CS 261: Data Structures

Week 6: Binary search

Lecture 6b: Optimal and self-adjusting trees

David Eppstein
University of California, Irvine

Spring Quarter, 2023

Static optimality

Static optimality

Suppose we know the frequencies p_i
of each search outcome (external node x_i)

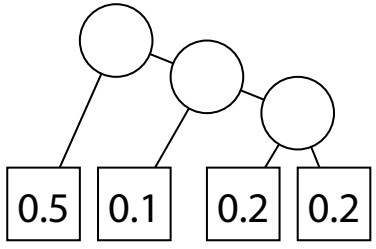
Then quality of a tree = average length of search path

$$= \sum_i p_i \times (\text{length of path to } x_i)$$

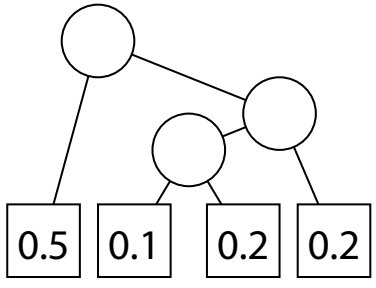
Uniformly balanced tree might not have minimum average length!

Example

With external node frequencies 0.5, 0.1, 0.2, 0.2:

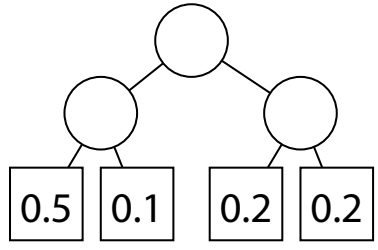


average height = 1.9
(1 x 0.5 + 2 x 0.1 + 3 x 0.2 + 3 x 0.2)

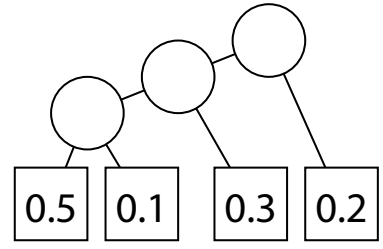


average height = 1.8
(1 x 0.5 + 3 x 0.1 + 3 x 0.2 + 2 x 0.2)

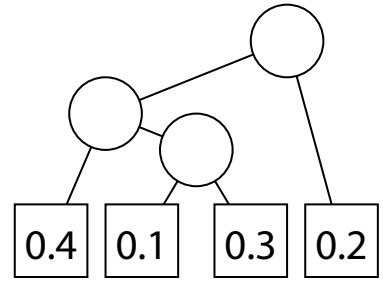
Optimal!



average height = 2
(2 x 0.5 + 2 x 0.1 + 2 x 0.3 + 2 x 0.1)



average height = 2.4
(3 x 0.5 + 3 x 0.1 + 2 x 0.2 + 1 x 0.2)



average height = 2.1
(2 x 0.5 + 3 x 0.1 + 3 x 0.2 + 1 x 0.2)

Dynamic program for optimal trees

For each subarray, in order by length:

- For each partition into two smaller subarrays:
  - Height = 1 + weighted average of subarray heights
- Choose partition giving smallest height
- Remember its height for later lookup

Optimal tree is given by best partition for full array,
and by recursive optimal choices for each subarray

Time for naive implementation: $O(n^3)$

Improved by Knuth 1971 to $O(n^2)$

Garsia–Wachs algorithm for optimal trees

Adriano Garsia and Michelle L. Wachs, 1977,
simplifying T. C. Hu and A. C. Tucker, 1971

Very rough sketch of algorithm:

- Add frequency values $+\infty$ at both ends of the sequence
- Use a dynamic balanced binary tree to implement a greedy algorithm that repeatedly finds the first consecutive triple of frequencies x, y, z with $x \leq z$, replaces x and y with $x + y$, and moves replacement earlier in the sequence
  (after rightmost earlier value that is $\geq x + y$)
- The tree formed by these replacements has optimal path lengths but is not a binary search tree (leaves are out of order); find a binary search tree with the same path lengths

Time is $O(n \log n)$

Self-adjusting dynamic trees

The main idea

When an operation follows a search path to node x, rotate x to the root of the tree so that the next search for it will be fast

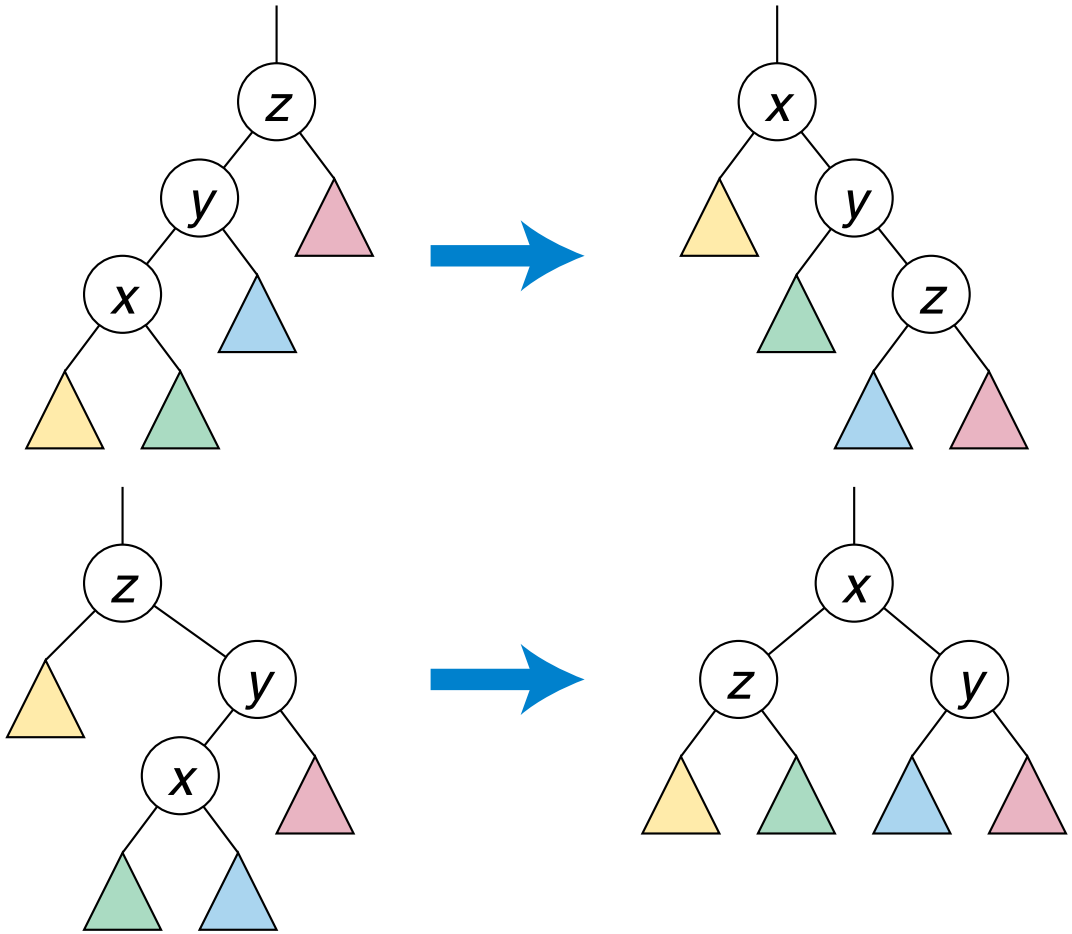
This operation is called "splaying"

Daniel Sleator and Robert Tarjan, 1985

Splay(x)

While x is not root:

If parent is root, rotate x and parent, else...



(and their mirror images)

Splay tree operations

Search

- Usual binary tree search (e.g. for successor)
- Splay the lowest interior node on the search path

Split into two subtrees at some key

- Splay the key
- Break link to its left child

Concatenate two subtrees

- Splay leftmost key in right subtree
- Add left subtree as its child

Add or remove item: split and concatenate

Simplifying assumptions for analysis

No insertions or deletions, only searches for members of an unchanging set of keys

- Deletion is similar to searching for the key and then not searching for it any more
- Insertion is similar to having a key in the initial set that you never searched for before
- Search for a missing key is similar to having another key where that key would be

We only need to analyze the time for a splay operation

- Actual time for search is bounded by time for splay

Amortized time for of weighted items

Suppose item x_i has weight $w_i > 0$, and let $W = \sum w_i$

For a node x_i with subtree T_i (including x_i and all its descendants), define rank $r_i = \lfloor \log_2 (\text{sum of weights of all nodes in } T_i) \rfloor$

Potential function Φ = sum of ranks of all nodes

Claim: The amortized time to splay x_i is $O(\log(W/w_i))$

Amortized analysis (sketch)

Main idea: look at the path from the previous root to x_i

Separate splay steps along path into two types:

- Steps where x and its grandparent z have different rank
- Steps where ranks of x and grandparent are equal

Rank at $x \geq \log_2 w_i$ and rank at root $\approx \log_2 W$ so number of different-rank steps is $O(\log(W/w_i))$

Each takes actual time $O(1)$ and can add $O(1)$ to Φ

There can be many equal-rank steps but each causes Φ to decrease (if rank is equal, most weight in grandparent's subtree is below x, so rotation causes parent or grandparent to decrease in rank)

Decrease in Φ cancels actual time for these steps

Consequences for different choices of weights

$O(\log(W/w_i))$ time is valid regardless of what the weights w_i are!
We can set w_i however we like; algorithm doesn't know or care

Uniform weights:

Set all $w_i = 1$

$$W = \sum w_i = n$$

$$W/w_i = n$$

Amortized time is $O(\log n)$

Consequences for different choices of weights

$O(\log(W/w_i))$ time is valid regardless of what the weights w_i are!

We can set w_i however we like; algorithm doesn't know or care

Optimal weights:

Let T be an optimal static binary tree

Set $w_i = 1/3^h$ where h is height of same node in T

$$W = \sum w_i = \sum_h \frac{\#\text{ nodes at height h}}{3^h} \leq \sum_{h=0}^{\infty} \frac{2^h}{3^h} = 3$$

$$W/w_i \leq 3^{h+1}$$

Amortized time is $O(\log 3^{h+1}) = O(h)$

Splay trees are as good as static optimal tree!

Consequences for different choices of weights

$O(\log(W/w_i))$ time is valid regardless of what the weights w_i are!

We can set w_i however we like; algorithm doesn't know or care

Weights from probabilities:

Suppose each search item is chosen randomly, independently of previous search items, with probability p_i of choosing key i

Set $w_i = p_i$

$W = 1$

Expected amortized time is $O(\sum p_i \log 1/p_i)$

This is the entropy of the distribution!

Consequences for different choices of weights

$O(\log(W/w_i))$ time is valid regardless of what the weights w_i are!

We can set w_i however we like; algorithm doesn't know or care

Weights from ranks:

Set weight of ith most frequently accessed item to $1/i^2$

$$W = \sum_{i=1}^{n} \frac{1}{i^2} \leq \sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}$$

$$\log W/w_i = O(\log i^2) = O(\log i)$$

Amortized time is $O(\log i)$

Consequences for different choices of weights

$O(\log(W/w_i))$ time is valid regardless of what the weights w_i are!

We can set w_i however we like; algorithm doesn't know or care

Weights from access times:

Set $w_i = 1/t_i^2$ where t_i = number of searches since last access

Weights are dynamic $\Rightarrow$ amortized analysis needs to include the change in potential caused by any change in weights

Weights change by increasing weight of (just-accessed) tree root, decreasing everything else $\Rightarrow$ change of weights cannot increase Φ

Amortized time is $O(\log t_i)$

Dynamic optimality

Dynamic optimality

But now suppose:

- We are adding and removing items as well as searching
- Different items are "hot" at different times

Maybe we can do better than a static tree?

(Idea: rearrange tree to move currently-hot items closer to root)

Competitive ratio

Question: How valuable is knowledge of the future?

Let A be any algorithm for handling a sequence S of dynamic requests, one at a time, without knowledge of future requests

Let OPT be an algorithm that can see the whole sequence of requests and then chooses optimally (somehow) what to do

Then the competitive ratio of A is

$$\max_S \frac{\text{cost of } A \text{ on sequence } S}{\text{cost of } OPT \text{ on sequence } S}$$

Dynamic optimality conjecture

Allow dynamic search trees to rearrange any contiguous subtree containing the root node, with cost per operation:

- Length of search paths for all operations, plus
- Sizes of all rearranged subtrees

Conjecture: There is a structure with competitive ratio $O(1)$

(I.e. it gets same O-notation as the best dynamic tree structure optimized for any specific input sequence)

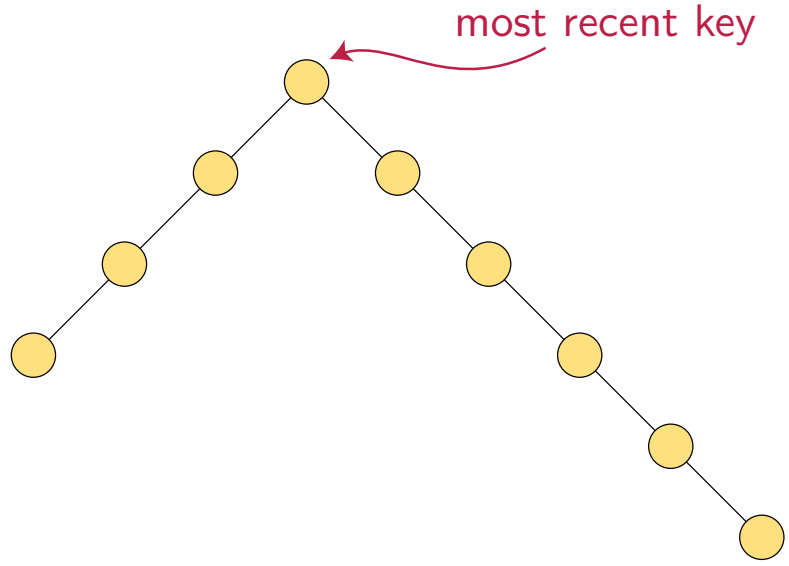
A simple example

For search sequences S where each search is previous ± 1:

Use a tree rooted at most recent search key, with two paths going left and right

For general searches this is a bad structure but for S it takes $O(1)$ per search (one rotation)

A competitive tree must also get $O(1)$ per search on S



Candidates for good competitive ratio

Splay trees

Conjectured to have competitive ratio $O(1)$

GreedyASS trees (next slides)

Conjectured to have competitive ratio $O(1)$

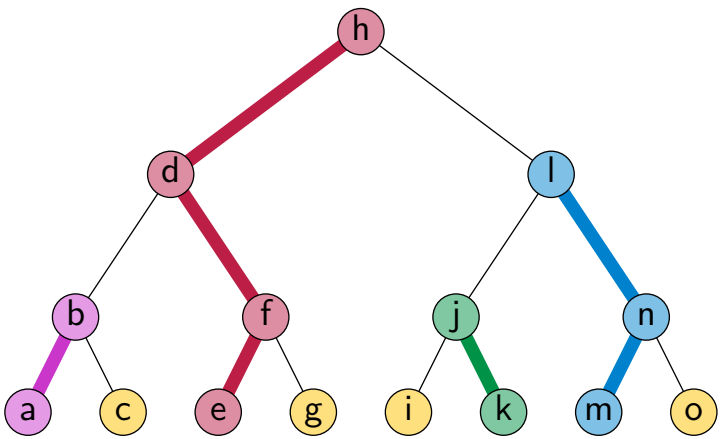
Tango trees (next slides)

Proven to have competitive ratio $O(\log \log n)$

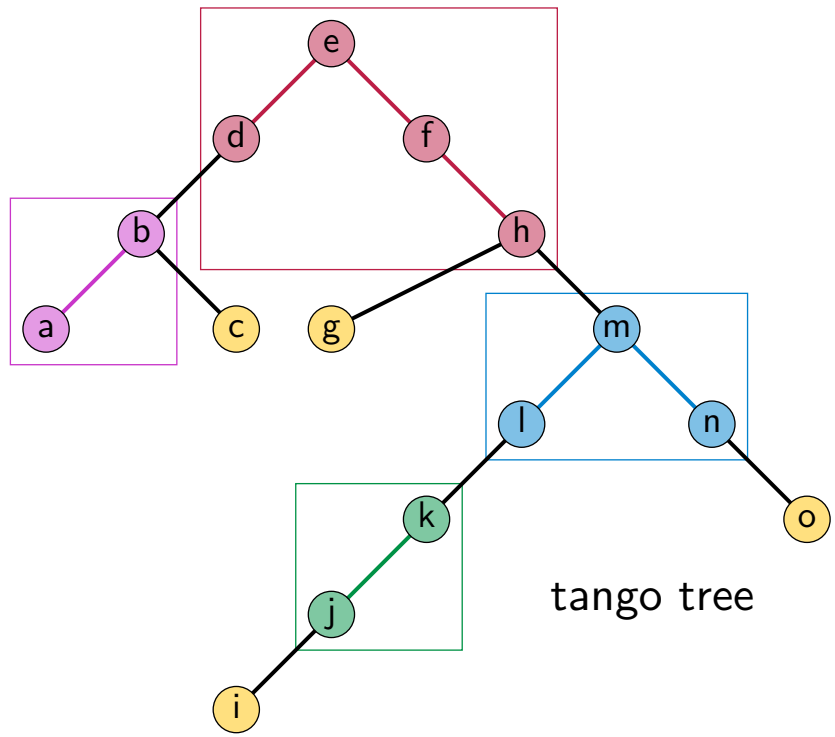
Tango trees

Consider a complete binary search tree on the keys ("reference tree")
+ "preferred paths" to most recently accessed descendant

Replace each preferred path by a balanced tree structure that can support cutting and linking operations (like a splay tree)



reference tree with paths to
recently accessed descendants

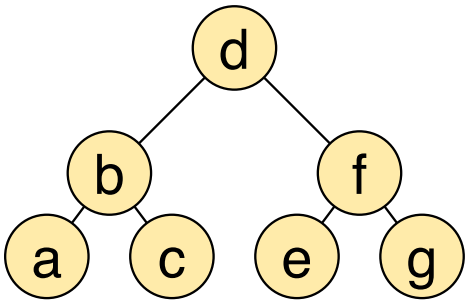


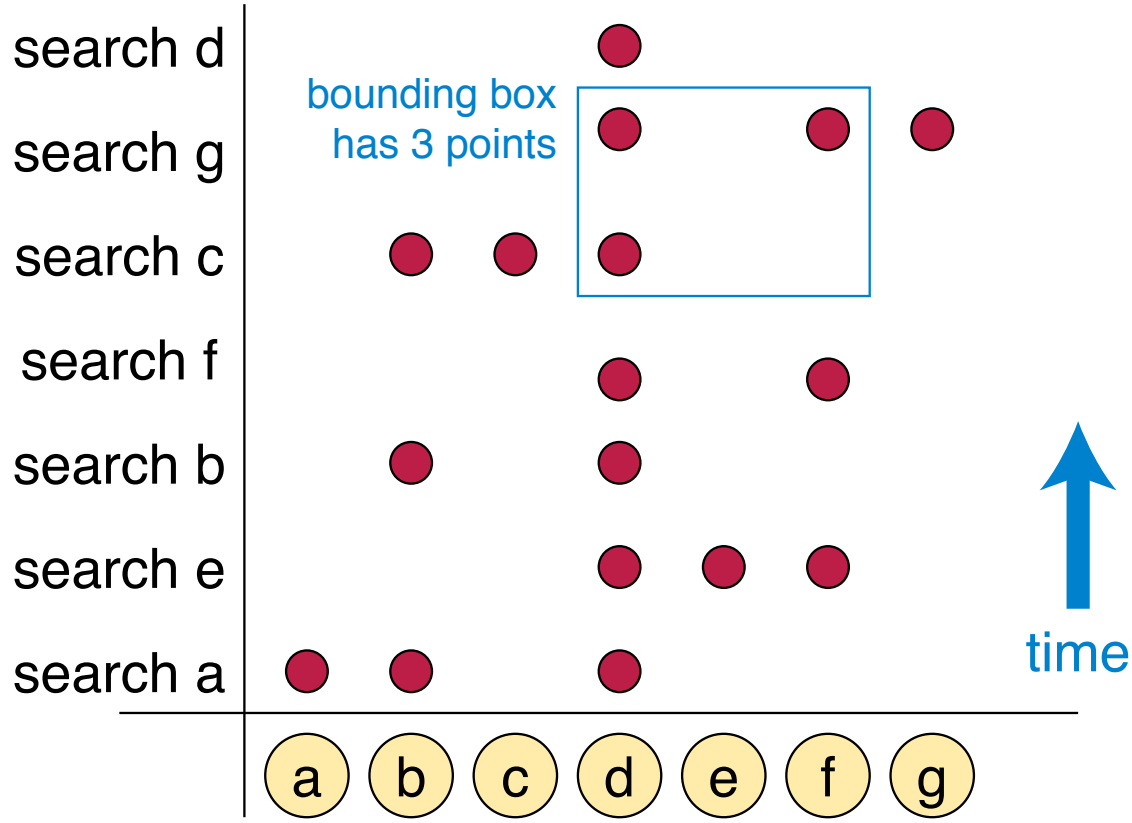
tango tree

Demaine, Harmon, Iacono, and Pătraşcu, 2004

The geometry of binary search trees

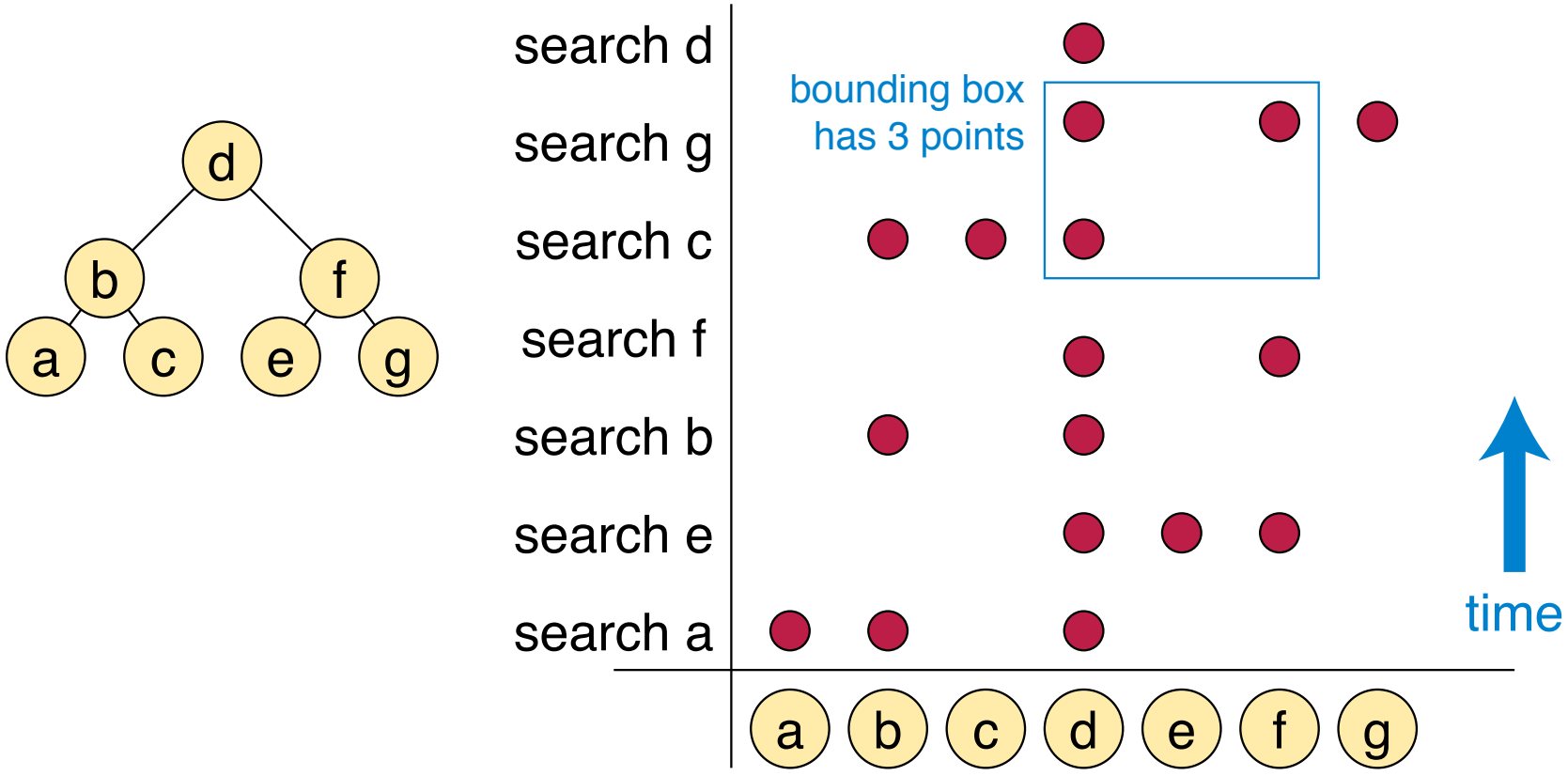
Given any (static or dynamic) binary search tree,
plot access to key i during operation j as a point (i, j)





Arborially satisfied sets

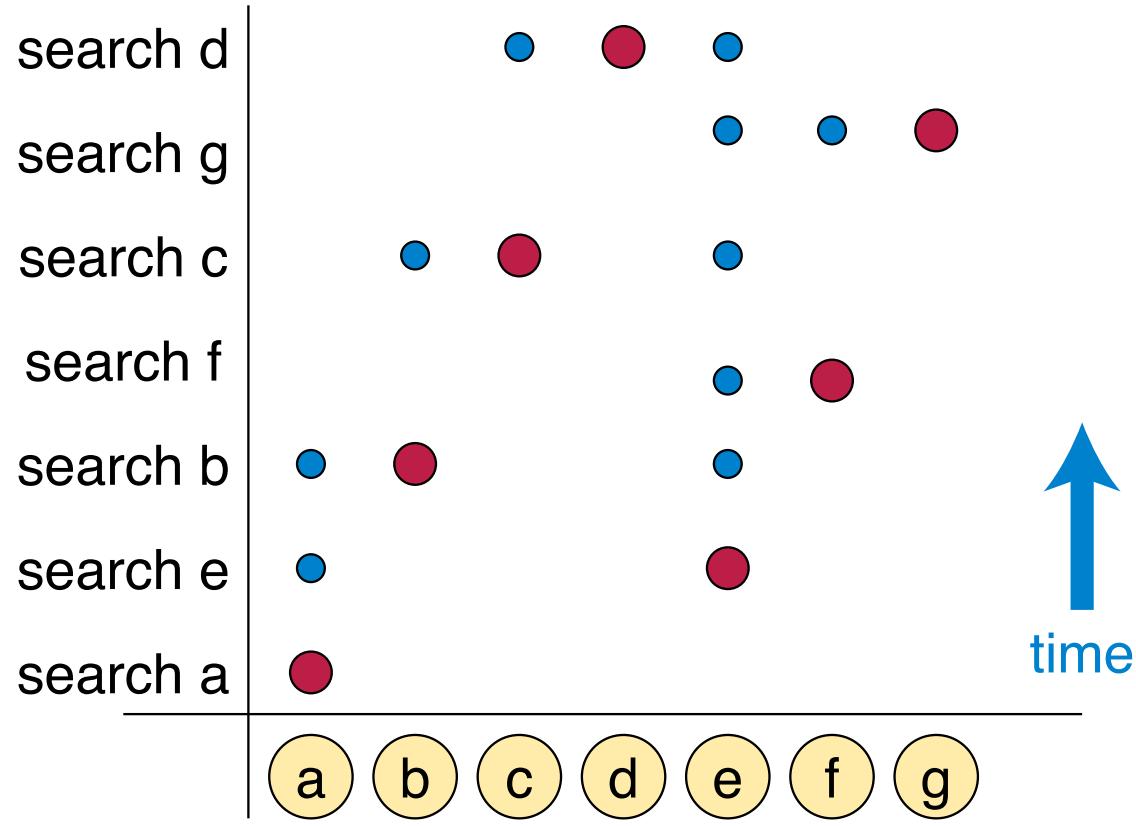
Key property: Every two points not both on same row or column have a bounding box containing at least one more point



Interpretation: if search reaches x, and later reaches y, it must pass through a common ancestor of both

Greedy arborially satisfied sets

In each row (bottom-up order) add the minimum number of extra points (blue) to make every bounding box have ≥ 3 points



Conjecture: uses $O(1) \times$ optimal # points

Can be turned into a dynamic tree algorithm (GreedyASS tree)

From geometry back to trees

Offline (if we know the future)

$\forall$ arborially satisfied set $\Rightarrow$ sequence of tree operations

Idea: Treap (a binary search tree that is heap-ordered by priorities), but replace random priority by next access time

Online

Can convert any row-by-row construction of arborially satisfied sets into a dynamic tree algorithm

Complicated

Greedy arborially satisfied set $\Rightarrow$ GreedyASS tree

Demaine, Harmon, Iacono, Kane, and Pătraşcu, 2009

Summary

Summary

- Hashing is usually a better choice for exact searches, but binary searching is useful for finding nearest neighbors, function interpolation, etc.
- Similar search algorithms work both for static data in sorted arrays and explicit tree structures
- Balanced trees: maintain log-height while being updated
- Many variations of balanced trees
- Static versus dynamic optimality
- Construction of static binary search trees
- Splay trees and their amortized analysis
- Static optimality of splay trees
- Dynamic optimality conjecture and competitive ratios