Anthony Redamonti

Syracuse university

Midterm Exam 2

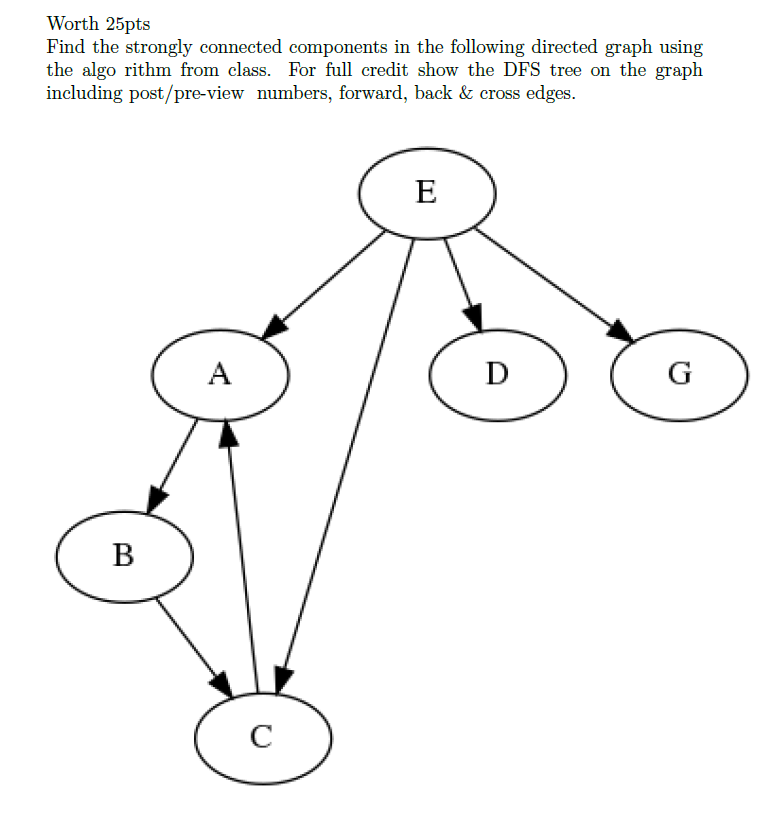
CIS-675 Design and analysis of algorithms

prof. Imani Palmer

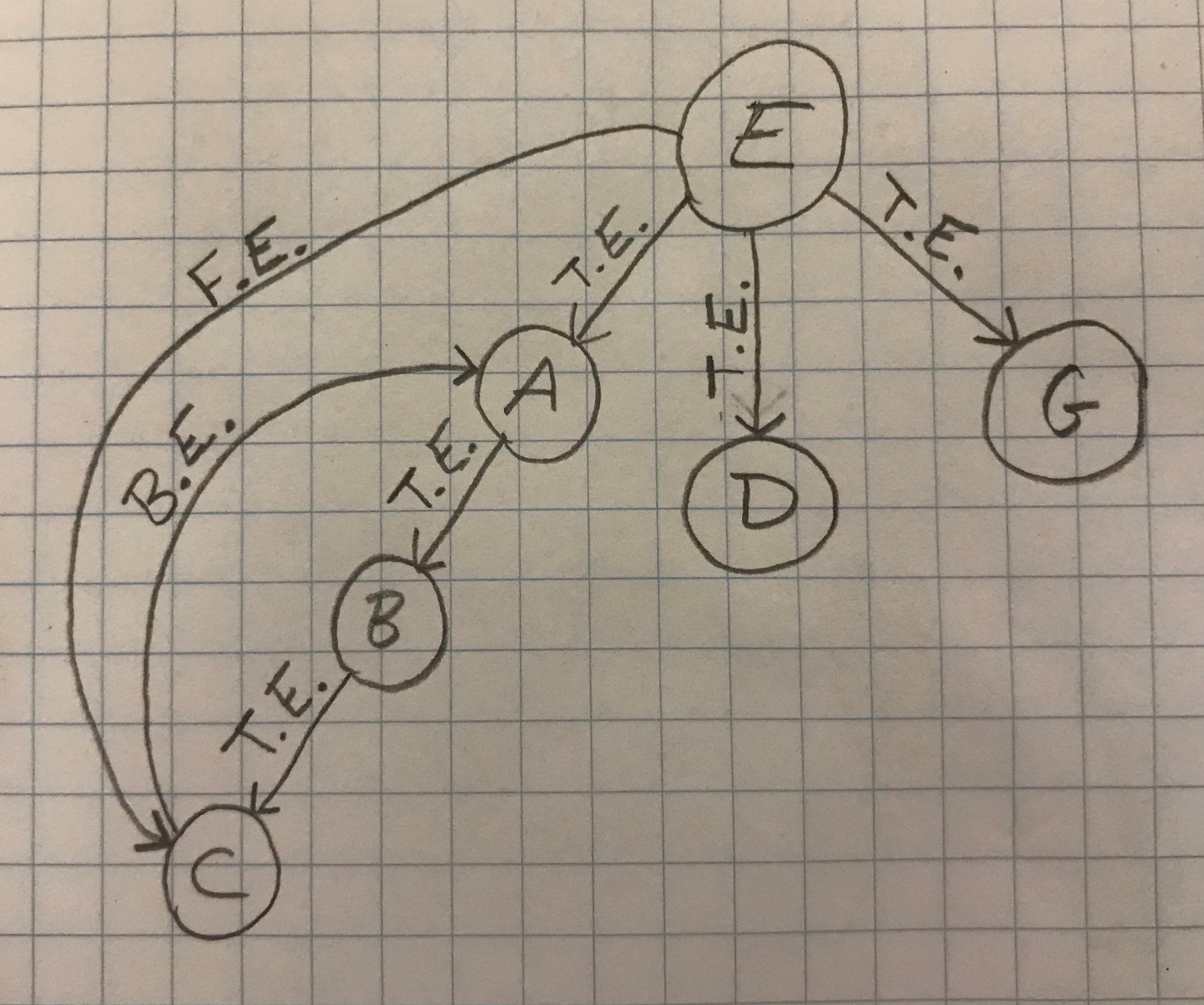
5/25/2021

Questions 1, 2, 4, and 6 have been answered.

Question 1:



The graph’s DFS tree is below with tree edges, cross edges, back edges, and forward edges labeled T.E., C.E., B.E. and F.E. respectively.



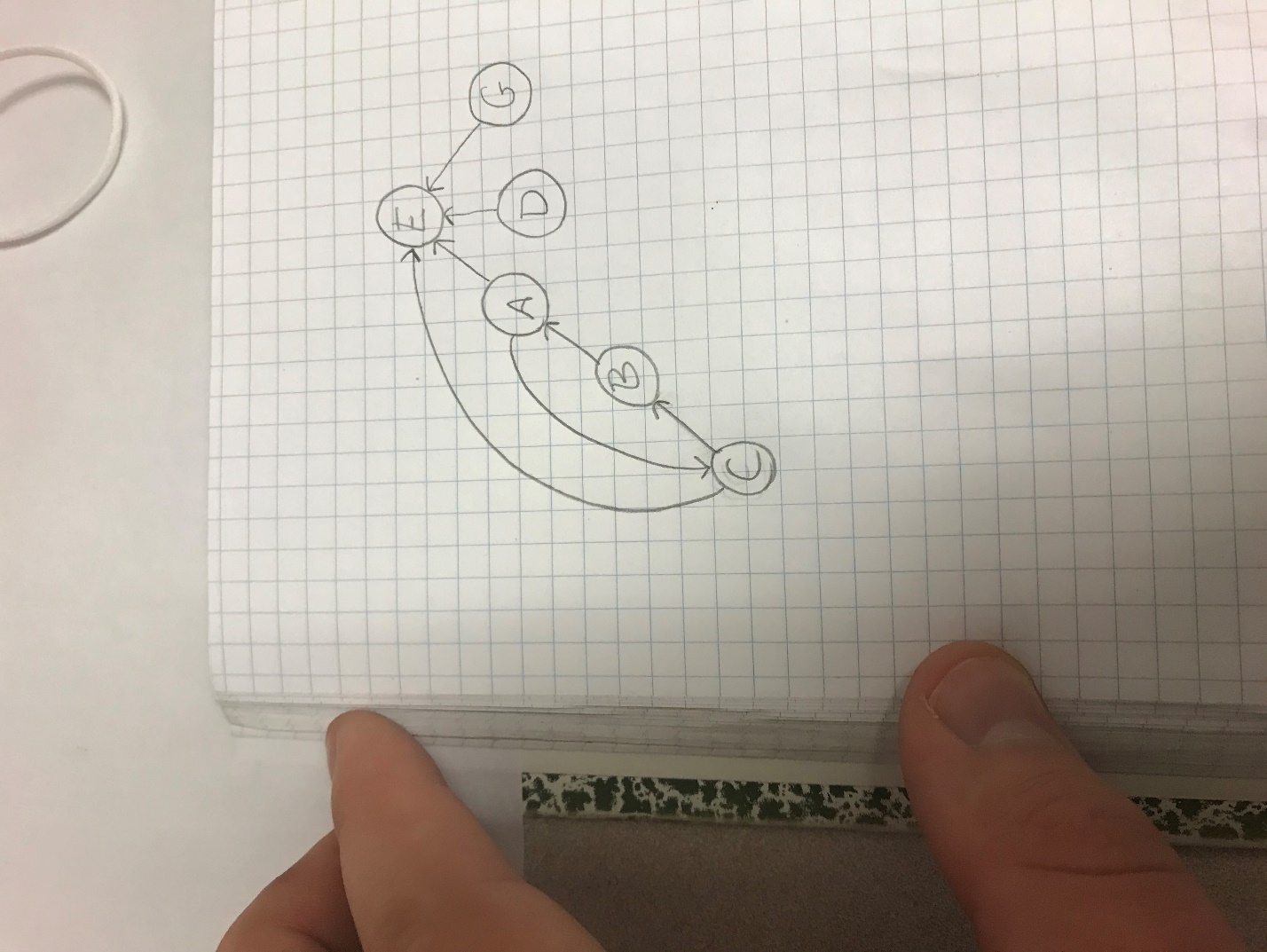
|  |  |  |
| --- | --- | --- |
| Vertex | Pre-order | Post-order |
| E | 1 | 12 |
| A | 2 | 7 |
| B | 3 | 6 |
| C | 4 | 5 |
| D | 8 | 9 |
| G | 10 | 11 |

Run the algorithm to find all the strongly connected components and provide an ordering of the nodes by the GR post-visit numbers.

The ordering of the post-visit numbers yields the following order on the stack.

|  |
| --- |
| Stack |
| E |
| G |
| D |
| A |
| B |
| C |

Now create GR, where GR is a directed graph with all edge directions reversed.



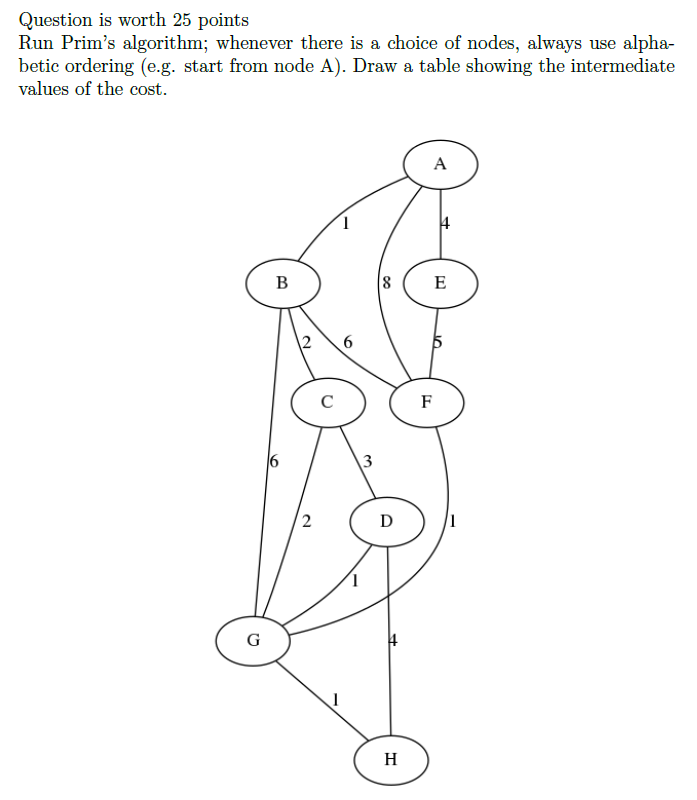
Pop the elements off the stack and perform DFS on them in GR.

|  |  |  |
| --- | --- | --- |
| Vertex | Pre-order | Post-order |
| E | 1 | 2 |
| G | 3 | 4 |
| D | 5 | 6 |
| A | 7 | 12 |
| C | 8 | 11 |
| B | 9 | 10 |

It is clear from the results of the algorithm that the following are the four strongly connected components in the graph:

* E
* G
* D
* {A,B,C}

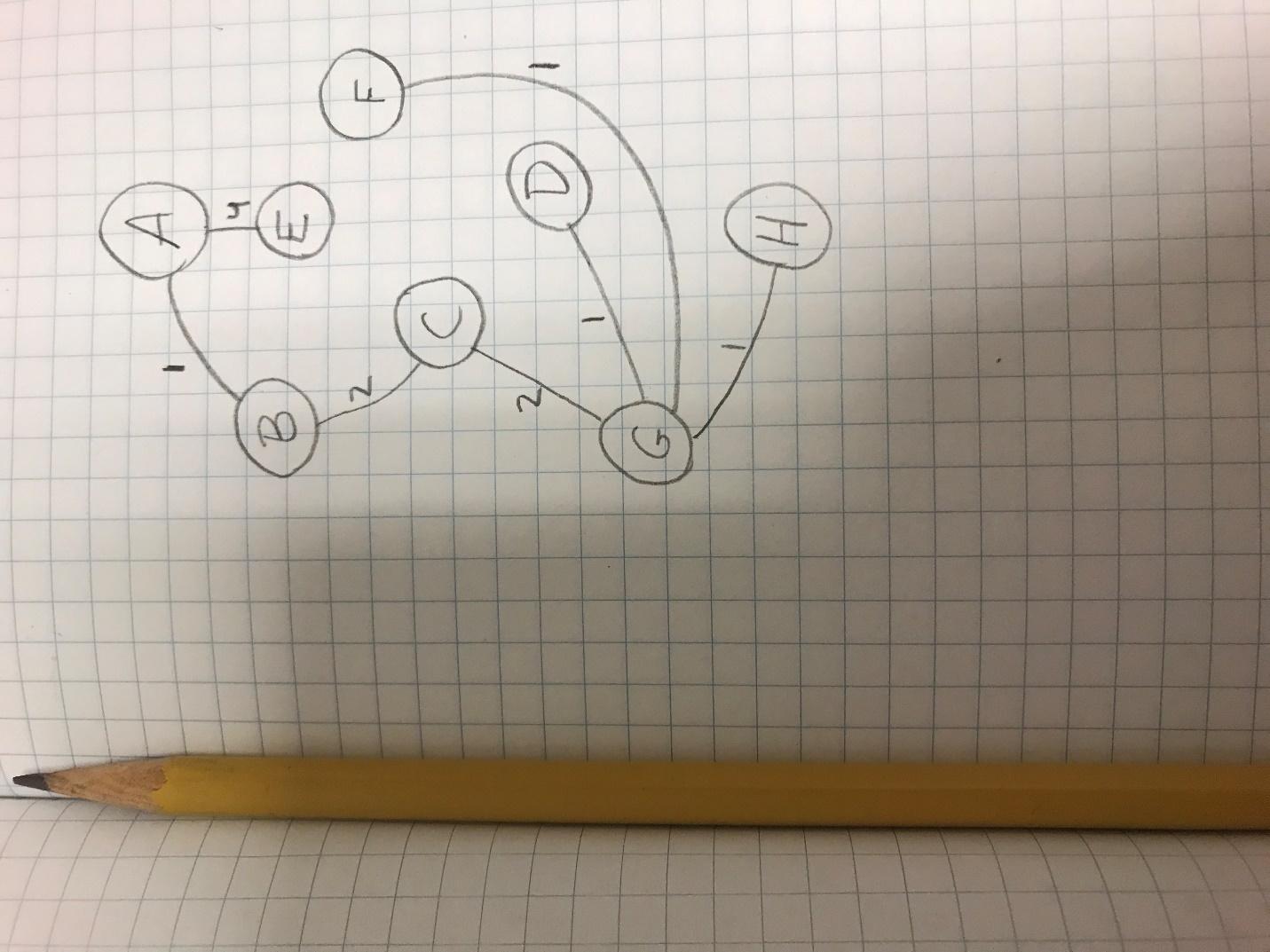
Question 2:



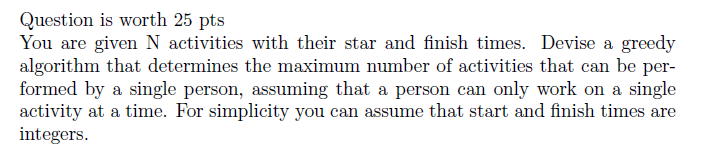
Initially the cost values of all vertices are set to infinity except for the starting vertex A, which is set to 0. Below is the table showing the intermediate values of the cost per iteration.

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

The minimum spanning tree is below. The total cost of the tree is 12.



Question 4:



Consider the following example.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 |
| Start Times | 1 | 3 | 2 | 6 | 6 | 8 | 4 | 4 | 9 | 10 | 2 | 7 |
| Finish Times | 6 | 5 | 7 | 8 | 9 | 9 | 7 | 7 | 12 | 14 | 6 | 10 |

Using the greedy approach to find the maximum number of activities that a single person can accomplish, the job with the lowest finish time is selected from the array. In the example above, A2 is chosen because it has the lowest finish time.

The algorithm will then swap A2 with the last element in the array and shrink its search size to exclude A2. The number of activities is now 1.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A1 | A12 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A2 |
| Start Times | 1 | 7 | 2 | 6 | 6 | 8 | 4 | 4 | 9 | 10 | 2 | 3 |
| Finish Times | 6 | 10 | 7 | 8 | 9 | 9 | 7 | 7 | 12 | 14 | 6 | 5 |

The algorithm will continue to search the array for activities with start times that are greater than or equal to the last activity chosen and with the smallest finish times. In the example, an eligible next activity has to have a starting time greater than or equal to 5 (A2’s finish time). Therefore, the following are all eligible next activities: A4, A5, A6, A9, A10, and A12. The algorithm will choose A4 as the next activity because it has the lowest finish time among all the eligible next activities.

The algorithm is greedy because it selects the most optimal choice from the given search area for each iteration. The final resulting array is below. The value 4 is returned as the maximum number of activities that can be performed by a single person.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A1 | A12 | A3 | A11 | A5 | A10 | A7 | A8 | A9 | A6 | A4 | A2 |
| Start Times | 1 | 7 | 2 | 2 | 6 | 10 | 4 | 4 | 9 | 8 | 6 | 3 |
| Finish Times | 6 | 10 | 7 | 6 | 9 | 14 | 7 | 7 | 12 | 9 | 8 | 5 |

The algorithm described above was implemented in C below.

#include <stdio.h>

#include <stdlib.h>

#define N 12 // number of activities

void swap\_elements(int\*\* array, int index1, int index2){

   int\* array1= \*array;

   int temp = array1[index1];

   array1[index1] = array1[index2];

   array1[index2] = temp;

}

//--------------------------- GREEDY ALGORITHM --------------------------------//

// Return the maximum number of activities able to be performed by a single person.

int greedy\_algorithm(int\* start\_times, int\* finish\_times){

   int i;

   int index\_min = 0; // index at lowest finish time

   int count = 1;

   int\*\* starttimes1 = &start\_times;

   int\*\* finishtimes1 = &finish\_times;

// find the activity with the lowest finish time

   for(i = 1; i < N; i++){

      if(finish\_times[i] < finish\_times[index\_min]){

         index\_min = i;

      }

   }

// swap it with the last activity in the array

   swap\_elements(finishtimes1, index\_min, N-1);

   swap\_elements(starttimes1, index\_min, N-1);

   index\_min = -1;

   int activity\_found = 0;

   for(i = 0; i < N-count; i++){

      // check for eligibility

      if(finish\_times[N-count] <= start\_times[i]){

         activity\_found = 1;

         if(index\_min == -1){

            index\_min = i;

         }

// update optimal finish time

         if(finish\_times[i] < finish\_times[index\_min]){

            index\_min = i;

         }

      }

      // end of this cycle

      if(i == N-count-1){

         if(activity\_found == 0){

            return(count);

         }

// increment count and swap

         count++;

         swap\_elements(finishtimes1, index\_min, N-count);

         swap\_elements(starttimes1, index\_min, N-count);

// reset values for next cycle

         i = -1;

         activity\_found = 0;

         index\_min = -1;

      }

   }

   return(count);

}

int main(){

   int start\_times[N] =  {1, 3, 2, 6, 6, 8, 4, 4, 9,  10, 2, 7};

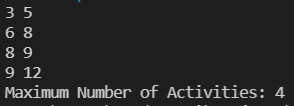
   int finish\_times[N] = {6, 5, 7, 8, 9, 9, 7, 7, 12, 14, 6, 10};

   int max\_number\_activities = greedy\_algorithm(start\_times, finish\_times);

   printf("Maximum Number of Activities: %d\n", max\_number\_activities);

}

The output of the program is below. The result matches what is expected: {A2, A4, A6, A9} = 4 activities.

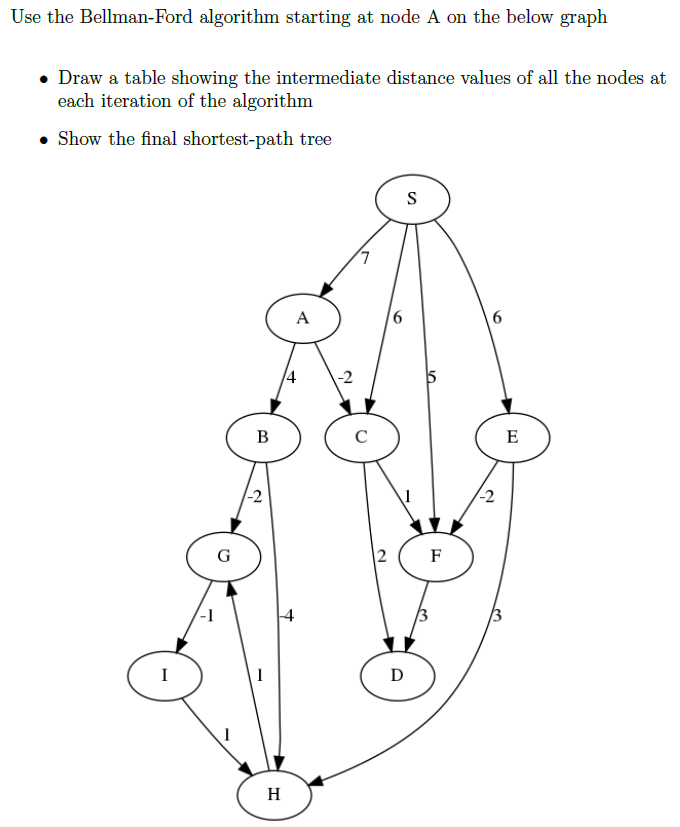


Other inputs were tested:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A1 | A12 | A3 | A11 | A5 | A10 | A7 | A8 | A9 | A6 | A4 | A2 |
| Start Times | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | 23 |
| Finish Times | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 |



Question 6:



Starting at node A on the graph.

1. The table showing the intermediate distance values of the nodes at each iteration is below.

The algorithm will run for 3 iterations because there is no improvement in the distances between vertices from the second iteration to the third. Before the first iteration, the starting vertex A is initialized with distance 0, and all other vertices are initialized with distance infinity.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Iteration | A | B | C | F | G | H | I | D |
| 1 | 0 | 4 | -2 | -1 | 1 | 0 | 1 | 0 |
| 2 | 0 | 4 | -2 | -1 | 1 | 0 | 0 | 0 |
| 3 | 0 | 4 | -2 | -1 | 1 | 0 | 0 | 0 |

1. The final shortest-path tree is below.

