DATA STRUCTURES

Heap applications. Exam details.

Lect. PhD. Diana-Lucia Miholca

2022 - 2023

 Babeş - Bolyai University
Faculty of Mathematics and Computer Science

In Lecture 12...

- Heap
- ADT Priority Queue

Today

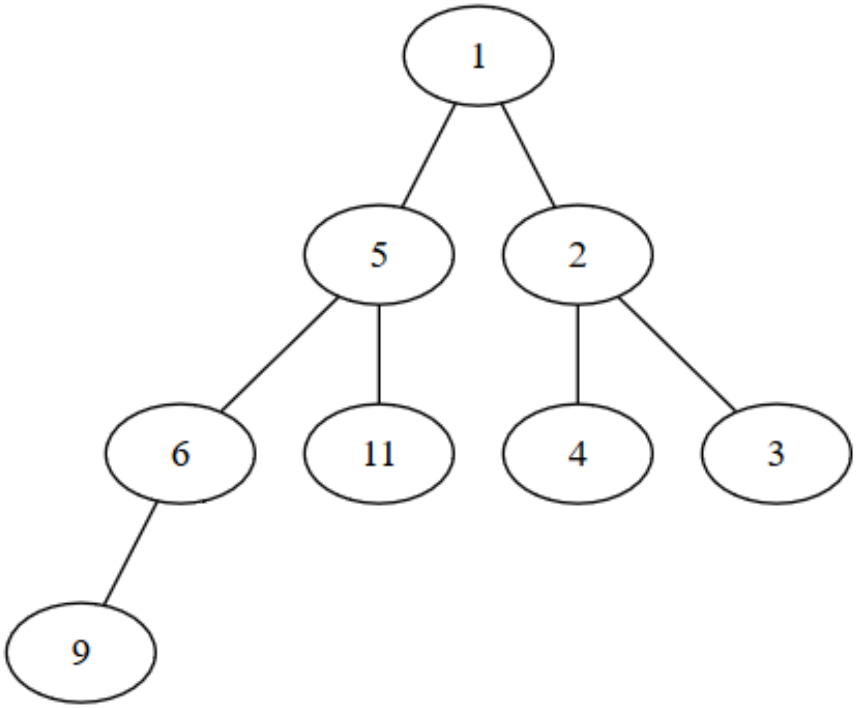
- Heap applications
- Exam details

Heap-sort - Naive approach

- The initial sequence: [6, 1, 3, 9, 11, 4, 2, 5]

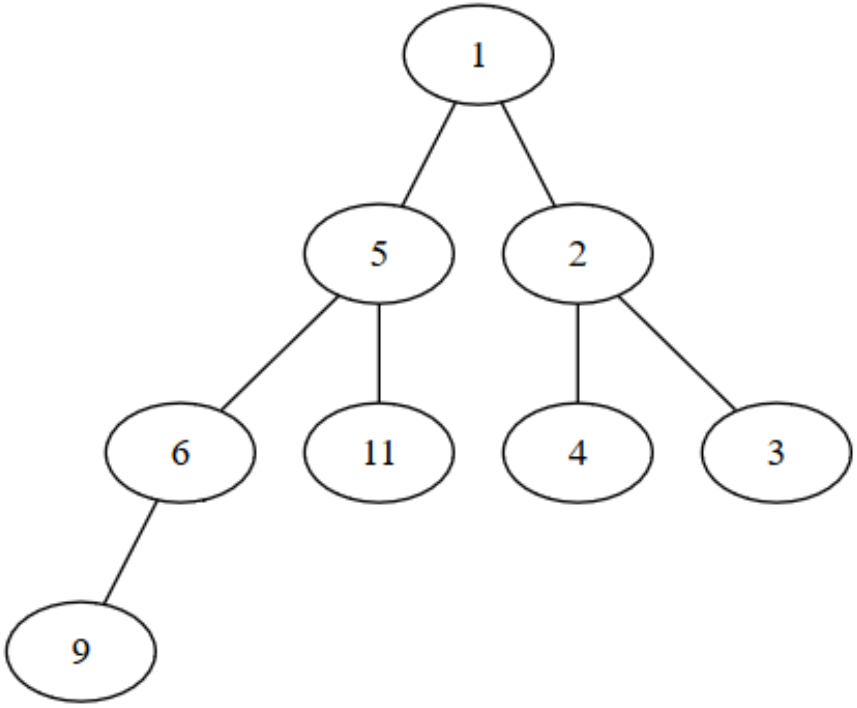
Heap-sort - Naive approach

- The initial sequence: [6, 1, 3, 9, 11, 4, 2, 5]
  - If we **add all elements into a min-heap**, we get:



Heap-sort - Naive approach

- The initial sequence: [6, 1, 3, 9, 11, 4, 2, 5]
  - If we **add all elements into a min-heap**, we get:



  - Then, if we **remove all the elements**, one-by-one, we obtain: 1, 2, 3, 4, 5, 6, 9, 11.

Heap-sort - Naive approach

❓ What is the time complexity of the heap-sort algorithm previously described?

Heap-sort - Naive approach

❓ What is the time complexity of the heap-sort algorithm previously described?

✓ $O(n \cdot log_2 n)$

Heap-sort - Naive approach

❓ What is the time complexity of the heap-sort algorithm previously described?

✓ $O(n \cdot log_2 n)$

❓ What is the extra space complexity of the heap-sort algorithm previously described?

Heap-sort - Naive approach

❓ What is the time complexity of the heap-sort algorithm previously described?

✓ $O(n \cdot log_2 n)$

❓ What is the extra space complexity of the heap-sort algorithm previously described?

✓ $\Theta(n)$

Heap-sort - Better approach

- We start by transforming the unsorted array into a max-heap.
  - The second half of the array contain leaves, so they can be left where they are.
  - Starting from the last non-leaf element (and going towards the beginning of the array), we just call *bubble-down* for every element.

Heap-sort - Better approach

Heap-sort - better approach:

```
function build-max-heap(a, n) is:
//a - an array of lenght n
    heap.elems ← a
    heap.length ← n
    heap.capacity ← n
    for i ← [n/2], 1, -1 execute
        bubble-down(heap, i)
    end-for
    build-max-heap ← heap
end-function
```

Time complexity: $O(n)$

Heap-sort - Better approach

- After transforming the unsorted array into a max-heap:
  - The maximum element is stored in the root, so at index 1 $\Rightarrow$ we swap it with the one at the last index
  - We discard the last element in the heap, by decrementing the length of the heap
  - The root element is the only that may violate the heap property $\Rightarrow$ we *bubble-down* it
  - We repeat the process until the length of the heap is 1

Heap-sort - Better approach

Heap-sort - better approach:

```
subalgorithm heapsort(a, n) is:
//a - an array of lenght n
    heap ← build-max-heap(a, n)
    for i ← n, 2, -1 execute
        aux ← a[i]
        a[i] ← a[1]
        a[1] ← aux
        heap.length ← heap.length-1
        bubbleDown(a, 1)
    end-for
end-subalgorithm
```

Heap-sort - Complexity

- Time complexity of this heap-sort is $O(n \cdot log_2 n)$.
  - *build-max-heap* runs in $O(n)$.
  - *bubble-down* runs in $O(log_2 n)$ and we call it $n - 1$ times.
- Extra-space complexity of this approach is $\Theta(1)$.

Sum of the largest k elements - problem statement

❗ Consider the following problem: Determine the sum of the largest k elements from a vector containing n distinct numbers.

🇪 If k = 3 and the array contains the following 10 elements:

- [6, 12, 9, 91, 3, 5, 25, 81, 11, 23]

, the result should be:

- 91 + 81 + 25 = 197.

Sum of the largest k elements - Solution I

 Use a binary max-heap. Add all the elements to the heap and remove the first k.

Sum of the largest k elements - Solution I - Implementation

```
function sumOfK(elems, n, k) is:
//elems is an array of unique integer numbers
//n is the number of elements from elems
//k is the number of elements we want to sum up. Assume k <= n
       init(h, "≥") //assume we have the Heap data structure implemented. We
       initialize a heap with the relation "≥" (a max-heap)
       for i ← 1, n execute
              add(h, elems[i])
       end-for
       sum ← 0
       for i ← 1, k execute
              elem ← remove(h)
              sum ← sum + elem
       end-for
       sumOfK ← sum
end-function
```

Sum of the largest k elements - Solution I - Complexity

 What's the complexity of adding an element to a heap with n elements?

Sum of the largest k elements - Solution I - Complexity

❓ What's the complexity of adding an element to a heap with n elements?

✓ O(log2n)

❓ What's the complexity of removing an element from a heap with n elements?

Sum of the largest k elements - Solution I - Complexity

 What's the complexity of adding an element to a heap with n elements?

✓ O(log2n)

 What's the complexity of removing an element from a heap with n elements?

✓ O(log2n)

Sum of the largest k elements - Solution I - Complexity

 What's the complexity of adding an element to a heap with n elements?

 O(log2n)

 What's the complexity of removing an element from a heap with n elements?

 O(log2n)

 In total we have O(n*log2n) + O(k*log2n). Since n >= k, this is O(n*log2n).

Sum of the largest k elements - Solution II

How can we reduce the complexity?

We can keep in the heap only the largest k elements (so far)

Example: [6, 12, 9, 91, 3, 5, 25, 81, 11, 23]

Sum of the largest k elements - Solution II

How can we reduce the complexity?

We can keep in the heap only the largest k elements (so far)

Example: [6, 12, 9, 91, 3, 5, 25, 81, 11, 23]

- Initially we add in the heap 6, 12, 9.
- When we get to 91, we can drop 6, because we know for sure that it is not going to be part of the 3 maximum numbers (we already have 3 numbers greater than this). So we keep 12, 91, 9.
- When we get to 3, we know it is not going to be part of the 3 maximum elements
- When we get to 5, we know it is not going to be part of the 3 maximum elements
- When we get to 25, we can drop 9, and go on with 12, 91, 25.
- etc.

Sum of the largest k elements - Solution II

 Should we keep the elements in a max-heap or a min-heap?

Sum of the largest k elements - Solution II

 Should we keep the elements in a max-heap or a min-heap?

 A min-heap

 When we have the k largest elements at a given point, we will be interested in the minimum of these elements, in order to compare it to the current element in the array.

Sum of the largest k elements - Solution I - Implementation

```
function sumOfK2(elems, n, k) is:
//elems is an array of unique integer numbers
//n is the number of elements from elems
//k is the number of elements we want to sum up. Assume k <= n
       init(h, "≤") //assume we have the Heap data structure implemented. We
       initialize a heap with the relation "≤" (a min-heap)
       for i ← 1, k execute //the first k elements are added "by default"
              add(h, elems[i])
       end-for
       for i ← k+1, n execute
              if elems[i] > getFirst(h) then //getFirst is an operation which returns
the first element from the heap.
                     remove(h)//it returns the removed element, but we do not need it
                     add(h, elems[i])
              end-if
       sum ← 0
       for i ← 1, k execute
              elem ← remove(h)
              sum ← sum + elem
       end-for
       sumOfK2 ← sum
end-function
```

Sum of the largest k elements - Solution I - Complexity

What's the complexity of adding an element to the heap?

Sum of the largest k elements - Solution I - Complexity

What's the complexity of adding an element to the heap?

O(log2k)

What's the complexity of removing from the the heap?

Sum of the largest k elements - Solution I - Complexity

What's the complexity of adding an element to the heap?

O(log2k)

What's the complexity of removing from the the heap?

O(log2k)

Sum of the largest k elements - Solution I - Complexity

What's the complexity of adding an element to the heap?

O(log2k)

What's the complexity of removing from the the heap?

O(log2k)

We call add at most n times (worst case, when every element is greater than the root of the heap) and remove at most n times. So, in total we have $O(n \cdot log_2 k)$.

Sum of the largest k elements - Solution I - Obs.

If we have access to the representation, we can make the implementation slightly more efficient:

- the last for will simply sum up the elements from the heap (array)
  - Without having access to the representation, we can keep a variable with the sum so far: whenever we add to the heap we add to the sum and whenever we remove from the heap we subtract from the sum.

- In the middle *for* loop we will not have to do a remove and an add. We can just overwrite the element from position 1 (this is what would be removed anyway) with the newly added element and do a bubble-down on it.

Exam - Date, location, duration

Date & location:

- **Regular exam**: **June 7** (Wednesday) from **14:00**
  - Tiberiu Popoviciu, No. 1 Mihail Kogalniceanu Street
- Retake exam: July 15 (Saturday) from 10:00,
  - G. Călugăreanu, No. 1 Mihail Kogalniceanu Street

Duration: **90 minutes**

Exam - Structure & grading scheme

- **1** point *ex officio*
- **3** points - Subject **A**: Short-answer & drawings problems
- **3** points - Subject **B**: Multiple choice problems
- **3** points - Subject **C**: Implementation

Exam - Subject A - Short-answer & drawings problems

- **1** point - Subject A.1. **Hash Tables** - operations (drawings)
- **1** point - Subject A.2. **Binary Trees** - terminology & traversals (short answers questions)
- **1** point - Subject A.3 **Binary Search Trees** / **Heaps** - operations (drawings)

Exam - Subject A.1: Hash Tables - Examples

Examples:

- Insert the integers 11, 12, 21, 33, 22 (in the given order) in an initially empty hash table with open addressing of size m = 11, using the division method and:
  - a) linear probing
  - b) quadratic probing with c1 = 0 and c2 = 1

Exam - Subject A.1: Hash Tables - Examples

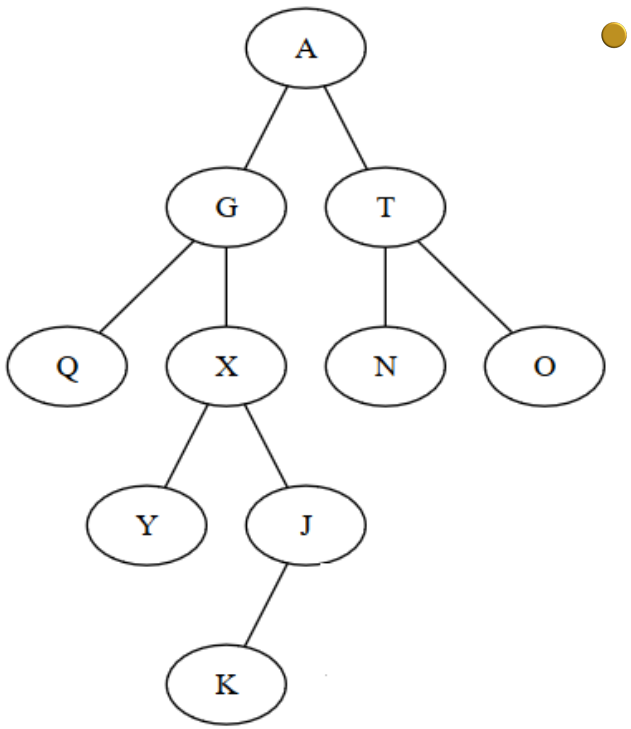
Examples:

- Insert the integers 11, 12, 21, 33, 22 (in the given order) in an initially empty hash table with open addressing of size m = 11, using the division method and:
  - a) linear probing
  - b) quadratic probing with c1 = 0 and c2 = 1

- Consider an initially empty hash table with coalesced chaining of size m = 10 positions, using the division method. Insert the following integers in the hash table: 13, 10, 163, 673, 30. After inserting all the elements, remove 13 from the hash table.

Exam - Subject A.2: Binary Trees - Terminology & traversals - Example

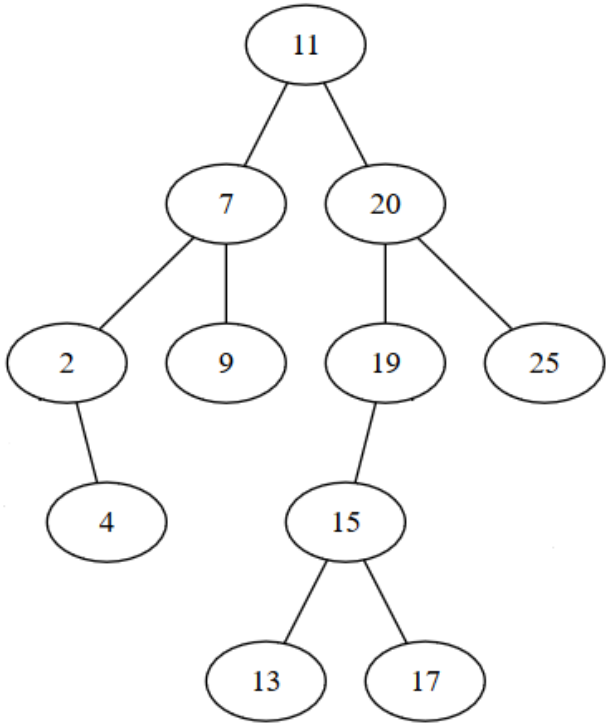
Example:



- Complete the following sentences about the binary tree on the left:
  - The internal nodes are:
  - The Height of node X is:
  - The inorder traversal visits the nodes in the following order:
  - The postorder traversal visits the nodes in the following order:
  - The tree is a balanced binary tree (yes/no):

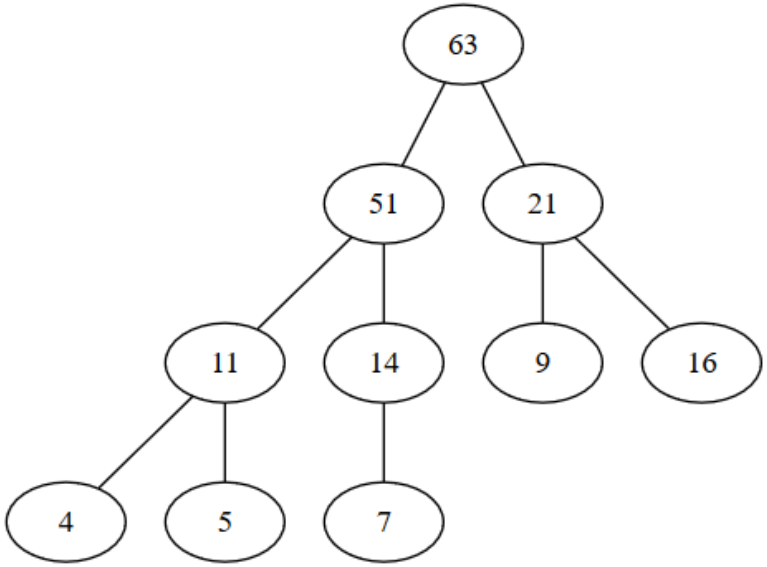
All the terminology discussed for binary trees as well as all types of traversals are possible for this problem.

Exam - Subject A.3: Binary Search Trees or Heaps - Examples



Example 1: Given the binary search tree above, insert the element 14 and then remove 7 and 19.

Exam - Subject A.3: Binary Search Trees or Heaps - Examples



Example 2: Given the heap above, insert the element 55 and then perform one remove operation.

Exam - Subject B: Multiple choice problems - Examples I

1. $n^2 * log_2 n + n^4$ belongs to which of the following complexity classes?

- a. $\Omega(n^3)$
- b. $O(n^5)$
- c. $\Theta(n^4)$
- d. $O(n^3)$
- e. $\Omega(n^5)$
- f. none of them

Exam - Subject B: Multiple choice problems - Examples I

1. $n^2 * log_2 n + n^4$ belongs to which of the following complexity classes?

- a. $\Omega(n^3)$
- b. $O(n^5)$
- c. $\Theta(n^4)$
- d. $O(n^3)$
- e. $\Omega(n^5)$
- f. none of them

2. Which of the following three sequences represent a binary heap?

- a. [1, 12, 23, 10, 15, 38, 45, 15, 18, 20, 21]
- b. [1, 8, 27, 10, 45, 83, 91, 31, 12, 52, 51]
- c. [1, 13, 20, 21, 65, 54, 67, 41, 30, 83, 52]
- d. none of them

Exam - Subject B: Multiple choice problems - Examples II

3. If we use a dynamic array as representation for a stack, where should we place the top for optimal performance for all stack operations?
   - a. beginning of the array
   - b. end of the array
   - c. either the beginning or the end
   - d. we cannot implement a stack on a dynamic array

Exam - Subject B: Multiple choice problems - Examples II

3. If we use a dynamic array as representation for a stack, where should we place the top for optimal performance for all stack operations?

- a. beginning of the array
- b. end of the array
- c. either the beginning or the end
- d. we cannot implement a stack on a dynamic array

4. For which of the following collision resolution methods might an insertion fail (we simply cannot add the element) even if the table contains empty slots?

- a.separate chaining
- b.coalesced chaining
- c.open addressing – linear probing
- d.open addressing – quadratic probing
- e.none of them

Exam - Subject B: Multiple choice problems - Examples III

5. What is the difference between the maximum and minimum possible depth of a binary tree with 7 nodes?

- b. 3
- c. 4
- d. 5
- e. 6
- f. 7

Exam - Subject B: Multiple choice problems - Examples III

5. What is the difference between the maximum and minimum possible depth of a binary tree with 7 nodes?

- b. 3
- c. 4
- d. 5
- e. 6
- f. 7

6. ADT Map and ADT MultiMap have a pretty similar interface. Which of the following operations do not have the same number of parameters? Select one or more.

- a. add
- b. remove
- c. size
- d. iterator
- e. isEmpty

Exam - Subject B: Multiple choice problems - Obs.

 Any aspect discussed during lectures and seminars might be tested here.

Exam - Subject C: Implementation

 Example:

- Consider a Set, represented on a doubly linked list on array. Give the representation of the Set and specify and implement the *remove* operation. Specify the time complexity of the operation (best case, worst case and overall complexity).

 Any operation discussed during lectures and seminars (in conjunction with any ADT) might be tested here.

Thank you