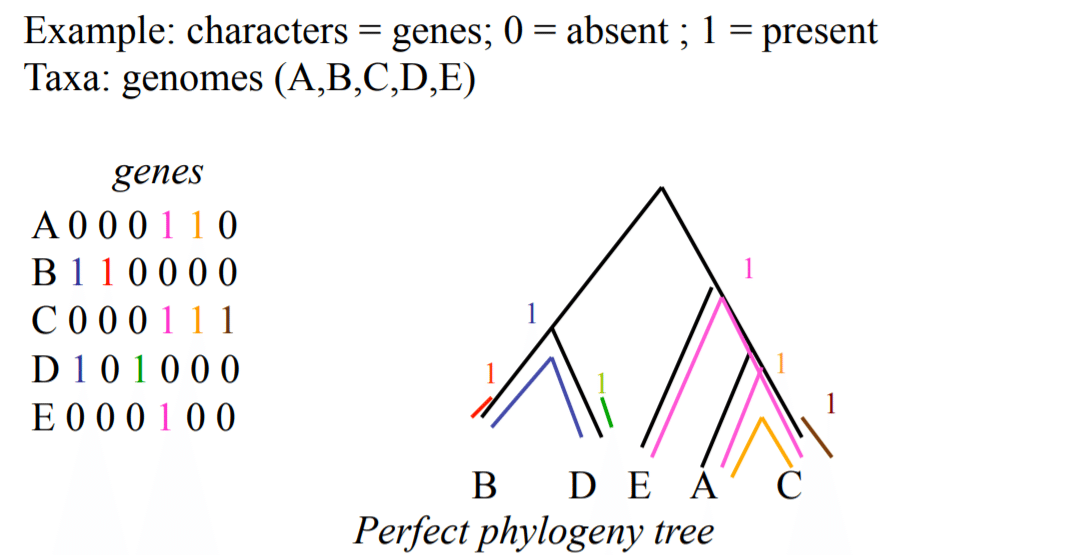
Assignment 2

This assignment is to determine if a given character-state matrix, M, has a perfect phylogeny, using the compatibility approach. If there exists a perfect phylogeny, then the tree is constructed.

* Implemented code works only for Binary matrix
* Algorithm assumes that all the species/rows are distinct
* No row should have all the characters 0

Below Image shows the example which can be used to test the algorithm:

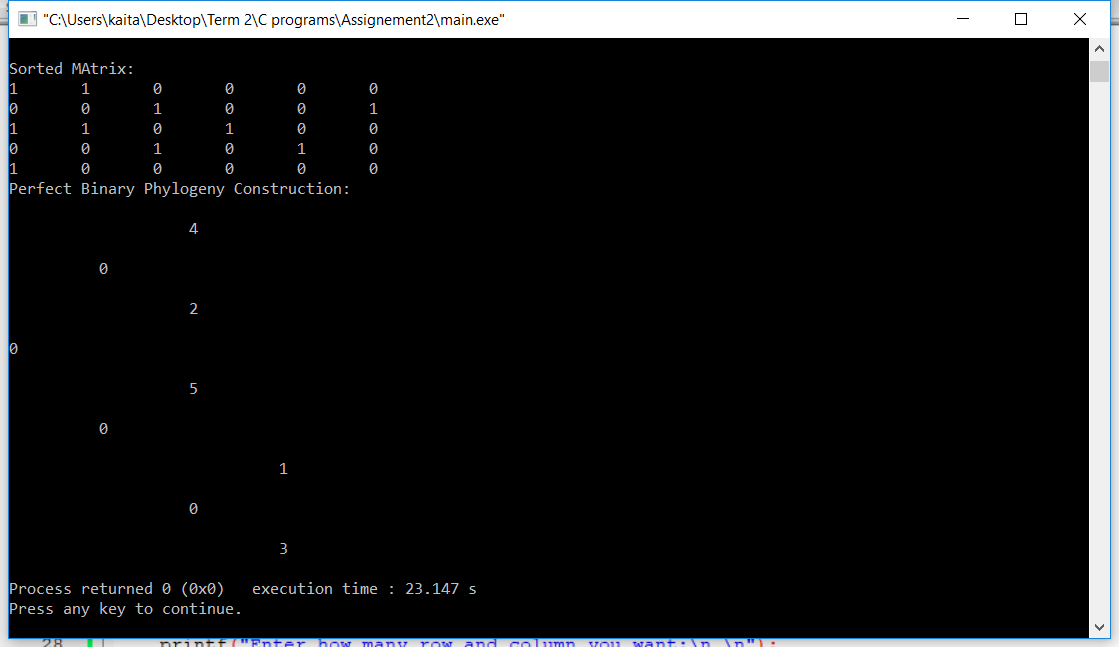
The

**Steps to run the code:**

* With the assignment, I have attached C main program file, just Import the file and run the program.

**Steps to test the code:**

* The code will first ask the no of rows and columns.
* User must enter the matrix column wise.
* It will then check for the perfect phylogeny, if the matrix is not perfect phylogeny then it will return stating the message.
* If the matrix is perfect phylogeny, then the tree will be constructed and printed as shown below.
* Please note that the index(objects) will start from 1**. 0/zero shown in the tree is empty node.**
* The tree is constructed side ways, as shown below in the output.



**Code:**

#include <stdio.h>

#include <stdlib.h>

#define COUNT 10

struct node

{

int data;

int leftEdge;

int rightEdge;

struct node \*left;

struct node \*right;

};

int mat[100][100];

int row,column,i,j,k,l;

void radixsort(int arr[], int n);

int perfectPhylogenyCheck();

struct node\* newNode(int data,int leftEdge,int rightEdge);

struct node\* perfectBinaryPhylogenyConstructionMethod(struct node \*root);

void print2DUtil(struct node \*root, int space);

void print2D(struct node \*root);

struct node\* leftNodeTraverse(struct node \*temp);

struct node\* rightNodeTraverse(struct node \*temp);

int main()

{

printf("Enter how many row and column you want:\n \n");

scanf("%d",&row);

scanf("%d",&column);

printf("Enter the matrix column wise:");

int count[column];

for(j=0; j<column; j++)

{

count[j] = 0;

}

for(i=0; i<row; i++)

{

for(j=0; j<column; j++)

{

scanf("%d",&mat[i][j]);

if(!(mat[i][j] ==0 || mat[i][j] ==1))

{

printf("Invalid matrix value added.. run the program again...");

return 0;

}

// Fill the count array, stating each column's number of ones in it

if(mat[i][j] == 1)

{

count[j]++;

}

}

printf("\n");

}

// Assumes all columns are distinct

//Sort columns based on the number of Is in each column using radix-sort

// (columns with more Is come first)

int n = sizeof(count)/sizeof(count[0]);

radixsort(count, n);

int perfectPhylogeny = perfectPhylogenyCheck();

if(perfectPhylogeny == 1)

{

/\*create root\*/

struct node \*root = newNode(0,NULL,NULL);

root = perfectBinaryPhylogenyConstructionMethod(root);

printf("Perfect Binary Phylogeny Construction: \n");

print2D(root);

}

}

struct node\* perfectBinaryPhylogenyConstructionMethod(struct node \*root){

struct node \*currentNode = root;

for(i=0; i<row; i++)

{

currentNode = root;

for(j=0; j<column; j++)

{

if(mat[i][j] == 1)

{

int leftEdge = (int)currentNode->leftEdge;

int rightEdge = (int)currentNode->rightEdge;

if(leftEdge == j+1)

{

currentNode = currentNode->left;

}

else if(rightEdge == j+1)

{

currentNode = currentNode->right;

}

else

{

if(currentNode->leftEdge <=0)

{

currentNode->left= newNode(0,NULL,NULL);

currentNode->leftEdge = j+1;

currentNode = currentNode->left;

}

else if(currentNode->rightEdge <=0)

{

currentNode->right= newNode(0,NULL,NULL);

currentNode->rightEdge = j+1;

currentNode = currentNode->right;

}

}

}

}

currentNode->data = i+1;

}

// for each node u except root do

// Create as many leaves linked to u as there are objects in u

traverseAndCreateNodeInorder(root);

return root;

}

void traverseAndCreateNodeInorder(struct node\* node)

{

if (node == NULL)

return;

struct node\* temp = node;

int leftEdge = (int)temp->leftEdge;

int rightEdge = (int)temp->rightEdge;

if(leftEdge > 0 && rightEdge <= 0 && temp->data > 0)

{

temp->right= newNode((int)temp->data,NULL,NULL);

temp-> data = 0;

}

traverseAndCreateNodeInorder(node->left);

traverseAndCreateNodeInorder(node->right);

}

int perfectPhylogenyCheck()

{

int L[row][column];

for(i=0; i<row; i++)

{

for(j=0; j<column; j++)

{

L[i][j]=0;

}

}

// Compute L- Auxilliary Matrix

for(i=0; i<row; i++)

{

k=-1;

for(j=0; j<column; j++)

{

if(mat[i][j]==1)

{

L[i][j] = k;

// k is the rightmost column to the left of j such that

// Mik =1. If no such column exists, k = -1

k = j;

k++;

}

}

}

//Check columns of L

for(j=0; j<column; j++)

{

for(i=0; i<row; i++)

{

for(l=0; l<row; l++)

{

if(L[i][j] != L[l][j] && L[i][j] != 0 && L[l][j] != 0)

{

printf("Given binary matrix is not perfect phylogeny");

return 0;

}

}

}

}

return 1;

}

// A utility function to get maximum value in arr[]

int getMax(int arr[], int n)

{

int mx = arr[0];

for (int i = 1; i < n; i++)

if (arr[i] > mx)

mx = arr[i];

return mx;

}

// A function to do counting sort of arr[] according to

// the digit represented by exp.

void countSort(int arr[], int n, int exp)

{

int output[n]; // output array

int outputMat[row][n];

int i, count[10] = {0};

// Store count of occurrences in count[]

for (i = 0; i < n; i++)

count[ (arr[i]/exp)%10 ]++;

// Change count[i] so that count[i] now contains actual

// position of this digit in output[]

for (i = 1; i < 10; i++)

count[i] += count[i - 1];

// Build the output array

for (i = n - 1; i >= 0; i--)

{

output[count[ (arr[i]/exp)%10 ] - 1] = arr[i];

for(int a=0; a<row; a++)

{

outputMat[a][count[ (arr[i]/exp)%10 ] - 1] = mat[a][i];

}

count[ (arr[i]/exp)%10 ]--;

}

// Copy the output array to arr[], so that arr[] now

// contains sorted numbers according to current digit

int j=0;

for(int r=column-1; r>=0; r--)

{

//copy the sorted matrix

for(i=0; i<row; i++)

{

mat[i][j] = outputMat[i][r];

}

j++;

}

printf("Sorted MAtrix: \n");

for(i=0; i<row; i++)

{

for(j=0; j<column; j++)

{

printf("%d \t",mat[i][j]);

}

printf("\n");

}

}

// Radix Sort

void radixsort(int arr[], int n)

{

// Find the maximum number to know number of digits

int m = getMax(arr, n);

// Do counting sort for every digit. Note that instead

// of passing digit number, exp is passed. exp is 10^i

// where i is current digit number

for (int exp = 1; m/exp > 0; exp \*= 10)

countSort(arr, n, exp);

}

/\* newNode() allocates a new node with the given data and NULL left and

right pointers. \*/

struct node\* newNode(int data,int leftEdge,int rightEdge)

{

// Allocate memory for new node

struct node\* node = (struct node\*)malloc(sizeof(struct node));

// Assign data to this node

node->data = data;

node-> leftEdge = leftEdge;

node-> rightEdge = rightEdge;

// Initialize left and right children as NULL

node->left = NULL;

node->right = NULL;

return(node);

}

// Wrapper over print2DUtil()

void print2D(struct node \*root)

{

// Pass initial space count as 0

print2DUtil(root, 0);

}

// Function to print binary tree in 2D

// It does reverse inorder traversal

void print2DUtil(struct node \*root, int space)

{

// Base case

if (root == NULL)

return;

// Increase distance between levels

space += COUNT;

// Process right child first

print2DUtil(root->right, space);

// Print current node after space

// count

printf("\n");

for (int i = COUNT; i < space; i++)

printf(" ");

printf("%d\n", root->data);

// Process left child

print2DUtil(root->left, space);

}