B-trees and kd-trees

Piotr Indyk

(slides partially by Lars Arge from Duke U)

Before we start

- If you are considering taking this class (or attending just a few lectures), send me an e-mail : indyk@theory.lcs.mit.edu
- Web page up and running: http://theory.lcs.mit.edu/~indyk/MASS
- Reading list updated – if you want to present, send me an e-mail

Today

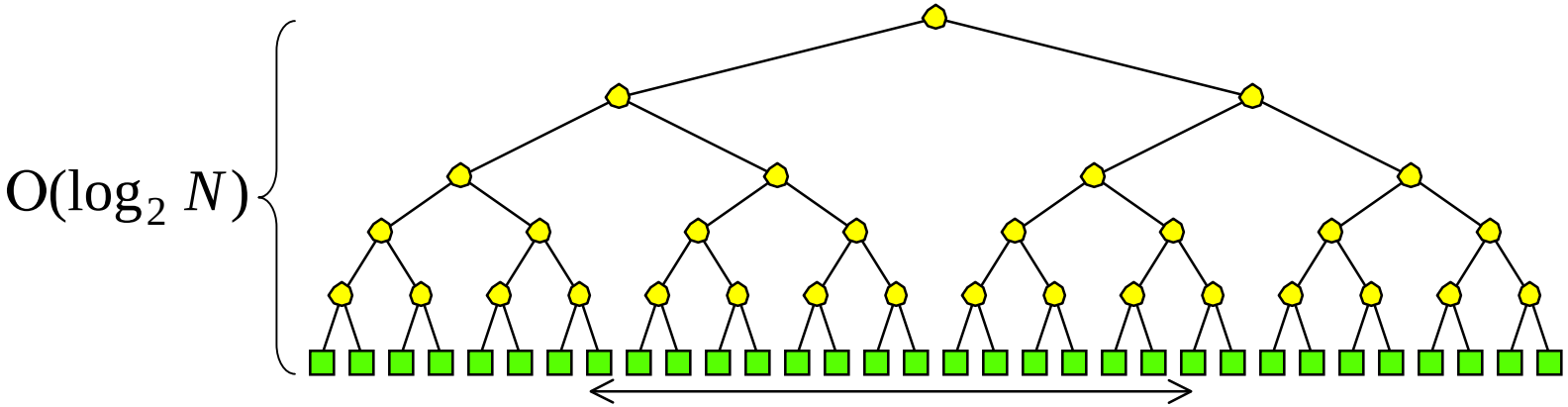
- 1D data structure for searching in *external* memory
  - O(log N) I/O's using standard data structures
  - Will show how to reduce it to O($\log_B N$)
- We already know how to sort using O(N/B $\log_{M/B} N$) I/O's
- Therefore, we will move on to 2D
- We will start from *main* memory data structure for range search problem:
  - Input: a set of points in 2D
  - Goal: a data structure, which given a query rectangle, reports all points within the rectangle
- Then we will continue with approximate nearest neighbor (also in main memory)

Searching in External Memory

- Dictionary (or successor) data structure for 1D data:
  - Maintains elements (e.g., numbers) under insertions and deletions
  - Given a *key* K, reports the successor of K; i.e., the smallest element which is greater or equal to K

Internal Search Trees

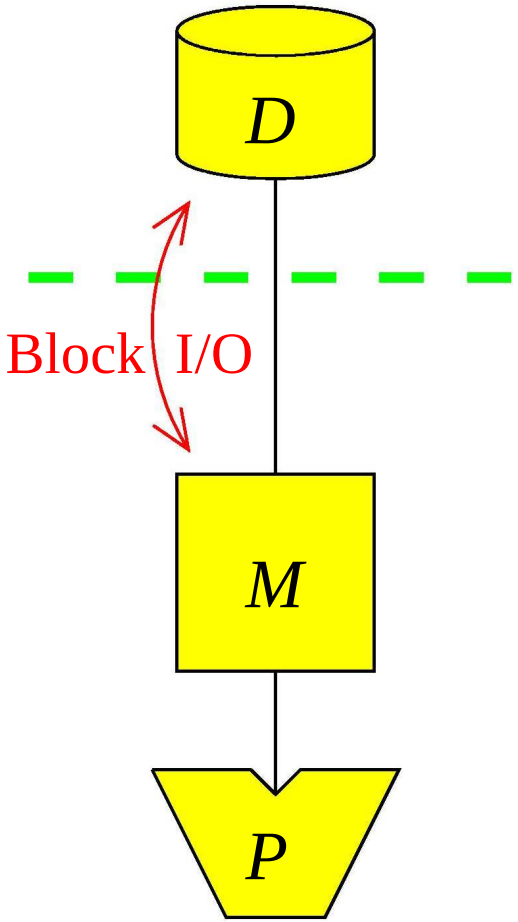
- Binary search tree:
  - Standard method for search among N elements
  - We assume elements in leaves



  - Search traces at least one root-leaf path

$\Rightarrow$ Search in $O(\log_2 N)$ time

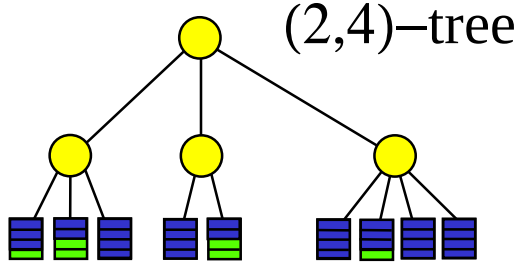
Model



- Model as previously
  - N : Elements in structure
  - B : Elements per block
  - M : Elements in main memory
  - T : Output size in searching problems

(a,b)-tree (or B-tree)

- T is an (a,b)-tree ($a \geq 2$ and $b \geq 2a-1$)
  - All leaves on the same level (contain between a and b elements)
  - Except for the root, all nodes have degree between a and b
  - Root has degree between 2 and b



- (a,b)-tree uses linear space and has height $O(\log_a N)$

$\Downarrow$

Choosing $a,b = \Theta(B)$ each node/leaf stored in one disk block

$\Downarrow$

$O(N/B)$ space and $O(\log_B N)$ query

(*a*,*b*)-Tree Insert

- Insert:

Search and insert element in leaf *v*

DO *v* has *b+1* elements

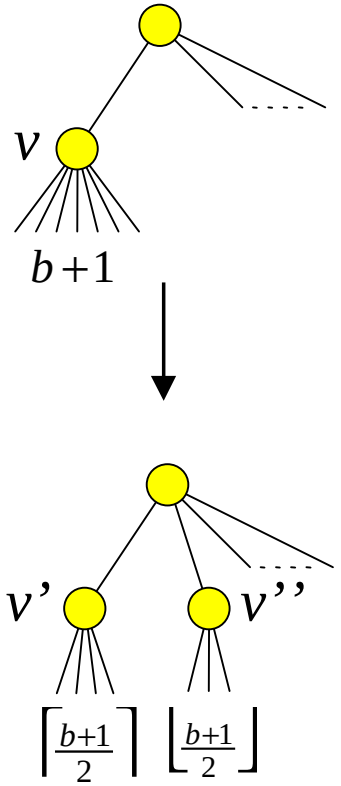
Split *v*:

make nodes *v'* and *v''* with $\lceil \frac{b+1}{2} \rceil \leq b$ and $\lfloor \frac{b+1}{2} \rfloor \geq a$ elements

insert element (ref) in *parent(v)*

(make new root if necessary)

v=parent(v)



- Insert touches $O(\log_a N)$ nodes

(*a*,*b*)-Tree Delete

- Delete:

Search and delete element from leaf *v*

DO *v* has *a-1* children

Fuse *v* with sibling *v'*:

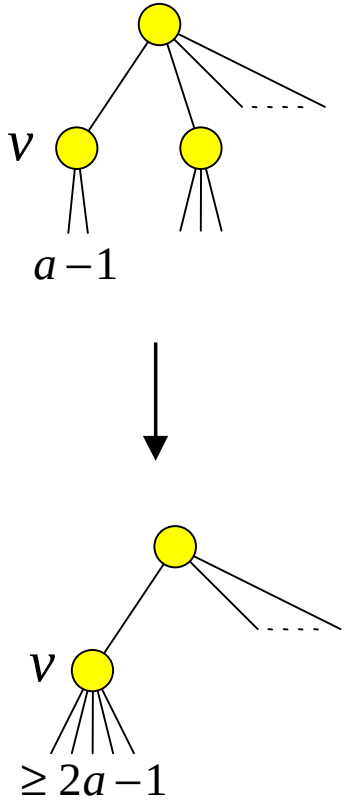
move children of *v'* to *v*

delete element (ref) from *parent(v)*

(delete root if necessary)

If *v* has $>b$ (and $\leq a+b-1$) children split *v*

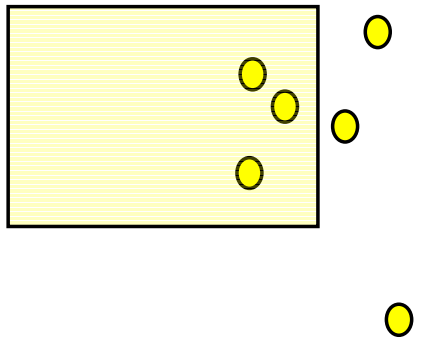
v=parent(v)



- Delete touches $O(\log_a N)$ nodes

Range Searching in 2D

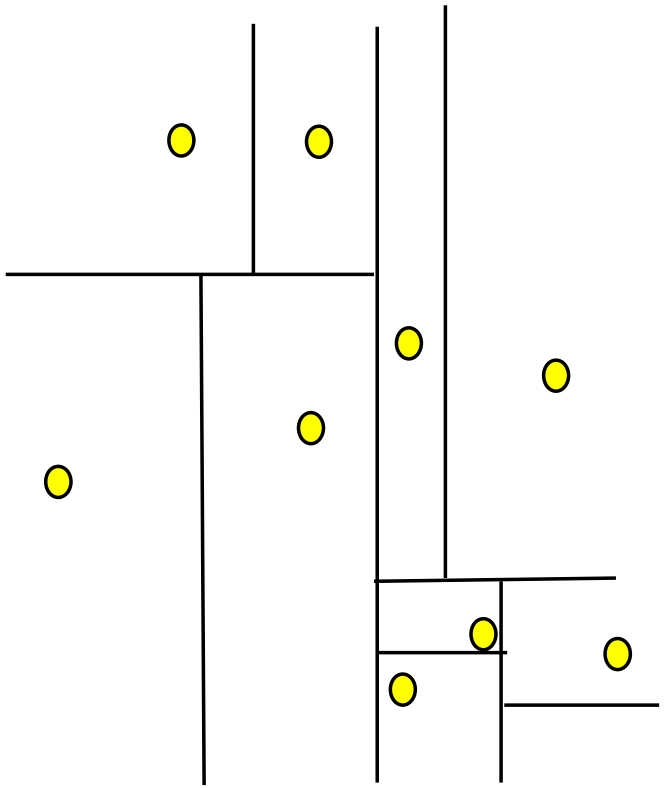
- Recall the definition: given a set of n points, build a data structure that for any query rectangle R, reports all points in R
- Updates are also possible, but:
  - Fairly complex in theory
  - Straightforward approach works well in practice



Kd-trees

- Not the most efficient solution in theory
- Everyone uses it in practice
- Algorithm:
  - Choose x or y coordinate (alternate)
  - Choose the median of the coordinate; this defines a horizontal or vertical line
  - Recurse on both sides
- We get a binary tree:
  - Size: O(N)
  - Depth: O(log N)
  - Construction time: O(N log N)

Kd-tree: Example



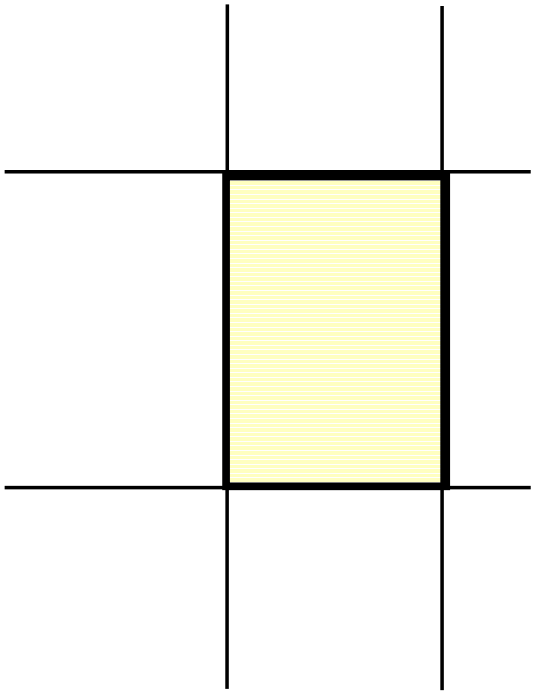
Each tree node v corresponds to a region Reg(v).

Kd-tree: Range Queries

- Recursive procedure, starting from v=root
- Search (v,R):
  - If v is a leaf, then report the point stored in v if it lies in R
  - Otherwise, if Reg(v) is contained in R, report all points in the subtree of v (*)
  - Otherwise:
    * If Region(left(v)) intersects R, then Search(left(v),R)
    * If Region(right(v)) intersects R, then Search(right(v),R)

Query Time Analysis

- We will show that Search takes at most O(sqrt{n}+k) time, where k is the number of reported points
  - The total time needed to report all points in all subtrees (i.e., taken by step (*)) is O(k)
  - We just need to bound the number of nodes v such that Reg(v) intersects R but is not contained in R. In other words, the boundary of R intersects the boundary of Reg(v)
  - Will make a gross overestimation: will bound the number of Reg(v) which cross *any* horizontal or vertical line



Query Time Continued

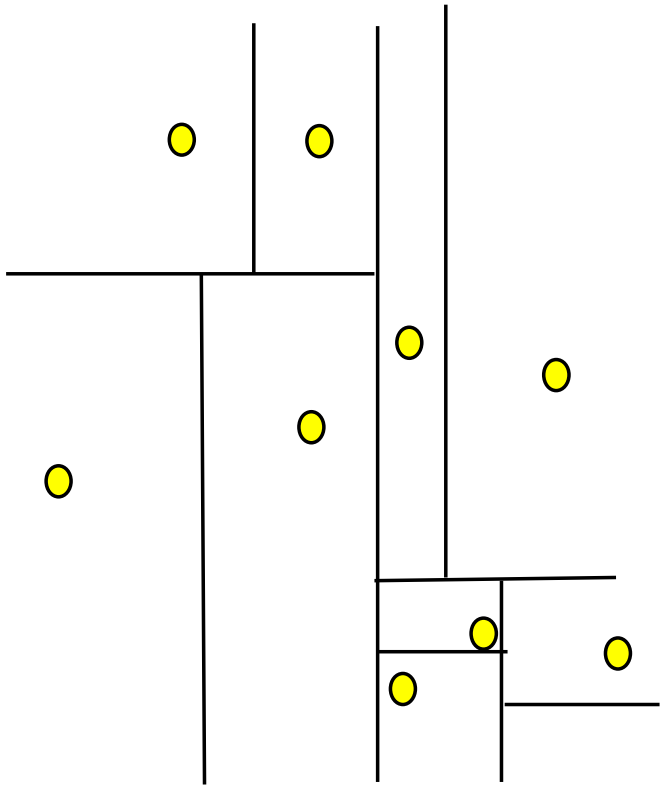
- What is the max number Q(n) of regions in an n-point kd-tree intersecting (say, vertical) line ?
  - If we split on x, Q(n)=1+Q(n/2)
  - If we split on y, Q(n)=2*Q(n/2)+2
  - Since we alternate, we can write Q(n)=3+2Q(n/4)
- This solves to O(sqrt{n})

Approximate Nearest Neighbor (ANN)

- Definition:
  - Given: a set of points P in 2D
  - Goal: given a query point q, and eps>0, find a point p' whose distance to q is at most (1+eps) times the distance from q to its nearest neighbor
- We will "solve" the problem using kd-trees…
- …under the assumption that all leaf cells of the kd-tree for P have bounded aspect ratio
- Assumption somewhat strict, but satisfied in practice for most of the leaf cells
- We will show
  - $O(\log n/eps^2)$ query time
  - O(n) space (inherited from kd-tree)

ANN Query Procedure

- Locate the leaf cell containing q
- Enumerate all leaf cells C in the increasing order of distance from q (denote it by r)
  - Let p(C) be the point in C
  - Update p' so that it is the closest point seen so far
- Stop when dist(q,p')<(1+eps)*r



Analysis

- Correctness:
  - We have touched all cells within distance r from q. Thus, if there is a point within distance r from q, we already found it
  - If there is no such point, then the p' provides a (1+eps)-approximate solution
- Running time:
  - All cells C seen so far (except maybe for the last one) have diameter > eps*r
  - …Because if not, then p(C) would have been a (1+eps)-approximate nearest neighbor, and we would have stopped
  - The number of cells with diameter eps*r, bounded aspect ratio, and touching a ball of radius r is at most $O(1/eps^2)$

References

- B-trees: “Introduction to Algorithms”, Cormen, Leiserson, Rivest, Stein, 2nd edition.
- Kd-trees: “Computational Geometry”, M. de Berg, M. van Kreveld, M. Overmars, O, Schwarzkopf. Chapter 5
- Approximate Nearest Neighbor (general algorithm without the bounded ratio assumption): Arya et al, ``An optimal algorithm for approximate nearest neighbor searching," *Journal of the ACM*, 45 (1998), 891-923. For implementation, see http://www.cs.umd.edu/~mount/ANN/