Before you start:

Homework Files

You can download the starter files for coding as well as this *tex* file (you only need to modify *homework4.tex*) on canvas and do your homework with latex. Or you can scan your handwriting, convert to pdf file, and upload it to canvas before the due date. If you choose to write down your answers by hand, you can directly download the pdf file on canvas which provides more blank space for solution box.

Submission Form

A pdf file as your solution named as VE281_HW4_[Your Student ID]_[Your name].pdf uploaded to canvas
Estimated time used for this homework: **3-4 hours.**

0 Student Info

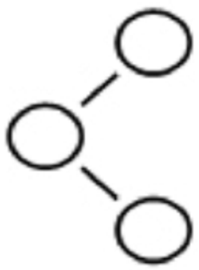
Your name and student id:

Solution:

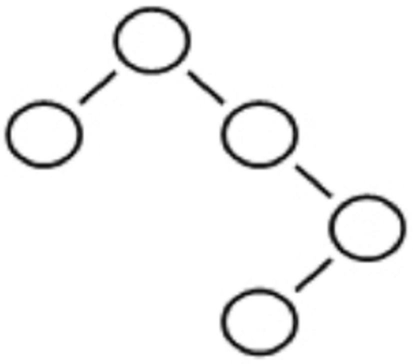
1 Basic knowledge of AVL treec

There are some questions about the basic knowledge of AVL tree.

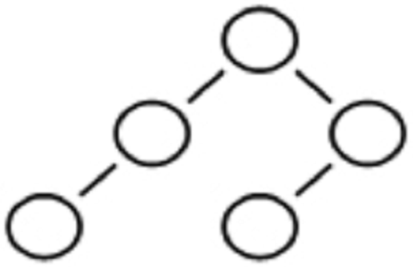
1. Which of the following binary search trees is an AVL tree?



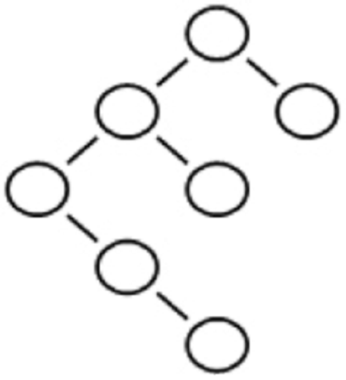
[A]



[B]



[C]



[D]

Solution:

2. Insert 1, 2, 3, 6, 5, 4 sequentially into an initially empty AVL tree, which of the following rotations will occur?

Solution:

3. Suppose an AVL tree of height N has at least A (n) nodes. Then, write A recursive formula to show how A (n) is generated.

Solution:

4. According to the above formula, we can get the following formula:

$$A(n) = \frac{(2+\sqrt{5})\left(\frac{1+\sqrt{5}}{2}\right)^n - (2-\sqrt{5})\left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}} - 1 \tag{1}$$

For large n we have the following approximation:

$$\text{n} \sim \frac{1}{\log\left(\frac{1+\sqrt{5}}{2}\right)} \log A(n) \sim 1.44 \log A(n) \tag{2}$$

Please think about how the formula (1) is obtained.

Solution:

5. Use the top two equations to answer the following questions: If the depth of the AVL tree is 6, what is the minimum number of nodes of the tree?

Solution:

What is the maximum depth of an AVL tree with 12 nodes?

Solution:

2 RB tree

2.1 Path length

Among the path from Node x to leaf node, is there one path that is greater than the double of that of another path? Prove your conclusion.

Solution:

2.2 Init

Init a tree by *insert* function. How many nodes are needed to ensure that there is a red node in the tree?

Solution:

2.3 Root node

Why the root node must be black? What will happen is root node is red?

Solution:

2.4 Insert

Draw the tree after insert 2,4,3,0,5 into the tree. There is one node (Black 1) in the tree at first.

Solution:

3 BST

3.1 New Edge in BST

Suppose we now have a tree. Try to prove that a new edge is added, there should be a cycle.

Solution:

3.2 K's smallest number

Suppose we have a binary search tree. Output the k's smallest number of the tree. Write the pseudocode or C++ code under. You can just call the root of the tree *root*
The difinition of the node and function in C++ is shown below:

```
Struct TreeNode{
   int data;
   TreeNode* left, right;
}
int kthSmallest(TreeNode* root, int k)

```

Solution:

4 Pyramid of numbers

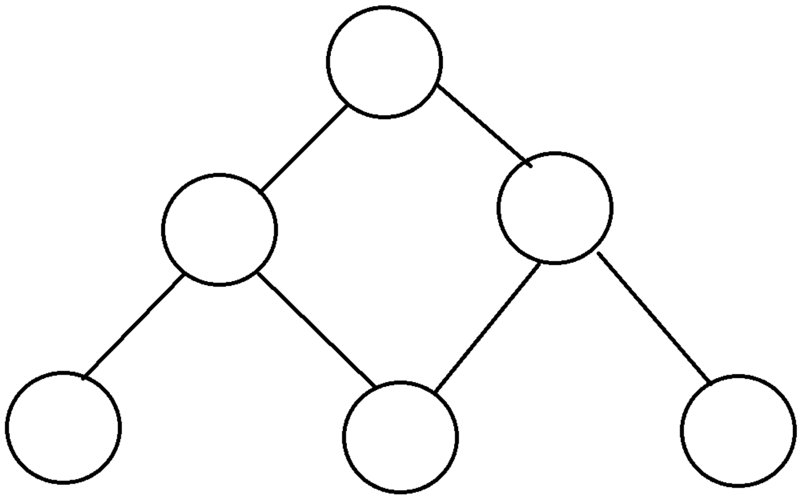
A number pyramid is an isosceles triangle of numbers with a total of n numbers in the NTH row (as shown below). The Blue Tiger needs to find a path from the vertex to the edge that maximizes the sum of numbers passed, each step leading to the lower left or lower right of the current point.

4.1 Search

Blue Tiger first considered the idea of brute force solution, starting point and end point clear, clear rules of the way, so it can be solved by searching. Please complete the code he didn't finish and write down the time complexity of the method.

```
int n, ans , A[100000][100000]
//n is the number of row, ans is the final answer, A store the number pramid.
void Dfs(int x, int y, int curr){
    if (x==n){
        if (       ) ans = curr;
        return ;
    }
    // Add code here if necessary


}
int main(){
    cin >> n
```



```
        for (int i =0; i<n; i++){
            for (int j=0; j<i; j++){
                cin >> A[i][j];
            }
        }
        ans=0;
        Dfs(0,0,A[0][0]);
        cout << ans << endl;
        return 0;
    }
```

Solution:

4.2 Whether the greedy algorithm works

Blue Tiger found the time complexity of the above method very high and decided to make some optimizations. He first came up with a greedy algorithm for optimization. Greedy algorithm is a method to obtain the global optimal solution by considering the local optimal solution. In this problem, it can be interpreted as choosing to go to the position with the larger number in the lower left or lower right at the current position. But that approach turned out to be wrong. Please find a case where the algorithm is incorrect.

Solution:

4.3 Memorized search

The Blue Tiger found that in the first question, multiple paths to each point were searched repeatedly. For example, from (0,0) to (1,2) there's a left path and a right path; So every path from (1,2) to the destination is searched twice. Therefore, the blue Tiger chose the memory search method to optimize. Please complete the code below and write the time complexity of the algorithm.

```
    int n, A[100000][100000] ,F[100000][100000]
    void Dfs(int x, int y){
        if (F[i][j]==-1){
            if(x==n) F[x][y]=A[x][y];
            else {
                // Add code here if necessary


            }
        }
        // Add code here if necessary

        return F[x][y];
    }
    int main(){
        cin >> n
        for (int i =0; i<n; i++){
            for (int j=0; j<i; j++){
                cin >> A[i][j];
                F[i][j]=-1;
            }
        }
        // Add code here if necessary

        cout <<      << endl;
        return 0;
    }
```

Solution:

4.4 Dynamic programming

Blue Tiger found that the memory search algorithm is already a dynamic programming algorithm in nature. So he tried to solve the problem using dynamic programming.

4.4.1 Boundary conditions and state transition equations

Please write down the boundary conditions and the state transition equation for this problem.

Solution:

4.4.2 Program implementation

Please help Blue Tiger complete the dynamic programming procedure.

Solution:

5 0-1 Knapsack problems

Knapsack problem is a combinatorial optimization problem: given n items, each with a weight $w_i > 0$ and a value v_i, determine the number of each item to include in a collection so that the total weight is less than or equal to a given capacity W and the total value is as large as possible.

5.1 Basic solution

For 0-1 problem, each item can be decided to be put into the knapsack once or not. Please fill in the missing code below.

```cpp
int knapsack(const vector<int> &w, const vector<int> &v){
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));
    for (int i = 1; i <= n; ++i){
        for (int j = 1; j <= W; ++j){
            // Add code here
            dp[i][j] = dp[i - 1][j];
            if (w[i - 1] <= j) {
```

```
 8                 dp[i][j] = max(dp[i][j], dp[i - 1][j - w[i]] + v[i]);
 9             }
10         }
11     }
12     return dp[n][W]// Add code here
13 }
```

What is the space-complexity?

Solution:

Can we exchange line 3 and 4?

Solution:

5.2 Can we do better?

Can we achieve a better space-complexity? If so, explain your optimizations and modify code and the improved space-complexity. Otherwise, explain in one sentence why we cannot improve space-complexity (Don't change line 3 and 4).

Solution:

5.3 Even better?

Actually we can use even more optimized space from dynamic programming (Hint: you can change the loop for your own sake). Write your code below.

Solution:

Can we exchange the two loops?

Solution:

What is the time complexity of the algorithm?

Solution: