Heapsort

Chapter 6

Review of Binary Trees

- What is a binary tree?
- What is the depth of the node?
- What is the height of a node?
- What is the height of the tree?
- What is a complete binary tree?

Facts about Perfect Binary Trees

Complete Binary Trees

- Nodes at depth h (the lowest level) are as far left as possible
- What is the relationship between the height and the number of nodes?

Heaps

- A ***heap*** is an complete binary tree
- Extra nodes go from left to right at the lowest level
- Where the value at each node is ≥ the values at its children (if any)
- This is called the ***heap property*** for max-heaps
- Max or Min Heap

Example

Storing Heaps

- As arrays!
- Root of tree is:
- Parent of A[i] is:
- Left child of A[i] is:
- Right child of A[i] is:

Example

- n = 13

92 85 73 81 44 59 64 13 23 36 32 18 54

Functions on Heaps

- MAX-HEAPIFY
- BUILD-MAX-HEAP
- HEAPSORT
- MAX-HEA-INSERT
- HEAP-EXTRACT-MAX
- HEAP-INCREASE-KEY
- HEAP-MAXIMUM

MAX-HEAPIFY, p 154

Max_Heapify(A, i) // A: Array, i: int

```
int L = left(i)
int r = right(i)
if (L <= A.heap_size and A[L] > A[i] )
  largest = L
else Largest = i
if (Right <= A.heap_size and A[R]> A[largest])
  largest = R
if largest != i
  swap ( A[i], A[largest] )
  Max_Heapify(A, largest)
```

Example

- 15 6 4 8 5 3 1 2 7   i = 2

Build_Max_Heap, p 157

Build_Max_Heap (A) // A: Array

```
A.heap_size = A.length
for i = floor ( A.length/2) to 1
  Max_Heapify(A,i)
```

Example

- 4 3 7 13 1 20 12 16 2 18

HeapSort, p 160

HeapSort(A) // A: Array

```
Build_Max_Heap(A)
for i = A.length to 2
  swap(A[1], A[i])
  A.heap_size = A.heap_size - 1
  Max_Heapify(A, 1)
```

Example

- 20 18 12 16 3 7 4 13 2 1

Priority Queues

- Priority Queues are an example of an application of heaps.
- A priority queue is a data structure for maintaining a set of elements, each with an associated key.

Priority Queues

- Max-priority queue supports dynamic set operations:
  - INSERT(S, x): inserts element x into set S.
  - MAXIMUM(S): returns element of S with largest key.
  - EXTRACT-MAX(S): removes and returns element S with largest key.
  - INCREASE-KEY(S, x, k): increases value of element x’s key to k. Assume k >= x’s current key value.

HEAP-MAXIMUM(A)

HEAP-MAXIMUM(A)

return $A[1]$

Time: $\Theta(1)$.

HEAP-EXTRACT-MAX

Heap_Extract_Max(A) // A: Array

```
if A.heap_size < 1
  error "underflow"
max = A[1]
A[1] = A[A.heap_size]
A.heap_size = A.heap_size -1
Max_Heapify(A,1)
return max
```

Example

- 15 6 4 8 5 3 1 2 7

Heap_Increase_Key, p 164

Heap_Increase_Key(A, i, key) // A: Array; i,key: ints

```
if key < A[i]
  error "new key is smaller than current key"
A[i] = key
while i > 1 and A[Parent(i)] < A[i]
  swap (A[i], A[Parent(i)] )
  i = Parent(i)
```

Why?

Example

- Increase key of node 6 in previous example to 20

MAX-HEAP-INSERT

- Given a key k to insert into the heap:
  - Insert a new node in the very last position in the tree with the key -infinity.
  - Increase the -infinity key to k using the HEAP-INCREASE-KEY procedure.

Max_Heap_Insert(A, key) // A:array; key:int

1	`A.heap_size = A.heap_size + 1`
2	`A[A.heap_size] = - infinity`
3	`Heap_Increase_Key(A, A.heap_size, key)`

// could replace Heap_Increase_Key with what?

Example

- Insert 12 into the above heap.