Data Structures and Algorithms Study Guide

Josepeh Sepich Oct 2

1 Running Time

One could use any measure of running time to classify an algorithm. Best time, average time, and worst case running time are all valid calculations; however using running time as the worst case running time, because better worst case guarantees a faster algorithm (usually).

We define the follow notation:

- O - big o notation denotes worst case running time (or upper bound)
- Ω - Big Omega notation denotes best case running time (or lower bound)
- Θ - Big Theta notation denotes something that is both upper and lower bound (exact running time)

When comparing running times remember if it is not exact (if f is bigger than g or smaller), then you can transform the function into an easy function to handle that is bound by that easier function. Another good trick to analyze running times is to use logairthms to deal with exponents. Also recall the order of precendence form constant to logarithmic to polynomial to exponential to factorial in ascending running time order.

2 Master Theorem

Master theorem is a shortcut to help analyze divide and conquer algorithm. It is based off using the height of a recursive tree and the amount of work done at each level to analyze the total running time.

Suppose we have a time of an algorithm in the form $T(n) = aT(\frac{n}{b}) + O(n^d)$

We have three cases:

1. If $d > log_b(a)$ then O(n^d)
2. If $d = log_b(a)$ then O($n^d log(n)$)
3. If $d < log_b(a)$ then O($n^{log_b(a)}$)

3 DFS

DFS is depth first search algorithm for traversing graphs. It runs in O(|V| + |E|) time. DFS algorithm is implemented using a stack (recursion stack).

Implementation of DFS including pre and post numbers:

```
DFS(G) {

  pre = new int[|v|]
  post = new int[|v|]
  int clock = 1
  visited = new boolean[|v|]
```

```
  foreach (w in V) {
    if (!visited[w]) {
      Explore(G, w);
    }
  }
}

Explore(G, w) {
  visited[w] = true;
  pre[w] = clock;
  clock++;
  foreach (edge {w,v} in E) {
    if (!visited[v]) {
      Explore(v);
    }
  }
  post[w] = clock;
  clock++;
}
```

A DFS tree is the tree created by the traversal of the DFS algorithm. THere are four types of edges in the DFS tree: tree edges, forward edges, back edges, and cross edges. Tree edges are edges found in the tree. Forward edges are an edge from an ancestor to a non-child descandent vertex. Back edges are the opposite of a forward edge and lead from a descendant to an ancestor. A cross edge is an edge from a vertex to another that is neither an ancestor or descendant, which implies it has already been assigned a post number.

4 DAG

A DAG is a directed acyclic graph. These graphs can be organized in topological order and exist if a DFS algorithm reveals no back edges. To find a topological sort run DFS and then order the numbers in decreasing post number order.

A strongly connected component exists if there is a connection from v to w such that you can go from v to w and then back to v.

Finding strongly connected components:

1. Build adjacency list for GR.
2. Run DFS on GR, assign pre and post numbers.
3. Sort vertices by their post numbers.
4. Run the undirected connected component algorithm in order of increasing post number.

```
color = 1
for (i=1 to n) {
  if (visited [i] == 0) {
    Explore(v_i, color) // in visited where pre number is assigned, assign color
    color ++
  }
}
```

5 BFS

BFS is breadth first search algorithm for traversing graphs. It runs in O(|V| + |E|) time. BFS is implemented using a Priority Queue.

Implementation of BFS including distance:

```
BFS-Explore(G, s){
  dist = new int[|V|]
  foreach (v in V) {
    dist[v] = infinity
  }
  dist[s] = 0
  Q is a queue
  add s to queue
  while (Q is not empty) {
    v = eject(Q)
    foreach ({v,w} in E) {
      if(dist[w] = inf) {
        add w to Q
        dist[w] = dist[v] + 1
      }
    }
  }
}
```

Dijkstra to find shortest path.