cpt_s 350

Homework 6

11641327 Yu-Chieh Wang

2020/3/20

1. The number of paths from $u \rightarrow v$.

   First of all, run the topo-sort algorithm to get a sequence A from the DAG which the first node is u. The sequence A is like what we get from BFS which is able to show the level of each node. Second, make a hash table B which's keys are elements of the sequence, and all values are 0 initially. Next, use the hash table to count how many other nodes can reach each node by following steps (psudo-code):

   For (i=0; i<len(sequence A); i++){

   Check which nodes A[i] goes next from the DAG,

   If so, B[next node] = B[next node] + B[i]

   }

   Finally, check the value of v (the node we look for) from the hash table B, and return it.

2. The number of gold paths($\#_{Green} > \#_{Yellow}$) from $u \rightarrow v$.

   We do the same steps from the previous question. After find the paths from $u \rightarrow v$, run every path to count the numbers of green and yellow nodes, and then return if the path satisfy $\#_{Green} > \#_{Yellow}$ or not.

   Finally, sum how many paths satisfy the requirement and then return the number.

3. The number of ugly paths which satisfy the regular expression γ from $u \rightarrow v$.

   First, run topo-sort algorithm to get the sequence from $u \rightarrow v$. Second, run the

dynamic program to count how many paths from $u \rightarrow v$. Next, if the result is not infinite, make a finite automata to recognized the regular expression γ. Finally, count how many paths are accepted by the automata. Else, run SCC algorithm to get the result.

4. Design an algorithm to compare the number of paths in two graphs.

   First of all, run topo-sort algorithm to get two sequences of two graphs. Next, run dynamic programming on each graph. During the running time, if any number exceeds the maximum value that can be calculated, directly determine the path of the graphic as infinite. Then, compare to the other graph to see which one has the largest number of path. If both are not infinite, return the result. Else, we can consider calculating how many points each point can connect to to determine which graph may have more paths. In general, the more complex the points are with each other, the more paths can be taken.

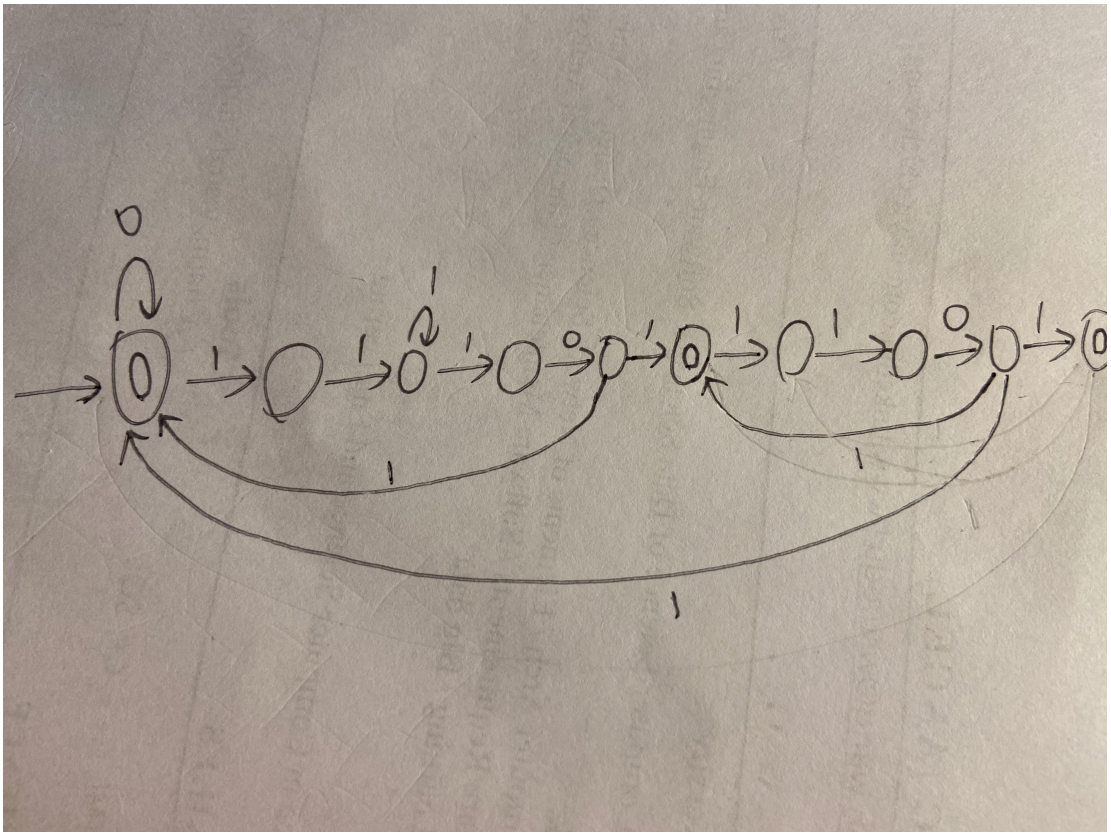
5. How graph help test a C-program?

   When running a C-program, we can see every operation as a node in a graph. In this graph, it is able to count how many paths from a node $u \rightarrow v$, which means from an operation to another operation. In this diagram, it is able to count the number of paths from one node to another operation. When a program has errors, it has some benefits. It can be used by programmers to show the many ways in which errors can occur, and to quickly grasp the scope of errors. In addition to completing a project, it is often more efficient to use a test program to detect the project for errors than to manually check. In the graph, they can set all the operations that the user can perform into a point. Then, find all feasible paths from this diagram, and actually test these paths on the project. If an error occurs in testing all possible operating procedures,

programmers can quickly locate the error and modify it. If no errors occur during the process, we can basically ensure that users will not make errors during the experience. However, if the project passes the testing process, users still find problems. The programmers need to go back and check the accuracy of the graphics to see if there are factors or procedures to consider. After revising the graphics, double check the project path to ensure the high quality of the project.

6. Design an algorithm to compute the number of binary strings with length n that satisfy the regular expression $((0 + 11 + 101)*(1101))*$.

   First of all, draw a graph of the regular expression:



   Second, enumerate all strings with length n.

   Finally, run all strings with length n on the graph, to see how many strings are accepted.