Anders Maraviglia

3/19/2014

Homework 2

2.24) a) Procedure quicksort(list lst){

Let low-elts = {}

Let high-elts = {}

Let pivot-pnt = a median value from the first, last, and middle values of lst

Remove pivot-pnt from lst

Foreach val in lst{

If val <= pivot-pnt:

Append val to low-elts

Else append val to high-elts

}

Return quicksort(low-elts) + pivot-pnt + quicksort(high-elts)

}

b) When pivot is the last point in the array, the algorithm is forced to recursively go over every element in the array as each new list going into quicksort is one less than the previous one, and when adding each call together results in the runtime:

2.29) a) Horner’s rule satisfies the requirements of the job and leaves the answer in z because its set to the relative position n, which is the last element in the polynomial. That number is multiplied by x every time, which is why n is multiplied by it under the for loop, and the addition between each is shown in the variable z.

b) This function does both n number of multiplications and additions. What would make this more efficient would be to break the polynomial down into sub problems and solve those individually, once at the lowest level, and then add each one together at the end.

3.4) a)

Graph i:

First: {C, D, F, G, H, I, J} source SCC

Second: {A. B, E} sink

Adding 1 edge from a vertex from sink to one in the source SCC would make it a strongly connected graph.

Graph ii:

SCC: {D, F, G, H, I, J} sink

SCC: {C}

SCC: {A, B, E} source SCC

Add 1 edge from a vertex to one in source SCC would make it a strongly connected graph

3.6) a) Edges u and v each contribute 1 to d(u) and d(v) respectively and thus each contribute 2 to the final sum.

b) = 2|E| - . V1 is all the odd degree vertices and v2 is all the even degree vertices. The right side of the equation is always going to be even and the left will always be the sum of the odds. Odd number sums are even if and only if it’s the sum of an even number of odds. Therefore V0 is even.

c) No, this is not the case.

3.7) a) Let the sets V1 and V2 each be an identifiable variable like squares and triangles. Do a DFS on both and recognize each level of the tree as red or green. If there is combination of the two colors at every level of the tree then it is indeed bipartite.

b) Let u and v be vertices such that (u, v) has one colored back edge, and also make it so that u is also an ancestor to v. Going from u to v must be even since they are one color, and this combined with the back edge would create an odd cycle.

c) Three colors can be used to fully shade in the graph if the graph in question had exactly one odd cycle.

3.11) A graph will have a cycle that contains e = (u, v) if and only if u and v are within the same connected component. One could do this in linear time by using a depth first search to check for u and v.

3.13) a) Doing a DFS from any vertex in a graph proves removing a single vertex will still leave the graph connected. If one vertex is removed the result is still a tree, and thus also a connected sub graph of the initial graph after the removal of that vertex. Thus the graph is still connected after that vertexes removal.

b) If the entire graph is a directed cycle then no vertices can be removed.

c) When there are 2 separate disjointed cycles that are also each strongly connected within said cycles. If one edge is added between the two, we can go from one graph to the next but not in the other direction.

4.1)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Node | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| B |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| C |  |  | 3 | 3 | 3 | 3 | 3 | 3 |
| D |  |  |  | 4 | 4 | 4 | 4 | 4 |
| E |  | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| F |  | 8 | 7 | 7 | 7 | 7 | 6 | 6 |
| G |  |  | 7 | 5 | 5 | 5 | 5 | 5 |
| H |  |  |  |  | 8 | 8 | 6 | 6 |

4.2)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Node | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| S | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A |  | 7 | 7 | 7 | 7 | 7 | 7 |
| B |  |  | 11 | 11 | 11 | 11 | 11 |
| C |  | 6 | 5 | 5 | 5 | 5 | 5 |
| D |  |  | 8 | 7 | 7 | 7 | 7 |
| E |  | 6 | 6 | 6 | 6 | 6 | 6 |
| F |  | 5 | 4 | 4 | 4 | 4 | 4 |
| G |  |  |  | 9 | 8 | 8 | 8 |
| H |  |  | 9 | 7 | 7 | 7 | 7 |
| I |  |  |  |  | 8 | 7 | 7 |

4.4)

C

B

A

E

D

Vertexes edges are solid black, back edges are dashed. The cycle that the algorithm will find is {A, B, C, D}, while the shortest cycle is actually {A, D, E}.

4.11) Using a matrix M and Mxy would be the shortest path from x to y. Computing row x of the matrix with Dijkstra’s gives us a runtime of O(|v|2) and thus we can then calculate all of M in a runtime of O(|v|3).

4.13) a) Preform s DFS from vertex s while ignoring all weights greater than L to determine in linear time whether there is enough fuel.