Before you start:

Homework Files

You can download the starter files for coding as well as this *tex* file (you only need to modify *homework5.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

For homework 5, there are only one part of submission, which is a pdf file as your solution named as VE281_HW5_[Your Student ID]_[Your name].pdf uploaded to canvas.
Estimated time used for this homework: **4-5 hours.**

Great credits to 2020FA VE281 TA Group and enormous thanks to 2021SU VE281 TA Roihn!!!

0 Student Info (0 point)

Your name and student id:

Solution:

1 Dynamic Programming (60 points)

1.1 Basic Case (12 points)

Suppose that we have an $n * n$ matrix filled with integers. Starting from the top left corner, we advance to either the downward or rightward block for each step and finally reach the bottom right corner of the matrix. During this process, we will pass through nodes with different integers. By applying dynamic programming, we can find the path with the largest sum of the passed integers. Write out the recurrence relation.

Solution:

1.2 Do it twice? (24 points)

Previously, we just go across the matrix for a single time. Assume that after the first travel, we set the visited integer in the matrix to be 0 and go across the matrix, back to the left corner again. How to maximize the sum of the passed integers in the whole procedure (top left -> bottom right -> top left)? In terms of this problem, William and Coned have different ideas again.

1.2.1 Simple repeat (10 points)

Coned thinks that since this problem is quite similar to what we have solved in 1.1, just run dynamic programming twice and add them up, and we will have the correct final result. Do you agree with him? If agree, write the recurrence relation for the second dynamic programming procedure and state whether there is a difference; if not agree, come up with a counter example and explain why it doesn't work.

Solution:

1.2.2 Double the result table (14 points)

William thinks that the dynamic programming strategy for this problem should be modified from the very beginning in this case. He gives out the main function as shown below. However, after testing, he found out there are some problems within this piece of code. Please correct the code between line 17 to line 26.

```
1  int main(){
2      // two paths are considered simultaneously, one at (i, j), the other at (k, l)
3      // integers stored in integer[n][n]
4      int n;
5      cin >> n;
6      int dp[100][100][100][100] = {0};
7      int integers[100][100] = {0};
8      // read all the inputs
9      for (int i = 0; i < n; i++)
10         for (int j = 0; j < n; j++)
11             cin >> integers[i][j];
12
13     dp[0][0][0][0] = integers[0][0]; // start point
14
15     // start dp
16
17     /* modify code within this part */
18     for (int i = 0; i < n; i++)
19         for (int j = 0; j < n; j++)
20             for (int k = 0; k < n; k++)
21                 for (int l = 0; l < n; l++)
22                     dp[i][j][k][l] = max(dp[i-1][j][k-1][l], dp[i][j-1][k-1][l],
23                                          dp[i-1][j][k][l-1], dp[i][j-1][k][l-1]);
24
25
26     /* modify code within this part */
27
28     cout << dp[n][n][n][n];
29
30 }
```

1.3 How to save memory? (12 points)

Coned takes a look at William's strategy and thinks that its memory usage is too bad. Regardless of correctness, it will have a space complexity of $O(n^4)$, which sounds horrible. Propose a modification to this dynamic programming algorithm so that the space complexity can be reduced to $O(n^3)$ and write out its recurrence relation.
Hint: one way to do so is to reduce the array dp to be 3-dimensional. There should be 1 dimension still for i and 1 dimension still for k.

Solution:

1.4 Does optimization end here? (12 points)

Looking at the modification proposed by Coned, William surprisingly agrees with him. After brainstorming, they find that this strategy can be further optimized in terms of space complexity. Briefly state how you can further reduce its space complexity to $O(2n^2)$.
Hint: Do we need every *dp* value in every iteration?

Solution:

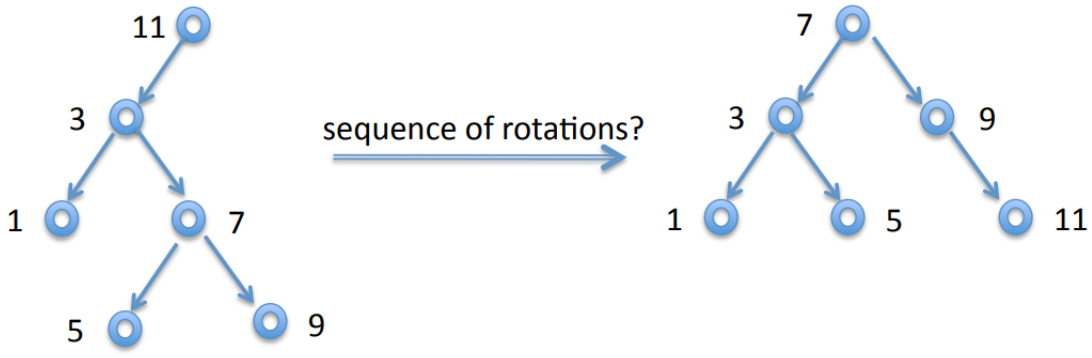
Notes: Actually we can further reduce its space usage from $O(2n^2)$ to $O(n^2)$ in this problem.

2 Self-balanced tree (40 points)

2.1 AVL Tree (20 points)

2.1.1 Rotate! (6 points)

Give the shortest sequence of single left or right rotations, and the two nodes being rotated, to transform the tree on the left to the tree on the right. Indicate in the rightmost column if the tree **after** the rotation satisfies the balance invariant of the AVL trees and also write out the balance factor of the **root** after rotation.
Notice that there might be unnecessary rotations, but make sure your sequence exactly matches the provided final result. You don't need to fill in all the rows in the table.



Step	Left or Right?	at nodes	balance factor	AVL?
1	left	3,7	2	no
2				
3				
4				
5				

2.1.2 Simulate! (8 points)

Based on the balanced tree in 2.1.1, give a value whose insertion will trigger a left-left rotation. If such a value does not exist, choose a non-leaf node to delete and again give such a value whose insertion will trigger a left-left rotation. Draw your final tree after insertion no matter whether a deletion takes place.
Assume that when deleting a non-leaf node, we always use the largest node in the left subtree to replace the deleted node.

Solution:

2.1.3 Code! (6 points)

In the lecture slides, the implementation of `Balance()` is given. In this problem, please implement `LLRotation` as well as `LRRotation` to make `Balance()` work properly. Suppose that `RRRotation` and `RLRotation` have been implemented and we follow the definitions in the slides.

```
1  void LLRotation(node *&n) {
2
3
4
5
6
7
8
9
10
11
12
13
14
15 }
16 void LRRotation(node *&n) {
17
18
19
20
21
22
23
24
25
26
27
28
29
30 }
```

2.2 Red Black Tree (20 points)

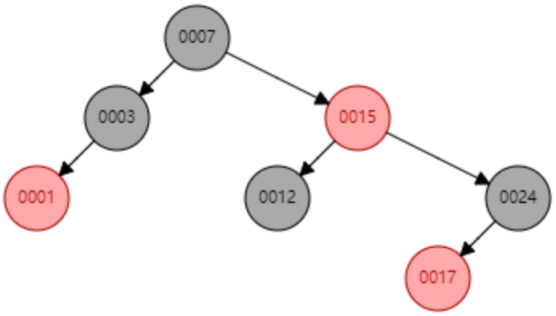
2.2.1 Balance? (4 points)

Briefly explain why red black tree is to some extent balanced.
Hint: what is the characteristic of a "balanced" tree?

Solution:

2.2.2 Complex? (8 points)

Given such a red black tree:



1) Show the result after inserting 20. Color the nodes clearly.

Solution:

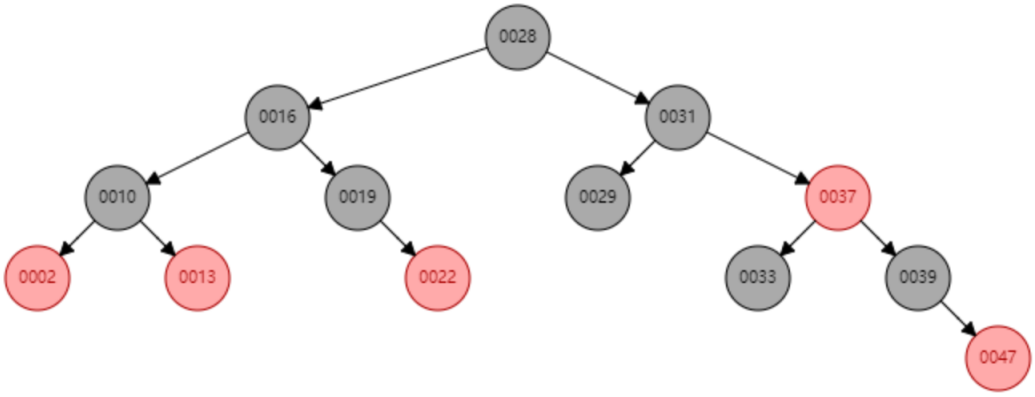
2) After 1), show the result after inserting 23. Color the nodes clearly.

Solution:

2.2.3 Correctness? (8 points)

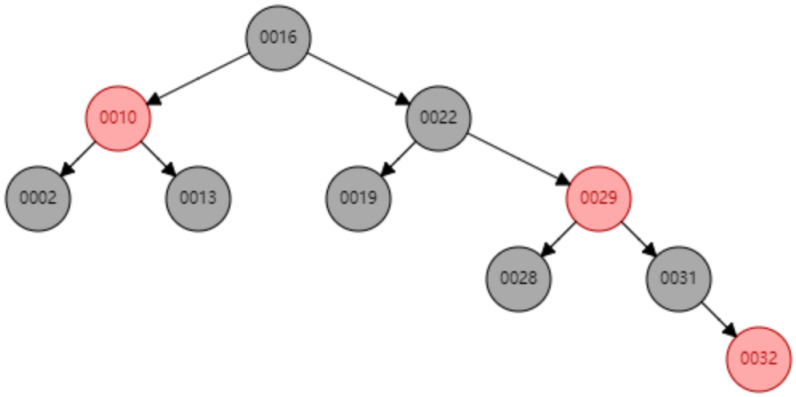
Judge whether the following red black tree is correct. If not correct, please point out which node is wrong and whether a recolor or rotation is needed (no need to draw the tree after fixing it).

a)



Solution:

b)



Solution:

Notes: In fact, although red black tree has a wide range of good application scenarios, its code implementation is quite complicated compared with other balanced tree such as AVL tree. It is also a reason why this assignment doesn't ask you to write related code in terms of red black tree.

3 Your Experience

1) Please state the thing you have most positive feeling towards/like most in this course.

2) Please state the thing you have most negative feeling towards/dislike most in this course (detailed explanation appreciated).

3) Feel free to say any comments here!

Reference

Homework 6, 15-122, Carnegie Mellon University
Exam 2, 15-122, Carnegie Mellon University