ICPC Training Material : Data Structures, Algorithms and Theorems

Contents

1 Data Structures

1.1 Elementary Data Structures

In Computer Science, in order to treat and store data, it first needs to be structured. Hence, multiple data structures were created : Array, Hash, Queue, Tree and multiple others.

1.1.1 Array

The array is the most used data structure. It consists on a collection of values, such as each value is identified by at least one index.

4	23	-2	7	0	22

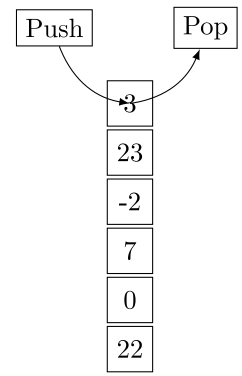
Arrays are useful because they exploit the addressing logic of computers. Generally, the memory is a one-dimensionnal array of words, whose indices are the addresses.

1.1.2 Stack

The stack consists on a collection of data that is added and retrieved according to the FILO(First In Last Out) method : The operations can only be applied on the top element of the stack.

Two operations are used to manipulate the data :

- push : Used to add a block on the top of the stack. A stack is overflowed if the number of blocks exceeds the capacity.
- pop : Used to retrieve the block at the top of the stack.



An intuitive application would be inversing strings or numbers. It is also used for memory management, as well as in expression evaluation.

1.1.3 Queue

A queue is

1.1.4 Heap

1.1.5 Hash

1.1.6 Trees

1.2 Advanced Data Structures

1.2.1 Priority queues

1.2.2 Fenwick Tree

1.2.3 K-D Tree

1.2.4 Interval Tree

2 Algorithms

2.1 Sorting and Searching

2.1.1 Binary Search

2.1.2 Quick Sort

The main concept behind this algorithm is the **divide and reign** principle. At each iteration, the initial array is progressively divided into a bigger number of sub-arrays. Each sub-array is sorted by choosing a **pivot** and placing the elements smaller than him at his left.[1]

Here is a pseudo-code scheme is attributed to *Nico Lomuto* :

```
function QUICKSORT(A,lo,hi)
    p ← PARTITION(A,lo,hi)
    QUICKSORT(A, lo, p − 1)
    QUICKSORT(A, p + 1, hi)
end function

function PARTITION(A,lo,hi)
    i ← lo
    for j ← lo to hi − 1 do
        if A[j] ≤ A[hi] then
            Switch A[i] with A[j]
            i ← i + 1
        end if
        j ← j + 1
    end for
    Switch A[i] with A[hi] return i
end function
```

[1] Hence, the bigger elements are placed on his right.

Quicksort can be easily parallelized, and the performance on parallel quicksort is better than mergesort and heapsort. However, it has the possibility to degenerate to $O(n^2)$, which can be devastating if used in large data sets.