**CIS 350 – SUMMER II**

**Final Exam**

**Demetrius Johnson**

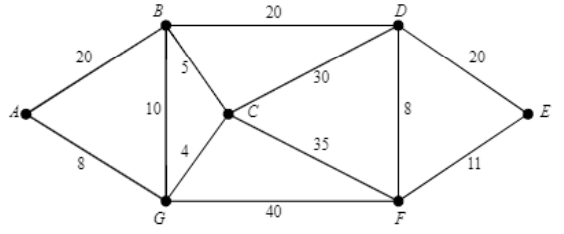
**Professor Thomas G. Steiner**

**August 23, 2021**

# Question 1 – (railway company): highlight in table == part of selection for minimum, \* == reevaluated

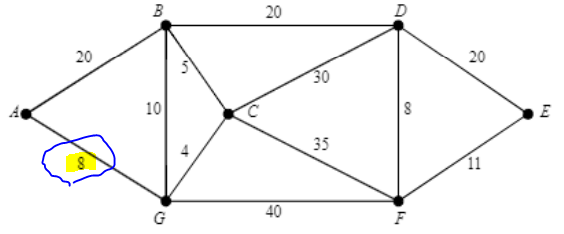
## A) Use Dijkstra’s Algorithm

***Iteration 1: (select A)***



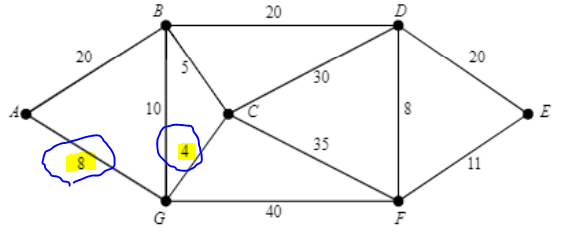
|  |  |  |  |
| --- | --- | --- | --- |
| **Vertex** | **Known** | **dv** | **pv** |
| A | true | 0 | 0 |
| B | false | ∞ | A |
| C | false | ∞ | 0 |
| D | false | ∞ | 0 |
| E | false | ∞ | 0 |
| F | false | ∞ | 0 |
| G | false | ∞ | A |

***Iteration 2: (select G)***



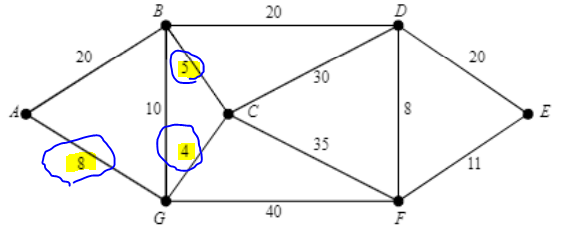
|  |  |  |  |
| --- | --- | --- | --- |
| **Vertex** | **Known** | **dv** | **pv** |
| A | true | 0 | 0 |
| B | false | 20 | A |
| C | false | ∞ | 0 |
| D | false | ∞ | 0 |
| E | false | ∞ | 0 |
| F | false | ∞ | 0 |
| G | true | 8 | A |

***Iteration 3: (select C)***



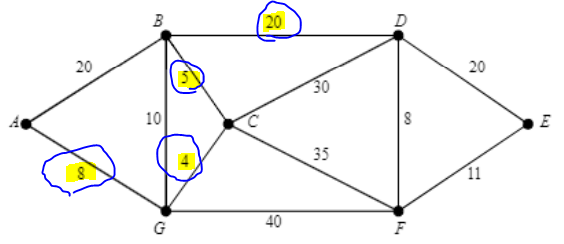
|  |  |  |  |
| --- | --- | --- | --- |
| **Vertex** | **Known** | **dv** | **pv** |
| A | true | 0 | 0 |
| B | false | 18\* | G |
| C | true | 12 | G |
| D | false | ∞ | 0 |
| E | false | ∞ | 0 |
| F | false | 48 | G |
| G | true | 8 | A |

***Iteration 4: (select B)***



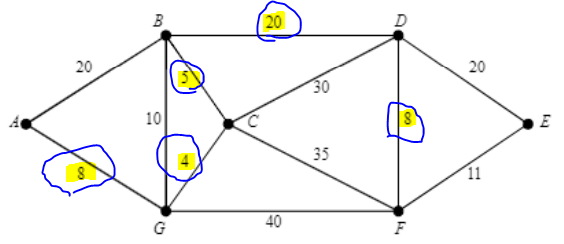
|  |  |  |  |
| --- | --- | --- | --- |
| **Vertex** | **Known** | **dv** | **pv** |
| A | true | 0 | 0 |
| B | true | 17\* | C |
| C | true | 12 | G |
| D | false | 42 | C |
| E | false | ∞ | 0 |
| F | false | 47\* | C |
| G | true | 8 | A |

***Iteration 5: (select D)***



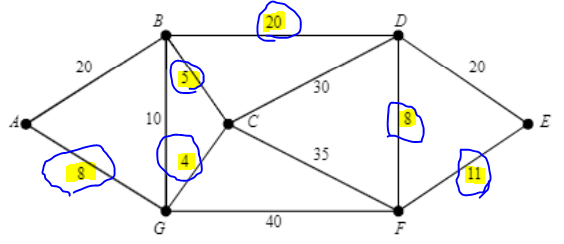
|  |  |  |  |
| --- | --- | --- | --- |
| **Vertex** | **Known** | **dv** | **pv** |
| A | true | 0 | 0 |
| B | true | 17 | C |
| C | true | 12 | G |
| D | true | 37\* | B |
| E | false | ∞ | 0 |
| F | false | 47\* | C |
| G | true | 8 | A |

***Iteration 6: (select F)***



|  |  |  |  |
| --- | --- | --- | --- |
| **Vertex** | **Known** | **dv** | **pv** |
| A | true | 0 | 0 |
| B | true | 17 | C |
| C | true | 12 | G |
| D | true | 37 | B |
| E | false | 57 | D |
| F | true | 45\* | D |
| G | true | 8 | A |

***Iteration 7: (select E) – FINAL ITERATION -> SOLUTION***



|  |  |  |  |
| --- | --- | --- | --- |
| **Vertex** | **Known** | **dv** | **pv** |
| A | true | 0 | 0 |
| B | true | 17 | C |
| C | true | 12 | G |
| D | true | 37 | B |
| E | true | 56\* | F |
| F | true | 45 | D |
| G | true | 8 | A |

**\*All nodes known (set to true). We are done. MST = 56.**

## B) Find min cost + path

D-C: path is D-B-C; cost = 20 + 5 = 25.

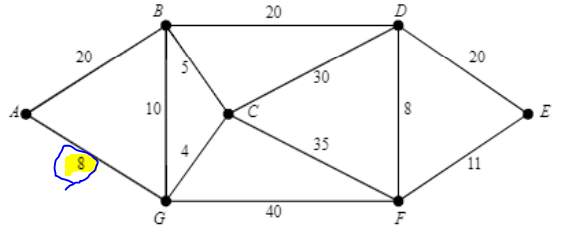
D-E: path is D-F-E; cost = 8 + 11 = 19.

D-F: path is D-F; cost = 8.

## C) Use Prim’s Algorithm

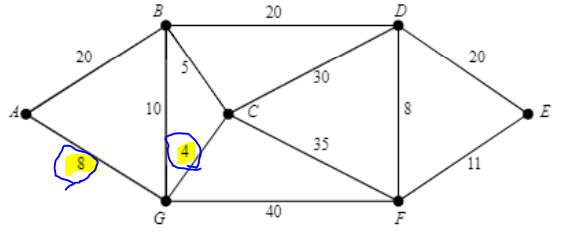
\*select smallest edges as you go – that is select the smallest of all adjacent edges to currently connected vertices (adding them to a heap/priority queue), and add the vertex if it is not already added to the tree; otherwise discard the edge if it is the smallest but connects to a vertex already added to the MST tree:

Iteration 1 (start at vertex A):



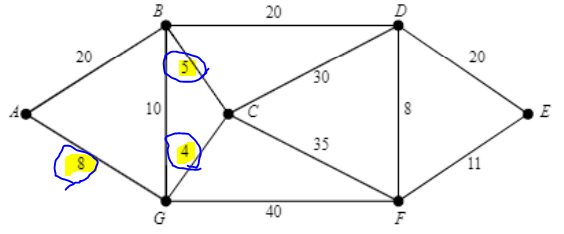
|  |  |
| --- | --- |
| **Vertex** | **Connected** |
| A | true |
| B | false |
| C | false |
| D | false |
| E | false |
| F | false |
| G | true |

Iteration 2:



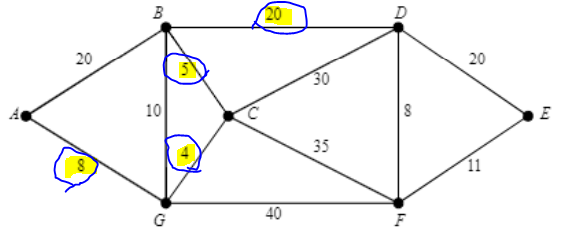
|  |  |
| --- | --- |
| **Vertex** | **Connected** |
| A | true |
| B | false |
| C | true |
| D | false |
| E | false |
| F | false |
| G | true |

Iteration 3:



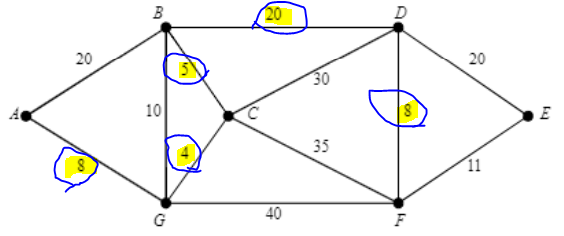
|  |  |
| --- | --- |
| **Vertex** | **Connected** |
| A | true |
| B | true |
| C | true |
| D | false |
| E | false |
| F | false |
| G | true |

Iteration 4:



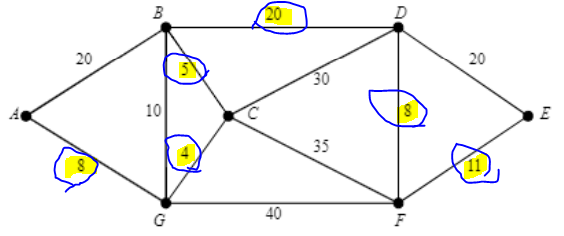
|  |  |
| --- | --- |
| **Vertex** | **Connected** |
| A | true |
| B | true |
| C | true |
| D | true |
| E | false |
| F | false |
| G | true |

Iteration 5:



|  |  |
| --- | --- |
| **Vertex** | **Connected** |
| A | true |
| B | true |
| C | true |
| D | true |
| E | false |
| F | true |
| G | true |

Iteration 6: FINAL ITERATION -> SOLUTION



|  |  |
| --- | --- |
| **Vertex** | **Connected** |
| A | true |
| B | true |
| C | true |
| D | true |
| E | true |
| F | true |
| G | true |

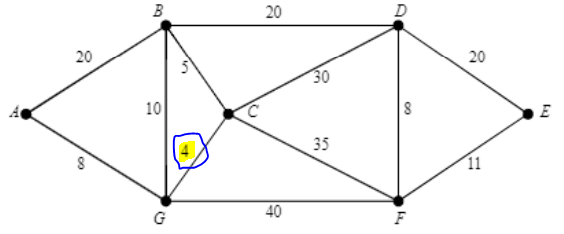
DONE: ALL EDGES ADDED. MST COMPLETE -> MST will simply be all edges that have been added to the MST.

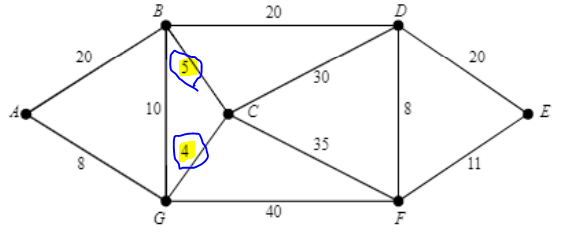
## D) Use Kruskal’s Algorithm

\*scan for all edges, and add all edges to a min heap/priority queue. Then, pop the smallest edges and add the associated vertex to the MST **as long as no cycles are formed (must be checked when trying to add a vertex each time)**; if multiple edge weights tie, arbitrarily pick which one to add to heap first to be popped first.

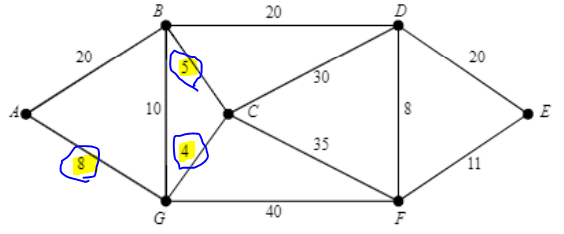
Build min heap: 4, 5, 8, 8, 10, 11, 20, 20, 20, 30, 35, 40

Iteration 1: add 4 (G-C)

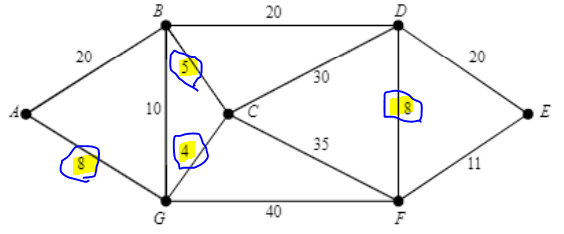
  
Iteration 2: add 5 (B-C)



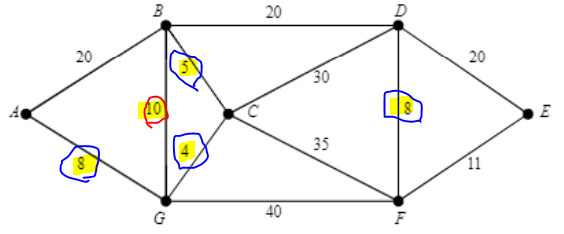
Iteration 3: add 8 (A-G)



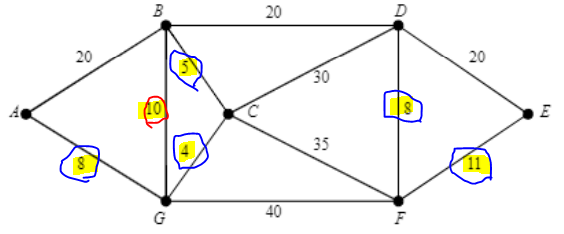
Iteration 4: add 8 (D-F)



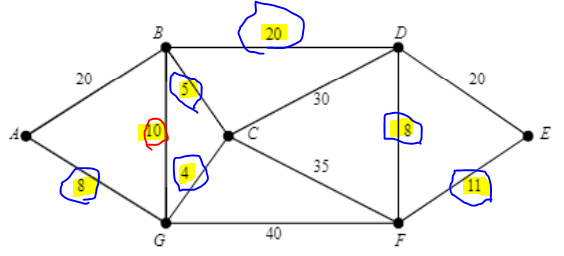
Iteration 5: CANNOT ADD 10 (B-G); cycle formed.



Iteration 6: add 11 (F-E)



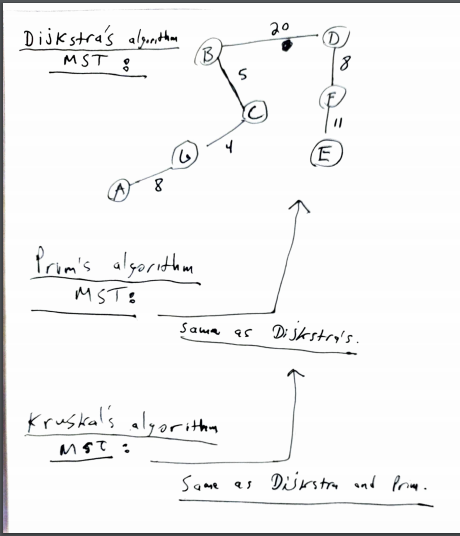
Iteration 7: add 20 (the non-cycle causing 20 edge: B-D) – FINAL ITERATION



DONE; ALL EDGES ADDED, WE HAVE SOLUTION TO MST.

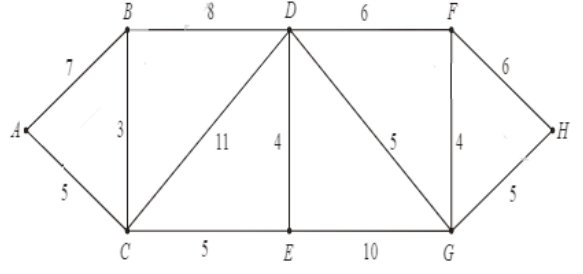
\*note: all MST ended up being exactly the same for Dijkstra, Prim, and Kruskal algorithms:

\*total cost of MST is: 56



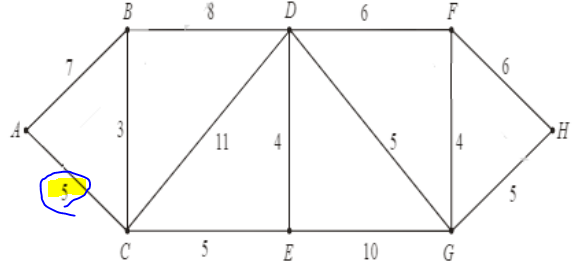
# Question 2 – (roads): highlight in table == part of selection for minimum, \* == reevaluated

## A) Dijkstra’s algorithm

***Iteration 1: (select A)***

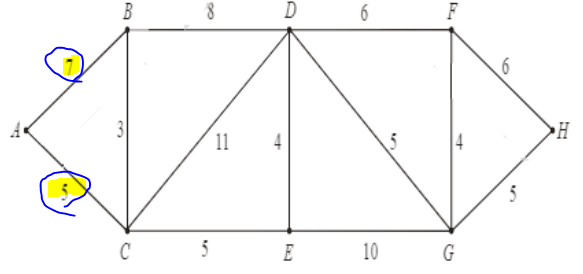
|  |  |  |  |
| --- | --- | --- | --- |
| **Vertex** | **Known** | **dv** | **pv** |
| A | true | 0 | 0 |
| B | false | ∞ | A |
| C | false | ∞ | A |
| D | false | ∞ | 0 |
| E | false | ∞ | 0 |
| F | false | ∞ | 0 |
| G | false | ∞ | 0 |
| H | false | ∞ | 0 |

***Iteration 2: (select C)***



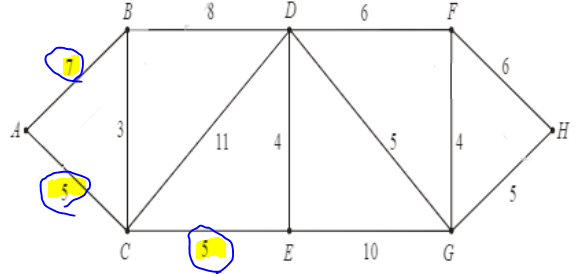
|  |  |  |  |
| --- | --- | --- | --- |
| **Vertex** | **Known** | **dv** | **pv** |
| A | true | 0 | 0 |
| B | false | 7 | A |
| C | true | 5 | A |
| D | false | ∞ | 0 |
| E | false | ∞ | 0 |
| F | false | ∞ | 0 |
| G | false | ∞ | 0 |
| H | false | ∞ | 0 |

***Iteration 3: (select B)***



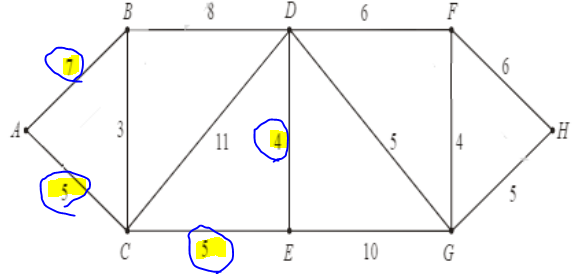
|  |  |  |  |
| --- | --- | --- | --- |
| **Vertex** | **Known** | **dv** | **pv** |
| A | true | 0 | 0 |
| B | true | 7 | A |
| C | true | 5 | A |
| D | false | 16 | C |
| E | false | 10 | C |
| F | false | ∞ | 0 |
| G | false | ∞ | 0 |
| H | false | ∞ | 0 |

***Iteration 4: (select E)***



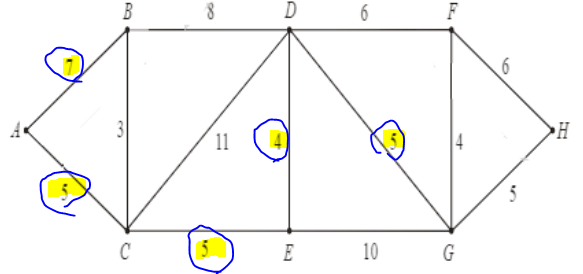
|  |  |  |  |
| --- | --- | --- | --- |
| **Vertex** | **Known** | **dv** | **pv** |
| A | true | 0 | 0 |
| B | true | 7 | A |
| C | true | 5 | A |
| D | false | 15\* | B |
| E | true | 10 | C |
| F | false | ∞ | 0 |
| G | false | ∞ | 0 |
| H | false | ∞ | 0 |

***Iteration 5: (select D)***



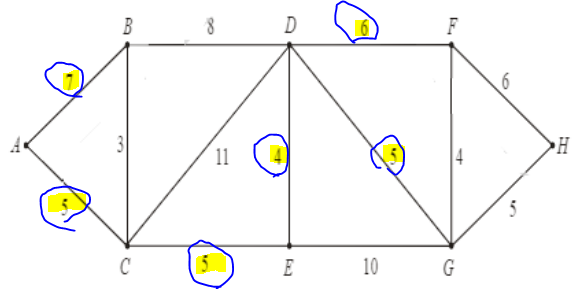
|  |  |  |  |
| --- | --- | --- | --- |
| **Vertex** | **Known** | **dv** | **pv** |
| A | true | 0 | 0 |
| B | true | 7 | A |
| C | true | 5 | A |
| D | true | 14\* | E |
| E | true | 10 | C |
| F | false | ∞ | 0 |
| G | false | 20 | E |
| H | false | ∞ | 0 |

***Iteration 6: (select G)***



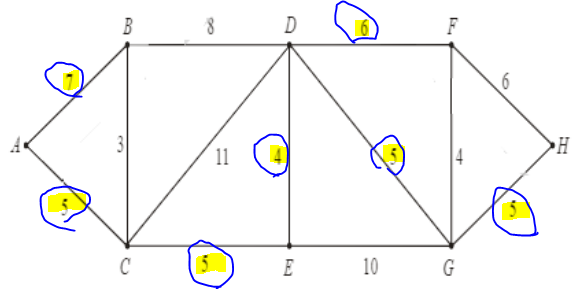
|  |  |  |  |
| --- | --- | --- | --- |
| **Vertex** | **Known** | **dv** | **pv** |
| A | true | 0 | 0 |
| B | true | 7 | A |
| C | true | 5 | A |
| D | true | 14 | E |
| E | true | 10 | C |
| F | false | 20 | D |
| G | true | 19\* | D |
| H | false | ∞ | 0 |

***Iteration 7: (select F)***



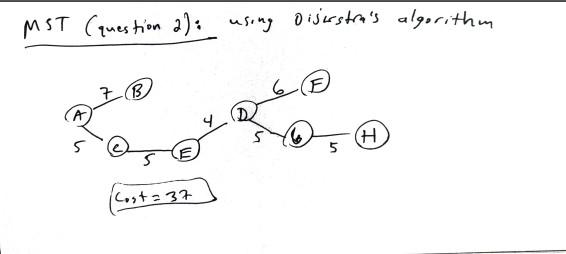
|  |  |  |  |
| --- | --- | --- | --- |
| **Vertex** | **Known** | **dv** | **pv** |
| A | true | 0 | 0 |
| B | true | 7 | A |
| C | true | 5 | A |
| D | true | 14 | E |
| E | true | 10 | C |
| F | true | 20 | D |
| G | true | 19 | D |
| H | false | 24 | G |

***Iteration 8: (select H) – FINAL ITERATION -> SOLUTION***



|  |  |  |  |
| --- | --- | --- | --- |
| **Vertex** | **Known** | **dv** | **pv** |
| A | true | 0 | 0 |
| B | true | 7 | A |
| C | true | 5 | A |
| D | true | 14 | E |
| E | true | 10 | C |
| F | true | 20 | D |
| G | true | 19 | D |
| H | true | 24 | G |

**\*All nodes known (set to true). We are done. MST = 37.**



## B) Find min distance and path for H-A connection

H-A: path is H-G-D-E-C-A; cost (distance) = 24 miles.

## C) cars travel on the road at 30 mph…

**a) How many minutes to get from H to A?:**

H-A is a path that is 24 miles long; 24mi/(30mi/1hr) = 0.8 hours = 0.8\*60 minutes = **48 minutes**.

**b) What is the length of the new D-A road?**

\*We know H-D minimum path is 10 miles; thus traveling 30 mph for 10 miles will take 10mi/30mph = 1/3rd of an hour = 20 minutes.

\*Now, it says total time with new D-A road is 36 minutes; thus the total time remaining possible for the this new H-A path must be 36 – 20 = 16 minutes in trip remaining to get from D to A.

\*if you travel for 16 minutes at 60mph, the length of the new D-A road must be the same distance you would travel: 60mph = 1 mile per minute; thus in 16 minutes you would travel 16 miles 🡪

60mph \* (16/60)hrs = 16 miles.

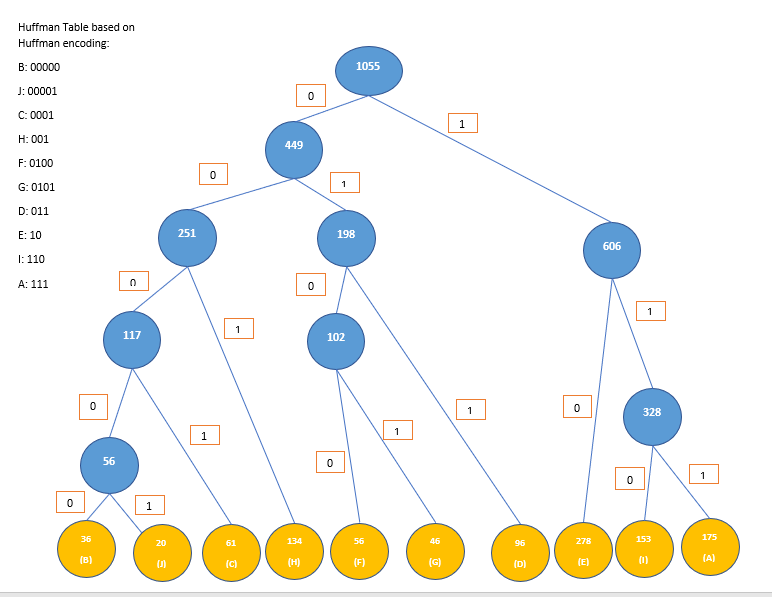
\*solution: the new D-A road must be a length of **16 miles**.

# Question 3 – Kruskal’s algorithm

A) The number of distinct MST for the weighted graph is 6

B) Yes, the algorithm always produces the same cost MST even though there may be many distinct MST that can be made; this is because the purpose of the algorithm is to find the MST; otherwise it would not be a MST algorithm; when there are multiple MST that provide the correct solution, it is usually due to edges with the exact same costs, where the algorithm must arbitrarily select which one on the heap to search first to see if it can be added to the MST without creating a cycle.

# Question 4 – Huffman Encoding – I made the tree in word and took a screenshot:



Thus, BIGHEAD using the above Huffman Encoding will be:

B I G H E A D

00000 110 0101 001 10 111 011

# Question 5- tree traversal

## A) Depth First

6-4-5-1-2-3

## B) Breadth First

6-4-3-5-2-1

# Question 8 – describing algorithms

## A) Greedy Algorithm

The greedy algorithm looks to do what is best first based on the current available options and continue building from there to find a solution. A perfect example of the use of greedy algorithms is honestly how people treat their careers and education life: its all about the best current pay rate options in the current present time. People will tend to continue to follow the money to the next best job – the same as following the next best available edge to reach the next vertex in a graph so as to eventually arrive at some destination (or destinations).

## B) Divide and Conquer algorithm

This approach has two parts: break a problem down in to smaller parts, and then finding solutions to those smaller problems; then, the solution to the original/larger problem is found from the solutions of the subproblems. This is done recursively – keep breaking down problems into smaller problems and find solutions to them as necessary until finally they all combine to give the final solution. Examples of using divide and conquer approach is in war; it is a strategy to divide up the enemy into smaller, less power (less complex) groups that can be targeted and destroyed (in smaller battles) – by destroying all divided groups of enemies, the whole of the enemy will be destroyed and thus the war will have been won.

## C) Dynamic Programming

Dynamic programming can be associated with Fibonacci sequences; the reason for this is because of the nature of the Fibonacci sequence which says that to find solutions, you must build from the previous solution; this is not like divide and conquer where separate solutions can be found and combined to find the higher level solution – rather you MUST have the solution to a previous problem before moving on to find the solution to the next subproblem. Once the entire solution has been sequentially built from the smaller, consecutive and reliant solutions, you find the final solution to the big/original problem. An example of this could be like playing a platform game – in order to beat a given level, you have to solve subproblems, but you cannot solve the next subproblem until you solve the one before it so that you can move on or gain a certain ability for the next problem that will require it. Once all level have been completed, you will likely have all upgrades and have passed all level consecutively in order to beat the entire game.

# Question 9 – Backtracking

## A) Describe the use of “pruning” in relation to a backtracking algorithm.

Pruning is the method of ignoring paths that you know will not lead to a solution; so when you backtrack, you can prune the path that was backtracked since you know for sure that continuing down that path is pointless.

## B) Is backtracking “pruning” different from the branch and bound pruning? Why or why not?

no, branch and bounding does not limit tree traversal methods and uses different standards (bounds) for pruning and is used only for optimization problems; other than that, the pruning is the same – ignore paths that will not lead to solution.

# Question 10 – is the algorithm efficient?

No the algorithm is not efficient because the recursive nature of it is so that each recursive call will cause a domino effect of calling the same recursive calls to find the solution: a better approach would be to use memory; once an h(x) is found, store that value in an array so that every time h(x) is called on for a solution, an array (the memory) is checked first so that a recursive call is not necessary and the h(x) can be returned quickly.

# Question 11 – film festival

Greedy approach would be to watch the movie that shows earliest in the day and then continuously choose the next earliest movie to watch.

# Question 13 – amortized analysis

## A) what is it?

It is a method for measuring the efficiency of an algorithm time complexity, specifically not just one operation but a sequence of operations on a given data structure – it seeks to give true average cost over a sequence of operations.

## B) why is it used?

It is used by computer scientists and programmer to get a better analysis on how efficient an algorithm will be so that it can be best optimized or changed.

## C) compare 3 methods

The aggregate analysis is the least used method, as it calculates the cost of n operations / n; it is usually unlikely you will know total operations during runtime. Next is accounting method, where each type of operation is given a cost, with a credit system for when operations are overcharged, so that when that credit needs to be accessed by other operations it can be; debit never goes below 0. Potential method is same as accounting method except that credit is stored as potential energy and as a whole.

# Question 15 – best sort method

## a. Only a few items

BUBBLE SORT

## b. Items are mostly sorted

insertion sort

## c. Concerned about worst-case scenarios

merge sort

## d. Interested in a good average-case result e. Avoid if you need a stable sort

radix sort

## f. Desire to write as little code as possible

bubble sort

## g. Items have a finite size with limited values

quick sort

