Object Oriented Programming

2022 midterm report

Deadline:2022/6/27

1 Exercise

Construct a binary heap class described below. You need to define the class using class template `T`, where `T` implements the `Comparable` interface. And observe it works with a list of instances implementing the `Comparable` interface. For this purpose, you also need to define a class of data. Finally show sorting examples as validation of the class.

2 Binary Heap

Binary heap is a one of types of data structure for extracting the minimum of data. Consider a set of elements, which have label and value fields. Each element is comparable by comparing the value field of the element with other. The image of binary heap is shown in Fig. 1. The minimum element locates at the root of the tree. The numbers locating above nodes indicate indexes discussed below.

The tree of the binary heap must be a complete binary tree which obeys the followings: The top level (the top in Fig. 1) is called $\ell = 0$. The second is $\ell = 1$ and so on. Each level ℓ has 2^ℓ nodes except the lowest level $\ell = L$. Remaining nodes in the $\ell = L$ level must be assigned from the leftmost.

Each node in the $\ell < L - 1$ level has two lower nodes (*daughters*). In the $l = L - 1$ level, only one node V of this level has one or two subnodes. The nodes locating left of the node V have two subnodes and the right nodes of the node V have no subnodes.

Each element v must not have greater value than its subnodes.

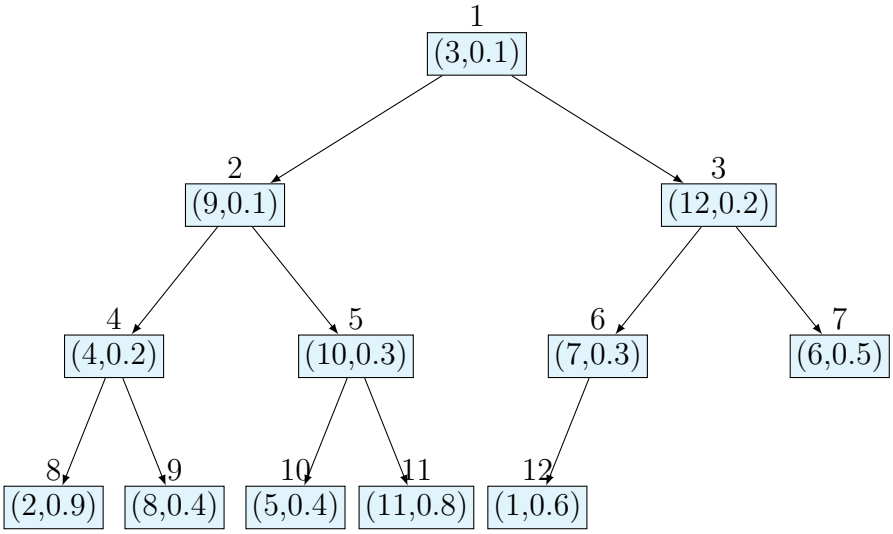

Fig. 1: Image of binary heap. Each node has label (left) and value (right). Comparison between nodes is relation between node values.

3 Implementation of Binary Heap

Binary heap stores elements as tree structure. However, it does not request to store elements in tree data structure, because the tree is restricted complete binary. The positions of elements are uniquely identified by natural number. You can find unique numberings (indexes) in Fig. 1.

The indexing scheme requires the followings (Fig. 2):

- The root node is V_1.
- Any node V_k ($k > 1$) has its upper node (*parent*) $V_{k'}(k' = \lfloor k/2 \rfloor)$, where $\lfloor x \rfloor$ returns the maximum integer y which is not larger than x.
- Any node V_k has zero, one or two lower nodes (*daughters*).
  - If a node V_k has one lower node, the lower node must be V_{2k}, where $V_k \leq V_{2k}$.
  - If a node V_k has two lower nodes, V_{2k} and V_{2k+1}, where $V_k \leq V_{2k}$ and $V_k \leq V_{2k+1}$.
  - There is no restriction between V_{2k} and V_{2k+1}.

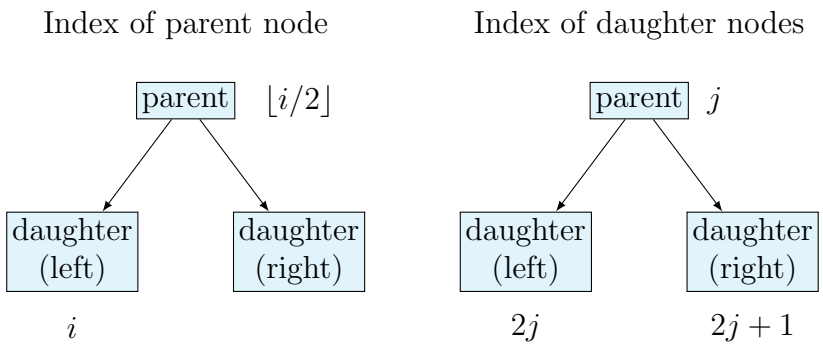

Fig. 2: Relation between a node and its subnodes (daughters).

As a result, a binary heap does not require tree like data structure. Data can be stored in a list, where the index start from one not zero. If you use a list library where the index starts zero, you have to insert *Null* (something not used) object at the beginning of the list.

4 Minimum methods of binary heap

The minimum methods of binary heap are

- constructor,
- adding a new node,
- polling the minimum node,
- checking the emptiness,

where *polling* means that returning the minimum and removing it from the binary heap.

4.1 Constructor

The constructor of a binary heap creates a list for data. If you use a list library where the index starts zero, you have to insert *Null* object at the beginning of the list. The number of data is set $n = 0$.

4.2 Adding a new node

For adding a new element into the binary heap, the new element is first added at the end of the list. Then the added element moves up to the adequate position for keeping the small-large relation in the system. See Algorithms 1 and 2.

Algorithm 1 Add a new element

```
procedure PUT(o)
    L.append(o)
    n++
    shiftUp(n)
end procedure
```

In Algorithm 2, `isLess(k,j)` returns true only if $V_k < V_j$. And `swap(i,j)` is a method exchange V_i and V_j.

Algorithm 2 Shift up element recursively

```
procedure SHIFTUP(k)
    j = ⌊k/2⌋
    if k > 1 ∧ isLess(k, j) then
        swap(k, j)
        shiftUp(j)
    end if
end procedure
```

4.3 Polling the minimum element

The minimum element locates in the list with index $k = 1$. If you remove the minimum, the tree structure is broken. To prevent the tree from broken, the last element in the list is moved to $k = 1$ and then shift down to the correct position. See Algorithms 3 and 4.

Algorithm 3 Polling the minimum element

```
procedure POLL
    t = L.get(1)                          ▷ Obtain the first element
    x = L.removeLast
    if n > 1 then
        L.set(1, x)                       ▷ Set x at the first position
        shiftDown(1)
    end if
    n − −
    return t
end procedure
```

Algorithm 4 Shift down element recursively

```
procedure SHIFTDOWN(k)
    if 2k ≤ n then
        j = 2k
        if j < n ∧ isLess(j + 1, j) then
            j + +
        end if
        if isLess(k, j) then
            return
        end if
        swap(k, j)
        shiftDown(j)
    end if
end procedure
```

5 How to submit your report

Your report should be prepared as a PDF file and submitted through *Teams*.

- Prepare your report digitally with such as `Word` or LaTeX.
- The file name should be `studentID.pdf`.
- Contain description of understanding and solutions of tasks, programs, and program outputs.
- Show multiple examples.
- Improve your program's readability with suitable naming of classes, variables, and methods. Also add suitable comments in programs.
- Write your document neatly with correct Japanese or English.
- Cite suitable references.

6 Scoring

C: Requested programs are coded, but not explained or not suitably constructed as OOP.

B: Classes are correctly defined and codes are well organized.

A: Class planning and workflows are well discussed in the report.

S: Over the level A, some notable points are found.