EÖTVÖS LORÁND
UNIVERSITY | BUDAPEST

Implementing Custom Data Types

Object Oriented Programming | 2024 Spring

Practice 4

By Tarlan Ahadli

Priority Queue

- $(data\colon\ \mathbb{S}, \mathrm{priority}\colon \mathbb{Z})$
  - Operations
    - Add – add element
    - Max – get Max element
    - RemMax – remove Max element
    - setEmpty
    - isEmpty

Problem: Competition

- Given group name and the points they earned creata a priority queue such that by callen remMax we can show them in order.

Data Type Definition Table:

- **Value**
  - PrQueue
- **Operations**
  - $A = (PQ: PrQueue, l: \mathbb{L})$
  - $l = PQ.isEmpty()$
  - --
  - $A = (PQ: PrQueue)$
  - $PQ.setEmpty()$

Data Type Definition Table: Cont.

- **Operations**
  - $A = (PQ: PrQueue, e: \mathbb{S} \text{ x } \mathbb{Z})$
  - $PQ.add(e)$
  - --
  - $A = (PQ: PrQueue, e: \mathbb{S} \text{ x } \mathbb{Z}) \mid Pre = (\neg PQ.isEmpty())$
  - $e = PQ.\mathrm{max}()$
  - --
  - $A = (PQ: PrQueue, e: \mathbb{S} \text{ x } \mathbb{Z}) \mid Pre = (\neg PQ.isEmpty())$
  - $e = PQ.\mathrm{remMax}()$

Data Type Definition Table: Cont.

- How do we store the elements? | Array, List?
- Ordered Vs Unordered Queue?
- ---
- **Ordered Priority Queue:**
  - **Sorted:** Keeps elements sorted by priority.
  - **Insertion:** Slower ($O(n)$).
  - **Deletion/Peek:** Fast ($O(1)$).
- **Unordered Priority Queue:**
  - **Unsorted:** No specific order; searches for priority.
  - **Insertion:** Fast ($O(1)$).
  - **Deletion/Peek:** Slower ($O(n)$).

Ordered vs Unordered Queue

Operation	Ordered	Unordered
isEmpty()	O(1)	O(1)
setEmpty()	O(1)	O(1)
Add	O(n)	O(1)
Max	O(1)	O(n)
RemMax	O(1)	O(n)

Ordered vs Unordered: Cont.

- Task Description
    - Add Element
    - Remove Max untill queue is Empty.
- isEmpty – O(1) for both
- Add – Unordered better
- removeMax – Ordered better

Although it may appear similar, adding to an ordered queue requires shifting elements, which leads to more frequent memory writes compared to the Unordered Add operation. Writing to memory consumes more time, making the unordered queue more efficient in this regard.

Data Type Definition Table. Cont.

- Representation
  - C#
    - List<Elem>
    - *struct Elem{*

      public string S;
      public int Priority;
      }
  - Paper
    - Vector: Elem
    - Elem = Record(data:S, pr: Z)

Data Type Definition Table. Cont.

- Implementation
  - isEmpty()
    - $l := (|vec| = 0)$
  - setEmpty()
    - $vec := <>$
  - add(e)
    - $vec := vec \oplus < e >$
    - < Concatanation must be performed between collection types >
  - Max
  - RemMax

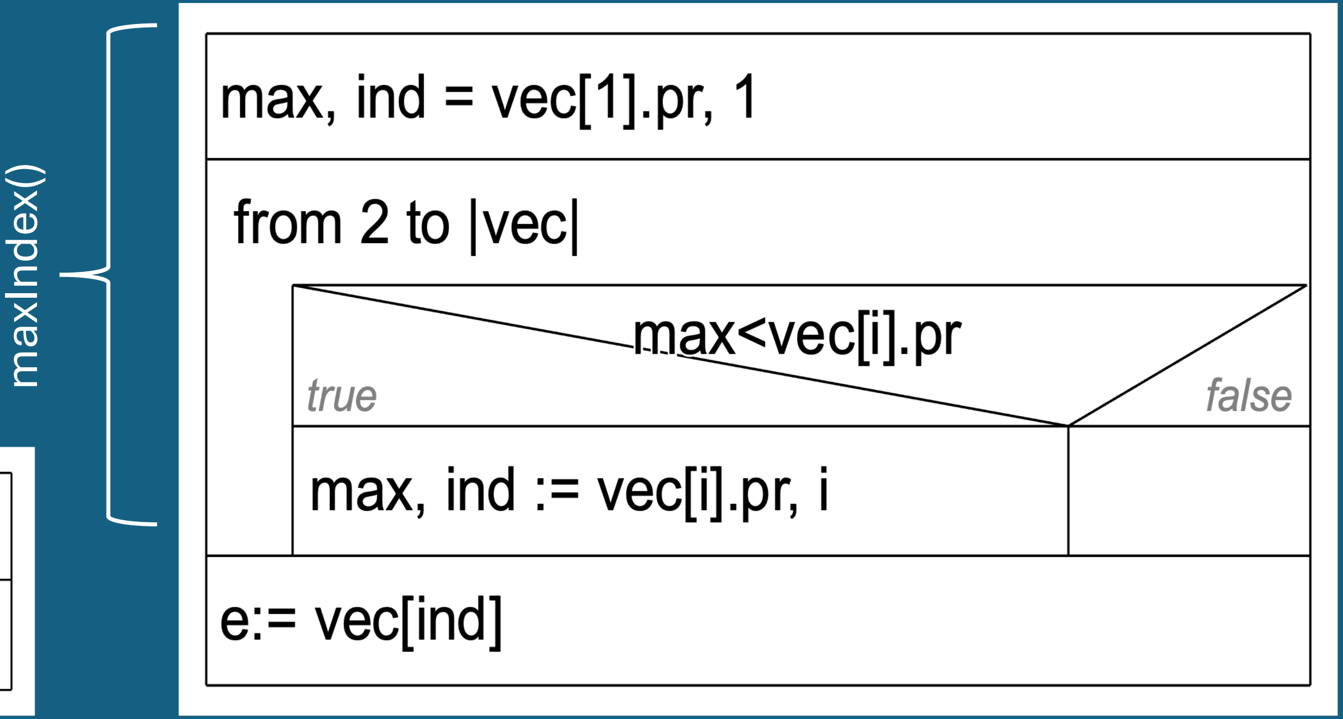
Implementation: Max

- $A = (vec: Elem, e: Elem)$
- $Pre = (vec = vec' \wedge |vec| > 0)$
- $Post = (Pre \ \wedge \underbrace{(ind, max) = MAX_{i=1}^{|vec|}\ (vec[i].pr)}_{\text{Post-cond. For maxIndex()}} \wedge e = vec[ind])$

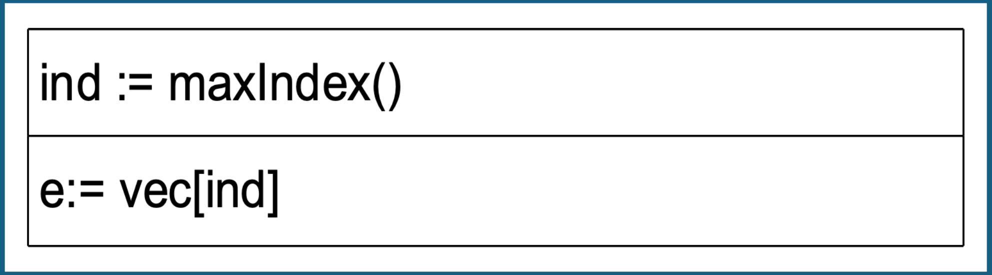
maxIndex() can be used for remMax() as well.

Implementation: Max. Cont.

- Analogy: MaxSearch/MaxIndex
  - $m \sim 1$
  - $n \sim |vec|$
  - $f(i) \sim vec[i].pr$
  - $(H,<) \sim (Z,<)$



```
max, ind = vec[1].pr, 1
from 2 to |vec|
    if max<vec[i].pr
        max, ind := vec[i].pr, i
e:= vec[ind]
```



```
ind := maxIndex()
e:= vec[ind]
```

* https://nigjo.github.io/structogramview/

Implementation: RemMax

- $A = (vec: Elem, e: Elem)$
- $Pre = (vec = vec' \wedge |vec| > 0) \mid vec': input, vec: output$
- $Post = ((ind, max) = MAX_{i=1}^{|vec'|}\left(vec'^{[i]}.pr\right) \wedge e = vec'^{[ind]} \wedge$
  $vec = vec'[1 \ldots ind - 1] + vec'[|vec'|] + vec'[ind + 1 \ldots |vec'| - 1])$
- $(\text{“”}, 5), (\text{“”}, \mathbf{7}), (\text{“”}, 1), (\text{“”}, 2), (\text{“”}, 2)$

```
ind := maxIndex()
e := vec[ind]
vec[ind] := vec[|vec|]
vec.pop_back()
```

Class Diagram

$PrQueue$

$-------- -$

$-vec: Elem[\quad]$

$-------- -$

$+PrQueue()$
$+isEmpty(\quad): bool\ \{query\}$
$+setEmpty(\quad): void$
$+Add(e: Elem): void$
$+Max(\quad): Elem\{query\}$
$+RemMax(\quad): Elem$
$-maxIndex(\quad): Int\{query\}$