ASSIGNMENT 7

Chitrarth Singh
A20387080

HOMEWORK ASSIGNMENT-7

1.

(a). BFS-WATERJUG

- Initially, the waterjug object is in the state (0, 0, 0)
- Add the object with initial state to Queue.
- do
- Check If the waterjug object at the start of queue is the required goal.
- If yes then,
- Print the visited nodes
- Exit
- Else
- Create all possible valid states of child to current state
- Add these possible states to the queue.
- Add this current state to the visited nodes array
- The array stores all the traversed nodes.
- Dequeue the current object.
- End If
- while queue is not empty.
- End.

ASSIGNMENT 7

(b) Correctness of the Algorithm.

- The algorithm explores all the possible node and hence we always get a finite count of the no. of nodes
- The algo. has finite states.
- With finite states the BFS is visiting all node in the graph.
- In case of impossible state it is not possible to plot the graph.

(c) Worst case Time complexity occurs, when all the nodes are visited. If there is a node, then the time required will be the last node visited.

So the complexity is:-

O (water jug total limit)

so $O(C_1 C_2 C_3)$

ASSIGNMENT 7

2.

Solution :-

We construct a directed graph G and test if it is acyclic.

Let's define b_i & d_i nodes for each person corresponding to their birth & death dates.

Edges will correspond to one event preceding another.

Let's start by including edges b_i, d_i

- For each i, we know that
- For some i & j, P_i died before P_j was born
- Further we include an edge d_i, b_j when for some i & j the lifespans of P_i & P_j overlapped at least partially

Then we add edges (b_i, d_j) & (b_j, d_i) to the graph G.

If G has a cycle -
- the events corresponding to nodes in the cycle must precede the next but this applies that no event among these can be put first in time and gives inconsistency. Therefore if there's a cycle the information is not internally consistent. To find the cycle we use DFS data structure

ASSIGNMENT 7

If G has no cycle then the information is consistent if we the events in the order of the birth & death dates of all people.

ASSIGNMENT 7

(3) Problem 23-1

Solution:

(a) MAYBE-MST-A(G, w)

1. sort the edges into nonincreasing order of edge weights w.
2. $T = E$
3. For each edge e taken in nonincreasing order by weight
4. If $T - \{e\}$ is a connected graph.
5. $T = T - \{e\}$.
6. return T.

The algorithm takes a greedy approach and removes edges in nonincreasing order as long as the graph is still connected.

Proof:-

Let M be a MST of graph G. Now each time we remove an edge e, either $e \in M$ or $e \notin M$.

If $e \notin M$ then we can remove it.

If $e \in M$ then removing e would make M a disconnected graph. The edge 'e' must be the smaller edge then others because we have assumed M to be a MST. If there exist a smaller edge than e then the algorithm could have chosen the smaller edge but since edge 'e' is a

ASSIGNMENT 7

part of M therefore all the edges which are of larger term e must have been removed by the algorithm.
Further, there may exist a path which is undiscovered and equal to the edge 'e'. In this case we can remove e and and attach, here undiscovered path and mark it visited.

Most efficient Implementation:

The algorithm MAYBE_MST_A(G, w) finds a MST and we can use adjacency list representation for T.
We can use merge-sort to sort the edges in $O(E \log E)$.
We use BFS or DFS to check if $T - \{e\}$ is a connected graph. This will take $O(V+E)$ time.

∴ Total time: $O(E \log E + E(V+E))$
$= O(E^2)$

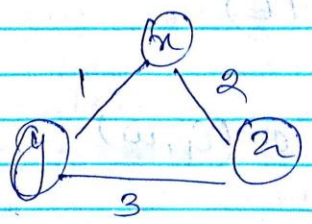
ASSIGNMENT 7

(b) MAYBE-MST-B (G, w).

1. T = ∅
2. For each edge e, taken in arbitrary order.
3. if T ∪ {e} has no cycles.
4. T = T ∪ {e}
5. return T.

Solution:

- The above algorithm does not follow greedy approach and arbitrarily chooses the edges and adds to the tree T.



- Since the edges are added in the arbitrary order if we first add (x, y) and (z, x) to the MST of G we get a weight of 3.
Now if first (y, z) and (x, z) is added to T, then a new edge (x, y) is added then the cycle is formed and the weight return would be T=5.

Implementation :-

To maintain T, use a UNION-FIND data structure.

ASSIGNMENT 7

For each vertex we need to MAKE-SET(v)
For each edge e = (u, v) if
FIND(u) ≠ FIND(v) then there is no
cycle in T ∪ {e}. and we UNION(u, v).

Total :- V MAKE-SET operations. —
2E FIND-SET operations.
V - 1 UNION-SET operations.

Total Running time :-

MAKE-SET + FIND-SET + UNION-SET
⇒ O(V) + O(E) + O(V log V)
⇒ O(V log V + E)

(c). MAKE-SET-C (G, w)

1. T = ∅
2. For each edge e, taken in arbitrary order
3. T = T ∪ {e}
4. if T has cycle c
5. let e' be a maximum-weight edge on C
6. T = T - {e'}
7. return T.

Solution :- The above algorithm gives a minimum spanning tree. It works by adding arbitrary edges to T

ASSIGNMENT 7

and checking if a cycle is formed or not.
If a cycle is formed then the maximum weight edge is removed.
Each time an edge is added either a cycle is formed or not.
If a cycle is detected after adding an edge e to some tree T' in T. In this case we remove the maximum-weighted edge e' from the cycle and T' - e' + e is less or equal weight than T' (because e' ≥ e)
If no cycle is formed we either add e to some tree in T or e connects two trees in T. In the second case if e is not the smallest weight edge that connects the trees then we haven't discovered it yet.
Thus the algorithm performs correctly if a cycle is formed by removing the maximum weight edge.

Implementations

We use adjacency list representation for T.
For each edge, we add it to T and check if T ∪ {e} forms a cycle.
There can be at most one cycle so in this case we use bfs to detect for cycle
If no cycle detected then the edge is kept.

ASSIGNMENT 7

otherwise output the cycle, find the maximum-weight edge & delete it from T.

Here we add an edge it takes constant time of O(1).

DFS takes O(V+E). In this algorithm as soon as a cycle is formed we break it by removing one edge of max-weight in T. So the numbe of edges in T at any point is greater than V.

Thus DFS takes O(V).

finding the maximum weight will take O(V) time and deleting an edge will take O(V) time.

Thus the final running time is O(EV)

ASSIGNMENT 7

REFERENCE

http://s3.alirezaweb.com/91-5/introduction-to-algorithms/solution-manual/CLRS-Exercises-Introduction-to-Algorithms_Borna66/CLRS-Introduction-to-Algorithms/H20-solution[www.alirezaweb.com].pdf

http://homepage.divms.uiowa.edu/~kvaradar/fall2014/hw3.pdf

http://projectsgeek.com/2011/09/water-jug-problem-artificial-intelligence.html