Priority Queue with Heap
Prim's Algorithm Revisited
Proofs

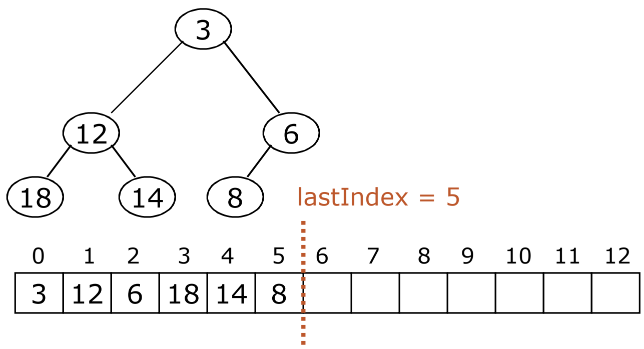
CSCI 3100

Review & Overview

Last week
- Minimum spanning trees (MST)
- Kruskal's algorithm
- Prim's algorithm
- Priority queue (started

Today
- Priority queue implementation using a heap
- Help with proofs (based on your feedback)
- Prim's algorithm revisited

Array representation of a heap



Left child of node i is 2*i + 1, right child is 2*i + 2

- Unless the computation yields a value larger than lastIndex, in which case there is no such child

Parent of node i is (i – 1)/2

- Unless i == 0

Using the heap

To add an element:

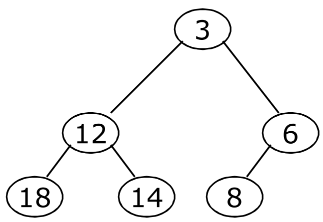
- Increase lastIndex and put the new value there
- Reheap the newly added node by swapping with parent node, until heap property is restored
  - This is called up-heap bubbling or percolating up
  - Up-heap bubbling requires O(log n) time

To remove an element:

- Remove the element at location 0
- Move the element at location lastIndex to location 0, and decrement lastIndex
- Reheap the new root node (the one now at location 0) by swapping with smallest child element until heap property is restored
  - This is called down-heap bubbling or percolating down
  - Down-heap bubbling requires O(log n) time

Thus, it requires O(log n) time to add *and* remove an element

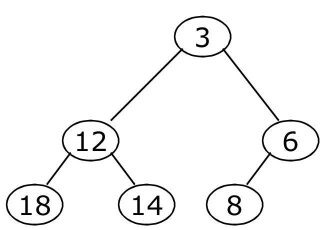
Example: up-heap (percolate up)



lastIndex = 5

0	1	2	3	4	5	6	7	8	9	10	11	12
3	12	6	18	14	8							

Example: down-heap (percolate down)



lastIndex = 5

0	1	2	3	4	5	6	7	8	9	10	11	12
3	12	6	18	14	8							

Comments

A priority queue is a data structure that is designed to return elements in order of priority

Efficiency is usually measured as the *sum* of the time it takes to add and to remove an element

Simple implementations take O(n) time

A heap implementation takes O(log n) time

Thus, for any sort of heavy-duty use, a heap implementation is better

Proof or Fluff

Let T=(V, E) be a tree. Prove that |E|=|V|-1

Consider the following proof by induction on V:

Base case: Clearly, this is true for |V|=1 and |V|=2.

Inductive hypothesis: suppose true for trees with |V|-1 vertices. Then this tree has |V|-1-1 edges. We can construct another tree by adding one new vertex and connecting it to one of vertices in the tree with one edge. Thus, we have a tree with |V| vertices and |V|-1-1+1= |V|-1 edges. This proves that |E|=|V|-1.

A. This proof is valid

B. This proof is flawed because it proves that there exists a tree with |E|=|V|-1 and not the general case

C. This proof is flawed because it doesn't prove that the newly constructed tree is in fact a tree.

D. This proof is flawed for some other reason

Let T=(V, E) be a tree. Prove that |E|=|V|-1

Consider the following proof by induction on V:

Base case: Clearly, this is true for |V|=1 and |V|=2.

Inductive hypothesis: suppose true for trees with n<|V| vertices.

Let T be a tree with |V| vertices. Let e be an edge connecting vertices u and v in T. Since T is a tree, there is a unique path from u to v and it has to be via edge e. If we remove e, T will become disconnected. Now T-{e} consists of two components T_1 and T_2 and those components are trees (since there were no cycles in T to begin with).

Let n_1 be the number of vertices in T_1 and n_2 be the number of vertices in T_2, so $n_1+n_2=|V|$.

Also $0 < n_1 < |V|$ and $0 < n_2 < |V|$. By inductive hypothesis the number of edges in T_1 is n_1-1 and the number of edges in T_2 is n_2-1. Thus, the number of edges in T is $n_1-1+n_2-1+1=n_1+n_2-1=|V|-1$.

A. This proof is valid

B. This proof is flawed because $n_1+n_2=|V|$ is false

C. This proof is flawed because T_1 and T_2 are not guaranteed to be trees.

D. This proof is flawed for some other reason

Claim: If G is an undirected graph on n vertices, where n is an even number, then if every vertex of G has a degree of at least n/2 then G is connected.

Proof: Assume, G is not connected, so there are at least two connected components c_1 and c_2. Since every vertex must have degree of at least n/2, a vertex in c_1 is connected to at least n/2 other vertices i.e. there are at least (n/2)+1 vertices in c_1. Similarly, in c_2 there must be at least (n/2)+1 vertices.

This gives total number of vertices n/2+1+n/2+1=n+2 which is a contradiction since we have only n vertices. Hence G must be connected.

A. This proof is valid

B. This proof is flawed because there may be more than two connected components

C. This proof is flawed because it assumes that G is not connected

D. This proof is flawed for some other reason

Claim: If G is an undirected graph on n vertices, where n is an even number, then if every vertex of G has a degree of at least n/2 then G is connected.

Proof: Assume that G is connected. Since it is connected, then by definition there exists a path between any two vertices, and there must be at least n=2 vertices in G. Each vertex in G has a degree of at least one.

Adding edges to a graph that is already connected (in order to satisfy the requirement that every node has a degree of at least n/2) does not destroy its connectivity, and so the claim is true.

A. This proof is valid

B. This proof is flawed because you cannot add edges to a graph

C. This proof is flawed because it assumes that G is connected

D. This proof is flawed for some other reason

Prim’s algorithm with priority queue

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

Prim’s algorithm example