Data Structures 1
NTIN066

Jirka Fink

Department of Theoretical Computer Science and Mathematical Logic
Faculty of Mathematics and Physics
Charles University in Prague

Summer semester 2025/26
Last change on March 4, 2026

Experimental assignment

- Run a program available on gitlab to obtain results of experiments.
- Determine the asymptotic behavior of the measured data. Note that certain results may depend on multiple parameters; therefore, the dependence on all relevant parameters should be analyzed.
- State the mathematical theorems that theoretically justify the observed empirical behavior. Cite all your sources.
- Discuss the correspondence between theory and experiment, explaining how the theoretical results account for, or deviate from, the empirical measurements.
- It may be helpful to apply regression to substantiate the asymptotic estimates. It is advisable to compare the outcomes of regression across different candidate functions. Any suitable mathematical software may be used to perform the regression analysis.
- Submit your report in PDF format.

Statically Optimal Binary Search Trees (SOBST)

Problem Setting

- We are given a set of keys $x_1 < x_2 < \cdots < x_n$.
- Each key x_i is accessed with frequency $w_i > 0$.
- Goal: construct a binary search tree T minimizing the total search cost

$$\sum_{i=1}^{n} w_i \cdot \mathrm{depth}_T(x_i),$$

where $\mathrm{depth}_T(x_i)$ is the depth of x_i in T.

Statically Optimal Binary Search Trees (SOBST)

Problem Setting

- We are given a set of keys $x_1 < x_2 < \cdots < x_n$.
- Each key x_i is accessed with frequency $w_i > 0$.
- Goal: construct a binary search tree T minimizing the total search cost

$$\sum_{i=1}^{n} w_i \cdot \mathrm{depth}_T(x_i),$$

where $\mathrm{depth}_T(x_i)$ is the depth of x_i in T.

Observation

If T is SOBST, then the subtree of every node in T is SOBST.

Dynamic Programming Construction

Subproblem

Let $C(i,j)$ denote the minimum cost of a subtree containing keys $x_i, \ldots, x_j$.

Dynamic Programming Construction

Subproblem

Let $C(i, j)$ denote the minimum cost of a subtree containing keys $x_i, \dots, x_j$.

Recurrence

If r is chosen as the root ($i \leq r \leq j$), then

$$C(i,j) = \min_{r=i,\dots,j} \left(C(i, r-1) + C(r+1, j)\right) + \sum_{k=i}^{j} w_k.$$

Dynamic Programming Construction

Subproblem

Let $C(i,j)$ denote the minimum cost of a subtree containing keys $x_i, \dots, x_j$.

Recurrence

If r is chosen as the root ($i \leq r \leq j$), then

$$C(i,j) = \min_{r=i,\dots,j} \left(C(i,r-1) + C(r+1,j)\right) + \sum_{k=i}^{j} w_k.$$

Complexity

- There are $\mathcal{O}\left(n^2\right)$ subproblems.
- Each requires $\mathcal{O}(n)$ choices of r.
- Total running time: $\mathcal{O}\left(n^3\right)$.
- With prefix sums and optimization: $\mathcal{O}\left(n^2\right)$.

Interval Queries Using Splay Trees

Problem

Given a splay tree T and two bounds a and b, find all keys between a and b in T.

Interval Queries Using Splay Trees

Problem

Given a splay tree T and two bounds a and b, find all keys between a and b in T.

Idea

Find and splay the smallest element a' greater than a in T, and then find and splay the largest element b' smaller than b.

Interval Queries Using Splay Trees

Problem

Given a splay tree T and two bounds a and b, find all keys between a and b in T.

Idea

Find and splay the smallest element a' greater than a in T, and then find and splay the largest element b' smaller than b.

Questions

- After splaying a' and b', is a' always the left child of b'?

Interval Queries Using Splay Trees

Problem

Given a splay tree T and two bounds a and b, find all keys between a and b in T.

Idea

Find and splay the smallest element a' greater than a in T, and then find and splay the largest element b' smaller than b.

Questions

- After splaying a' and b', is a' always the left child of b'?
- Where are all elements in the interval $[a, b]$ stored?

Interval Queries Using Splay Trees

Problem

Given a splay tree T and two bounds a and b, find all keys between a and b in T.

Idea

Find and splay the smallest element a' greater than a in T, and then find and splay the largest element b' smaller than b.

Questions

- After splaying a' and b', is a' always the left child of b'?
- Where are all elements in the interval $[a, b]$ stored?
- What is the time complexity of this interval query?

Find the k-th Smallest Element

Problem

Given a splay tree T and an integer k, find the k-th smallest element in T.

Find the k-th Smallest Element

Problem

Given a splay tree T and an integer k, find the k-th smallest element in T.

Questions

- Can we efficiently find the k-th smallest element if no additional information is stored in T?

Find the k-th Smallest Element

Problem

Given a splay tree T and an integer k, find the k-th smallest element in T.

Questions

- Can we efficiently find the k-th smallest element if no additional information is stored in T?
- What additional information needs to be stored in T?

Find the k-th Smallest Element

Problem

Given a splay tree T and an integer k, find the k-th smallest element in T.

Questions

- Can we efficiently find the k-th smallest element if no additional information is stored in T?
- What additional information needs to be stored in T?
- What is the time complexity of this operation?

Find the k-th Smallest Element

Problem

Given a splay tree T and an integer k, find the k-th smallest element in T.

Questions

- Can we efficiently find the k-th smallest element if no additional information is stored in T?
- What additional information needs to be stored in T?
- What is the time complexity of this operation?
- How can this additional information be updated during the operations SPLAY, INSERT, and DELETE?

Splay tree: Operations INSERT and DELETE

Procedure

1. Insert/delete the given node as in a BST
2. SPLAY the deepest visited node

Splay tree: Operations INSERT and DELETE

Procedure

1. Insert/delete the given node as in a BST
2. SPLAY the deepest visited node

Amortized complexity

- Finding the required nodes has the same complexity as SPLAY: $\mathcal{O}(\log n)$ amortized
- The actual deletion of a node takes $\mathcal{O}(1)$ time and the potential of the tree decreases
- When adding a leaf, the potential of the nodes $u_1, \dots, u_h$ on the path to the root increases by $\sum_{i=1}^{h} \log(s(u_i) + 1) - \log(s(u_i)) \leq \log(n)$
- The amortized complexity of the operations INSERT and DELETE is $\mathcal{O}(\log n)$

Properties of splay trees

Theorem (searching a subset of elements)

If we perform k searches for elements from a subset of size m, then the total cost is $\mathcal{O}(n \log n + k + k \log m)$.

Properties of splay trees

Theorem (searching a subset of elements)

If we perform k searches for elements from a subset of size m, then the total cost is $\mathcal{O}(n \log n + k + k \log m)$.

Theorem (working set)

Let $x_1, \dots, x_k$ be a sequence of searched elements, and let z_i denote the number of distinct elements searched for since the previous search of the element x_i. The total cost of searching for the elements $x_1, \dots, x_k$ is $\mathcal{O}\left(n \log n + k + \sum_i \log(1 + z_i)\right)$.

Properties of splay trees

Theorem (searching a subset of elements)

If we perform k searches for elements from a subset of size m, then the total cost is $\mathcal{O}(n \log n + k + k \log m)$.

Theorem (working set)

Let $x_1, \dots, x_k$ be a sequence of searched elements, and let z_i denote the number of distinct elements searched for since the previous search of the element x_i. The total cost of searching for the elements $x_1, \dots, x_k$ is $\mathcal{O}\big(n \log n + k + \sum_i \log(1 + z_i)\big)$.

Example

Search sequence	a	b	c	a	c	b	b	a	b
Values z_i	0	1	2	2	1	2	0	2	1

Properties of splay trees

Theorem (searching a subset of elements)

If we perform k searches for elements from a subset of size m, then the total cost is $\mathcal{O}(n \log n + k + k \log m)$.

Theorem (working set)

Let $x_1, \dots, x_k$ be a sequence of searched elements, and let z_i denote the number of distinct elements searched for since the previous search of the element x_i. The total cost of searching for the elements $x_1, \dots, x_k$ is $\mathcal{O}\big(n \log n + k + \sum_i \log(1 + z_i)\big)$.

Example

Search sequence	a	b	c	a	c	b	b	a	b
Values z_i	0	1	2	2	1	2	0	2	1

Theorem (searching elements in increasing order)

If the search sequence S contains elements in increasing order, then the total time required to search for S in a splay tree is $\mathcal{O}(n)$.

Comparison of Statically Optimal Trees and Splay Trees

Questions

- Does there exist a search sequence on which a splay tree is asymptotically faster than a statically optimal tree constructed for that sequence?
- Does there exist a search sequence on which a statically optimal tree is asymptotically faster than a splay tree?

Comparison of Statically Optimal Trees and Splay Trees

Questions

- Does there exist a search sequence on which a splay tree is asymptotically faster than a statically optimal tree constructed for that sequence?
- Does there exist a search sequence on which a statically optimal tree is asymptotically faster than a splay tree?

Theorem (Static Optimality)

Let $x_1, \dots, x_k$ be a sequence of searches for elements from a set X, where every element of X is searched for at least once. Let T be a static tree on X, and suppose the total cost of searching for the elements in the sequence in T is f. Then the total cost of performing the searches in a splay tree is $\mathcal{O}(f)$.

Dynamic Optimality of Splay Trees

Are double rotations in Splay trees the best possible?

Dynamic Optimality of Splay Trees

Are double rotations in Splay trees the best possible?

Informally, a binary search tree algorithm is said to be *dynamically optimal* if for every sufficiently long sequence of accesses, it performs within a constant factor of the cost of the best possible dynamic BST algorithm (including offline algorithms that can restructure the tree between accesses).
The *Dynamic Optimality Conjecture* asserts that splay trees are dynamically optimal.

Dynamic Optimality of Splay Trees

Are double rotations in Splay trees the best possible?

Informally, a binary search tree algorithm is said to be *dynamically optimal* if for every sufficiently long sequence of accesses, it performs within a constant factor of the cost of the best possible dynamic BST algorithm (including offline algorithms that can restructure the tree between accesses).
The *Dynamic Optimality Conjecture* asserts that splay trees are dynamically optimal.

Dynamic Optimality Conjecture

Let A be any binary search tree algorithm that accesses an element x by traversing the path from the root to x at a cost of $d_x + 1$, and that between accesses can make any rotations in the tree at a cost of 1 per rotation where d_x is the depth of x. Let $A(S)$ be the cost for A to perform the sequence S of accesses. Then the cost for a splay tree to perform the same accesses is $\mathcal{O}(n + A(S))$.