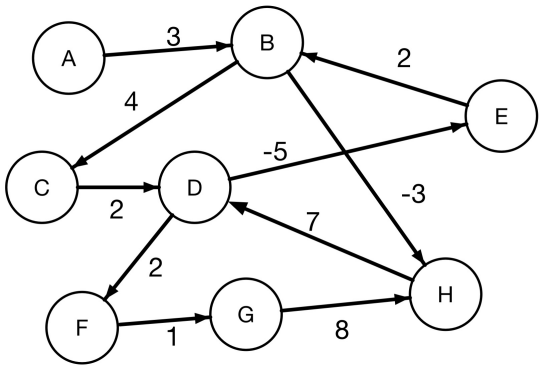
Problem Set 6

Hongrui(Ray) Liu

(1).



Since there are 8 vertices in the graph, there would be 8 - 1 = 7 iterations for traversing all edges in the graph.

The 1st iteration

Edge	Weight
A -> B	3
B -> C	4
C -> D	2
B -> H	-3
D -> E	-5
E -> B	2
D -> F	2
F -> G	1
G -> H	8
H -> D	7

	A	B	C	D	E	F	G	H	I
d	0	3	7	7	4	11	12	0	13

The 2nd iteration

Edge	Weight
A -> B	3
B -> C	4
C -> D	2
B -> H	-3
D -> E	-5
E -> B	2
D -> F	2
F -> G	1
G -> H	8
H -> D	7

	A	B	C	D	E	F	G	H	I
d	0	3	7	7	2	9	10	0	13

The 3rd iteration

Edge	Weight
A -> B	3
B -> C	4
C -> D	2
B -> H	-3
D -> E	-5
E -> B	2
D -> F	2
F -> G	1
G -> H	8
H -> D	7

	A	B	C	D	E	F	G	H	I
d	0	3	7	7	2	9	10	0	13

Since not one single node is updated in the 3rd iteration, we can terminate the loop and the final answer is :

	A	B	C	D	E	F	G	H	I
d	0	3	7	7	2	9	10	0	13

(2).

(a)

This algorithm takes $O(n^2 \log n)$

Expalanation:

In order to execute this algorithm, we need to write a loop first, and the predicate for this loop is the number of nodes in the graph(n).

Within the loop, we are supposed to find the smallest unused color of the neighbors colored so far. In this case, we need to initialize a set of color(in a number form). Since the maximal number of colors is the number of nodes, we should initiliaze a set with a length of n.

Then we should delete all colors that have been taken by the neighbors, After that, we can put nodes left into a priority queue and pop out the smallest element which would take O(nlogn).

Therefore, the time complexity is $O(n^2 logn)$

(b) Yes, it is an algorithm with optimality. When we are traversing a single path of a graph, if we always pick the smallest coloring value, we can ensure that the number of the color that we use is the least one. For example:

red -> blue -> ?

according to the algorithm, the color for the question mark should be red(assuming the value of red is 0, blue is 1 and 0 is the smallest) since we are always picking the color with the smallest value and hence obtain a result with the least number of colors.

5.2

(a)

1. Initialize a priority queue which contains the cost of node with the distance from the source node and set all nodes as INF except the source node.
2. Set the cost of the source node A to be 0

   PQ = {A = 0, the rest = INF}
3. Delete the smallest element in the queue and update the costs in the queue in each iteration

Cost	Value
A	0
B	INF
C	INF
D	INF
E	INF
F	INF
G	INF
H	INF

After the 1st iteration:

Cost	Value
B	1
C	INF
D	INF
E	4
F	8
G	INF
H	INF

Visited : A

After the 2nd iteration:

Cost	Value
C	3
D	INF
E	4
F	7
G	7
H	INF

Visited : A, B(pre = A)

After the 3rd iteration:

Cost	Value
D	6
E	4
F	7
G	5
H	INF

Visited : A, B(pre = A), C(pre = B)

After the 4th iteration:

Cost	Value
D	6
F	7
G	5
H	INF

Visited : A, B(pre = A), C(pre = B), E (pre = A).

After the 5th iteration:

Cost	Value
D	6
F	7
H	6

Visited : A, B(pre = A), C(pre = B), E (pre = A), G(pre = C)

After the 6th iteration:

Cost	Value

I F I 7 I I H I 6 I

Visited : A, B(pre = A), C(pre = B), E (pre = A), G(pre = C)， D(pre = C)

After the 7th iteration:

Cost	Value

| F | 7 |

Visited : A, B(pre = A), C(pre = B), E (pre = A), G(pre = C)， D(pre = C), H(pre = G)

After the 8th iteration:

Cost	Value

Visited : A, B(pre = A), C(pre = B), E (pre = A), G(pre = C)， D(pre = C), H(pre = G), F(pre = B)

(b)

1. Initializa a set which contains a bunch of set with independent nodes in the graph. And put all edges into a edge set E and set the answer set X as null.

   S = {{A}, {B}, {C}, {D}, {E}, {F}, {G}, {H}}

   X = {}

   E = {{AE}, {AF}, {AB}, {BF}, {EF}, {BC}, {BG}, {FG}, {GC}, {DC}, {GD}, {GH}, {DH}}

2. Pop out each edge in E, in a increasing order of weight. And if the ending points of the edge don't exist in the same set, add the edge to X, and merge u,v to the same set.

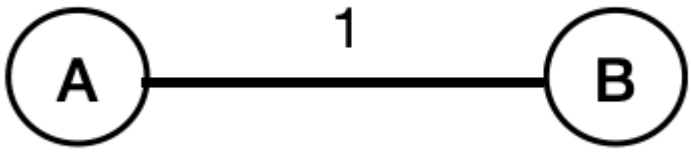
   Here's the step-by-step approach for this:

   - Edge = AB

     S = {{A, B}, {C}, {D}, {E}, {F}, {G}, {H}}

     E = {{AE = 4}, {AF = 8}, {BF = 6}, {EF = 5}, {BC = 2}, {BG = 6}, {FG = 1}, {GC = 2}, {DC = 3}, {GD = 1}, {GH = 1}, {DH = 4}}

     X = {AB}

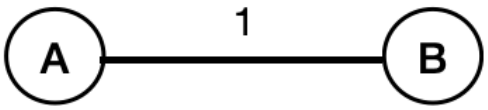


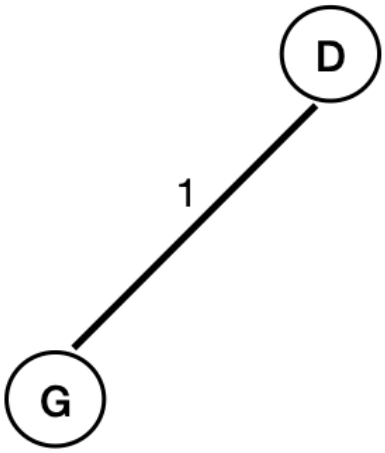
- Edge = GD

  S = {{A, B},{C}, {D, G}, {E}, {F},{H}}

  E = {{AE = 4}, {AF = 8}, {BF = 6}, {EF = 5}, {BC = 2}, {BG = 6}, {FG = 1}, {GC = 2}, {DC = 3}, {GH = 1}, {DH = 4}}

  X = {AB, GD}



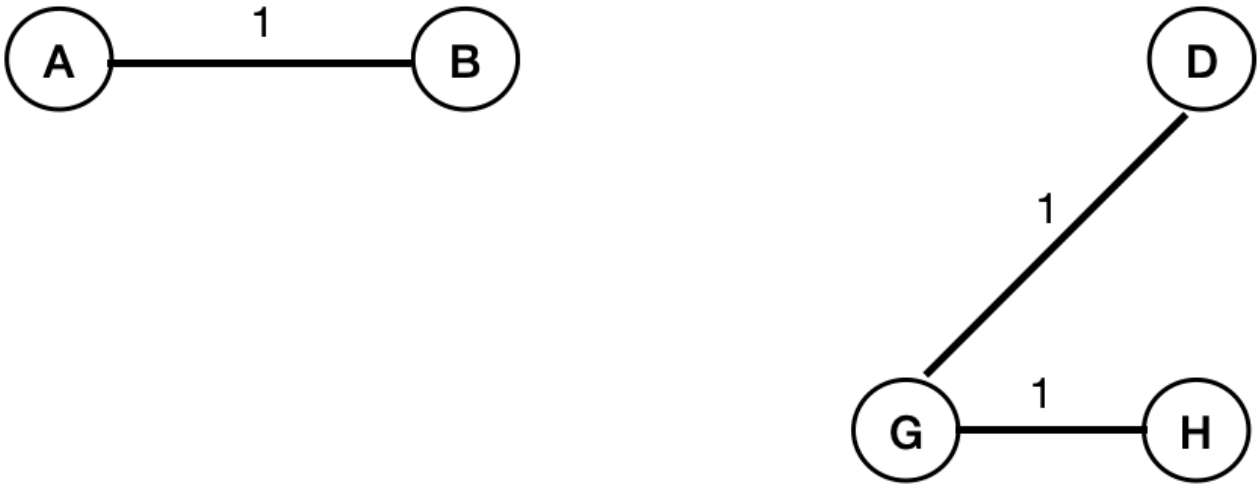


- Edge = GH

  S = {{A, B},{C}, {D, G, H}, {E}, {F}}

  E = {{AE = 4}, {AF = 8}, {BF = 6}, {EF = 5}, {BC = 2}, {BG = 6}, {FG = 1}, {GC = 2}, {DC = 3}, {DH = 4}}

X = {AB, GD, GH}

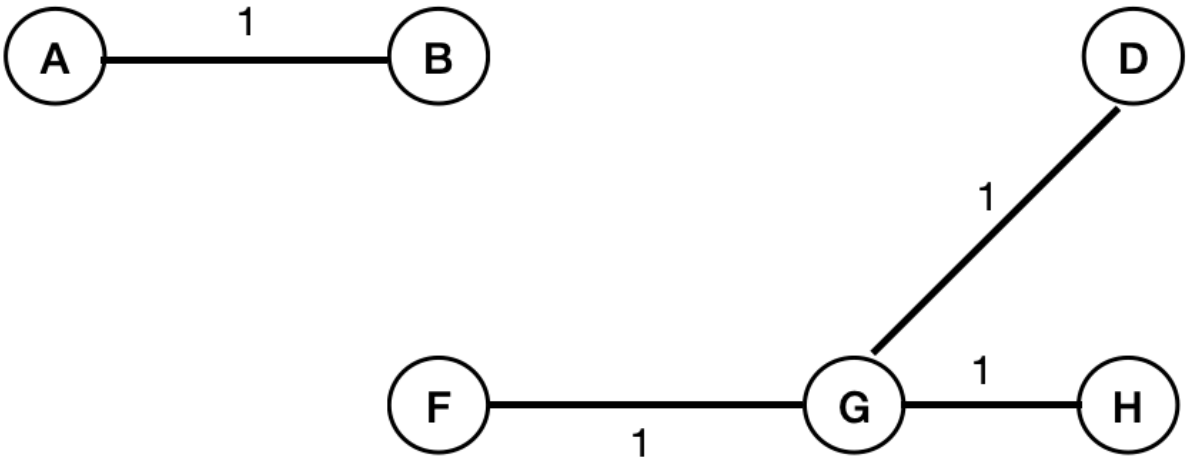


- Edge = FG

  S = {{A, B},{C}, {D, G, H, F}, {E}}

  E = {{AE = 4}, {AF = 8}, {BF = 6}, {EF = 5}, {BC = 2}, {BG = 6}, {GC = 2}, {DC = 3}, {DH = 4}}

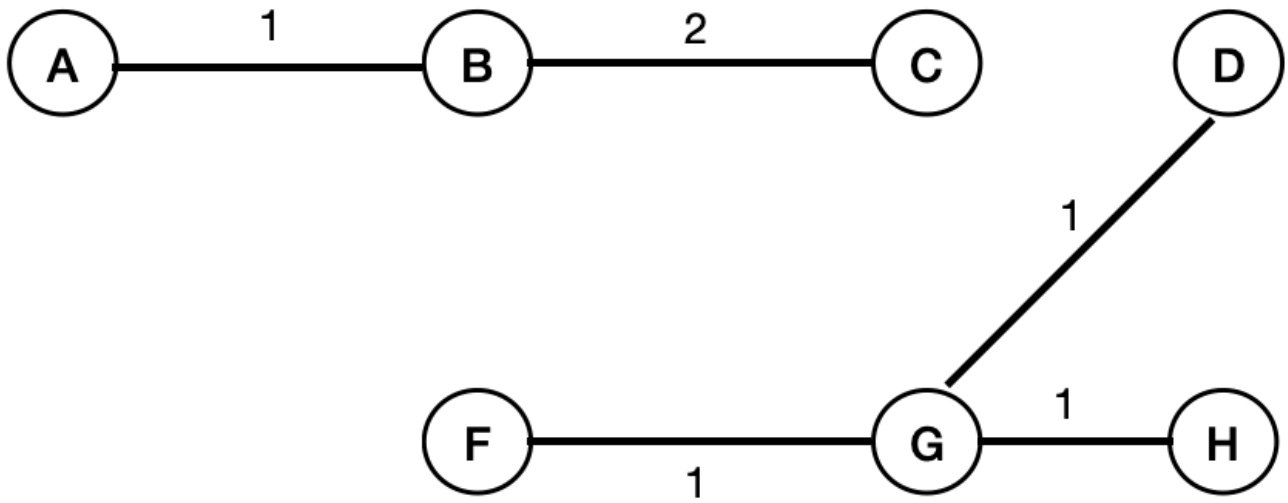
  X = {AB, GD, GH, FG}



- Edge = BC

  S = {{A, B, C}, {D, G, H, F}, {E}}

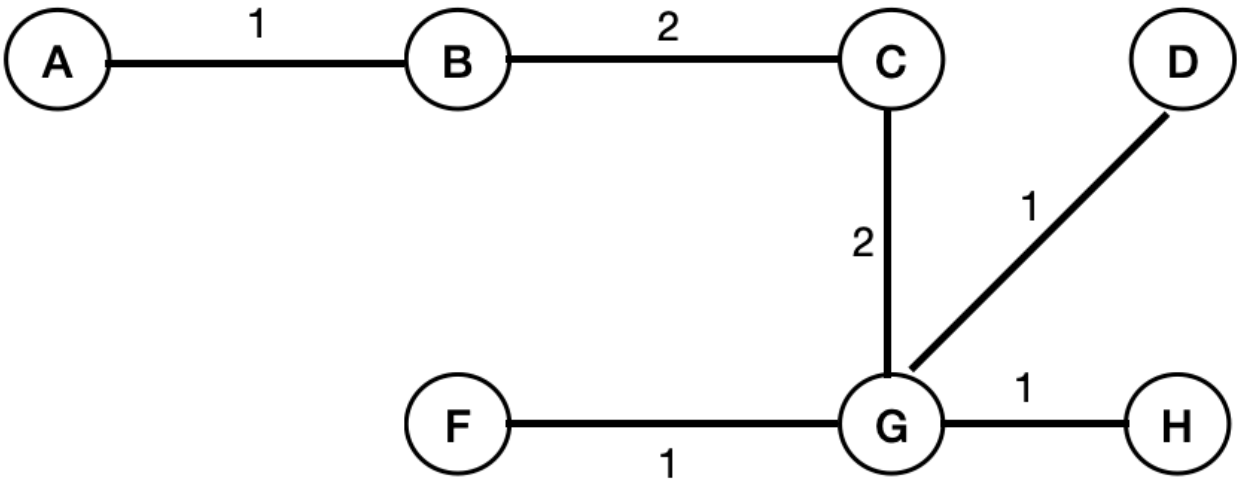
```
  E = {{AE = 4}, {AF = 8}, {BF = 6}, {EF = 5}, {BG = 6}, {GC = 2}, {DC = 3}, {DH
X = {AB, GD, GH, FG, BC}
```



- Edge = GC

  S = {{A, B, C, D, G, H, F}, {E}}

```
  E = {{AE = 4}, {AF = 8}, {BF = 6}, {EF = 5}, {BG = 6}, {DC = 3}, {DH = 4}}
X = {AB, GD, GH, FG, BC, GC}
```

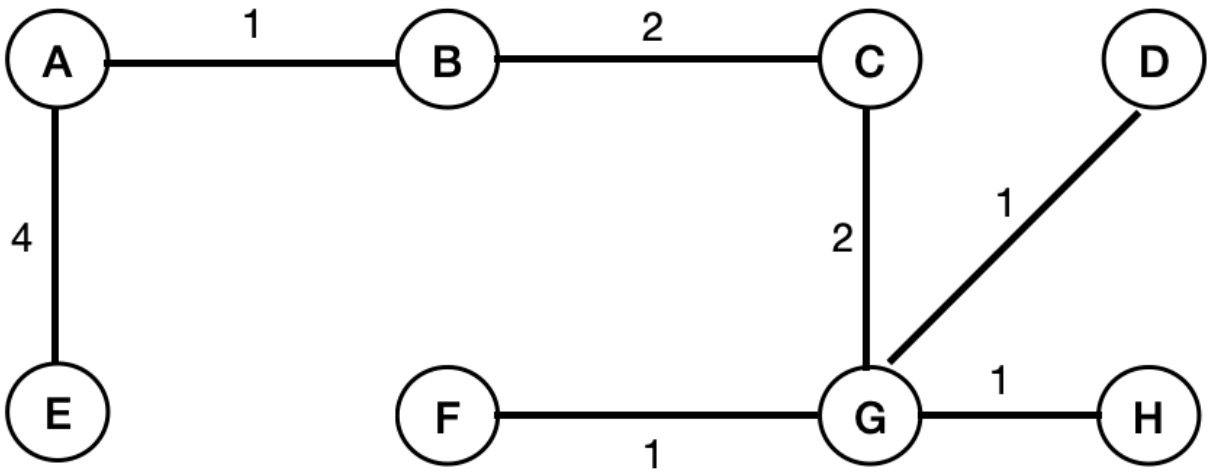


- Edge = AE

  S = {{A, B, C, D, G, H, F, E}}

```
  E = {{AF = 8}, {BF = 6}, {EF = 5}, {BG = 6}, {DC = 3}, {DH = 4}}

X = {AB, GD, GH, FG, BC, GC, AE}
```

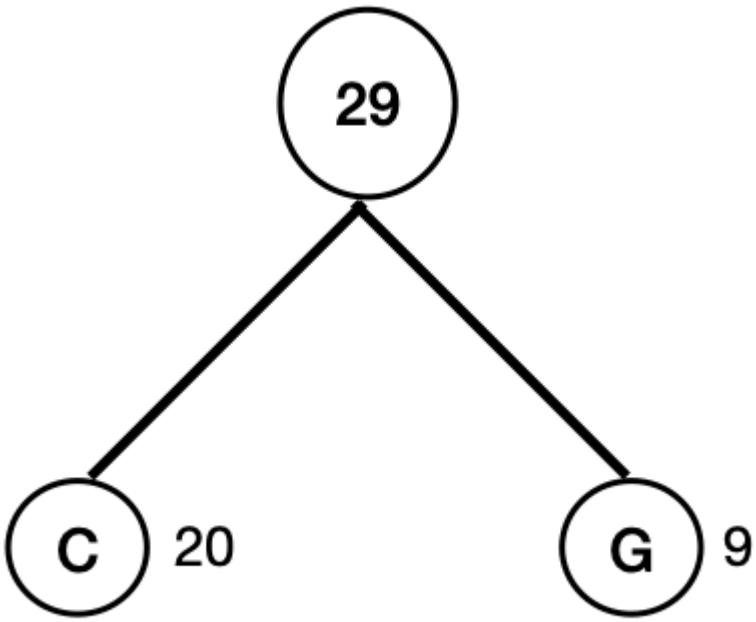


5.13

1. At first, the frequency stack shows as below: H = {G = 9, C = 20, A = 31, T = 40}

i = dequeue(G), j = dequeue(C)

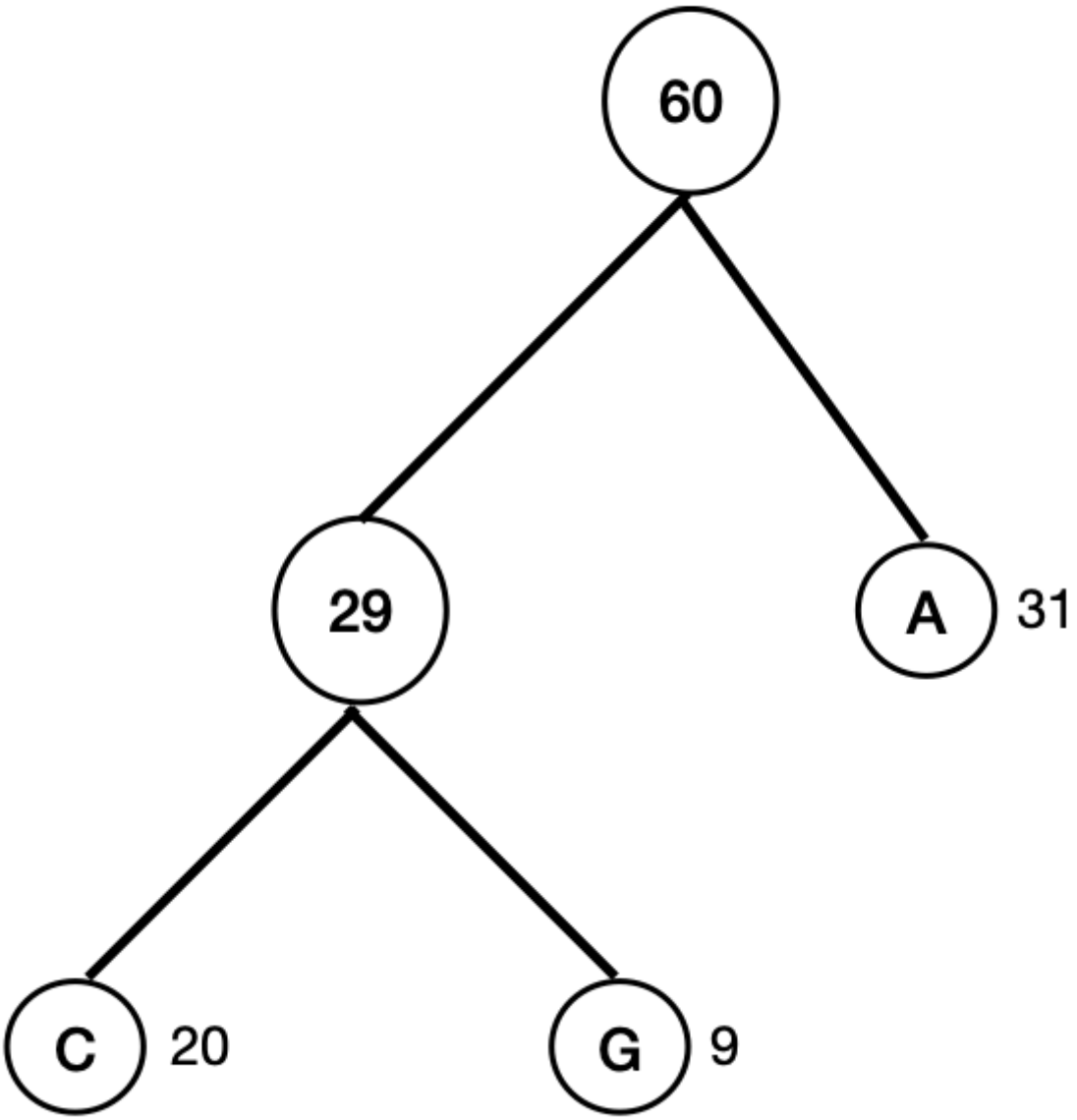
And we create a new tree node based on i and j




```
And enqueue treenode 29

H = {treenode = 29,  A = 31, T = 40}
```

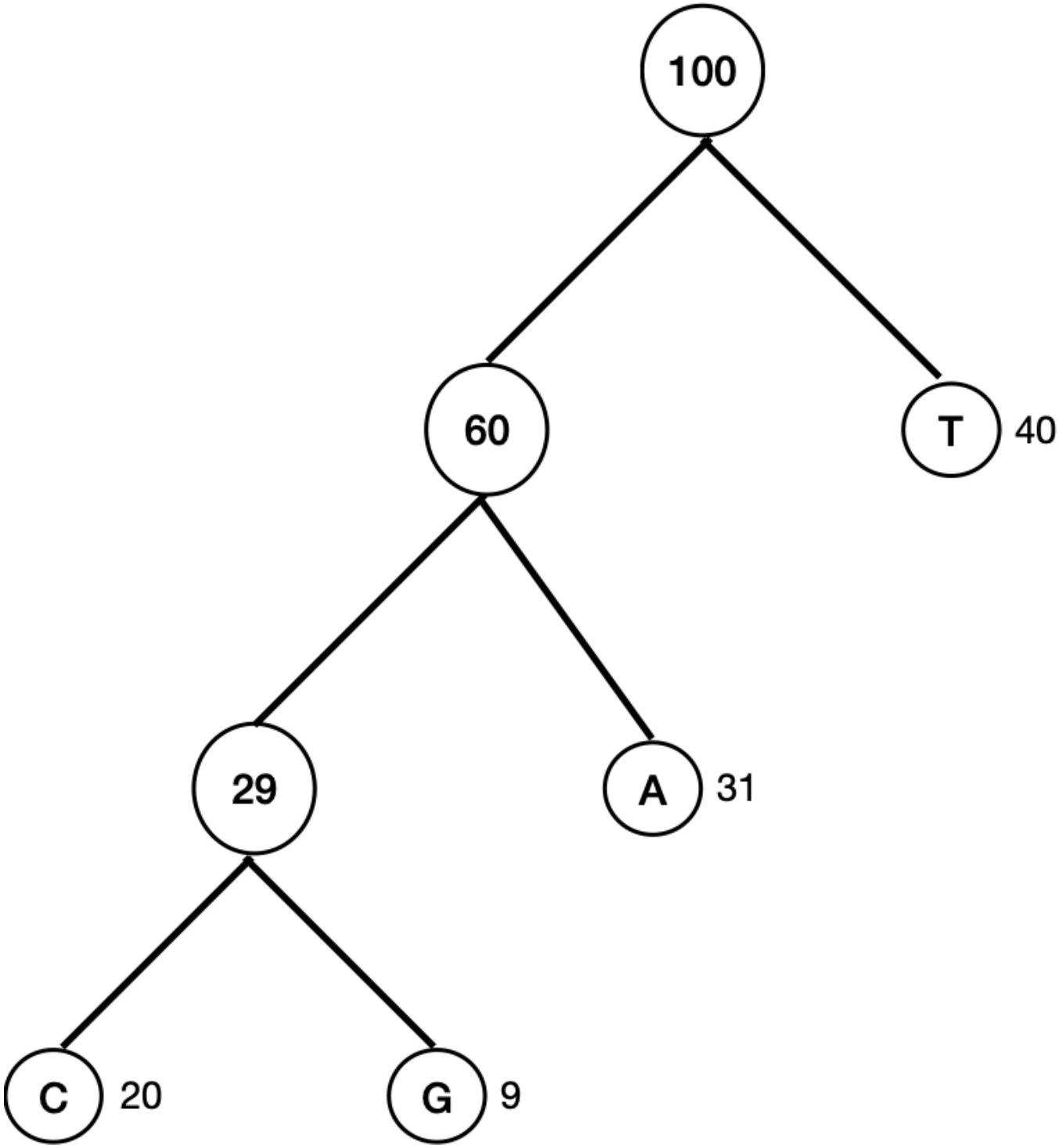
2. i = dequeue = treenode29, j = dequeue = A And we create a new tree node based on i and j




```
And enqueue treenode 60

H = {treenode = 60, T = 40}
```

3. i = dequeue = treenode60, j = dequeue = T And we create a new tree node based on i and j



Therefore, A = 01, C = 000, G = 001, T = 1 | Node | Value | | ------ | ---------- | | A | 01 | | C | 000 | | T | 1 | | G | 001 |