Yueling Qin

CS 4800

Problem 1

1. Third-min (A):

If n<3,

We have A [1,1], A [1,2], A [1,3]

A [2.1]

A [3,1]

If A [1,2]=A [2,1]

Return A [2,1]

If A [1,2]<A [2,1]

Return Min (A [1,3], A[2,1])

If A [1,2]>A [2,1]

Return Min (A [3,1], A [1,2])

For this problem, we have Θ (1), independent of the size.

1. Changing A[i , j] to k.

If A[I, j]<k,

Compare A[i+1, j] with A[i, j+1]

If A[i+1, j] and A[i, j+1] >= A[i, j], finish.

Otherwise, find Min (A[i+1, j] and A[i, j+1])

Swap with A[i, j]

Keep doing this until we do not find any violation. To A[n.n]

If A[I, j]>k,

Compare A[i-1, j] with A[i, j-1]

If A[i-1, j] and A[i, j-1] <= A[i, j], finish.

Otherwise, find Max (A[i-1, j] and A[i, j-1])

Swap with A[i, j]

Keep doing this until we do not find any violation. To A[1,1]

For each iteration it takes O (n).

Problem2

1. cannot be satisfied, if we assume it right, we go through the whole statement,

(x1=x3)>(x5=x6)>(x2=x4), but we have to make x2>x3, it’s a violation here,

so we can justify this statement is wrong.

1. 1.Build an directed graph G in which have a vertex for each x, and an edge between

xn and xm when they are non strict inequality constraint.

2.count the strongly connected components In G, if two x are in same strongly connected component, then they are equal because they are part of a cycle containing <=.

3.Build the meta-graph G’ of strongly connected components of G, then process the strict inequality constraints. For each strict inequality constraint xn<xm, add an edge from xn’ to xm’ in G’.

4.is G’ have a cycle, if is, constraints are inconsistent,

return No,

5.else, G’ is a dog, count the topological ordering, set a value to each component in G’ in increasing order of sort,

return the value of the component for each variable.

So ,for each one is Θ(n + m).we have running time, Θ(n + m).

Problem3

Firstly, we need to find a no-incoming vertex, then using DFS in G,

when we go over graph, we always go to the vertex that have smaller number of incomings until we finish DFS. Finally, we need to check is visiting every vertex in G exactly once, compute the size of simple path in G.

so for this process,

Θ(1)

Θ(n + m)

Θ(n + m)

Θ(1)

Θ(n + m)

So, we have a linear running time of this problem.

Problem4

1.find no incoming vertexes, because we have large robot, we can do some thing at same time, we just count time once for these.

2.create a list for non incoming vertexes

3.find connected vertexes with non incoming vertexes, based on rule, we also just count time once.

4.add those connected node to list we create.

5.keepping track the process until we done all graph(finish all tasks)

So, for these problem, we have linear time Θ(n + m) running time.

Problem5

1. go through triples, create 2 vertices for each and add directed edges for both sides, also we need remember which vertices has already created, if we found there is same computer that time is earlier but already exist in graph, we make a single directed graph, finding Ca, X’ and Cb,y’. then we run DFS to check is there have path. It takes linear time when we add each one. Θ (n + m) running time