Advanced Data Structures

Lecture 06: BSP Trees and Packed and Compressed Hash Tables

Florian Kurpicz
Recap: 2-Dimensional Rectangular Range Searching

Important
- Assume now two points have the same x- or y-coordinate
- Generalize 1-dimensional idea
- 1-dimensional
  - Split number of points in half at each node
  - Points consist of one value
- 2-dimensional
  - Points consist of two values
  - Split number of points in half w.r.t. one value
  - Switch between values depending on depth
Motivation

- hidden surface removal
- which pixel is visible
- important for rendering
**z-Buffer Algorithm**

- transform scene such that viewing direction is positive z-direction
- consider objects in scene in arbitrary order
- maintain two buffers
  - frame buffer (currently shown pixel)
  - z-buffer (z-coordinate of object shown)
- compare z-coordinate of z-buffer and object
- first sort object in depth-order
- depth-order may not always exist 🎈
- how to efficiently sort objects?
BSP Trees (1/2)

- partition space using hyperplanes
- binary partition similar to kd-tree
- hyperplanes create half-spaces and cut objects into fragments

\[
\begin{align*}
    h^+ &= \{(x_1, \ldots, x_d) : a_1 x_1 + \cdots + a_d x_d > 0\} \\
    h^- &= \{(x_1, \ldots, x_d) : a_1 x_1 + \cdots + a_d x_d < 0\}
\end{align*}
\]

- each split creates two nodes in a tree
- if number of objects in space is one: leaf
- otherwise: inner node
BSP Trees (2/2)

- for leaf: store object/fragment
- for inner node $v$: store hyperplane $h_v$ and the objects contained in $h_v$
- left child represents objects in upper half-space $h^+$
- right child represents objects in lower half-space $h^-$

- space of BSP tree is number of objects stored at all nodes
- what about fragments?
- too many fragments can make the tree big
Auto-Partitioning

- sorting points for kd-trees worked well
- BSP-tree is used to sort objects in dept-order
- auto-partitioning uses splitters through objects
  - 2-dimensional: line through line segments
  - 3-dimensional: half-plane through polygons
Painter’s Algorithm

- consider view point $p_{\text{view}}$
- traverse through tree and always recurse on half-space that does not contain $p_{\text{view}}$ first
- then scan-convert object contained in node
- then recurse on half-space that contains $p_{\text{view}}$
Constructing Planar BSP Trees (1/3)

- use auto-partitioning
- construction similar to construction of kd-tree
- store all necessary information
  - hyperplane
  - objects in hyperplane
- how to determine next hyperplane?
- creating fragments increases size of BSP tree

- let \( s \) be object and \( \ell(s) \) line through object
- order matters
Constructing Planar BSP Trees (2/3)

Lemma: Number Line Fragments

The expected number of fragments generated when iterating through the line segments using a random permutation is $O(n \log n)$.

Proof (Sketch)

- distance of lines $\text{dist}_{s_i}(s_j) = \begin{cases} \# \text{ segments inters. } \ell(s_i) \text{ between } s_i \text{ and } s_j \\ \ell(s_i) \text{ inters. } s_j \\ \infty \quad \text{otherwise} \end{cases}$
- example on the board

Proof (Sketch, cnt.)

- let $\text{dist}_{s_i}(s_j) = k$ and $s_{j_1}, \ldots, s_{j_k}$ be segments between $s_i$ and $s_j$
- what is the probability that $\ell(s_i)$ cuts $s_j$?
- this happens if no $s_{j_x}$ is processed before $s_i$
- since order is random

\[ P[\ell(s_i) \text{ cuts } s_j] \leq \frac{1}{\text{dist}_{s_i}(s_j) + 2} \]
Proof (Sketch, cnt.)

- expected number of cuts

\[
\mathbb{E}[\# \text{ cuts generated by } s_i] \leq \sum_{j \neq i} \frac{1}{\text{dist}_{s_i}(s_j)} + 2 \leq 2 \sum_{k=0}^{n-2} \frac{1}{k + 2} \leq 2 \ln n
\]

- all lines generate at most \(2n \ln n\) fragments

**Lemma: BSP Construction**

A BSP tree of size \(O(n \log n)\) can be computed in expected time \(O(n^2 \log n)\)

**Proof (Sketch)**

- computing permutation in linear time
- construction is linear in number of fragments to be considered
- number of fragments in subtree is bounded by \(n\)
- number of recursions is \(n \log n\)
New Topic: Hash Tables

- now hash tables
- first packed and compressed hash table
- presented in January ’23 at ALENEX
Motivation

Setting
- static hash table for objects of variable size
- storing objects in external memory
- ideally retrieve objects in single I/O
- very small internal memory data structure

Objects of Variable Size
- only blocks of size $B$ bits can be transferred
- one I/O per block transfer

External Memory
## Space-Efficient Object Stores from Literature

<table>
<thead>
<tr>
<th>Method</th>
<th>$I_b$</th>
<th>Load factor</th>
<th>I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fixed</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Larson et al. [LR85]</td>
<td>96</td>
<td>&lt;96%</td>
<td>1</td>
</tr>
<tr>
<td>SILT SortedStore [Lim+11]</td>
<td>51</td>
<td>100%</td>
<td>1</td>
</tr>
<tr>
<td>Linear Separator [Lar88]</td>
<td>8</td>
<td>85%</td>
<td>1</td>
</tr>
<tr>
<td>Separator [GL88; LK84]</td>
<td>6</td>
<td>98%</td>
<td>1</td>
</tr>
<tr>
<td>Robin Hood [Cel88]</td>
<td>3</td>
<td>99%</td>
<td>1.3</td>
</tr>
<tr>
<td>Ramakrishna et al. [RT89]</td>
<td>4</td>
<td>80%</td>
<td>1</td>
</tr>
<tr>
<td>Jensen, Pagh [JP08]</td>
<td>0</td>
<td>80%</td>
<td>1.25</td>
</tr>
<tr>
<td>Cuckoo [Aza+94; Pag03]</td>
<td>0</td>
<td>&lt;100%</td>
<td>2</td>
</tr>
<tr>
<td>PaCHash, $a = 1$</td>
<td>2</td>
<td>100%</td>
<td>2*</td>
</tr>
<tr>
<td>PaCHash, $a = 8$</td>
<td>5</td>
<td>100%</td>
<td>1.13*</td>
</tr>
<tr>
<td><strong>variable</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SILT LogStore [Lim+11]</td>
<td>832</td>
<td>100%</td>
<td>1</td>
</tr>
<tr>
<td>SkimpyStash [DSL11]</td>
<td>32</td>
<td>≤98%</td>
<td>8</td>
</tr>
<tr>
<td>PaCHash, $a = 1$</td>
<td>2</td>
<td>99.95%</td>
<td>2.06*</td>
</tr>
<tr>
<td>PaCHash, $a = 8$</td>
<td>5</td>
<td>99.95%</td>
<td>1.19*</td>
</tr>
</tbody>
</table>
PaCHash Overview

- Objects of variable size
- Hash function $h: K \rightarrow 1..am$
- EM sorted objects in bins
- First bin (partially) in block
- Store offset in EM
- $\bar{B}$ bits remaining per EM block
- Tuning parameter
- # EM blocks
- No fragmentation
- $p = \langle p_1, \ldots, p_m \rangle$
Query Algorithm

- $b_x = h(x)$
- find first $i$ with $p_i \leq b_x$
- if $p_i = b_x$ let $i = i - 1$
- find first $j$ with $p_j > b_x$
- return $i..(j - 1)$

Elias-Fano Coding

- given $k$ monotonic increasing integers in $1..u$
- store $\log k$ MSBs encoded in bit vector
- store $\log(u/k)$ LSBs plain
- $k(2 + \log(u/k)) + 1 + o(k)$ bits in total
- predecessor in $O(k)$ time

Lemma: Space with Elias-Fano Coding

When using Elias-Fano coding [Eli74; Fan71] to store $p$, the index needs $2 + \log a + o(1)$ bits of internal memory per block.
Lemma: Expected Predecessor Time

When using Elias-Fano coding to store $p$, the range of blocks containing the bin of an object $x$ can be found in expected constant time.

Proof (Sketch)

- consider $\lceil \log m \rceil$ MSB
- let bin $b_x$ have MSBs equal to $u$
- expected size $\mathbb{E}(Y_u)$ of all bins with MSB $u$ that are $< b_x$ is

$$\sum_{y \in S} |y| \cdot \mathbb{P}(h(y) \text{ w/ MSB } = u; h(y) < h(x))$$

$$\leq \sum_{y \in S} |y| \cdot \mathbb{P}(h(y) \text{ w/ MSB } = u)$$

$$= \frac{1}{m} \sum_{y \in S} |y| = \frac{m\bar{B}}{m} = \bar{B}$$

- number of entries to scan is $\mathbb{E}(Y_u)/\bar{B} = 1$
Lemma: Additional Blocks Loaded

Retrieving an object \( x \) of size \( |x| \) from a PaCHash data structure loads \( \leq 1 + |x|/\bar{B} + 1/a \) consecutive blocks from the external memory in expectation.

Proof (Sketch)

- expected size of bin \( b_x = h(x) \)

\[
E(|b_x|) = |x| + \sum_{y \in S, y \neq x} |y|P(y \in b_x)
\]

\[
\leq |x| + \sum_{y \in S} |y|P(y \in b_x)
\]

\[
= |x| + \sum_{y \in S} |y| \cdot \frac{1}{am} = |x| + \frac{\bar{B}}{a}
\]

Proof (Sketch, cnt.)

- expected number of blocks overlapped by \( b_x \)

\[
E(X) = 1 + \left( E(|b_x|) - 1 \right)/\bar{B}
\]

\[
= 1 + \frac{|x|}{\bar{B}} + \frac{1}{a} - 1/\bar{B}
\]

- \( P(\text{bin and block border align}) = 1/\bar{B} \)
# Experimental Evaluation

## Hardware and Software
- Intel i7 11700 (base clock speed: 2.5 GHz)
- 1 TB Samsung 980 Pro NVMe SSD
- Ubuntu 21.10 (Kernel 5.13.0)
- io_uring for I/O operations
- GCC 11.2.0 (-O3 -march=native)
- \( b = 4096 \) bytes

## Objects
- here only **fixed size**
- more in the paper (very similar results)

## Competitors
- LevelDB [Goo21]
- RocksDB [Fac21]
- SILT [Lim+11]
- std::unordered_map
- RecSplit [EGV20]
- CHD [BBD09; CR+12]
- PTHash [PT21]
### Construction

#### Space

- **Internal Space** (B/object):
  - CHD (16-perfect) [BBD09]
  - Cuckoo (here)
  - LevelDB (Static part) [Goo21]
  - PaCHash (here)
- **External Space** (B/object):
  - RecSplit [EGV20]
  - RocksDB [Fac21]
  - SILT (Static part) [Lim+11]
  - Separator (here)
  - std::unordered_map

#### Construction Throughput

- Buffered I/O [Objects/s]:
  - CHD (16-perfect) [BBD09]
  - LevelDB [Goo21]
  - RecSplit [EGV20]
  - SILT (Static part) [Lim+11]
  - Separator (here)
  - std::unordered_map
Queries

<table>
<thead>
<tr>
<th>Number of objects [Millions]</th>
<th>Query Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>60</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>0.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of objects [Millions]</th>
<th>Query Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>4</td>
<td>0.3</td>
</tr>
<tr>
<td>6</td>
<td>0.1</td>
</tr>
<tr>
<td>8</td>
<td>0.05</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of objects [Millions]</th>
<th>Query Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.5</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0.5</td>
</tr>
<tr>
<td>8</td>
<td>0.1</td>
</tr>
</tbody>
</table>

- CHD (16-perfect) [BBD09]
- LevelDB [Goo21]
- RecSplit [EGV20]
- SILT (Static part) [Lim+11]
- Cuckoo (here)
- PTHash [PT21]
- RocksDB [Fac21]
- Separator (here)
- LevelDB (Static part) [Goo21]
- PaCHash (here)
- SILT [Lim+11]
- std::unordered_map
Maximum Load Factor of Competitors

Cuckoo Hashing

Separator Hashing

Average object size [B]

Identical size
Normal distribution
Uniform distribution
When using Succincter [Pat08] to store \( p \), the index needs 
\[
1.44 + \log(a + 1) + o(1) \text{ bits of internal memory per block.}
\]

**Structure of Bit Vector**
- runs of 0s and 10s
- sometimes additional 1s

**Entropy Encoding**
- encode positions directly
- compress bit vector using Huffman codes
- encode blocks of size 8, 16, 32, or 64

![Graph showing space and query throughput vs. parameter a](image-url)
Conclusion and Outlook

This Lecture
- BSP trees
- PaCHash

Next Lecture
- more on hashing

Advanced Data Structures
- PaCHash
- Successor
- RMQ
- Kd- & Range Tree
- static BV
- static succ. trees
- range min-max tree
- succ. graphs
measuring memory
measuring time
std::map vs std::unordered_map
more questions?
Bibliography I


Bibliography II


Bibliography III


Bibliography IV


