BRAC University

Algorithms

Running Times of Graph Based Algorithms

Rubayat Khan
12/5/2015

1 BFS:

We are given a graph with V vertices and E edges. At first we create an array of V vertices.

```
for every vertex v in graph G{
        colour[v] = white
        pi/parent[v] = null
        level[v] = -1
}
```

As there are V vertices and we work for each, this loop runs V times.

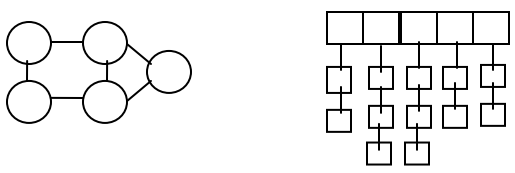
```
enqueue s in queue Q        [O(1)]
colour[s] = grey            [O(1)]
level[s] = 0                [O(1)]

while (Q not empty){        [this loop will enque/check all the vertices, hence will run V times]
        d = dequeue[Q]
        for each neighbor vertex u of d{
```

[this loop will run (V-1) times in the worst case because one vertex might be connected to all other vertices. V-1 can be written an V]

```
                if (color[u]==white){
                        enqueue[Q]=u
                        colour[u] = grey
                        level[u] = level[d]+1
                        pi/parent[u]=d
                }
        colour[d]=black
}
```

a) using a matrix will give a $O(V+V^2) = O(V^2)$ because we have to traverse each and every slot of the matrix to check whether there is a connection between vertices or not.

b) In case of an adjacency list the run time will be V + (V + E) which is O(V+E). Now the question is if there is a nested loop then why there is a plus? Shouldn't it be VxV or VxE ?



Above shows the list representation of the given graph. Every vertex needs to be traversed which is basically traversing the ARRAY, [V times]. Now for each vertex not all

the edges are traversed at the same time. For any vertex only those edges are visited that are connected to it. Therefore for each slot in the array, only the number of nodes connected to it are traversed. In case of the above diagram, for the first vertex only 2 nodes are traversed. There are 5 vertices and 6 edges. If we count the number of traverses we did is all vertices + edge of each vertex(which is all edges); 5 +6 = 11. There proved the runtime of BFS is O(V + E).

2. DFS

Same as BFS. Both BFS and BFS traverse in the similar way except for the fact that one uses a queue and the other a stack(recursion).

3. Toposort

The algorithm of topological sort is:
DFS [O(V+E)]
Sort [O(VlgV)]
Total runtime = O((V+E) + (VlgV)]
**** Find out the running time of finding the number of strongly connected components.**

4. Dijkstra

Calculating the run time of dijkstra is critical. In this algorithm we need to extract the vertex with minimum value and also update the value to a lower one is there is a shorter path. Therefore if the vertices are stored in a linear array searching to find the minimum values vertex will take O(n). Can we do better? What is the worst case of extracting min from min heap? It is lgn, better than n. Thus we will be using a heap data structure to store the vertices [value, parent].

```
Q = all the vertices
while (Q!empty){      [O(V)]
        u = extract the vertex with min value       [O(lgV)]
        for all vertex v adjacent to u{       [O(E) while using adjacency list. Reason same
                                                as BFS, DFS]
                if (v is in Q && d[v]>d[u]+w(u,v)){
                        d[v] = d[u] + w(u,v)  [O(lgV), because we are using a heap, changing
                                                a value of node requires swim operation]
                         p[v] = u
                }
        }
}
```

Let us compute the running time. The outer loop runs V times. If one extraction takes lgV then V times take VlgV. The inner loop runs E times and as modifying the value lgV times, time for this portion is ElgV times. Therefore the total time complexity is O(VlgV + ElgV). We can keep it like this or further simplify is to ElgV as the number of edges E >= V.

6. Bellman: O(VE) Find out how.

7. Huffman: We know the domain of the plain text then counting the number of occurrences of each character takes O(n). Then we sort them so that each time we can pick the character with the minimum frequency. Sorting takes O(nlgn). Then for each character we create the tree and this takes O(n). Total running time T = O(n) + O(nlgn) + O(n), which can approximated to O(nlgn).
A min heap can also be used for Huffman encoding. For that too the running time is O(nlgn).
****How long does decoding take?****