Binary Heaps

Autumn 2018

Shrirang (Shri) Mare
shri@cs.washington.edu

Thanks to Kasey Champion, Ben Jones, Adam Blank, Michael Lee, Evan McCarty, Robbie Weber, Whitaker Brand, Zora Fung, Stuart Reges, Justin Hsia, Ruth Anderson, and many others for sample slides and materials ...
Problems

1. Merging multiple sorted arrays
   - OutArray[k] = min(Array1[i1], Array2[i2])  // for 2 sorted arrays
   - OutArray[k] = min(Array1[i1], Array2[i2], …, Arrayk[ik])  // for k sorted arrays

2. Given n 2D points, find k points which are closest to point P(x, y)
   S = Set of k distances
   For each point Q in the remaining points:
     if dist(P, Q) is less than max(S)
       removeMax from S
       Insert dist(P, Q) in S

3. Priority queues: Job schedulers
   - Among all the job in a queue, get the job with the highest priority: removeMax(Priority Queue)
Problems

1. Merging multiple sorted arrays
   - OutArray[k] = min(Array1[i1], Array2[i2])                          // for 2 sorted arrays
   - OutArray[k] = min(Array1[i1], Array2[i2], …, Arrayk[ik])          // for k sorted arrays

2. Given \( n \) 2D points, find \( k \) points which are closest to point \( P(x, y) \)
   
   \( S = \) Set of \( k \) distances
   
   For each point \( Q \) in the remaining points:
   
   if dist\((P, Q)\) is less than max\((S)\)
     removeMax from \( S \)
     Insert dist\((P, Q)\) in \( S \)

3. Priority queues: Job schedulers
   
   - Among all the job in a queue, get the job with the highest priority: removeMax(Priority Queue)
Desired behavior: Get extreme values (min or max)

1. Merging multiple sorted arrays
   - OutArray[k] = min(Array1[i1], Array2[i2]) // for 2 sorted arrays
   - OutArray[k] = \textbf{min} (Array1[i1], Array2[i2], …, Arrayk[ik]) // for k sorted arrays

2. Given n 2D points, find k points which are closest to point P(x, y)
   
   S = Set of k distances
   
   For each point Q in the remaining points:
   
   if dist(P, Q) is less than max(S)
   
   removeMax from S

   Insert dist(P, Q) in S

3. Priority queues: Job schedulers
   
   - Among all the job in a queue, get the job with the highest priority: removeMax(Priority Queue)
Min Priority Queue ADT

- Collection where elements ordered based on priority.

- Behavior:
  - `removeMin()`: return element with smallest priority, removes element from collection
  - `peekMin()`: find, but do not remove, the element with smallest priority
  - `insert(element)`: add element to the collection

- Max Priority Queue ADT:
  - Same as Min Priority Queue ADT, just returns the largest instead of the smallest
Binary heap data structure

- Invented in 1964 for sorting
- Priority Queues is one of the main applications for binary heaps
- Lots of other applications: greedy algorithms, shortest path

- Basically, min-heap (or max-heap) is ideal when you want to maintain a collection of elements where you need to add arbitrary values but need an efficient removeMin (or removeMax).
Binary heap

Binary heap is a

1. binary tree
2. that satisfies the heap property, and
3. where every heap is a “complete” tree
Binary heap: Heap property
Binary heap: Heap property

$max$-heap: Every node is larger than (or equal to) its children
Binary heap: Heap property

*max-heap*: Every node is larger than (or equal to) its children

*min-heap*: Every node is smaller than (or equal to) its children
A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

There are no “gaps” in a complete tree.
Complete binary tree

There as not complete binary trees
Question: Valid min-heap?

Complete binary tree?  
Heap property satisfied?
Question: Valid min-heap?

Complete binary tree? Yes!
Heap property satisfied? Yes!
Question: Valid min-heap?

Complete binary tree?
Heap property satisfied?
Question: Valid min-heap?

Complete binary tree?  No!
Heap property satisfied? No!
Binary heap insert

insert(12)
Binary heap insert

insert(12)
Binary heap insert

insert(12)
insert(7)
Binary heap insert

insert(12)
insert(7)
Binary heap insert

insert(12)
insert(7)

Heap broken
Binary heap insert

insert(12)
insert(7)

Heap broken
Binary heap insert

insert(12)
insert(7)
Binary heap insert

insert(12)
insert(7)
insert(1)
Binary heap insert

insert(12)
insert(7)
insert(1)
Binary heap insert

insert(12)
insert(7)
insert(1)

Heap broken
Binary heap insert

insert(12)
insert(7)
insert(1)

Heap broken
Binary heap insert

insert(12)
insert(7)
insert(1)

Heap broken
Binary heap insert

insert(12)
insert(7)
insert(1)

Heap broken
Binary heap insert

insert(12)
insert(7)
insert(1)
Binary heap insert

insert(12)
insert(7)
insert(1)
Binary heap removeMin

removeMin()
Binary heap `removeMin`
Binary heap removeMin

removeMin()
Binary heap removeMin

removeMin()

Heap broken
Binary heap removeMin

removeMin()

Heap broken
Binary heap removeMin

removeMin()

Heap broken
Binary heap removeMin

removeMin()

Heap broken

percolateDown
Binary heap `removeMin`

`removeMin()`
Binary heap: removeMin() runTime
Binary heap: removeMin() runTime

\[ \text{findLastNodeTime} + \text{removeRootTime} + \text{numOfSwaps} \times \text{swapTime} \]
Binary heap: `removeMin()` runtime

\[
\text{findLastNodeTime} + \text{removeRootTime} + \text{numOfSwaps} \times \text{swapTime}
\]

\[
n + 1 + \log(n) \times 1 = O(n)
\]
Binary heap: removeMin() runtime

\[
\text{findLastNodeTime} + \text{removeRootTime} + \text{numOfSwaps} \times \text{swapTime}
\]

\[
n + 1 + \log(n) \times 1 = O(n)
\]

How can we do better?
findLastNodeTime + removeRootTime + numOfSwaps * swapTime

\[ n + 1 + \log(n) \times 1 = O(n) \]

How can we do better?
What do we make efficient in the above expression?
Array implementation of a complete binary tree
Array implementation of a complete binary tree
Binary heap: Array implementation
Binary heap: Array implementation

Indices

0 1 2 3 4 5 6 7 8 9 10 11 12
Binary heap: Array implementation
Binary heap: Array implementation
Binary heap: Array implementation
Binary heap: Array implementation

leftChild(i) = 2i

rightChild(i) = 2i + 1

parent(i) = i / 2