CSCI 260 Fall 2024 --- Written Assignment 2

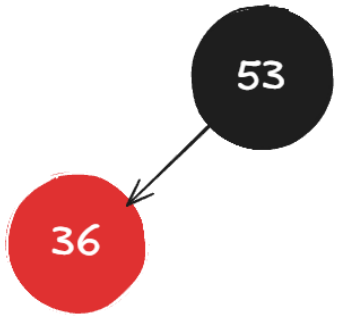
Luka Karanovic – 665778833

Trees

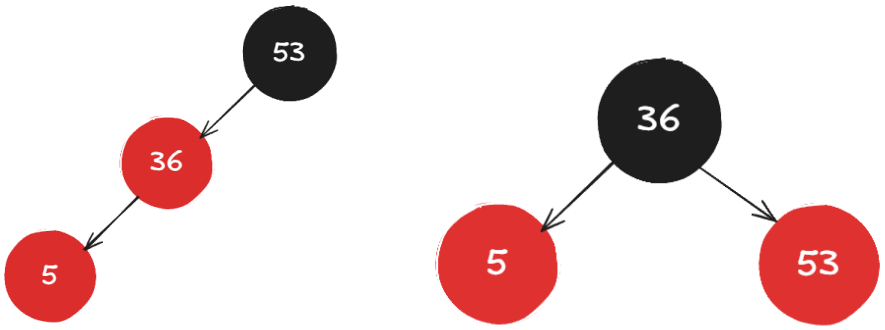
1a. Draw the red-black tree generated by inserting the following sequence of numbers one by one to an initially empty red-black tree: {53, 36, 5, 50, 43, 13, 8, 88, 62, 78, 97, 91, 33, 28, 4, 58}, clearly identify the color of each node in your tree.

Insert 53 - It is the root node so it's black.

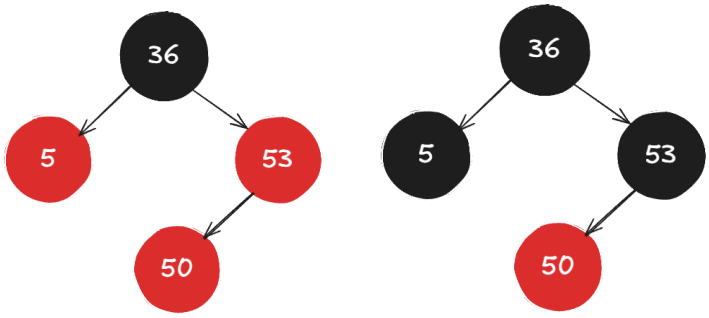
Insert 36 - Less than 53 and a leaf node, so it's red.



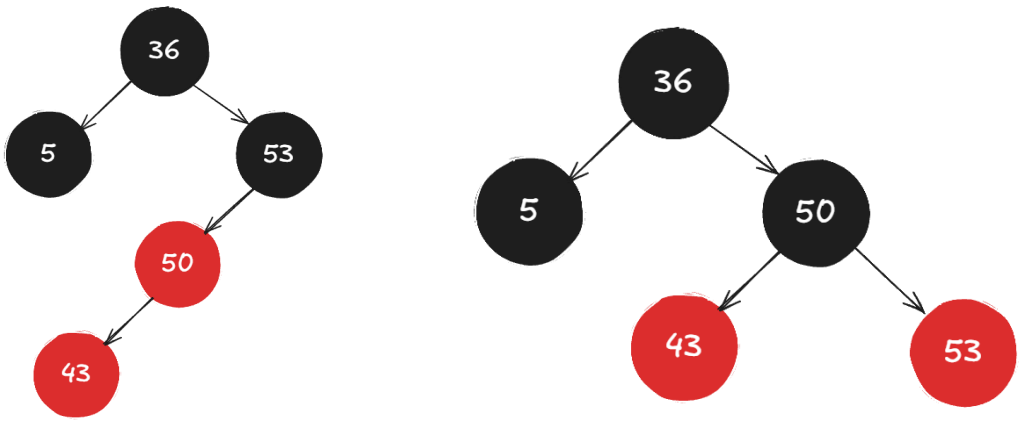
Insert 5 - Less than 36 and leaf node, so insert to the left and colour red. We now have a double red problem! As 5's uncle node is black (nullptr), we need to do a trinode restructure, which is a right rotation.



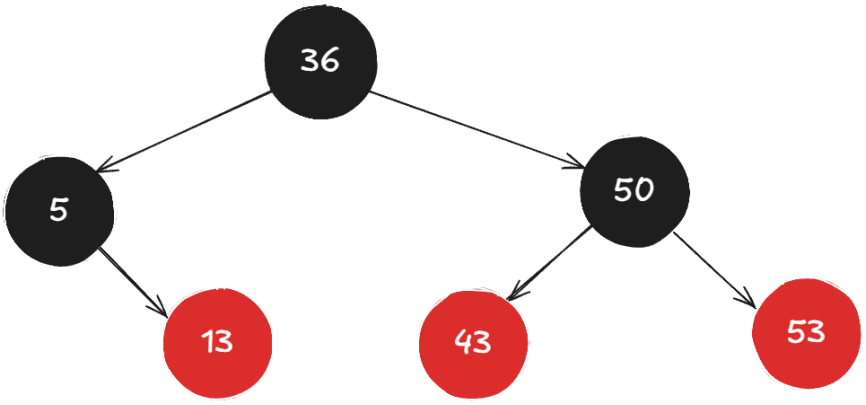
Insert 50 - Add to the left of 53 and since it is a leaf, colour it red. Double red problem and 50's uncle node is red so we change both 53 and 5 to black, but we don't change 36 to red as it's the root node.



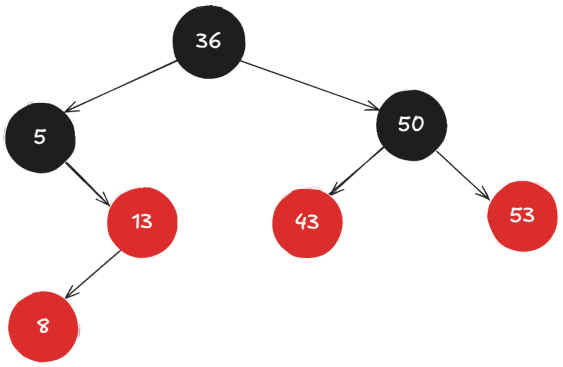
Insert 43 - Insert to the left of 50, double red problem -> trinode restructure through right rotation.

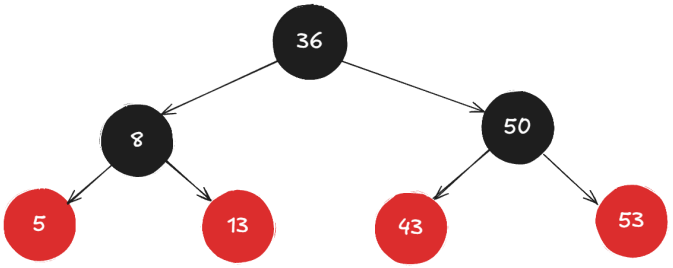


Insert 13 - To the right of 5, it's a leaf node so colour it red

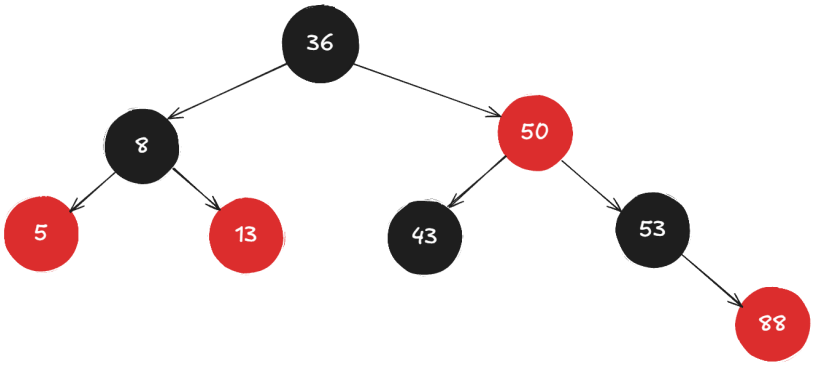


Insert 8 - To the left of 13 -> double red problem -> 8's uncle is black (nullptr), so do a trinode restructure through right-left rotation.

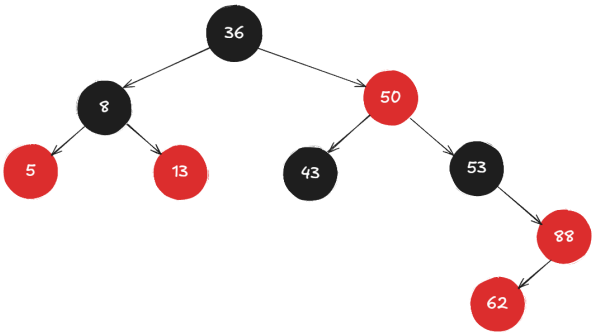


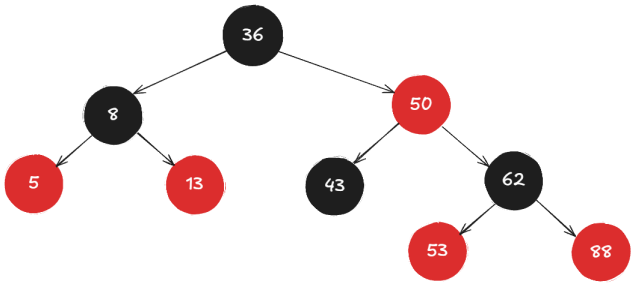


Insert 88 - To the right of 53 -> double red problem -> 88's uncle is red -> colour 43 and 53 black, colour 50 red, propagate problem to 50. There is no double red problem at 50.

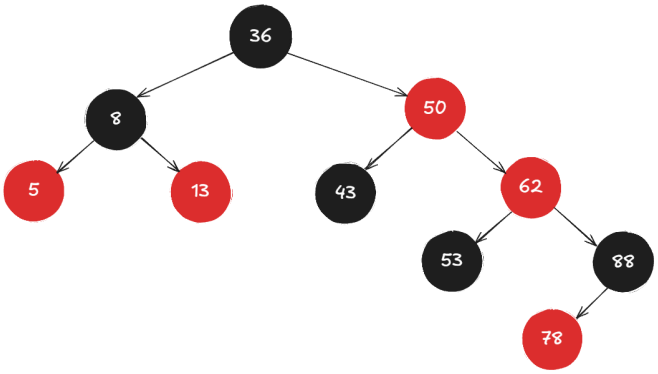


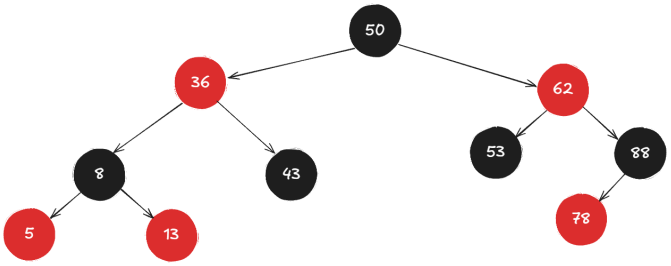
Insert 62 - To the left of 88 -> double red problem -> 62's uncle is black (nullptr) -> trinode restructure through a right-left rotation.



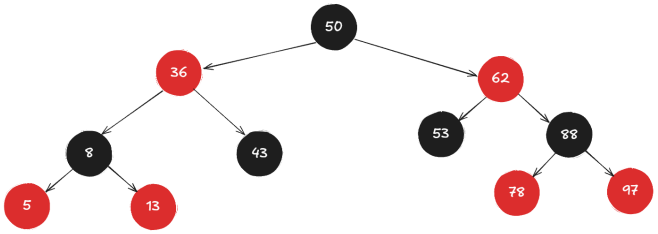


Insert 78 - To the left of 88 -> double red problem -> 78's uncle is red -> recolour 53 and 88 to be black, 62 to red. Propagate double red problem to 62 -> 62's uncle is black -> trinode restructure through left rotation.

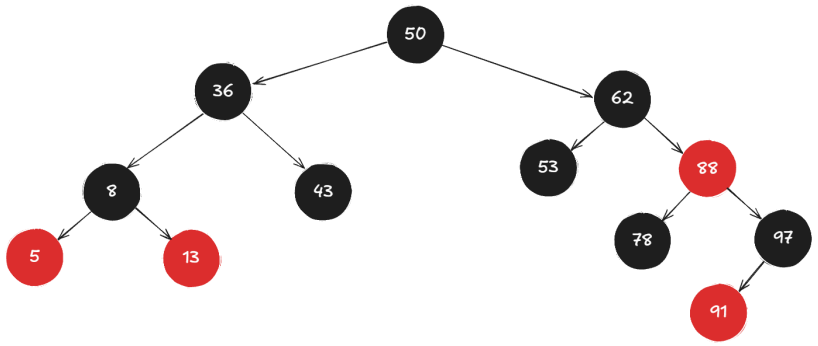




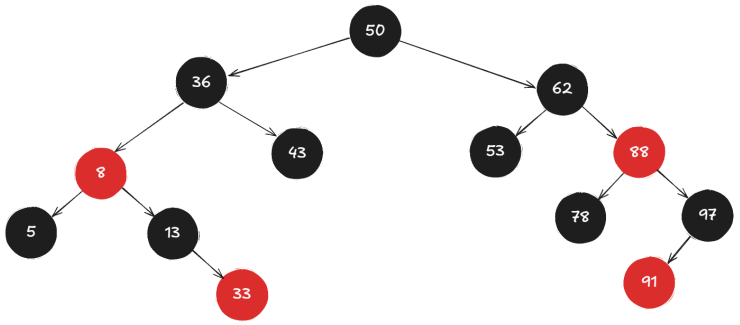
Insert 97 - To the right of 88 -> leaf node so colour it red, no double red problem.



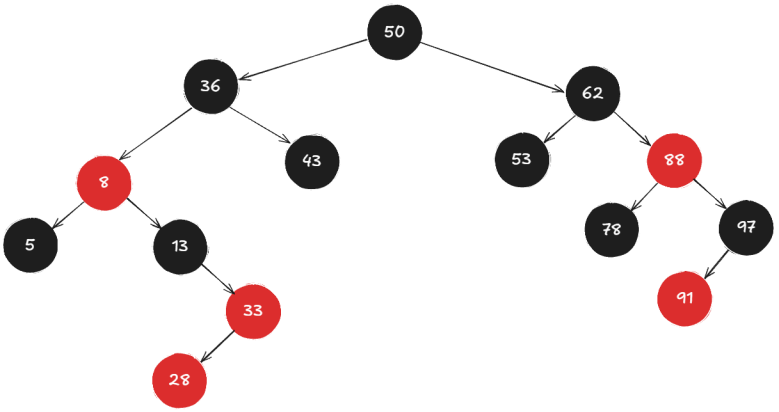
Insert 91 - To the left of 97 -> double red problem -> 91's uncle is red -> colour 78 and 97 black, colour 88 red. Propagate double red problem to 88 -> 88's uncle is red -> colour 62 and 36 black, keep 50 black as it's the root node.

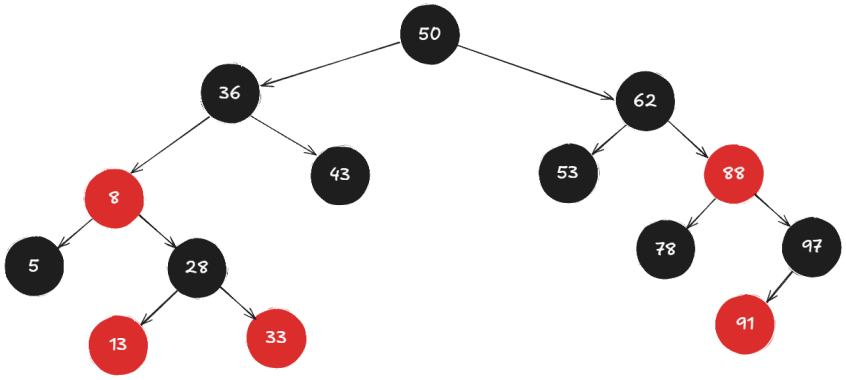


Insert 33 - To the right of 13 -> double red problem -> 33's uncle is red -> colour 5 and 13 black, colour 8 red. There is no double red problem at 8.

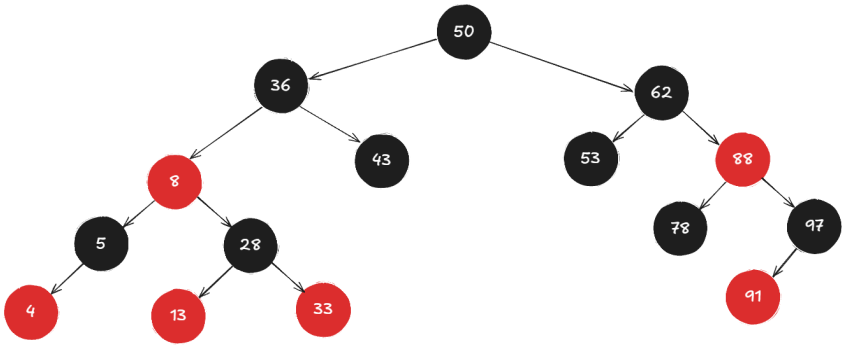


Insert 28 - To the left of 33 -> double red problem -> 28's uncle is black -> trinode restructure through left rotation.

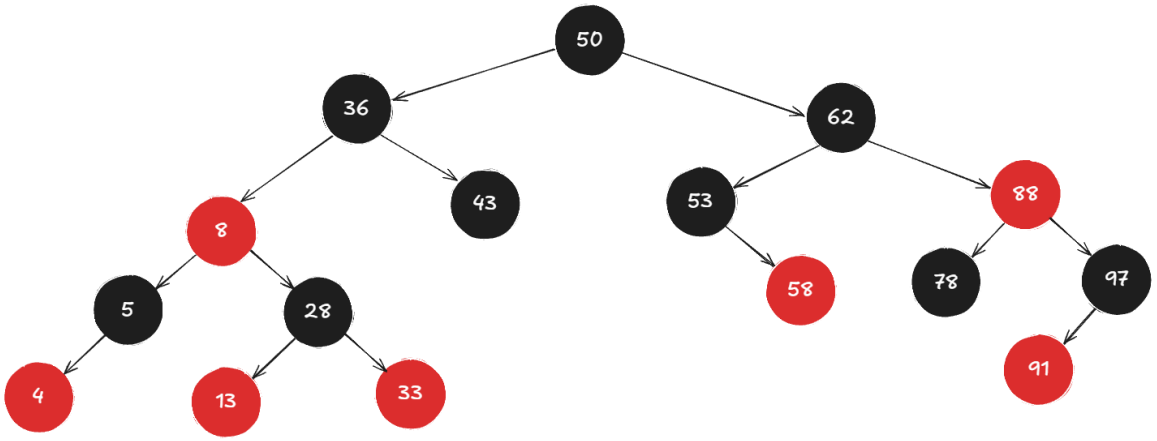




Insert 4 - To the left of 5, no double red problem.

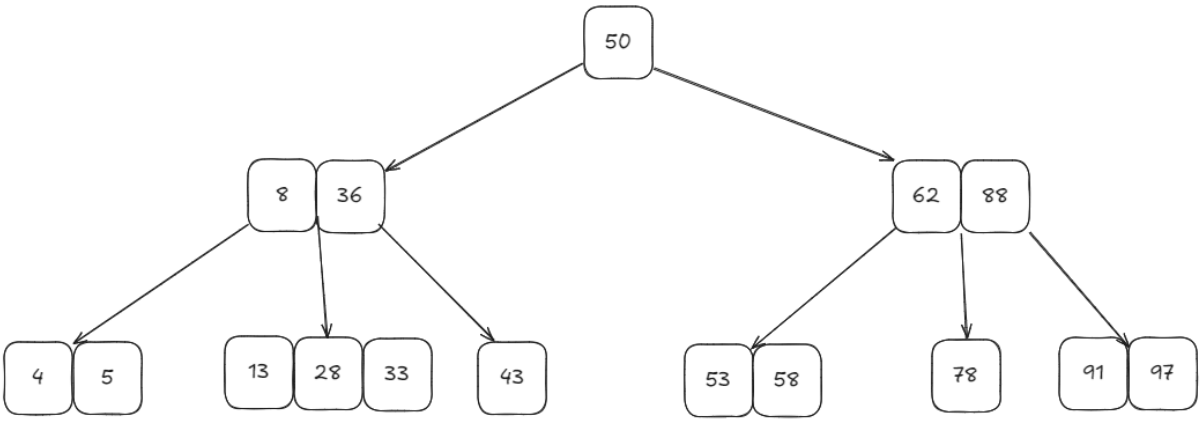


Insert 58 - To the right of 53, no double red problem.



This is the final red black tree.

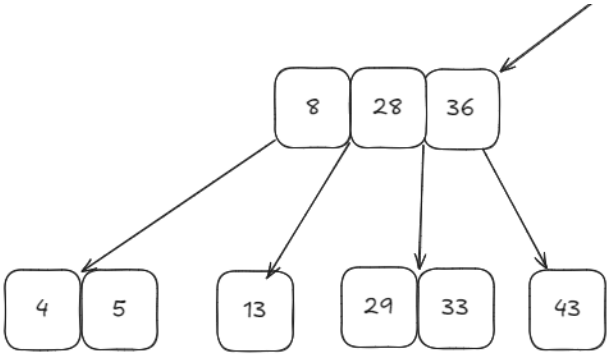
1b. Draw the (2,4)-tree that is equivalent to the red-black tree you gave in the previous question.



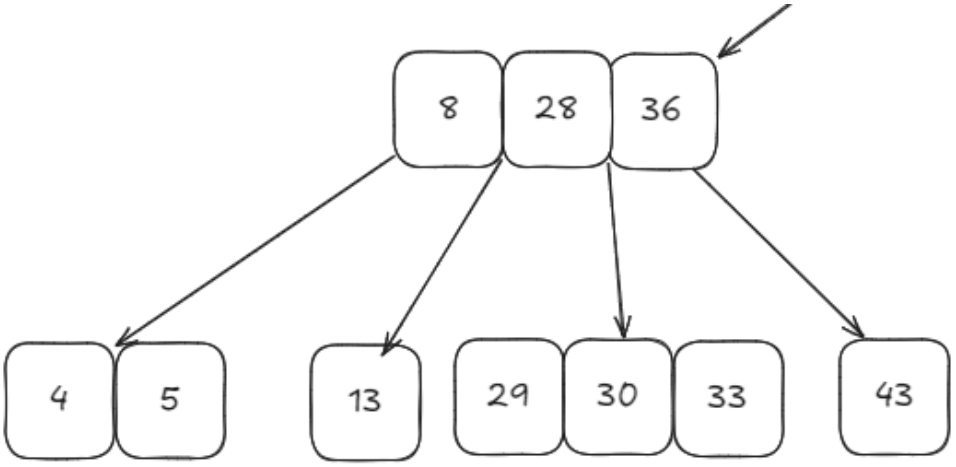
1c. What is the minimum number of data items (numbers) to be inserted into your (2,4)-tree you gave in the previous question to result in a split operation of an internal node in your (2,4)-tree? Show an example of such a set of data items (numbers).

You can cause an internal node split through 3 data insertions.

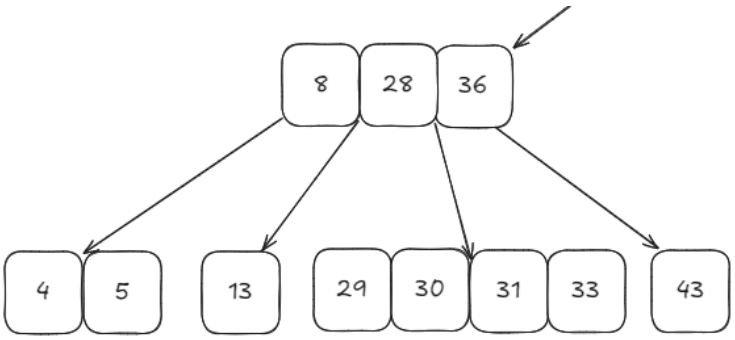
Step 1. If you first insert 29, you cause the (13, 28, 33) node to split, pushing 28 to the (8, 36) node so that it is (8, 28, 36).

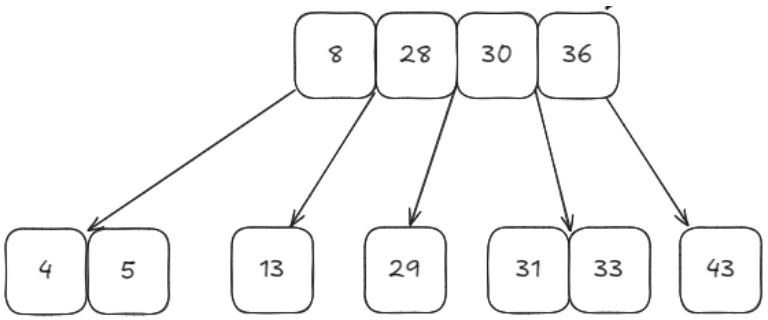


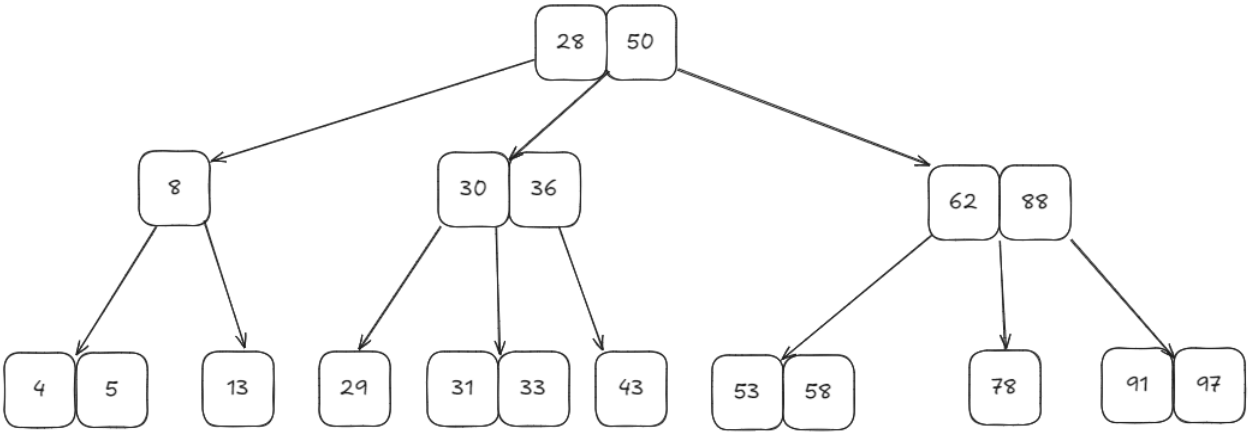
Step 2. If you then insert 30, it will be added to the (29, 33) node making it (29, 30, 33).



Step 3. If you insert 31, the (29, 30, 33) node will split, causing 30 to be pushed up into the (8, 28, 36) node, causing a split on an internal node.







2. Compute a table representing the KMP failure function for the following pattern string within the double quotation marks: "tattarrattat"
(Note: The double quotation marks are not part of the string. The meaning of the word is "the imitation of the sound of knocking on a door".)

Index (j)	Substring from 0 to j	Longest prefix-suffix match	Length of prefix-suffix match
0	t	none	0
1	ta	none	0
2	tat	t	1
3	tatt	T	1

4	tatta	ta	2
5	tattar	none	0
6	tattarr	none	0
7	tattarra	none	0
8	tattarrat	t	1
9	tattarratt	t	1
10	tattarratta	ta	2
11	tattarrattat	tat	3

Final array: {0, 0, 1, 1, 2, 0, 0, 0, 1, 1, 2, 3}

3. Given the following text string:
“huffman encoding algorithm uses optimized variable length bit strings to encode characters in a given string x over some alphabet. the optimization is based on the frequencies of the characters used in string x. the basic idea of the optimization is to use fewer digits to represent the characters with high frequencies. it is a greedy algorithm.”

3a. Assume the alphabet contains the lower case letters only (i.e., you can ignore the word delimiters and punctuations). Compute the frequency table for letters appearing in the text only. Show both the algorithm you used to generate the table and the resulting table itself.

Symbol	**Frequency**
e	37
i	31
t	29
s	21
a	20
r	19
h	17
n	17
o	17
c	11
g	11

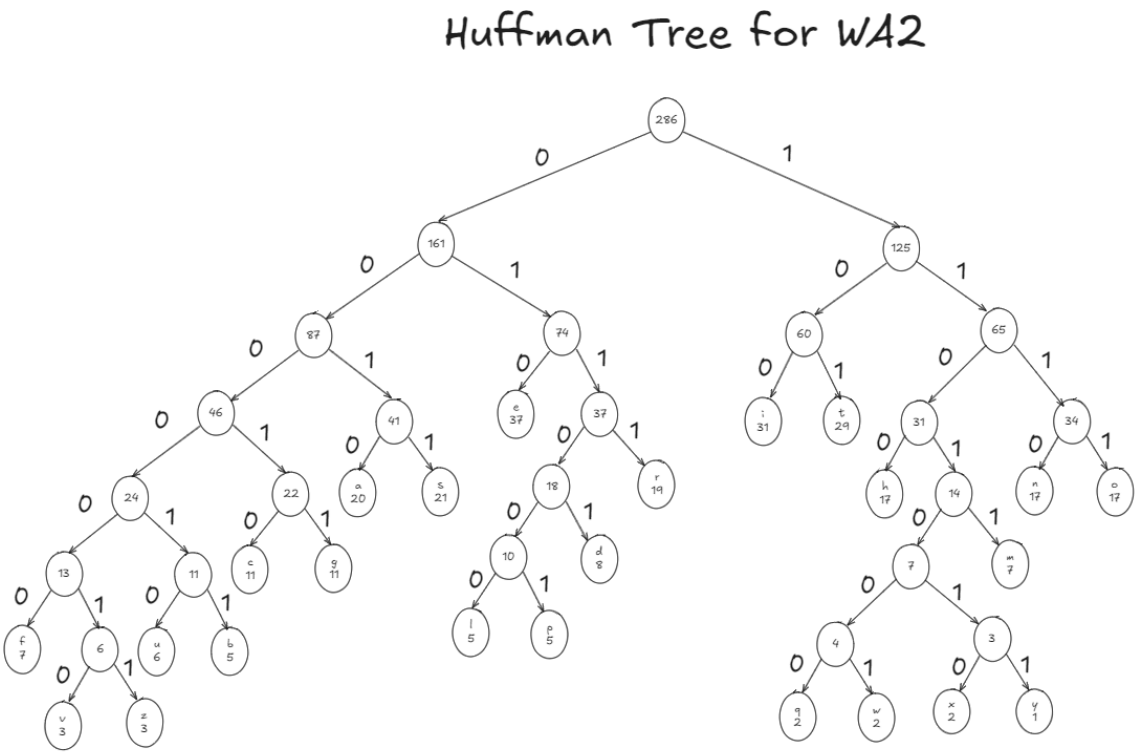
d	8
f	7
m	7
u	6
b	5
l	5
p	5
v	3
z	3
q	2
w	2
x	2
y	1

Here is the C++ script I wrote to count the frequency of all letters.

huffman.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {

    // string given by question 3 on the assignment:
    string input = "huffman encoding algorithm uses optimized variable length bit strings to encode char

    int frequency[26]; // for lowercase a-z

    for (int i = 0; i < 26; i++) {
        frequency[i] = 0; // initialize all values
    }

    for (int i = 0; i < input.length(); i++) {
        int c = (int)input[i];
        if (c < 97 || c > 122) { // ascii range for lowercase a-z
            continue;
        } else {
            frequency[c-97]++; // adjust for frequency array
        }
    }

    // print results, on the google doc the table was sorted by frequency number rather than letter.
    for (int i = 0; i < 26; i++) {
        cout << char(i+97) << ": " << frequency[i] << endl;
    }


    return 0;
}
```

3b. Draw the Huffman tree based on your frequency table.



Note: I put 0s on all the left edges and 1s on all the right edges for the encoding in 3c.

3c. Show the Huffman code assigned to each letter.

The resulting encoding:

Symbol	**Huffman Code**
e	010
i	100
t	101
s	0011
a	0010
r	0111
h	1100
n	1110
o	1111
c	00010
g	00011
d	01101

f	000000
m	11011
u	000010
b	000011
l	011000
p	011001
v	0000010
z	0000011
q	1101000
w	1101001
x	1101010
y	1101011

4. Using the Lempel-Ziv algorithm to decode the following binary code string, where white space and newline characters are used to separate the codes.

010000 010010 001111 000010 000001 000010 001100 000101 000000
001001 001101 010000 001111 010011 010011 001001 000010 001001 001100
001001 010100 001001 000101 010011 000000 000001 010010 100010
010100 001111 000000 000010 100010 011011 000101 000110 000101 010010
110101 000100 000000 110111 100011 100101 011100 011110 100000 111011
100111 101001 101011 101101 101111 110001

The following rules are used when encoding a message to the above code string:

- **The alphabet should contain the white space and the 26 lower case letters (in that order).**
- **6 bits are used to encode each entry in the codebook dictionary.**
- **When fully filled, the codebook dictionary simply stops accepting new entries. And the rest of the message is encoded using this codebook dictionary.**
- **If you are curious how the above code is generated, here is the source code file I used to encode the message.**

4a. Show the decoded message.

The decoded message is:

probable impossibilities are to be preferred to improbable possibilities

4b. If you did the decoding manually (not recommended), attach your final codebook dictionary. If you did the decoding using a program, attach your program's source code.

My program's source code:

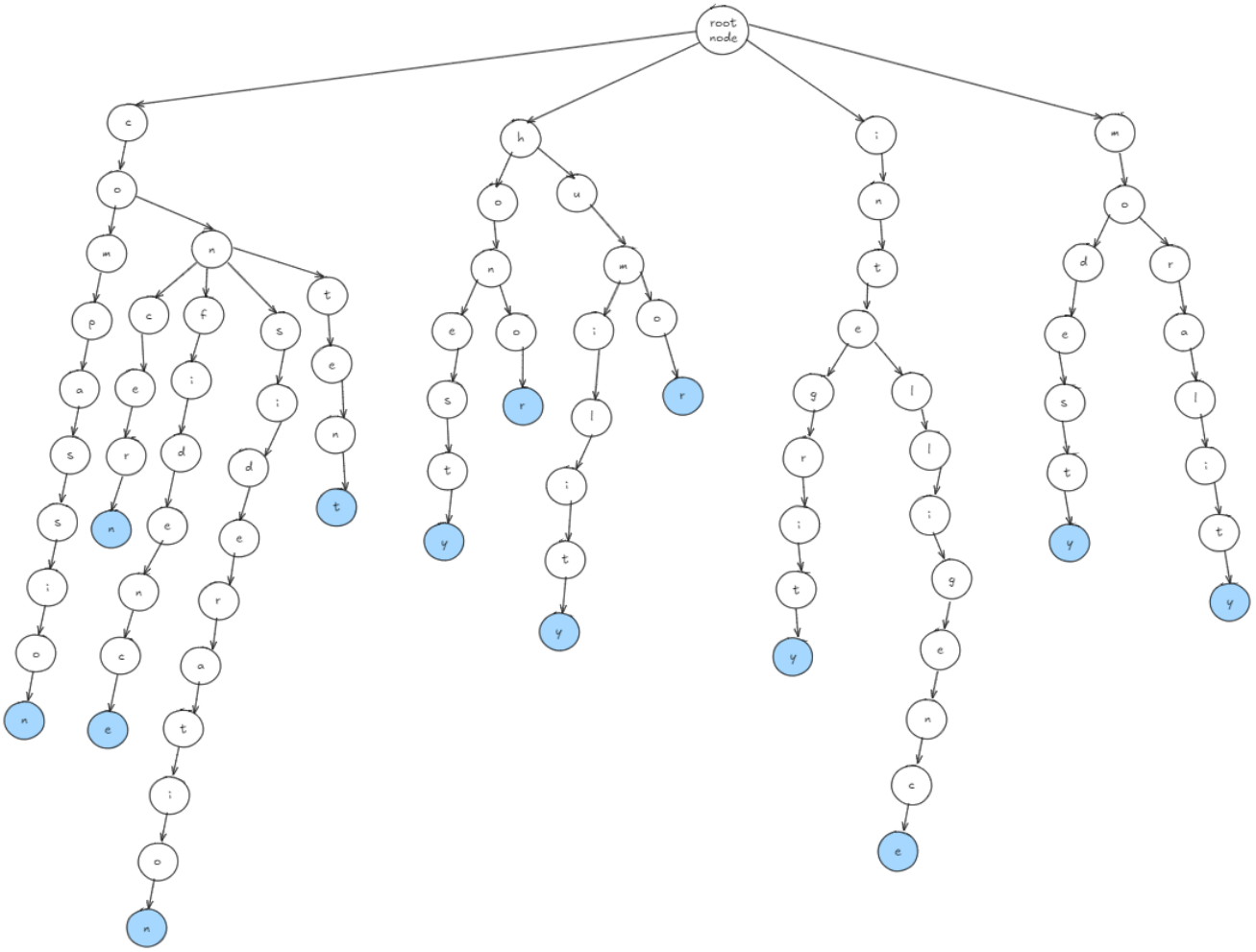
```
/**
 * @file lempelziv.cpp
 * @author Luka Karanovic, 665778833, F24N03, CSCI 260, VIU
 * @version 1.0.0
 * @date December 2, 2024
 *
 * Script to decode lempel-ziv encoding for question 4 of Written Assignment 2.
 */

#include <iostream>
#include <string>
#include <cmath>
using namespace std;

int main() {

    // string given by question 4 on the assignment: i had to add a space at the very end to make the last else statement trigger
    string input = "010000 010010 001111 000010 000001 000010 001100 000101 000000 001001 001101 010000 001111 010011 010011 001001 000010 001001 001100 001001 010100 001001 000101 010011 000000 000001 010010 100010 010100 001111 000000 000010 100010 011011 000101 000110 000101 010010 110101 000100 000000 110111 100011 100101 011100 011110 100000 111011 100111 101001 101011 101101 101111 110001 ";

    // space then lowercase a-z
    string dictionary[64];

    // when fully filled, codebook dictionary simply stops accepting new entries and rest of message is encoded using this codebook dictionary

    // initliaze array
    dictionary[0] = ' ';
    for (int i = 1; i < 27; i++) {
        dictionary[i] = 'a' + (i-1);
    }

    // placeholder values for dictionary
    for (int i = 27; i < 63; i++) {
```

```
        dictionary[i] = "0";
    }

    // output
    string X;

    // string from dictionary for current code
    string curWord;

    // string from dictionary for previous code
    string prevWord;

    // new codeword entry into dictionary
    string codeWord;

    // used to get each binary code in its own string
    string temp;
    // for binary calcs
    int C = 0;
    int power = 0;

    int length = input.length();


    for (int i = 0; i < length; i++) {
        // compute binary code C
        if (input[i] == '1' || input[i] == '0') {
            temp += input[i];
        } else {
            for (int i = 5; i >= 0; i--) {
                if (temp[i] == '1') {
                    C += pow(2, power);
                }
                power++;
            }
            temp.clear();
            // for first code C
            if (prevWord.length() == 0) {
                curWord = dictionary[C];
                X += curWord;
                prevWord = curWord;
                power = 0;
```

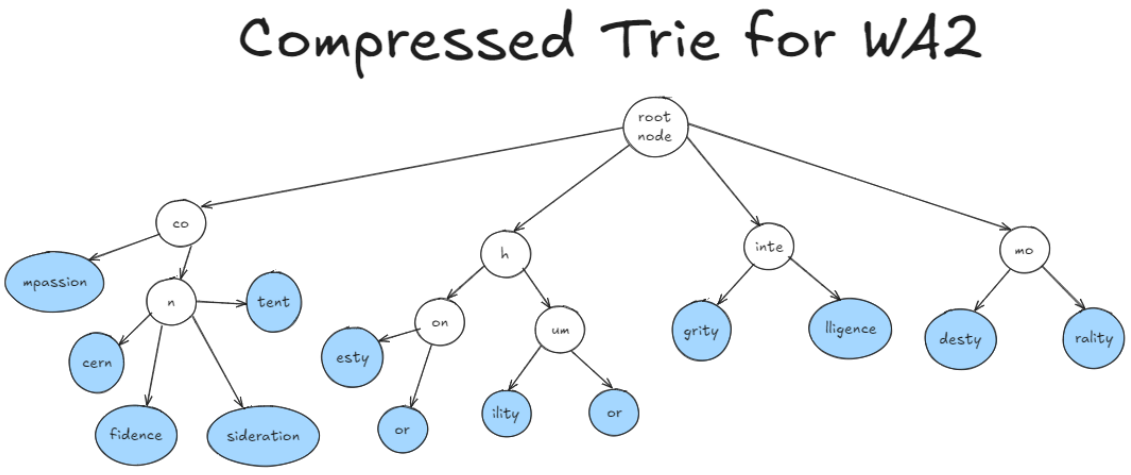
```
                C = 0;
                continue;
            }
            // for every other code C
            curWord = dictionary[C];
            X += curWord;
            codeWord = prevWord + curWord[0];

            // check if codeword is in dictionary or if dictionary is already full, and
find next available spot
            int i = 0;
            bool duplicate = true;
            while (dictionary[i] != "0" && i < 64) {
                i++;
                if (codeWord == dictionary[i]) {
                    duplicate = false;
                }
            }
            // add new codeword to dictionary
            if (i < 64 && duplicate == true) {
                dictionary[i] = codeWord;
            }
            prevWord = curWord;

            power = 0;
            C = 0;
        }
    }

    // final output string
    cout << X << endl;

    return 0;
}
```

5. Draw a standard trie and a compressed trie respectively to store the following set of strings: {compassion, concern, confidence, consideration, content, honesty, honor, humility, humor, integrity, intelligence, modesty, morality}

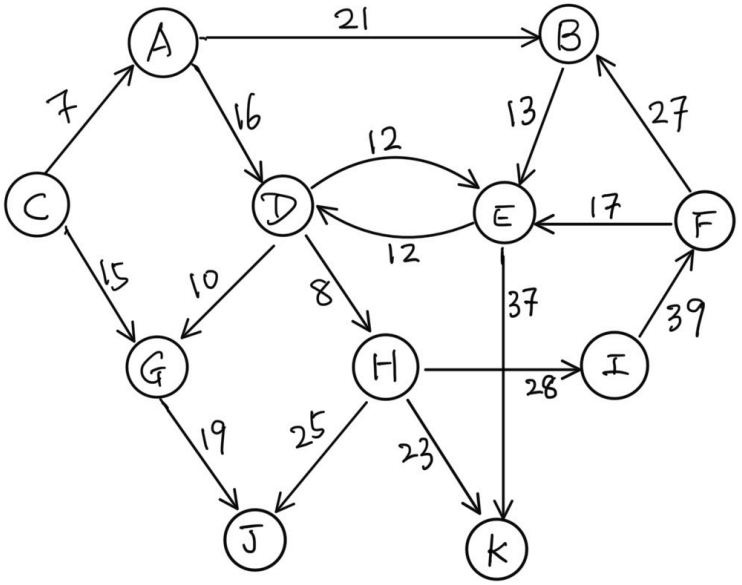
Standard Trie: Terminal nodes are coloured.



Compressed Trie: Terminal nodes are coloured.



6. Based on the directed and weighted graph shown below:



6a. Draw the adjacency matrix of the given graph.

Row, then column

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**
A	0	21	0	16	0	0	0	0	0	0	0
B	0	0	0	0	13	0	0	0	0	0	0
C	7	0	0	0	0	0	15	0	0	0	0
D	0	0	0	0	12	0	10	8	0	0	0
E	0	0	0	12	0	0	0	0	0	0	37
F	0	27	0	0	17	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	19	0
H	0	0	0	0	0	0	0	0	28	25	23
I	0	0	0	0	0	39	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0

6b. Show the adjacent vertices list of each vertex.

Node	**Index**	**List (destination node, weight)**
A	0	(1, 21), (3,16)

B	1	(4, 13)
C	2	(0, 7), (6, 15)
D	3	(4, 12), (6, 10), (7, 8)
E	4	(3, 12), (10, 37)
F	5	(1, 27), (4, 17)
G	6	(9, 19)
H	7	(8, 28), (9, 25), (10, 23)
I	8	(5, 39)
J	9	
K	10	

6c. Show the order of the vertices as they are visited in a DFS (depth first search) traversal starting from vertex A.

I decided to visit unvisited neighbours alphabetically, so suppose that this "randomly happened" (as the vertex selection is random).

Using this pseudocode algorithm from Wikipedia:

```
procedure DFS(G, v) is
    label v as discovered
    for all directed edges from v to w that are in
G.adjacentEdges(v) do
        if vertex w is not labeled as discovered then
            recursively call DFS(G, w)
```

Order of vertices visited: A, B, E, D, G, J, H, I, F, K

6d. Show the order of the vertices as they are visited in a BFS (breadth first search) traversal starting from vertex A.

I decided to visit unvisited neighbours alphabetically, so suppose that this "randomly happened" (as the vertex selection is random).

Using this pseudocode algorithm from Wikipedia:

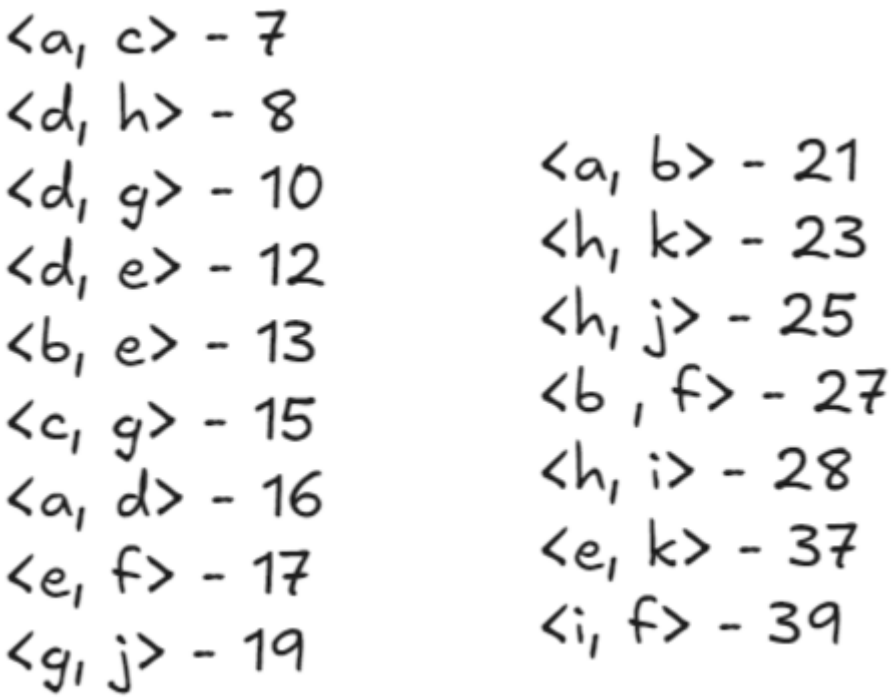
```
procedure BFS(G, root) is
    let Q be a queue
    label root as explored
    Q.enqueue(root)
    while Q is not empty do
        v := Q.dequeue()
        if v is the goal then
            return v
        for all edges from v to w in G.adjacentEdges(v) do
            if w is not labeled as explored then
                label w as explored
                w.parent := v
                Q.enqueue(w)
```

Order of vertices visited: A, B, D, E, G, H, K, J, I, F

6e. Draw the minimum spanning tree of the given graph. For the purpose of this question only, treat all the edges as undirected ones.

Using Kruskal's Algorithm (since edges are undirected):

1. Sort the edges in the tree by minimum weight



2. While there are less than n-1 edges
    a. Add the edge to the minimum spanning tree as long as u and v are not in the same cluster (then merge u and v's clusters to one).
3. Repeat step 2 until all edges are in the tree.

Edges

<a, c> - 7
<d, h> - 8
<d, g> - 10
<d, e> - 12
<b, e> - 13
<c, g> - 15
<a, d> - 16
<e, f> - 17
<g, j> - 19
<a, b> - 21
<h, k> - 23
<h, j> - 25
<b , f> - 27
<h, i> - 28
<e, k> - 37
<i, f> - 39

Construct T:
d, h, g, e, b, c, a, f, j, k, i

<a, c> - 7
<d, h> - 8
<d, g> - 10
<d, e> - 12
<b, e> - 13
<c, g> - 15
a and d are in the same cluster
<e, f> - 17
<g, j> - 19
a and b are in the same cluster
<h, k> - 23
h and j are in the same cluster
b and f are in the same cluster
<h, i> - 28
T has n-1 edges

T: