Transform and Conquer

Chapter 6

Transform and Conquer

There are 3 Variations:

- Instance simplification
- Representation change
- Problem reduction

Transform and Conquer

- Instance simplification:
  - Transformation to a simpler or more convenient instance of the same problem.
  - Ex:
    - Presorting
    - Gauss elimination, AVL Trees, 2-3 Trees (not discussing now)
- Representation change:
  - Transformation to a different representation of the same instance.
  - Ex:
    - Heaps and Heapsort
- Problem reduction: (not discussing now)
  - Transformation to an instance of a different problem for which an algorithm is already available.
  - Ex:
    - Linear programming
    - Reductions to graph problems

Presorting

- Importance of Sorting
- Efficiency calculations of sorting algorithms

Presorting

ALGORITHM PresortElementUniqueness(A[0….n-1])

//Solves the element uniqueness problem by sorting the array first.

//Input: An Array A[0…n-1]

//Output: Returns “true” if A has no equal elements, “false” otherwise.

Sort the array A

for i<-0 to n-2 do

    If A[i]==A[i+1] return false

return true

ALGORITHM *PresortMode*($A[0..n-1]$)

```
//Computes the mode of an array by sorting it first
//Input: An array A[0..n − 1] of orderable elements
//Output: The array's mode
Sort the array A
i ← 0                       //current run begins at position i
modefrequency ← 0           //highest frequency seen so far
while i ≤ n − 1 do
    runlength ← 1; runvalue ← A[i]
    while i+runlength ≤ n − 1 and A[i+runlength] = runvalue
        runlength ← runlength+1
    if runlength > modefrequency
        modefrequency ← runlength; modevalue ← runvalue
    i ← i+runlength
return modevalue
```

Heap

- Heap is a binary tree with following properties:
- 1. Tree shape requirement: Binary Tree is complete - all levels full except for last level where only some rightmost leaves may be missing
- 2. Parental dominance holds - value at a node greater than or equal to the values of its children
- Traversal from root to leaf is always decreasing
- No order among siblings (no ordering from left to right)

Characteristics of Heap

- 1. Root is largest element
- 2. Heap with n elements can be stored as a list:
- – Elements in slots 1 .. n
- – Non-leaf nodes in first $\lfloor n/2 \rfloor$ positions
- – Leaf nodes stored in last $\lceil n/2 \rceil$ positions
- – Given node in slot i, its children are in slots 2i and 2i + 1
- – Given node in slot i, its parent is in slot $\lfloor i/2 \rfloor$

Heap Bottom Up Algorithm

- 1. Given n nodes, place them from positions 1 to n
- 2. Starting with last parental node, compare parent with children
- 3. If parental dominance does not hold, exchange parent with largest child and
- repeat checking for parental dominance in the new position for parent

Heap Bottom Up Algorithm

Algorithm *HeapBottomUp*$(H[1..n])$
//Constructs a heap from the elements of a given array
// by the bottom-up algorithm
//Input: An array $H[1..n]$ of orderable items
//Output: A heap $H[1..n]$
for $i \leftarrow \lfloor n/2 \rfloor$ **downto** 1 **do**
    $k \leftarrow i; \quad v \leftarrow H[k]$
    $heap \leftarrow$ **false**
    while not $heap$ **and** $2 * k \leq n$ **do**
        $j \leftarrow 2 * k$
        if $j < n$ //there are two children
            if $H[j] < H[j+1] \quad j \leftarrow j + 1$
        if $v \geq H[j]$
            $heap \leftarrow$ **true**
        else $H[k] \leftarrow H[j]; \quad k \leftarrow j$
    $H[k] \leftarrow v$

Root deletion algorithm

Sorting:

- Swap root with rightmost leaf
- Decrease heap size by 1
- Heapify the smaller tree

The End

Thank You