EECE7205: Fundamental of Computer Engineering
Homework 5

Ankesh Kumar (002208893)

Problem 1

TODO Binary Search Tree (BST) Problem

Answer the following questions about the Binary Search Tree (BST) data structure:

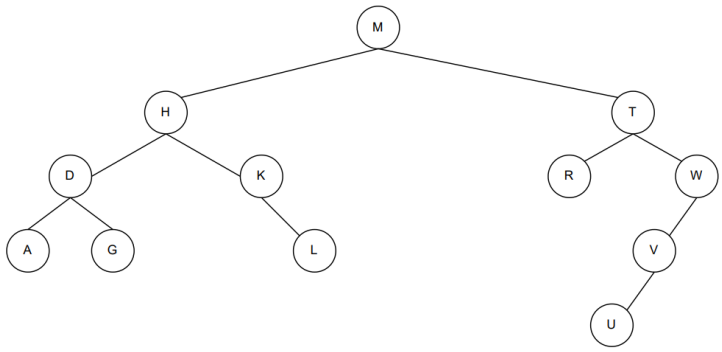
Ques 1. *Without drawing any trees, what are the minimum and maximum heights of binary trees with 63 nodes.*

Maximum occurs when Tree is completely unbalanced. i.e. when there will be all single leaf to each node. Considering this case, for 63 nodes, we will have a **height maximum of 63-1 = 62**.
The minimum will be the case for the perfectly balanced tree. Each parent node can have a maximum 2 children, for height h=1. We have only node.
For h=2, it is 2^1 = 2 (1+2 = 3 nodes total) For h=3, it is 2^2 = 4 (1+2+4 = 7 nodes total) ..
Similarly, for 63 nodes, we can calculate is (2^0 + ...2^5 = 63 nodes), having **minimum height = 6**

Ques 2. *Starting from an empty BST, draw the tree after inserting the following values in the order given (from left to right): M, H, D, A, G, K, L, T, R, W, V, U*

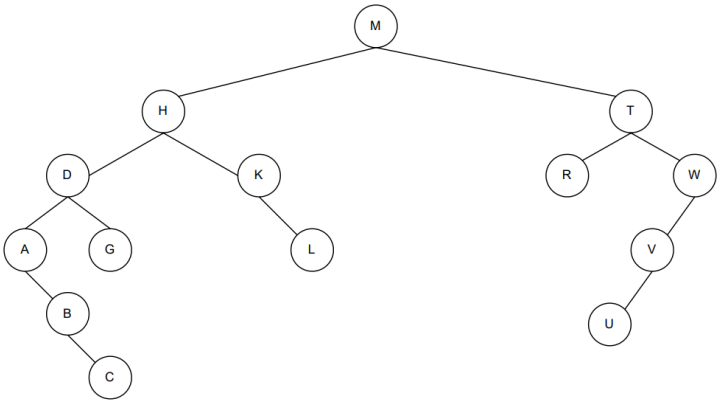
Following the steps or algorithm provided in class , we come up with following BST.

- Insert M: M becomes the root.
- Insert H: H goes to the left of M.
- Insert D: D goes to the left of H.
- Insert A: A goes to the left of D.
- Insert G: G goes to the right of D.
- Insert K: K goes to the right of H.
- Insert L: L goes to the right of K (since L is greater than K).
- Insert T: T goes to the right of M.
- Insert R: R goes to the left of T (since R is between M and T).
- Insert W: W goes to the right of T.
- Insert V: V goes to the left of W (since V is between T and W).
- Insert U: U goes to the left of V (since U is between T and V).



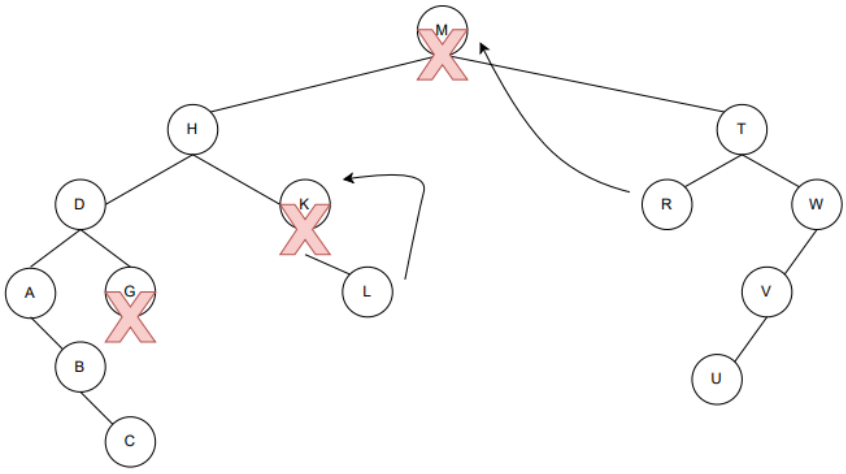
Ques 3. *On the same BST you created in part (b), add the following values B then C*

In BST, B will go till A to find the minimum and get placed as right leaf of A as B is greater than A.
For, C, it searches for value where it is alphabetically greater and it is greater than B.
Hence **C becomes a new right leaf of B**.

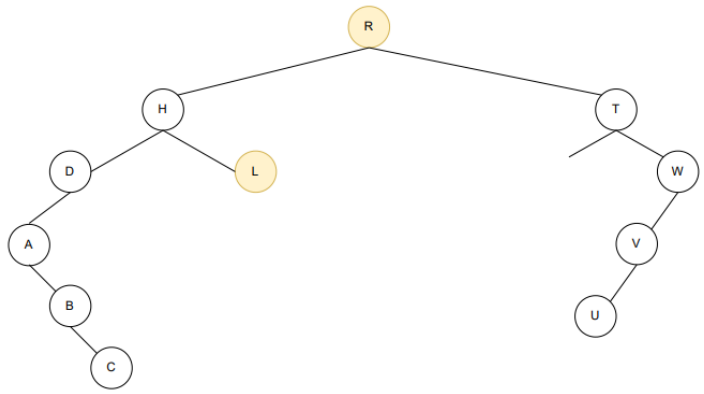


Ques 4. *Redraw the BST you have from part (c) after removing the following values: G - K - M (in this order)*

As per the lecture notes, modified the G then K and then M
Removing G : G is removed as it doesn't have and successor, hence Graph structure remains the same.
Removing K : When K is removed, it has successor L, and L is replaced by K
Removing M : As M is the root node and has two children. When deleting a node with two children, we generally replace it with its in-order successor or in-order predecessor. The in-order successor is the smallest node in the right subtree, which in this case is R (the leftmost node in the right subtree of M).



New BST formed



Ques 5. *Is the tree resulted from part (d) balanced? Why? Explain a step-by-step procedure to convert it to a balanced BST.*

Yes, the resultant tree is balanced as the difference between the left and right subtrees heights differ by not more than 1.
Steps to convert it to balanced BST :

- Find the middle element of the array and make it the root.
- Recursively do the same for the subarray left of the middle element and make the returned node the left child of the root.
- Recursively do the same for the subarray right of the middle element and make the returned node the right child of the root.
- This approach will guarantee a balanced BST because it always picks the middle of the remaining elements as the root, thus ensuring the left and right subtrees have nearly the same number of nodes and thereby minimizing the height of the tree.

Problem 2

TODO B-Tree Problem

Answer the following questions about the B-Tree data structure:

Ques 6. *What is the maximum number of keys that can be stored in a B-tree of height 2 and minimum degree 3?*

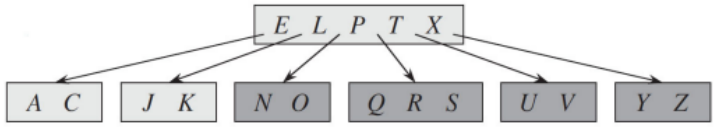
If a non-leaf B-tree node x contains x.n keys, then x has x.n+1 children.
Every node, other than root must have **atleast t-1 keys** where t is the degree.
Every node, including root should have **almost 2t-1 keys**.
For B-tree, degree 3, possible maximum keys is each node is 2*3-1 = 5 keys.
From the above things, for height 2, if the root has 5 elements, it will have 6 children. 6 children can have 5 keys each.
So, the maximum number of keys is calculated as follows:
For the root node: 5 keys
For the children of the root node:
6 children nodes × 5 keys per child = 30 keys
Adding them together, the maximum number of keys in the entire B-tree is 5 (from the root node) + 30 (from the children nodes) = **35 keys**.

Ques 7. *For a B-tree with a minimum degree 3, what is its possible maximum height to store 17 keys?*

As mentioned earlier, the root node can have maximum of 5 keys and hence 6 children.
Also, minimum number of keys in node except root node is 2 and minimum number of children is 2 for root node.
Utilising this information, given a minimum degree of 3, each node must have at least 2 keys, and each internal node must have at least 3 children. Here's a step-by-step calculation using the minimum criteria to find the maximum height:
Level 0 (Root): Can have a single key (in the worst-case scenario, to maximize height).
Level 1: Since the root has 1 key, it splits into 2 children (each with at least 2 keys, since we're looking for the maximum height, not the minimum).
Level 2: Each of these 2 nodes can have 3 children (since they're not the root, they follow the minimum children rule). That gives us 2 x 3 = 6 nodes on this level, each with at least 2 keys.

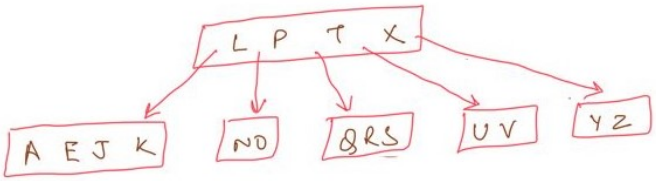
Total keys in this configuration:
Level 0: 1 key in the root.
Level 1: 2 nodes with at least 2 keys each, giving 4 keys at this level.
Level 2: 6 nodes with at least 2 keys each, giving 12 keys at this level.
Adding them up, we have 1 (root) + 4 (level 1) + 12 (level 2) = 17 keys.
Hence, the maximum height would be 3 considering each nodes filled with minimum required keys i.e. 2 (except root which is 1).

Ques 8. *Show the results of deleting C, P, and V in order, from the following B-Tree that has a minimum degree t = 3:*



As the B-Tree has t=3, we need to have minimum of 2 and maximum of 5 keys stored in each node. Also, after deletion, if any node keys becomes less than the required, we need to perform the rebalancing method to satisfy all the condition of the B-Tree.
Deleting C

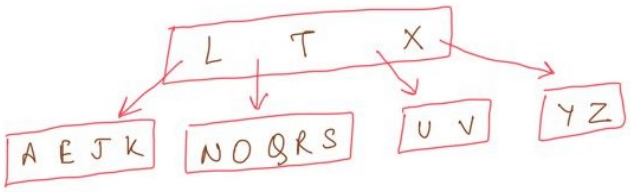
- 'C' is a leaf node, after removal its respective node has now 1 key.
- We need to maintain the property and have to borrow its predecessor, so we bring E from the root hence now all the B-Tree property is satisfied.



Deleting P

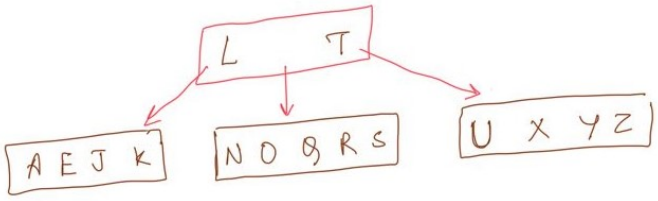
- Position of 'P' is in root node.
- On removal of P, as per the B-Tree property that *n key must have n+1 child* rule will violate.
- So, we have option to borrow Q which is successor OR we can join the leaves to reduce the child to 4 for 3 keys ('L', 'T', 'X') in root.

- Used the past approach mentioned and joined the children nodes and the resultant B-Tree satisfies all properties.



Deleting V

- Here, we need to remove element from the leaf node.
- After removing 'V', the node 'U' has few keys and will need to merge with a sibling node or borrow a key.
- We merge the sibling node and hence, all B-Tree properties are still maintained.



Problem 3

TODO Graph Problem - C++ programming

Approach to the solution

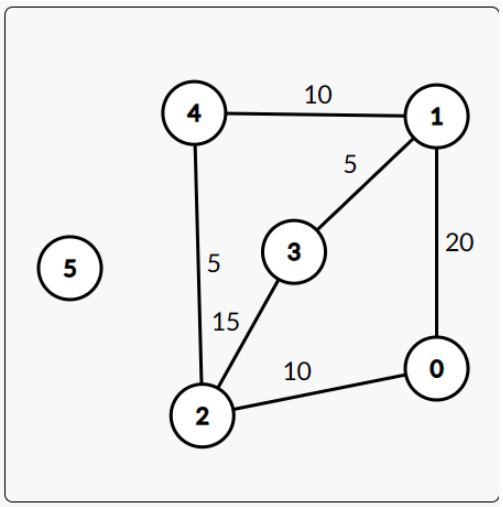
Here, I have followed the steps asmentioned in the question.
Used the Algorithm logic mention in course slides. Briefly, will go through the steps used to create the solution :

- Firstly created the File Upload logic as mentioned as Hint in the Assignment.
- Created Logic to form the Adjacency Matrix where first element from the file (*GraphData.txt*) was used to get number of Vertex/ Nodes and rest data as values of the Graph(vertex1, vertex2, weight).
- Then, implemented the logic of Depth First Search and then put a check on Adjacency Matrix to validate if it is connected. If the Graph is connected, it runs further, else it exits with a message.
- Implemented Logic of both Kruskal's and Prim's to get the Minimum Spanning Tree. Output of it contains the Edge and corresponding weights. Also, Sum of weights is present.
- Implemented method for Shortest Path using both - Dijkstra's and Bellman-Ford's. Here, in this for Bellman-Ford's, as asked, have implemented the optimized code where it runs for V times to check for any negative cycle. The code indicates if it catches a negative cycle.
- The code was created and is compiled successfully in the Linux server. A corresponding Screenshot has been placed.
- 3 different files (Connected, Not Connected and Negative-cycle) data was created in file and execution result along with graph diagram is placed in Execution Screenshots section.

Execution Screenshot

`Note` `Used 2 external libraries`
#include <climits> to set infinity for certain values
#include <algorithm> for Sorting method and used in Kruskal's algorithm

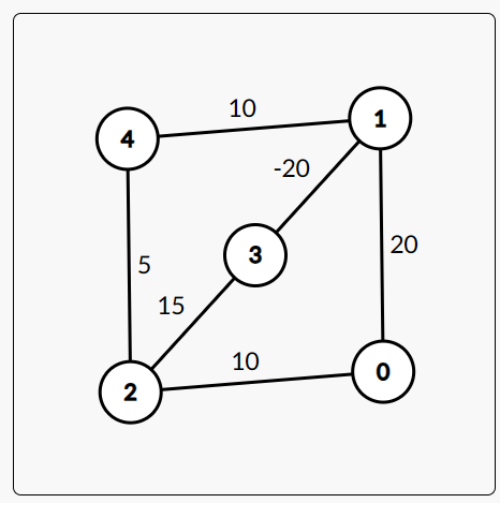
Not connected Graph



cmd output
The program returns "Graph not connected" and exits the program.

```
No negative weight cycle detected
akuma67@LAPTOP-DCEM5HVC:~/EECE7205CProgram/HW5$ ./HW5_Problem3
The graph is not connected.
akuma67@LAPTOP-DCEM5HVC:~/EECE7205CProgram/HW5$
```

Negative cycle graph



cmd output

Here, the program runs and we get Adjacency Matrix at starting and then, we calculate the MST and Shortest Path.

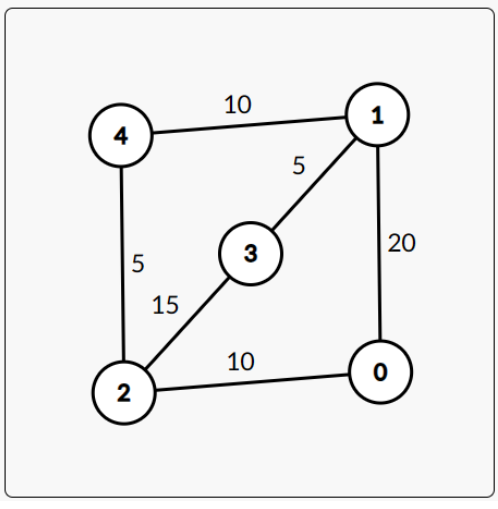
As we have used the optimised Bellman-Ford algorithm to detect the negative cycle the program writes "**Graph contains a negative-weight cycle**"

Also, the **MST weights calculated by both Kruskal's and Prim's are equal** and is validated from the screenshot.

```
akuma67@LAPTOP-DCEM5HV\
HW2_Problem1.cpp  HW2_Problem3.cpp  HW4_Problem1  HW5              folder
akuma67@LAPTOP-DCEM5HVC:~/EECE7205CProgram$ cd HW5
akuma67@LAPTOP-DCEM5HVC:~/EECE7205CProgram/HW5$ g++ HW5_Problem3.cpp -o HW5_Problem3
akuma67@LAPTOP-DCEM5HVC:~/EECE7205CProgram/HW5$ ./HW5_Problem3
The graph is connected.
0 20 10 0 0
20 0 0 -20 10
10 0 0 15 5
0 -20 15 0 0
0 10 5 0 0
-----------------------------Minimum Spanning Tree-----------------------------------------------
Using Kruskal Algorithm
Edge    Weight
1 -- 3 == -20
2 -- 4 == 5
0 -- 2 == 10
1 -- 4 == 10
Weight of Minimum Spanning Tree is 5
Using Prims Algorithm
Edge    Weight
4 -- 1 == 10
0 -- 2 == 10
1 -- 3 == -20
2 -- 4 == 5
Weight of Minimum Spanning Tree is 5

-------------------------------------Single-Source Shortest Path-----------------------------------------
Using Dijkstra Algorithm
Vertex  Distance        Path
0 -> 0   0              0
0 -> 1   20             0 -> 1
0 -> 2   10             0 -> 2
0 -> 3   0              0 -> 1 -> 3
0 -> 4   15             0 -> 2 -> 4
Using Bellman-Ford's Algorithm
Graph contains a negative-weight cycle
akuma67@LAPTOP-DCEM5HVC:~/EECE7205CProgram/HW5$
```

Connected graph



cmd output

The program returns all the values.

MST weights provided by both algorithm is same.

Shortest Path is also same by both algorithms.
Important : - Although, the MST weight is same, if you see, the path or edges within MST varies in both algorithm and it is based on their approach but minimum remains the same.
Also, for Shortest path, as to reach one point, 0 to 3, we have same weight, both algorithm took different shortest path from 0->3, but as weights were same, we get equal and their minimum value.

```
akuma67@LAPTOP-DCEM5HVC:~/EECE7205CProgram/HW5$ ./HW5_Problem3
The graph is connected.
0 20 10 0 0
20 0 0 5 10
10 0 0 15 5
0 5 15 0 0
0 10 5 0 0
------------------------------Minimum Spanning Tree-----------------------------------------------
Using Kruskal Algorithm
Edge    Weight
1 -- 3 == 5
2 -- 4 == 5
0 -- 2 == 10
1 -- 4 == 10
Weight of Minimum Spanning Tree is 30
Using Prims Algorithm
Edge    Weight
4 -- 1 == 10
0 -- 2 == 10
1 -- 3 == 5
2 -- 4 == 5
Weight of Minimum Spanning Tree is 30

---------------------------------------Single-Source Shortest Path-------------------------------------------
Using Dijkstra Algorithm
Vertex  Distance        Path
0 -> 0   0              0
0 -> 1   20             0 -> 1
0 -> 2   10             0 -> 2
0 -> 3   25             0 -> 2 -> 3
0 -> 4   15             0 -> 2 -> 4
Using Bellman-Ford's Algorithm
Vertex  Distance        Path
0 -> 0   0              0
0 -> 1   20             0 -> 1
0 -> 2   10             0 -> 2
0 -> 3   25             0 -> 1 -> 3
0 -> 4   15             0 -> 2 -> 4
No negative-weight cycle detected
akuma67@LAPTOP-DCEM5HVC:~/EECE7205CProgram/HW5$
```

This is the submission for the assignment 5.