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 February 16, 2026

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

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

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

Experimental assignments

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

Organization

Web

https://ktiml.mff.cuni.cz/˜fink/

Organization

Web

https://ktiml.mff.cuni.cz/˜fink/

E-mail

fink@ktiml.mff.cuni.cz

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`

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`

General rules

- Do not share code nor reports (except with the instructor).
- 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. This also applies for theorems formulated by AI where citing AI is not considered as a proper citation, and a student is responsible for finding the original source.
- Using AI to prepare solutions is allowed but students are fully responsible for submitted solutions. Students must completely understand all details of their solutions, so students have to be able to explain their solutions when asked by a teacher.
- Details at `https://gitlab.kam.mff.cuni.cz/datovky/assignments`

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

Complexity

What is the time complexity of the following algorithms or operations

- Add an element to the end of a linked list

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

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

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

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

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

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

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

Heap

Definition

Heap

Definition

- A heap is a binary tree data structure.
- In all levels except the last two, every node has both children.
- The last level is filled from left.
- It satisfies the heap property:
  - In a max-heap, for any given node *u*, the value of *u* is greater than or equal to the values of its children.
  - In a min-heap, the value of *u* is less than or equal to the values of its children.

Heap

Definition

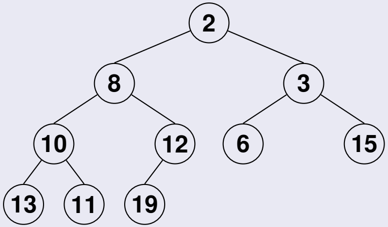
- A heap is a binary tree data structure.
- In all levels except the last two, every node has both children.
- The last level is filled from left.
- It satisfies the heap property:
  - In a max-heap, for any given node u, the value of u is greater than or equal to the values of its children.
  - In a min-heap, the value of u is less than or equal to the values of its children.

Applications

- Priority Queues
- Heap Sort
- Graph Algorithms (e.g., Dijkstra's Algorithm)
- Memory Management

Heaps in arrays

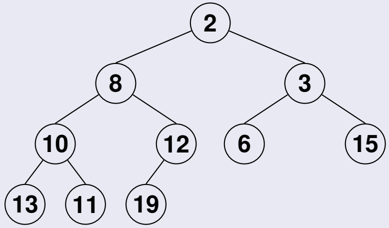
Min-heap stored in a tree



Min-heap stored in an array

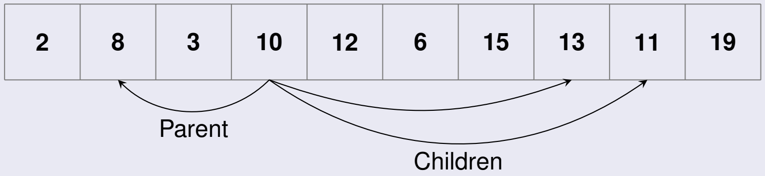
Heaps in arrays

Min-heap stored in a tree



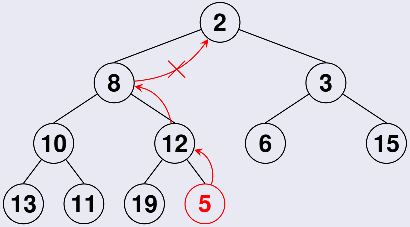
Min-heap stored in an array

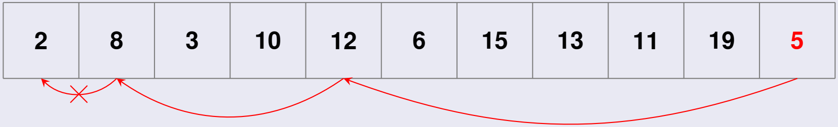
An element on the index i has its parent at $\lfloor (i-1)/2 \rfloor$ and children at $2i+1$ and $2i+2$:



Min-heap: Operation INSERT

Insert a new element 5

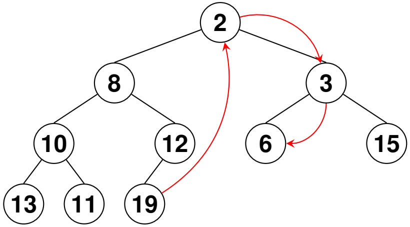




2	8	3	10	12	6	15	13	11	19	5

What is time complexity of operation INSERT?

Operation DELETEMIN



What is time complexity of operation DELETEMIN?

d-regular heaps

Definition

d-regular heaps

Definition

- **Each node has at most d children**
- In all levels except the last two, every node has both children.
- The last level is filled from left.
- It satisfies the heap property:
  - In a max-heap, for any given node u, the value of u is greater than or equal to the values of its children.
  - In a min-heap, the value of u is less than or equal to the values of its children.

d-regular heaps

Definition

- **Each node has at most d children**
- In all levels except the last two, every node has both children.
- The last level is filled from left.
- It satisfies the heap property:
  - In a max-heap, for any given node u, the value of u is greater than or equal to the values of its children.
  - In a min-heap, the value of u is less than or equal to the values of its children.

Exercise

Modify operations INSERT and DELETEMIN for d-regular heaps, and determine their complexity.

Dijkstra’s Algorithm

Dijkstra's Algorithm

```
   Input: A graph $G$ with edge lengths $c \geq 0$, a source vertex $s$, and a target vertex $t$
1  Mark all vertices as unvisited
2  Mark the source vertex as visited
3  For every vertex $u$ set $d[u] \leftarrow \infty$, except $d[s] \leftarrow 0$
4  while there exists a visited vertex and the target vertex has not been explored do
5      $u \leftarrow$ the visited vertex with the smallest value of $d[u]$
6      for $v$ a neighbor of $u$ do
7          if $d[v] > d[u] + c(u, v)$ then
8              $d[v] \leftarrow d[u] + c(u, v)$
9              Mark $v$ as visited
10     Mark $u$ as explored
```

Dijkstra's Algorithm

```
Input: A graph $G$ with edge lengths $c \geq 0$, a source vertex $s$, and a target vertex $t$
1  Mark all vertices as unvisited
2  Mark the source vertex as visited
3  For every vertex $u$ set $d[u] \leftarrow \infty$, except $d[s] \leftarrow 0$
4  while there exists a visited vertex and the target vertex has not been explored do
5      $u \leftarrow$ the visited vertex with the smallest value of $d[u]$
6      for $v$ a neighbor of $u$ do
7          if $d[v] > d[u] + c(u, v)$ then
8              $d[v] \leftarrow d[u] + c(u, v)$
9              Mark $v$ as visited
10     Mark $u$ as explored
```

Remarks

- The value $d[u]$ is the length of the shortest currently known path from s to u

Dijkstra's Algorithm

```
Input: A graph $G$ with edge lengths $c \geq 0$, a source vertex $s$, and a target vertex $t$
1  Mark all vertices as unvisited
2  Mark the source vertex as visited
3  For every vertex $u$ set $d[u] \leftarrow \infty$, except $d[s] \leftarrow 0$
4  while there exists a visited vertex and the target vertex has not been explored do
5      $u \leftarrow$ the visited vertex with the smallest value of $d[u]$
6      for $v$ a neighbor of $u$ do
7          if $d[v] > d[u] + c(u, v)$ then
8              $d[v] \leftarrow d[u] + c(u, v)$
9              Mark $v$ as visited
10     Mark $u$ as explored
```

Remarks

- The value $d[u]$ is the length of the shortest currently known path from s to u
- If u is explored, then $d[u]$ equals the length of the shortest path from s to u

Dijkstra's Algorithm

```
   Input: A graph $G$ with edge lengths $c \geq 0$, a source vertex $s$, and a target vertex $t$
1  Mark all vertices as unvisited
2  Mark the source vertex as visited
3  For every vertex $u$ set $d[u] \leftarrow \infty$, except $d[s] \leftarrow 0$
4  while there exists a visited vertex and the target vertex has not been explored do
5      $u \leftarrow$ the visited vertex with the smallest value of $d[u]$
6      for $v$ a neighbor of $u$ do
7          if $d[v] > d[u] + c(u, v)$ then
8              $d[v] \leftarrow d[u] + c(u, v)$
9              Mark $v$ as visited
10     Mark $u$ as explored
```

Remarks

- The value $d[u]$ is the length of the shortest currently known path from s to u
- If u is explored, then $d[u]$ equals the length of the shortest path from s to u
- Vertices are marked as explored in nondecreasing order of their distance from s

Dijkstra's Algorithm

```
Input: A graph $G$ with edge lengths $c \geq 0$, a source vertex $s$, and a target vertex $t$
1  Mark all vertices as unvisited
2  Mark the source vertex as visited
3  For every vertex $u$ set $d[u] \leftarrow \infty$, except $d[s] \leftarrow 0$
4  while there exists a visited vertex and the target vertex has not been explored do
5      $u \leftarrow$ the visited vertex with the smallest value of $d[u]$
6      for $v$ a neighbor of $u$ do
7          if $d[v] > d[u] + c(u, v)$ then
8              $d[v] \leftarrow d[u] + c(u, v)$
9              Mark $v$ as visited
10     Mark $u$ as explored
```

Remarks

- The value $d[u]$ is the length of the shortest currently known path from s to u
- If u is explored, then $d[u]$ equals the length of the shortest path from s to u
- Vertices are marked as explored in nondecreasing order of their distance from s
- Visualization: `https://qiao.github.io/PathFinding.js/visual/`

Dijkstra's Algorithm: Complexity

- Excluding the time required to select the visited vertex u with the minimum value $d[u]$, what is the overall time complexity?

Dijkstra's Algorithm: Complexity

- Excluding the time required to select the visited vertex u with the minimum value $d[u]$, what is the overall time complexity?
- By which data structure can the vertex u be selected efficiently?

Dijkstra’s Algorithm: Complexity

- Excluding the time required to select the visited vertex u with the minimum value $d[u]$, what is the overall time complexity?
- By which data structure can the vertex u be selected efficiently?
- How many times are the operations INSERT and DELETEMIN invoked during the execution of the algorithm?

Dijkstra’s Algorithm: Complexity

- Excluding the time required to select the visited vertex u with the minimum value $d[u]$, what is the overall time complexity?
- By which data structure can the vertex u be selected efficiently?
- How many times are the operations INSERT and DELETEMIN invoked during the execution of the algorithm?
- When a binary heap is employed, what is the resulting time complexity of Dijkstra’s algorithm?

Dijkstra’s Algorithm: Complexity

- Excluding the time required to select the visited vertex u with the minimum value $d[u]$, what is the overall time complexity?
- By which data structure can the vertex u be selected efficiently?
- How many times are the operations INSERT and DELETEMIN invoked during the execution of the algorithm?
- When a binary heap is employed, what is the resulting time complexity of Dijkstra’s algorithm?
- How does the time complexity change if a d-regular heap is used instead?

Dijkstra's Algorithm: Complexity

- Excluding the time required to select the visited vertex u with the minimum value $d[u]$, what is the overall time complexity?
- By which data structure can the vertex u be selected efficiently?
- How many times are the operations INSERT and DELETEMIN invoked during the execution of the algorithm?
- When a binary heap is employed, what is the resulting time complexity of Dijkstra's algorithm?
- How does the time complexity change if a d-regular heap is used instead?
- For which value of d is the time complexity minimized?