter der er
plottet efter to
attributer. Nu kan
der søges.
Opdeling af akser.

Preliminaries

Orthogonal range searching

- Svar effektivt på forespørgslen $q = [x_1, x_2] \times [y_1, y_2]$.
- $p \in [x_1, x_2] \times [y_1, y_2] \Leftrightarrow p_x \in [x_1, x_2] \wedge p_y \in [y_1, y_2]$

Preliminaries

- n punkter fra \mathbb{R}^2 . Alle koordinater er unikke
- Rank space. Sorteret array. Vi finder \hat{y}_1 og \hat{y}_2 og så ved vi hvor mange elementer der er imellem dem.
- n er en potens af 2
- static og output-sensitive

Outline

1 Introduction

- Orthogonal Range Searching i 2D
- kd-tree

2 Ball Inheritance Search

- Ball Inheritance Problem
- Ball Inheritance Search Data Structure

3 Resultater

- Resultater

kd-træ

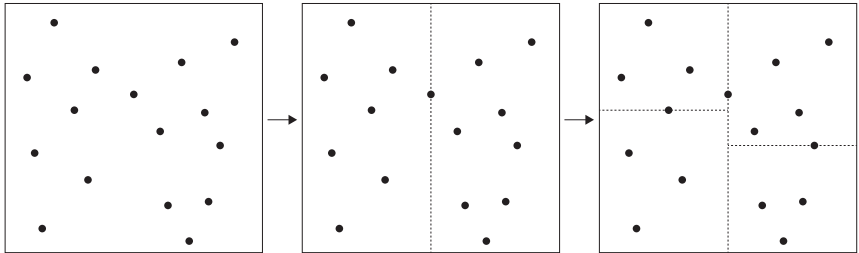
Jon L. Bentley. 1975.

- $\mathcal{O}(n)$ plads
- $\mathcal{O}(\sqrt{n} + k)$ tid

Givet n punkter: x eller y på skift. Et punkt per blad i træet.

Opbygning af kd-træ

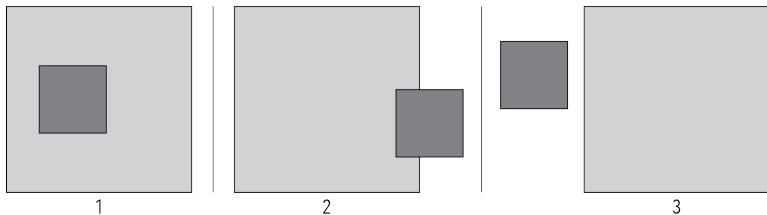
Det $\lceil \frac{n}{2} \rceil$ 'te element bliver valgt som median. Dette element fungerer som en skille-linje mellem de to punkt-mængder.



Punkter i region, punkter i undertræ. Under-opdel.

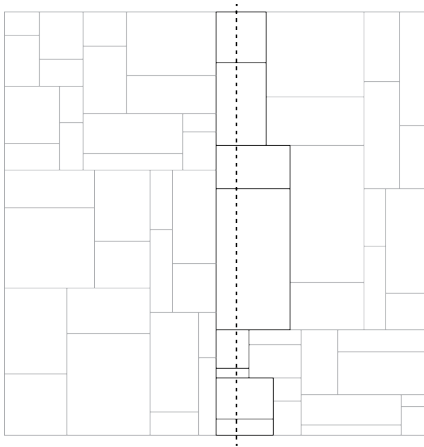
Søgning i kd-træ

■ = node region
■ = search query



1 tager $\mathcal{O}(k)$ tid, 3 tager $\mathcal{O}(1)$ tid. Bound på 2.

Søgning i kd-træ



$O(\sqrt{n})$ regioner.

Outline

- 1 Introduction
 - Orthogonal Range Searching i 2D
 - kd-tree
- 2 Ball Inheritance Search
 - Ball Inheritance Problem
 - Ball Inheritance Search Data Structure
- 3 Resultater
 - Resultater

Ball Inheritance

- Vi er givet et perfekt binært træ.

Ball Inheritance

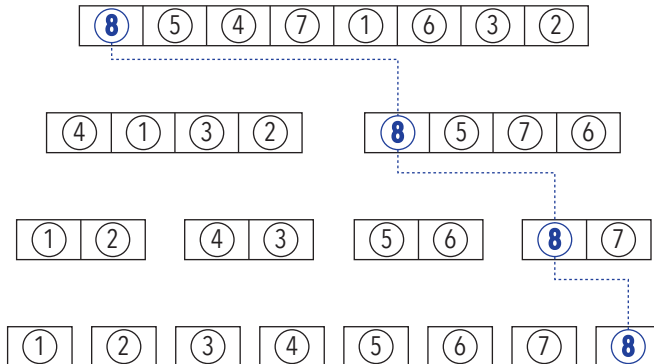
- Vi er givet et perfekt binært træ.
- Roden indeholder n punkter(bolde) som er blevet fordelt sådan at hvert blad lagrer et punkt. Boldene i roden er sorteret.

- Vi er givet et perfekt binært træ.
- Roden indeholder n punkter(bolde) som er blevet fordelt sådan at hvert blad lagrer et punkt. Boldene i roden er sorteret.
- Hver knude har en liste over hvilke bolde der går igennem den. Boldene i knudens liste har samme rækkefølge som boldene i forældre-knudens liste.

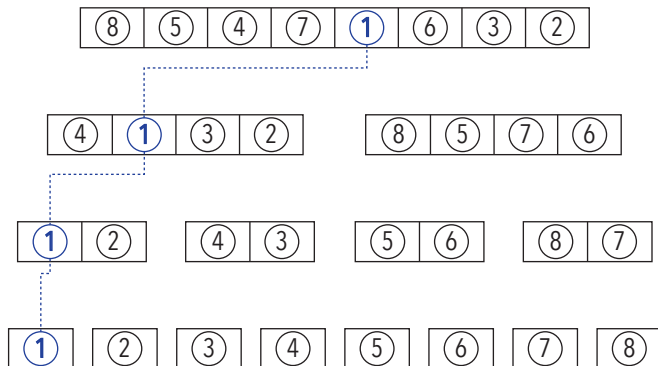
Ball Inheritance

- Vi er givet et perfekt binært træ.
- Roden indeholder n punkter(bolde) som er blevet fordelt sådan at hvert blad lagrer et punkt. Boldene i roden er sorteret.
- Hver knude har en liste over hvilke bolde der går igennem den. Boldene i knudens liste har samme rækkefølge som boldene i forældre-knudens liste.
- Løs: Givet en knude og et index i knudens liste, hvilket blad ender denne bold ved?
- Vi kan nu følge en bold fra en knude til et blad med $\mathcal{O}(\lg n)$ skridt.

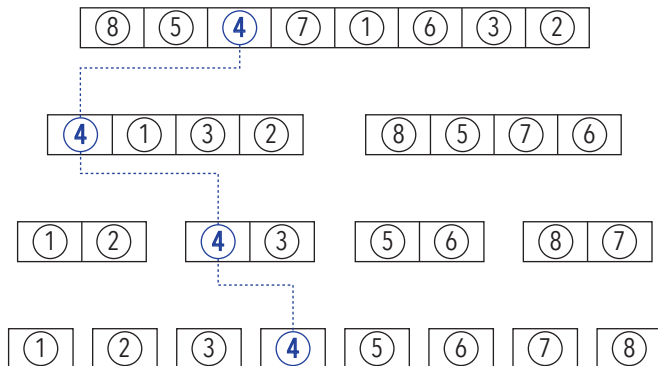
ball inheritance



ball inheritance



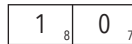
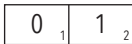
ball inheritance



RANK SELECT

Givet en bitvektor, så er Rank-Select query en constant-time query der finder boldens nye position i barnets bitvektor.
Man kan nu komme ned med $\mathcal{O}(\lg n)$ tid. $\mathcal{O}(1 \cdot \lg n)$.

ball inheritance



Faster Queries

Færre skridt fra knude til blad. Knuder på niveau deleligt med B^i hopper B^i niveauer over.

Det bruger

- $\mathcal{O}(n \cdot \lg_B \lg n) = \mathcal{O}(\frac{n}{\epsilon}) = \mathcal{O}(n)$ plads
- $\mathcal{O}(B \cdot \lg_B \lg n) = \mathcal{O}(\lg^\epsilon n)$ tid

hvor $\epsilon > 0$ er en arbitrær lille konstant. Space-time tradeoff.

Faster Queries

Færre skridt fra knude til blad. Knuder på niveau deleligt med B^i hopper B^i niveauer over.

Det bruger

- $\mathcal{O}(n \cdot \lg_B \lg n) = \mathcal{O}(\frac{n}{\epsilon}) = \mathcal{O}(n)$ plads
- $\mathcal{O}(B \cdot \lg_B \lg n) = \mathcal{O}(\lg^\epsilon n)$ tid

hvor $\epsilon > 0$ er en arbitrær lille konstant. Space-time tradeoff.

$B = \Omega(\lg^\epsilon n)$ og $B = \lg^{\frac{\epsilon}{2}} = \Omega(\lg \lg n)$.

$\lg_B \lg n$ er valgt fordi $B^i \leq \lg n \Leftrightarrow i \leq \lg_B \lg n$.

Outline

- 1 Introduction
 - Orthogonal Range Searching i 2D
 - kd-tree
- 2 Ball Inheritance Search
 - Ball Inheritance Problem
 - Ball Inheritance Search Data Structure
- 3 Resultater
 - Resultater

BISintro

Ball Inheritance Search (BIS) er en datastruktur som er en simplificering af den datastruktur der findes i **Orthogonal Range Searching on the RAM, Revisited**[1] af Chan et al.

$\mathcal{O}(\lg n + k \cdot \lg^\epsilon n)$ imod $\mathcal{O}(\lg \lg n + (1 + k) \cdot \lg^\epsilon n)$.

Ball Inheritance Search

- $\mathcal{O}(n + n \cdot \lg_B \lg n) = \mathcal{O}(n + \frac{n}{\epsilon})$ plads.
- $\mathcal{O}(\lg n + k \cdot B \lg_B \lg n) = \mathcal{O}(\lg n + k \cdot \lg^\epsilon n)$ tid, hvor $\epsilon > 0$ er en arbitrær lille konstant

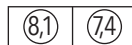
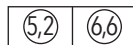
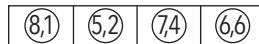
da vi har valgt $B = \Omega(\lg^\epsilon n) = \lceil \frac{1}{2} \lg^{\frac{1}{3}} n \rceil$.

Ball Inheritance Search

Når vi kigger på BIS, så er punkterne **fordelt** efter x og **sorteret** efter y . Det betyder

- Undertræer i knuder i mellem x_1 og x_2 kun indeholder punkter i $[x_1, x_2]$.
- Bolde mellem \hat{y}_1 og \hat{y}_2 kun indeholder punkter i $[y_1, y_2]$.

ball inheritance



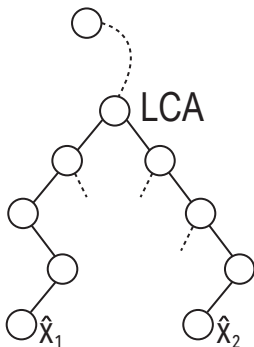
Ball Inheritance Search

I denne datastruktur bruger vi ball-inheritance til

- Følge en bold ned når vi ved den ligger i vores søge-område.
Dvs dekode fra bold-til-blad(punkt).
- Fra roden, opdatere et interval af hvilke bolder der ligger i $[y_1, y_2]$.

Nu har vi markeret de bolde der ligger i $[y_1, y_2]$ på de knuder hvis undertræ kun indeholder punkter i $[x_1, x_2]$. Og det er netop de punkter der ligger i $[x_1, x_2] \times [y_1, y_2]$.

ball inheritance



- Rank space opslag ved roden.
- Vedligehold $[\hat{y}_1, \hat{y}_2]$ ned til LCA.
- Find fully contained knuder og deres $[\hat{y}_1, \hat{y}_2]$ interval.
- Ball Inheritance fra knuder.

Ball Inheritance Search

Vi har nu nogle knuder og lister over indeces i disse knuder. Det er præcis det problem ball inheritance løser. Vi kan nu bruge ball inheritance på alle disse knuder til at finde ud af hvilke blade der indeholder punkter i $[y_1, y_2]$.

Det giver en kørselstid på $\mathcal{O}(\lg n + k \cdot \lg^\epsilon n)$ for at finde k punkter.

plads

Denne datastruktur bruger $\mathcal{O}(n)$ plads.

- Bit vectors. $\mathcal{O}(n)$ bits per level.

plads

Denne datastruktur bruger $\mathcal{O}(n)$ plads.

- Bit vectors. $\mathcal{O}(n)$ bits per level.
- Store hop $\mathcal{O}(\lg \Sigma)$ bits per entry hvert $\lg \Sigma$ level. ($\lg \Sigma = B^i$).

plads

Denne datastruktur bruger $\mathcal{O}(n)$ plads.

- Bit vectors. $\mathcal{O}(n)$ bits per level.
- Store hop $\mathcal{O}(\lg \Sigma)$ bits per entry hvert $\lg \Sigma$ level. ($\lg \Sigma = B^i$).
- Egentlig punkter

plads

Denne datastruktur bruger $\mathcal{O}(n)$ plads.

- Bit vectors. $\mathcal{O}(n)$ bits per level.
- Store hop $\mathcal{O}(\lg \Sigma)$ bits per entry hvert $\lg \Sigma$ level. ($\lg \Sigma = B^i$).
- Egentlig punkter
- Binær søgning

1 Introduction

- ## 2 Ball Inheritance Search

- ### 3 Resultater

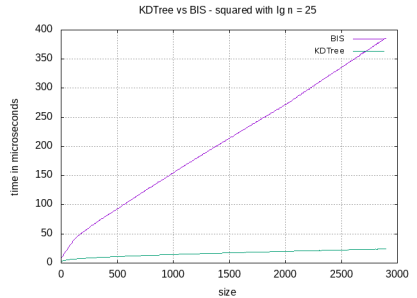
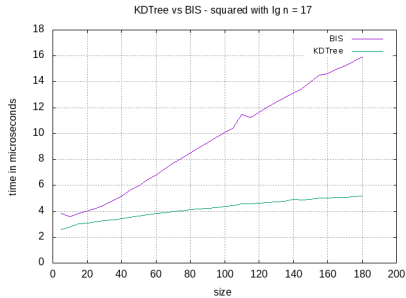
- 
- AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

setup

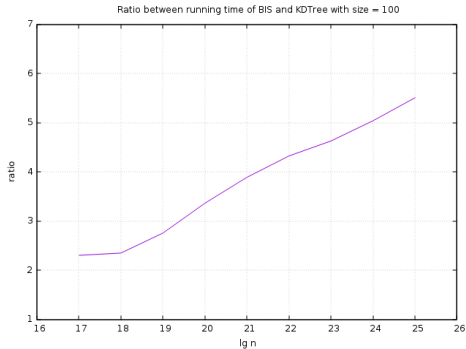
- Square area $\sqrt{n} \cdot \sqrt{k} \times \sqrt{n} \cdot \sqrt{k}$ returnerer k punkter.
- Slices af størrelse k returnerer k punkter. $[0, n] \times [y, y + k]$
- $\sqrt{n} + k = \lg n + k \cdot \lg^\epsilon n \Leftrightarrow k = \frac{\sqrt{n} - \lg n}{\lg^\epsilon n - 1}$

- $\mathcal{O}(\lg n + k \cdot \lg^\epsilon n)$ vs $\mathcal{O}(\sqrt{n} + k)$.
- BIS er mere stabil når vi ændrer shape. $\mathcal{O}(\sqrt{n})$ er problemet.
- Slice er godt for BIS og square er godt for kd-træ.
- kd-træ er hurtigere jo flere punkter, og jo mindre $\mathcal{O}(\sqrt{n})$ er.
- $\lg n < \sqrt{n}$, så vi forventer at en dårlig shape betyder meget for kd-træet.
- Vi har også kigget på $k \leq 200$ for personlig interaktion. Der er BIS god.

Squared

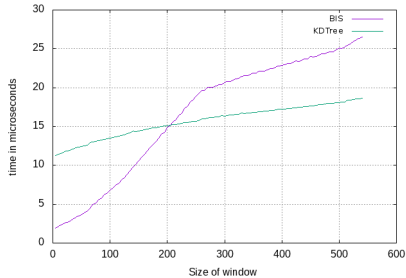


Squared

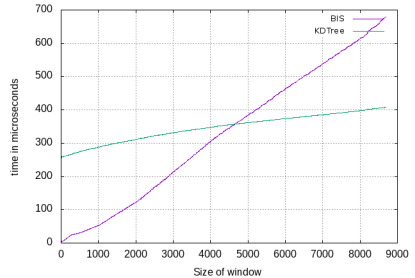


vertical

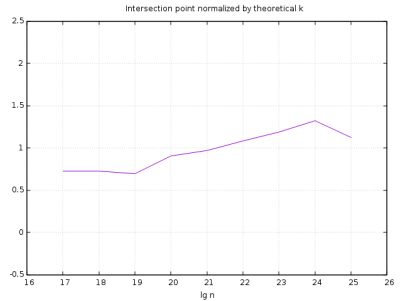
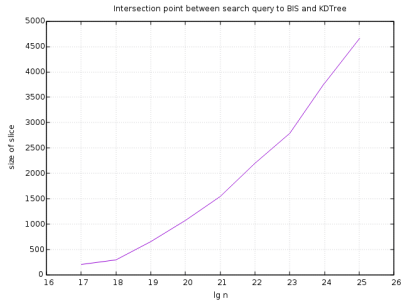
KDTree vs BIS - vertical 17



KDTree vs BIS - vertical 25

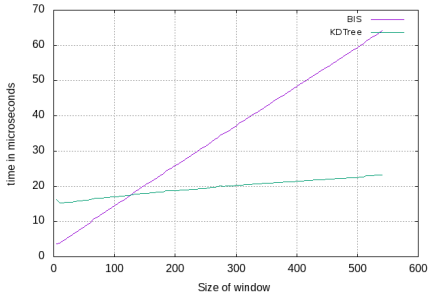


vertical

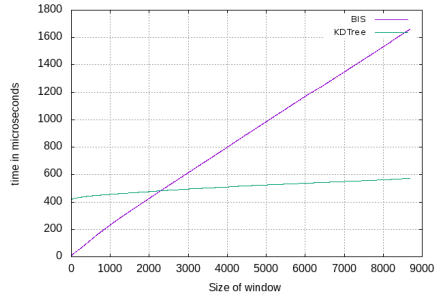


horizontal

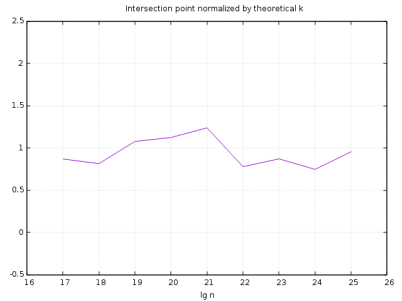
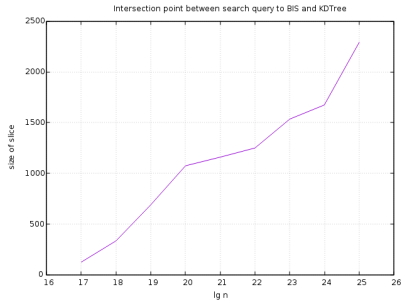
KDTree vs BIS - horizontal 17



KDTree vs BIS - horizontal 25

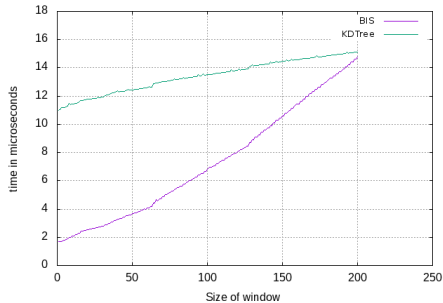


horizontal

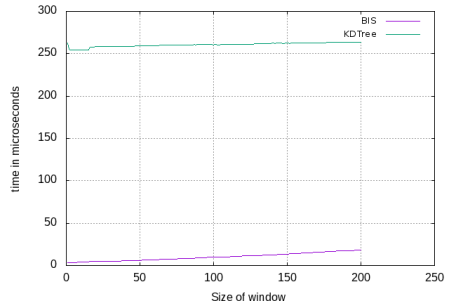


small k

KDTree vs BIS - vertical 17

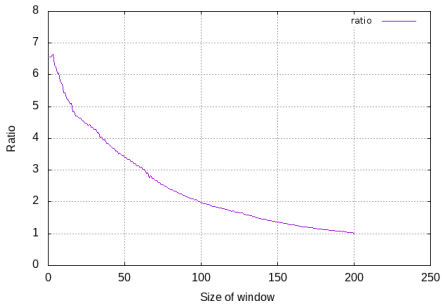


KDTree vs BIS - vertical 25

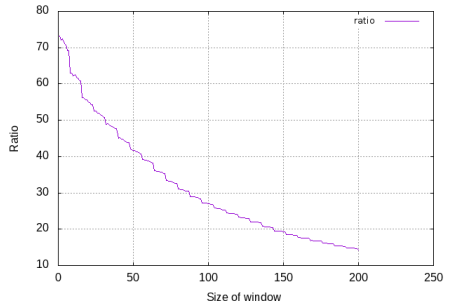


small k

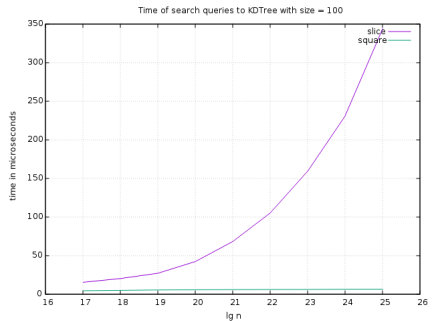
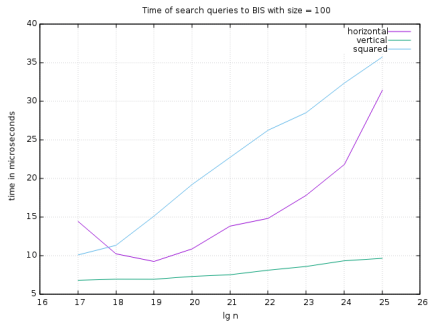
KDTree vs BIS - vertical 17



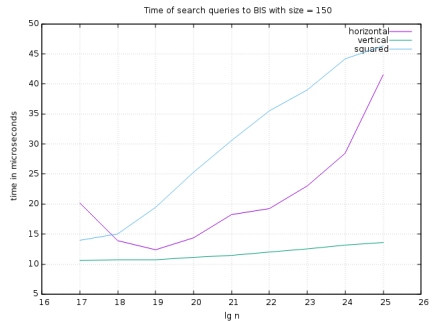
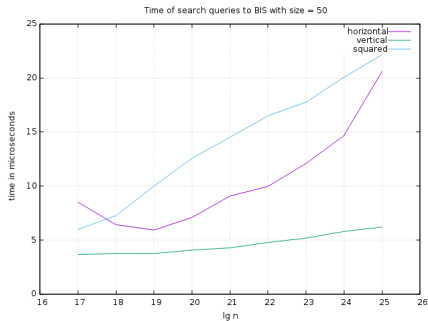
KDTree vs BIS - vertical 25



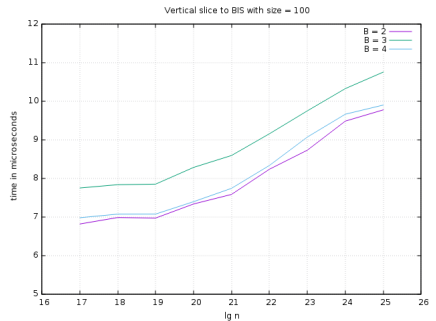
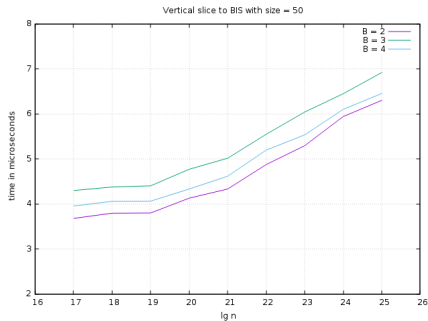
small k



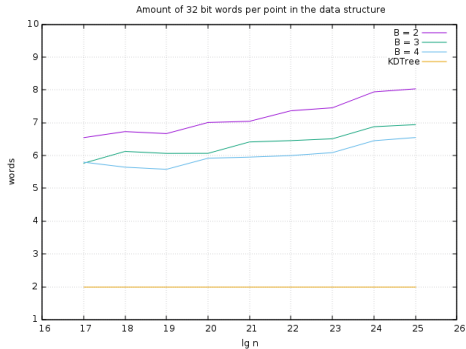
small k



small k



sizes

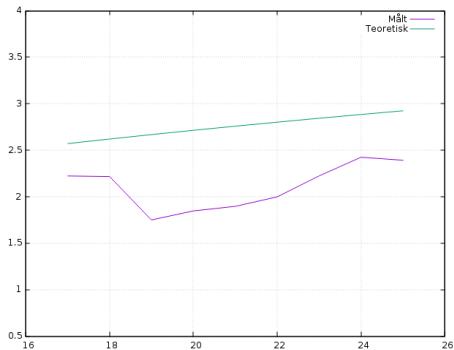


- Bedre udpakning
- Cache
- Concurrency

Færdig

Spørgsmål?

praktisk log epsilon n



Små hop

Hvert niveau gemmer n bits som indikerer om bolden er gået til højre eller venstre. Hvert 32 bit gemmer vi et 32 bit major checkpoint. Precomputed tabel med 16 bit tal som tæller antal 1-entries. $\mathcal{O}(n)$ bits per level.

Store hop

$\mathcal{O}(\lg \Sigma)$ per entry. $\Sigma = 2^{B^i}$. Så plads er $\mathcal{O}(B^i)$ bits per entry.
Det er

$$\sum_{i=1}^{\lg_B \lg n} \frac{\lg n}{B^i} \cdot \mathcal{O}(B^i) = \mathcal{O}(\lg n \cdot \lg_B \lg n)$$

for hele kæden. Vælg nu $B = \Omega(\lg^\epsilon n)$.

Vi har n punkter, hvilket giver $\mathcal{O}(n \lg n \cdot \lg_B \lg n)$ bits. Det er $\mathcal{O}(n \cdot \frac{\lg \lg n}{\epsilon \lg \lg n}) = \mathcal{O}(\frac{n}{\epsilon})$ ord.

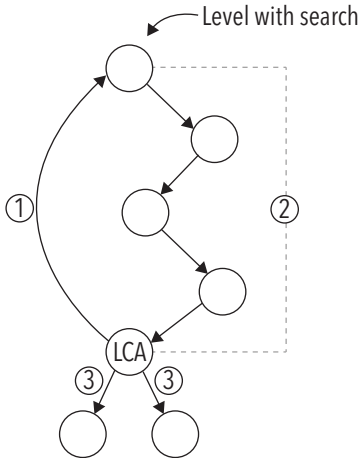
Store hop

Tiden for de store hop er højst $\mathcal{O}(B \lg_B \lg n)$. Vælg
 $B = \lg^{\epsilon/2} n = \Omega(\lg \lg n)$.

OBIS

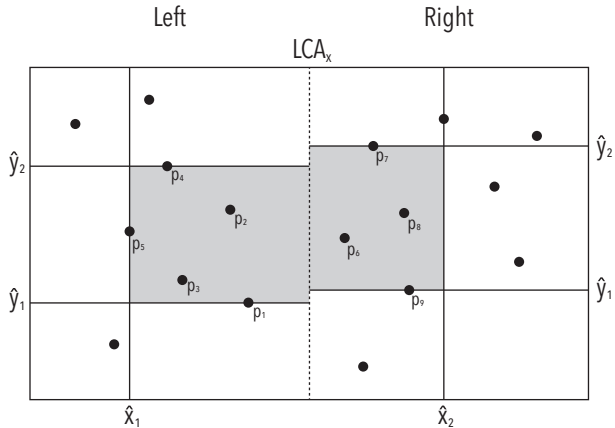
OBIS af Chan et al. Med $\mathcal{O}(n)$ plads og $\mathcal{O}(\lg \lg n + (1 + k) \cdot \lg^\epsilon n)$.
Bruger også Ball Inheritance til at finde de k punkter.

OBIS

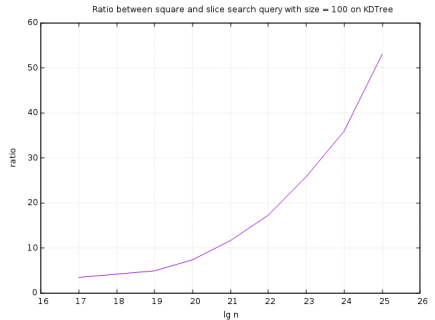
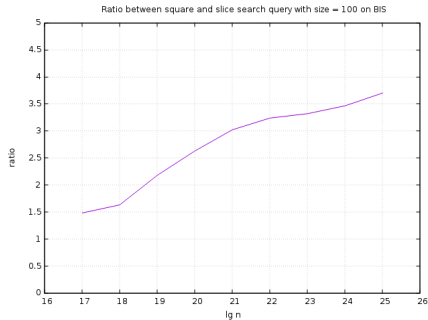


- Op til nærmeste level med pred-search
- Gå ned til LCA højst $\lg \lg n$ levels nede.
- Gå ned og find resultater i begge børn af LCA.

OBIS



small k



References I



Timothy M. Chan, Kasper Green Larsen, Mihai Patrascu.
Orthogonal Range Searching on the RAM, Revisited.