Lecture 1

Monday, August 26, 2013
5:54 PM

Administrivia

- TA: Jiangwei Pan
- Info sheet + website
- Midterm + Final + 6 assignments
- Scribe / Grade
- signup sheet / Mailing list
- Saxai and Piazza

Fibonacci Heaps

Dijkstra's Algorithm

```
d[s] ← 0 ; ENQUEUE(s) in Q
for i = 1 to n-1 do
  u ← DEQUEUE-MIN(Q)
  for each (u,v) ∈ E do
      if (d[u] + l[u,v] < d[v])
          d[v] ← d[u] + l[u,v]
          if v ∉ Q
              ENQUEUE(v) in Q
          else
              DECREASE-KEY (v, d[v]) in Q
          endif
      endif
  endfor
endfor
```

Data structure to implement Q?

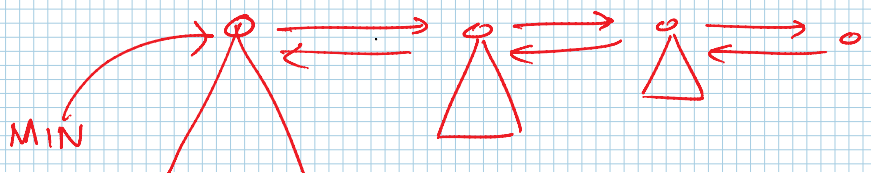
- MIN-HEAP: $O(\log n)$ time for all operations
  Dijkstra has $O(n \log n)$ running time (dominated by $O(m)$ DECREASE-KEY operations)

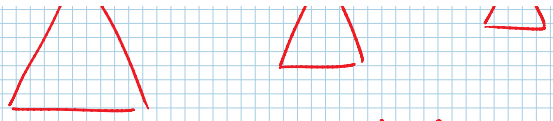
Q: Can we do DECREASE-KEY in $O(1)$ time and other operations in $O(\log n)$ time?
(Note: There are $O(m)$ DECREASE-KEY operations and $O(n)$ ENQUEUE/DEQUEUE-MIN operations!)

Fibonacci Heaps achieve this, thereby reducing the running time of Dijkstra to $O(m + n \log n)$.

Amortization: Individual operations can be expensive, but will lead to structural simplification that makes future operations cheaper. Overall, the average cost of each operation will be smaller than the worst-case cost of an operation.

Fibonacci Heaps were introduced by Fredman and Tarjan in 1984.
Many min-heaps (recall: key[parent] ≤ key[child]) with external pointers to min

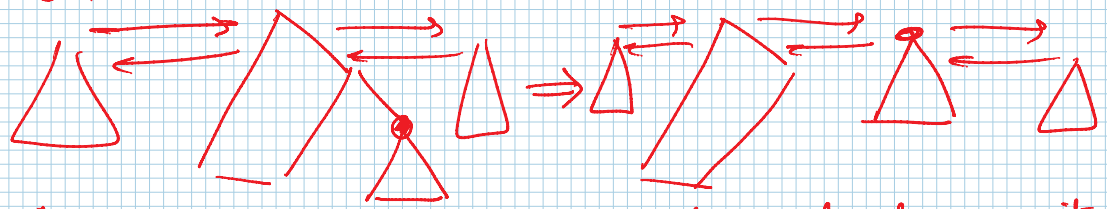


MIN 

- ENQUEUE : Add a new singleton tree : O(1) time

Problem : If we enqueue n elements and then try to implement a DEQUEUE-MIN operation, it will take O(n) time!
Indeed it will !!! But remember, we are interested in amortized running times - not worst-case bounds.

- MERGE TREES : Compare roots and make the larger root a child of the smaller one ; update root list.
- CUT A NON-LEAF NODE :



- DECREASE-KEY : Cut the node and decrease its key
- DEQUEUE-MIN : Remove MIN (external pointer) and add children to list of roots.

Problem: Updating the MIN pointer takes time $\propto$ # of roots
Solution: When we find a MIN, we also do additional work to consolidate trees and make the next MIN cheaper.

Rank : # of children
Rule : When finding MIN, merge heaps until each rank value has ≤ 1 heap. (Note: Mergers are free with MIN operations!)

Ideally, every heap should have exponential # of entries in its rank. ($2^i + 2^i = 2^{i+1}$) to avoid super-logarithmic ranks.

(Recall : # of distinct ranks = cost of MIN)

Problem : How to ensure this when nodes are cut due to DECREASE-KEY operations?

Mark : A node is marked if it loses

are cut due to [illegible]

Marking: A node is marked if it loses a child. Root nodes are always unmarked. After cutting a node, move up towards the root cutting marked nodes until you reach an unmarked node which you now mark. The cut nodes are unmarked since they are roots now.

Analysis: $\phi = K$ (# of roots + 2 · # marked nodes)

K ← constant to be determined later

- ENQUEUE: $O(1)$ real work; $\Delta\phi = K = O(1)$
- DEQUEUE-MIN: # of roots = r; # of new roots = $O(\log n)$

  $O(r + \log n)$ real work; $\Delta\phi = K(O(\log n) - r)$

  For large enough K, overall = $O(\log n)$
- DECREASE-KEY: "cascading cuts"

  $O($# of cut nodes$)$ real work;

  # of cut nodes = c

  $\Delta\phi = -2Kc + Kc + O(K)$

  For large enough K, overall = $O(K) = O(1)$

What remains? To show that every heap of rank r has α^r nodes where α is some constant > 1.

Lemma: The ith child to be added to a vertex has rank $\geq i - 2$.

Proof: When it was added, it had rank at least $i-1$ (equal rank roots are merged). It couldn't have lost more than one child without being cut.

Lower bound on heap size

$$S_k \geq \sum_{i < k-2} S_i$$

$$\Rightarrow S_k - S_{k-1} \geq S_{k-2}$$

$$\Rightarrow S_k \geq S_{k-1} + S_{k-2}$$

Now you know why it's called a fibonacci heap!