CpSc 2120 Fall 2020

Quiz 1 In-Class Part. 30 possible points

Please fill out your solutions in this text file and upload it on

handin; files with upload timestamps beyond the end of lecture will

not be counted. You can use any resources you want for this quiz

except other humans (although given the time constraint, you are

highly encouraged not to waste time searching online for material; the

questions below have been designed specifically so this won't be a

helpful strategy). When describing any algorithm, it's ok to be

concise. You can use either code or English or a mix (if using

English, be as precise as possible). Whenever you describe an

algorithm, give its running time as well. Faster algorithms are worth

more points than slower algorithms.

1. Running Times (6 points)

Please write the running time of each example below using Theta()

notation as a function of N. Assume each one is called with a

positive integer value of N.

(i) void john(int n) { while (n--); } Theta(N)

(ii) void taylor(int n) { if(n) { taylor(n/2); } } Theta(log(N)

(iii) void matthew(int n) { if(n) { john(n); matthew(n/2); } } Theta(Nlog(N)

(iv) void jacob(int n) { if(n) { john(n); jacob(n-1); } } Theta(n^2)

(v) void tahj(int n) { if(n) { tahj(n-1); tahj(n-1); } } Theta(N)

(vi) void madison(int n) { if(n>=23) { madison(n/2); madison(n/2); } } Theta(log(N)

2. Predfind on a Linked List (6 points)

Consider our standard linked list node structure:

struct Node {

int key;

Node \*next;

};

Please fill in the following function, which should operate like the

"predfind" function we wrote for a BST. Given the head pointer to a

linked list of distinct keys as well as a value V, it should return a

pointer to the node in the list whose key is the largest among all

keys in the list that are at most V. If a node in the list exists

with key equal to V, that node should be returned. As an example, if

the list looks like this:

4 -> 15 -> 7 -> 43 -> 12 -> 13

(head)

then a call to predfind(head, 9) should return a pointer to the node

with key 7. If there is no node in the list with key at most V, you

should return NULL.

For full credit on this problem and a general sense of fulfillment in

life, you should write your function recursively.

Node \*predfind(Node \*head, int v)

{

if(head==nullptr || head->key == v) {return}

if (v < root->key) return predFind(root->next, v);

Node \*result = predFind(root->next, v);

return result == nullptr ? root : result;

}

3. Fall Colors (6 points)

You are given as input a file containing N strings representing fall

colors (e.g., "orange"); these strings are listed in arbitrary order.

You are also given the pointer to the root of a binary search tree

with N total nodes, each with a string as its key (this is a standard

binary search tree, where nodes on the left are smaller and nodes on

the right are larger). You may assume each string is short (at most

12 characters). Please describe an algorithm that checks if every

\_leaf\_ in the binary search tree is a fall color. Remember that for

this and the remaining questions, you can describe your algorithm in

English if you prefer along with or instead of code.

Take all the fall colors and add up their ASCII codes. Whatever corresponding codes represent fall colors that's what we want.

void print\_fall(Node \*T) {

if (T == nullptr) return;

print\_fall(T->left);

if(T->key == whatEverOurFallAsciiIs){

cout << T->colorString << endl;

}

print\_inorder(T->right);

}

4. Like 2150, But More Fun (6 + 6 points)

The students in 2150 just finished a project where they had to

implement a tic-tac-toe game on a board of size 8x8. In an effort

that will surely in no way trigger these students, we've re-imagined

below what a more algorithmically fun version of this project could

have been.

(a) Minecraft tic-tac-toe. Consider 3D game board of size N x N x N.

N of the cells in this board are occupied by X's. You are told their

locations as input, but not in any particular order. Each location is

described by 3 coordinates (just like the three indices you would use

to address a location in a 3D array). Please describe an algorithm

that computes the longest consecutive run of X's. For simplicity, we

don't count diagonals here, so a "run" consists of series of X's that

have consecutive values in one coordinate, while they agree in the

other two coordinates. For example, X's at locations (2, 2, 7), (2, 3,

7), (2, 4, 7), and (2, 5, 7) form a "run", since they have consecutive

values (2, 3, 4, 5) in the second coordinate, and they have consistent

values in the first and third coordinates (all have the common value 2

as the first coordinate and 7 as the third coordinate).

for (int i = 0; i < getNumRows(); i++) {

for (int j = 0; j < getNumColumns(); j++) {

for(int k = 0; k<getNumWidth(); k++){

/\* Check if have a player, if so add to the streak till it equals getNumToWin \*/

if (board[i,j,k] == player) {

countForWin++;

} else {

countForWin = 0;

}

if (countForWin == getConsecutiveStreakToWin()) {

return true;

}

}

}

}

(b) Expanding tic-tac-toe. Consider a 1D game board of length N. The

"X" player and "O" player take turns. In the first move, the "X"

player can place a single X on one of the N cells in the game board.

In the next move, the "O" player places Os on two consecutive empty

positions on the board. In the next move, the "X" player places Xs on

three consecutive empty positions on the board. In general, each step

requires a player to cover a consecutive segment of the board with a

run of pieces, and the size of this segment increments at each move.

A player loses if the run they attempt to play overlaps with any other

piece. You want to design an algorithm that detects when a loss

occurs, and then terminates. The input to your algorithm is a stream

of integers specifying where players move. For example, if the first

integer in the input stream is 7, this means an X is played at

location 7. If the next integer is 3, this means Os are played at

locations 3 and 4. If the next integer is 10, this means Xs are

played at locations 10, 11, and 12. In general, each integer

specifies the starting position of each run.

Int myCounter = 1;

bool detectLoss(int someIntegerStream){

Int myCountertmp = myCounter-1;

std::vector<char> myVec; /\* Assume this has already 1000 spots available \*/

while(myCountertmp!=0) {

if(myVec.at(myCounter+(myCounterTmp) == enemyPlayer) {return true}

myVec.insert(currentPlayer,myCounter+(myCounterTmp);

myCounter++;

myCountertmp--;

}