Data Structures 1

NTIN066

Jirka Fink

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

Winter semester 2023/24

Last change on February 23, 2024

Content

- Search trees (BB[α]-tree, Splay tree, B-tree)
- Cache-oblivious algorithms
- Hashing
- Suffix array
- Geometric data structures
- Parallel data structures

Assignments

Overview

- There are at least 7 programming assignments per 10 points
- and at least 3 experimental assignments per 15 points
- You need 75 points for class credit
- Deadline: 2 weeks

Programming assignments

- You are given a partial implementation of a DS
- Implement the missing bits
- Automatic checking, tests are public
- Instructor looks at the source code
- C++ and (usually) Python available

Programming assignments

- Measure properties of a given implementation
- Write a report (and submit PDF)

General rules

- Do not share code nor reports (except with the instructor).
- Do not share example solutions with anybody.
- Deadlines are strict.
- Before deadline, you can re-submit.
- The code must pass all tests.
- Quality of your code and reports contributes to grading.
- Do not use non-trivial code you didn't write yourself. This includes other peoples' implementations and non-obvious library functions. Trivial cases of growing arrays (appending to std::vector in C++ or list.append() in Python) are permitted, anything more complicated isn't. When in doubt, ask your instructor.
- All theorems used in your reports must be stated in full and their source must be properly cited. If the theorem was stated at the lecture, citing the lecture is considered sufficient.

Requisite knowledge

- Programming (Python or C++)
- Basic algorithms and data structures: e.g., balanced search trees (AVL/red-black/. . .)
- Discrete math (combinatorics, basic number theory)
- Basic probability theory (linearity of expectation, . . .)
- Computer architecture

Organization

Web

https://ktiml.mff.cuni.cz/~fink/

E-mail

fink@ktiml.mff.cuni.cz

GIT

- Problem statements
- Templates to be filled
- `https://gitlab.kam.mff.cuni.cz/datovky/assignments`

Recodex

- Submissions and unit tests
- Comments to your solutions
- `https://recodex.mff.cuni.cz`

Complexity

What is the time complexity of the following algorithms or operations

- Add an element to the end of a linked list
- Find an element in a linked list
- Find an element in a sorted array
- Find the smallest element in a sorted array
- Sort an array of elements
- Determine the number of components of a graph
- Find a shortest path in a graph
- Find a cycle visiting all vertices in a graph

Amortized complexity

Motivation

- Consider a data structure which is usually very fast
- However in rare cases, it needs to reorganize its internal structure
- So, the worst-case complexity is quite slow
- This data structure may be used by an algorithm
- We are interested in the total complexity or average complexity of many operations

Problem description (Stack version)

- We have an array of length p storing n elements, and we need to implement operations Insert and Delete
- If $n = p$ and an element has to be inserted, then the length of the array is doubled
- If $4n = p$ and an element has to be deleted, then the length of the array is halved
- What is the number of copied elements during k operations Insert and Delete?

Aggregated analysis

- Let k_i be the number of operations between $(i-1)$-th and i-th reallocation
- The first reallocation copies at most $n_0 + k_1$ elements where n_0 is the initial number of elements
- The i-th reallocation copies at most $2k_i$ elements for $i \geq 2$
- Every operation without reallocation copies at most 1 element
- The total number of copied elements is at most $k + (n_0 + k_1) + \sum_{i \geq 2} 2k_i \leq n_0 + 3k$

Potential method

- Consider the potential

$$\Phi = \begin{cases} 0 & \text{if } p = 2n \\ n & \text{if } p = n \\ n & \text{if } p = 4n \end{cases}$$

  and piece-wise linear function in other cases
- Explicitly,

$$\Phi = \begin{cases} 2n - p & \text{if } p \leq 2n \\ p/2 - n & \text{if } p \geq 2n \end{cases}$$

- Change of the potential without reallocation is $\Phi_i - \Phi_{i-1} \leq 2$ ①
- Let T_i be the number of elements copied during i-th operation
- Hence, $T_i + \Phi_i - \Phi_{i-1} \leq 3$
- The total number of copied elements during k operations is $\sum_{i=1}^{k} T_i \leq 3k + \Phi_0 - \Phi_k \leq 3k + n_0$

1

$$\Phi' - \Phi = \begin{cases} 2 & \text{Insert and } p \leq 2n \\ -2 & \text{Delete and } p \leq 2n \\ -1 & \text{Insert and } p \geq 2n \\ 1 & \text{Delete and } p \geq 2n \end{cases}$$

Average of the aggregated analysis

- The amortized complexity of an operation is the total time of k operations over k assuming that k is sufficiently large.
- For example, the amortized complexity of operations Insert and Delete in the dynamic array is $\frac{\sum_{i=1}^{k} T_i}{k} \leq \frac{3k+n_0}{k} \leq 4 = \mathcal{O}(1)$ assuming that $k \geq n_0$.

Potential method

- Let Φ a potential which evaluates the internal representation of a data structure
- Let T_i be the actual time complexity of i-th operation
- Let Φ_i be the potential after i-th operation
- The amortized complexity of the operation is $\mathcal{O}(f(n))$ if $T_i + \Phi_i - \Phi_{i-1} \leq f(n)$ for every operation i in an arbitrary sequence of operations
- For example in dynamic array, $T_i + \Phi_i - \Phi_{i-1} \leq 3$, so the amortized complexity of operations Insert and Delete is $\mathcal{O}(1)$

Why dynamic array is doubling the size?

Consider that an array is full, and it has n elements.
What happens when we change its size to be

- $3n$
- $n + 10$
- n^2?

Queue

- Propose an efficient implementation of a queue.
- You have two stacks, supporting only the POP and PUSH operations. Propose an algorithm, that would simulate a Queue with operations ENQUEUE and DEQUEUE. Besides the two stacks, you have only a constant amount of memory. Show that the queue operations have a constant amortized time complexity.

```
class Queue:
  def __init__(self):
    self.input = [] # For elements that are enqueued
    self.output = [] # For elements that will be dequeued

  def enqueue(self, element):
    self.input.append(element)

  def dequeue(self):
    if not self.output: # If the output list is empty
      while self.input: # While the input list is not empty
        self.output.append(self.input.pop())
    return self.output.pop()
```

More details on Wikipedia

Dequeue

Propose an efficient implementation of a dequeue; e.i. an array that allows inserting, and removing elements from both ends.

1st assignment: Successor in a binary tree

- Successor of a node A is the smallest node larger than A
- Given a tree and its node, find the successor
- If the given node is the last one, return None
- If the given node is None, return the first node
- When whole tree is traversed by your program, the total time complexity has to be $O(n)$