Algorithms and Data Structures

with implementations in Java and C++

Data structures

Algorithms

C++

Books

Forum

Feedback

C++ code snippets

Economics Textbook

TOP3 Articles

Quicksort

Depth-first search

Binary search tree

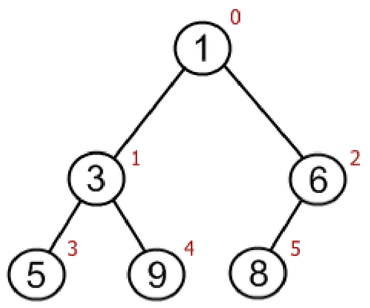
Binary heap. Array-based internal representation.

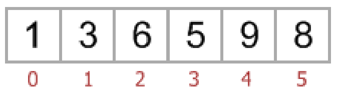
Heap is a binary tree and therefore can be stored inside the computer using links. It gives various benefits; one of them is the ability to vary number of elements in a heap quite easily. On the other hand, each node stores two auxiliary links, which implies an additional memory costs. As mentioned above, a heap is a complete binary tree, which leads to the idea of storing it using an array. By utilizing array-based representation, we can reduce memory costs while tree navigation remains quite simple.

As for the heapsort algorithm, array-based implementation is kind of native.

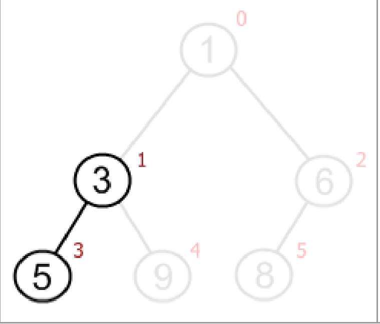
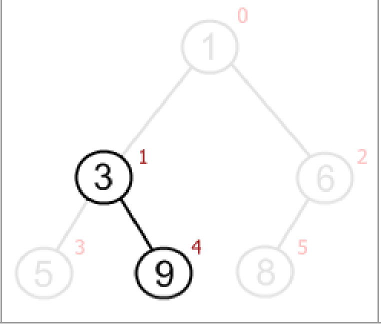
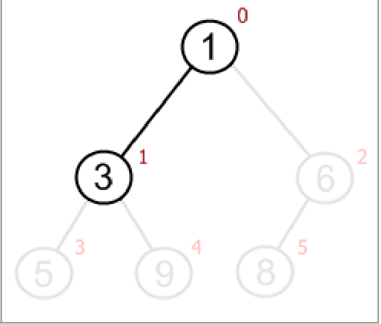
Mapping heap to array

A heap is stored in array level by level. The topmost level contains root only. It is mapped to the first element of an array (with index 0). Root's children are mapped to the second and third elements and so on. A heap is a complete binary tree, which guarantees, that heap's nodes take up places in the array compactly, making the mapping quite efficient.





Such mapping answers to formulas below:

`Left(i) = 2 * i + 1`	`Right(i) = 2 * i + 2`	`Parent(i) = (i - 1) / 2`
		

You can see now, that navigation over a heap, mapped to an array, is, actually, very easy.

Code snippets

Java implementation

```
public class BinaryMinHeap {
      private int[] data;
      private int heapSize;

      public BinaryMinHeap(int size) {
            data = new int[size];
            heapSize = 0;
      }

      public int getMinimum() {
            if (isEmpty())
                  throw new HeapException("Heap is empty");
            else
                  return data[0];
      }

      public boolean isEmpty() {
            return (heapSize == 0);
      }

      …

      private int getLeftChildIndex(int nodeIndex) {
            return 2 * nodeIndex + 1;
      }

      private int getRightChildIndex(int nodeIndex) {
            return 2 * nodeIndex + 2;
      }

      private int getParentIndex(int nodeIndex) {
            return (nodeIndex - 1) / 2;
      }

      public class HeapException extends RuntimeException {
            public HeapException(String message) {
                  super(message);
            }
      }
}
```

C++ implementation

```
class BinaryMinHeap {
private:
      int *data;
      int heapSize;
      int arraySize;

      int getLeftChildIndex(int nodeIndex) {
            return 2 * nodeIndex + 1;
      }

      int getRightChildIndex(int nodeIndex) {
            return 2 * nodeIndex + 2;
      }

      int getParentIndex(int nodeIndex) {
            return (nodeIndex - 1) / 2;
      }

public:
      BinaryMinHeap(int size) {
            data = new int[size];
            heapSize = 0;
            arraySize = size;
      }

      int getMinimum() {
            if (isEmpty())
                  throw string("Heap is empty");
            else
```

```
                  return data[0];
      }

      boolean isEmpty() {
            return (heapSize == 0);
      }

      …

      ~BinaryMinHeap() {
            delete[] data;
      }
};
```

Previous: Binary heap   Next: Inserting an element into a heap

Like 11 people like this.

Contribute to AlgoList

Liked this tutorial? Please, consider making a donation. Contribute to help us keep sharing free knowledge and write new tutorials.

Donate

PayPal

Every dollar helps!

Partners Ads

Leave a reply

Your name (optional):	
Your e-mail (optional):	
Message:	

Send