Mandatory Project-4

Implementation Of Data Structures

Manoj Kumar Ellanti(mxe150630)

- The edges are added from the Dummy source and Dummy destination vertex to all the vertices whose revadjList is empty and adjacency list is empty respectively. This edge addition is done in readGraph().
- These edges are stored in the graph as src and dest.
- TopologicalOrdering1() and TopologicalOrdering2() are implemented in the graph class which implements the take1 and take 2 algorithms respectively. Both returns a list<Vertex> in topological order.
- CriticalPath class contains inputG for input Graph, listOAllCriticalPAths(List<List<Vertex>> to store all the critical paths) , noOfCriticalNodes, noOfCriticalPaths, globali a global Variable used in the recursion for findcriticalpaths(). Initially, noofCriticalnodes=-2 since dest and src are critical and we don't consider them.
- earliestTimeCalculation() is implemented to set the earliest time to all the nodes in the graph. It uses TopologicalOrdering1() and process the nodes based on the list returned by the topological1().
- latestTimeCalculation() is implemented to set the latest time to all the nodes in the graph. It uses TopologicalOrdering2() to retrieve a Topological list of nodes. The list is converted to array and the nodes are processed in the reverse order to set the latest time to all the vertices.
- PrintAllVerticesDetails() is used to print the earliestTime , LatestTime, slackTime of all the vertices except src and dest.
- findCriticalPAths() is used to find all the critical paths and to add them to the listOfAllCriticalPAths.
- Intially, instead of List<List<Vertex>> an array of vertices is used to work on all possible paths. But, since array is not dynamic for larger inputs.
- A criticalPath{List<List<Vertex>>} is created to store the list of all possible critical paths and this is used to trace back in Dfs to see for a possible second path from the previous critical path.
- For ex:

    - `1 3 25 30 12 7 26 10 15 14`
    - `1 3 25 30 12 7 26 10 15 18`
    - `1 3 25 30 12 7 28 10 15 14`

- Are all critical paths criticalPAth.get(0) has 1 3 52 30 12 7 26 10 15 14 after first Dfs. Since, The Dfs now continues from 15, Now The entire List except 14 is used to calculate criticalpsth2. Since, 1 3 52 30 12 7 26 10 15 are all critical nodes .So, criticalPAth.add(1) is added with the above list .
- Since, The Critical Path Start from Src. The for loop contains Adj List of Src.
- If the other end Slack==0 and Edge is tight(). We increment the globali, and add a new LinkedList() there. Now, to that linked List this node is added as starting vertex.
- A DfsCritical(Vertex v,List<List<Vertex>>) which is kind of DFS is done on that vertex.
- For the Dfs Visit an all its adjacency list{at any point in DFS criticalPAth.get(globali) is the path that is being processed now.}
- { A path is critical if it contains all critical nodes(slack=0) and TightEdges e(u,v) {u.latestTime=v.latesttime-v.d}}.
- A flagToathWithV is used to see if a path can be constructed with v. IF so it is true , and v is added to criticalPAth.get(globali).
- It the destination is reached the node is not added but the list is added to listOfAllCriticalPAths as one of critical paths. Now, since the list is done .
- Consider it be following
    - `1 3 25 30 12 7 26 10 15 14 //criticalPAth.get(0) //criticalPAth.get(1) is list that now being processed.`
    - `Now DFS starts at 15 and look for a path to dest; //by examination 1 3 25 30 12 7 26 10 15 14 are all critical so these are already seen in one DFS. They are added to the criticalPAth.get(1). Here , 14 if the adj list is done sees whether it is used in the list using flag if so it is removed.`
    - `So, when the dFS trace back to 15. Only 1 3 25 30 12 7 26 10 15 will be there as possible path.`
    - `This continues if a node is used in the path and seems to fininsh its DFS life time it is removed from the path.`
- lengthOfCriticalPAth() is used to display last vertex latestTime.
- printAllCriticalPAths method iterates through the listOfallCriticalPAths and print each one of them.
- printOneCriticalPAth() is used to print the first critical path in the listofallcriticalpaths.
- The listOfallCriticallPAths.size() gives the total number of critical paths possible from src to dest.
- An order that is first execution of earliesttimes, Latesttimes, findcriticalpaths Is done in the same order before using any other method.