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

For homework 4, there are only one part of submission, which is 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: **4-5 hours.**

0   Student Info (0 point)

Your name and student id:

> **Solution:** Yinchen Ni 520370910026

1   Graph Search MCQs (16 points)

Choose the right answer of the following multiple choice questions, no explanation is needed.

1. Suppose that we have an undirected graph $G = (V, E)$, where

   - $V = \{a, b, c, d, e\}$
   - $E = \{(a, b), (a, e), (a, c), (b, e), (c, f), (f, d), (e, d)\}$

   If we perform a DFS from node a, which of the following could be a possible node sequence?

   A. a, b, e, c, d, f

   B. a, c, f, e, b ,d

   C. a, e, b, c, f ,d

   D. a, e, d, f, c, b

   > **Solution:** D

2. After learning about the concept of graph, you can know that tree can be represented by an acyclic directed graph. Comparing tree with such representation and other kinds of directed graph, which of the following is right?

   A. A node in **a tree** could have multiple parents

   B. A node in **a directed graph** could have multiple predecessors

   C. A node in **a tree** could **NOT** have multiple children

   D. A node in **a directed graph** could **NOT** have multiple successors

   Hint: If there exists an edge from node u to node v in a directed graph, we then call u the predecessor of v, and call v the successor of u.

   > **Solution:** B

3. The Breadth First Search (BFS) algorithm has been implemented using the queue data structure. Which one of the following is a possible order of visiting the nodes in the graph below?
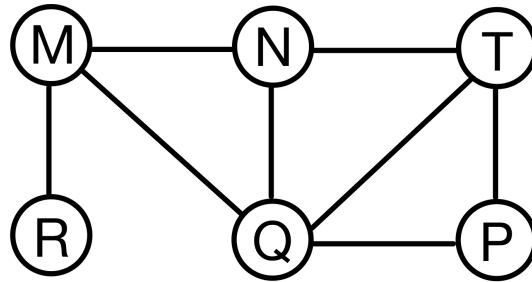
A. MNTPQR

B. NQMPTR

C. QMNRTP

D. PTQNMR

> **Solution:** D

4. Let G be a directed graph whose vertex set is the set of numbers from 1 to 100. There is an edge from a vertex i to a vertex j if and only if either $j = i + 1$ or $j = 3i$. The minimum number of edges in a path in G from vertex 1 to vertex 100 is

A. 23

B. 99

C. 4

D. 7

> **Solution:** D

2 Topological Sort (8 points)

In the lecture, we have introduced how to implement topological sorting algorithm by queue, whose main idea is quite similar to BFS. Mr. Blue Tiger argues that without **in—degree**, we can also implement topological sorting by using DFS, with an array **visited**. Complete the following pseudo code provided by him:

Algorithm 1 Algorithm to implement topological sorting with DFS

  1: **Input:** an adjacency list representing the graph
  2: **Output:** a stack with the topologically sorted nodes.
  3: Create a stack S and a boolean array $visited[]$ initialized with false.
  4: **for** each node v in the graph **do**
  5:     dfs_helper(v, $visited$, S)
  6: **end for**

  7: dfs_helper(v, $visited$, S):
  8: // add codes here **if necessary**

  9: **for** each node u adjacent to v **do**
10:     **if** (! $visited[u]$) **then**
11:         S.push(u)
12:         $visited[u] \leftarrow true$
13:         dfs_helper(u,$visited$,S)
14:     **end if**

15: **end for**
16: // add codes here **if necessary**

---

3 Minimum Spanning Tree (12 points)

3.1 Delete an edge (6 points)

Given a minimum spanning tree of a connected undirected graph G, an edge is deleted from G now. Suppose after deletion, G is still connected. Describe how to find the new minimum spanning tree with the old one in O(E) time.

> **Solution:** If the original spanning tree does not contain the deleted edge, we do nothing. Otherwise, for the vertex that is isolated, we search through all the edges that it connects and pick the one with minimum weight.

3.2 Add an edge (6 points)

Given a minimum spanning tree of a connected undirected graph G, an edge is added to G now. Describe how to find the new minimum spanning tree with the old one in O(V) time.

> **Solution:** Say the added edge connects u and v. Search through the unique path from u to v (one can use dfs to find such a path within $O(V)$ time since there are only $|V| - 1$ edges), find the edge with maximum weight. If it is larger than the weight of the newly added edge, then replace it.

4 Shortest Path (4 points)

As introduced in the lecture, we can use Dijkstra's algorithm when the graph only has non-negative edges. Give a simple example of a directed graph with negative-weight edges for which Dijkstra's algorithm produces incorrect answers. Then briefly explain why Dijkstra's algorithm fails on your example.

Solution:
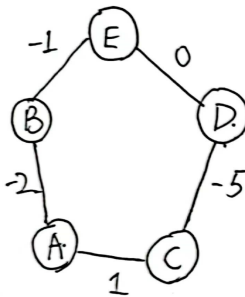
Using Dijkstra's Algorithm to calculate distance: $A \rightarrow E$

first. B is picked, distance -2

Then, E is picked, distance -3

In later searching procedure, the distance to E will not be updated

But actually a shorter path, $A \rightarrow C \rightarrow D \rightarrow E$, weigh -4

It seems that Dijkstra's Algorithm would fail
when there are cycles with negative value. edges.

5 Dynamic Programming (60 points)

5.1 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: Suppose $a_{i,j}$ stands for the integer in i-th row, j-th column; $f_{i,j}$ stands for the largest sum from the left corner to the block, we have

$$f_{i,j} = \max\{f_{i-1,j} + f_{i,j-1}\} + a_{i,j}.$$

To find out the path, let $p_{i,j}$ stores the coordinate of previous block in the path , then

$$p_{i,j} = \left\{ \begin{array}{ll} p_{i-1,j} & , f_{i-1,j} > f_{i,j-1} \\ p_{i,j-1} & , f_{i-1,j} \leq f_{i,j-1} \end{array} \right.$$

5.2   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, Alice and Bob have different ideas again.

5.2.1   Simple repeat (10 points)

Bob thinks that since this problem is quite similar to what we have solved in Section 5.1, 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: No. Here is one counter example:
First round:
$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Second round:
$$\begin{pmatrix} 0 & 2 & 3 \\ 0 & 0 & 6 \\ 7 & 0 & 0 \end{pmatrix}$$

However, the largest path would be (top left -> bottom right, bottom right -> top left):

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} + \begin{pmatrix} 0 & 2 & 3 \\ 0 & 5 & 6 \\ 0 & 0 & 0 \end{pmatrix}$$

5.2.2   Double the result table (14 points)

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

```cpp
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 = 1; i <= n; i++)
10         for (int j = 1; j <= n; j++)
11             cin >> integers[i][j];
12
13     dp[1][1][1][1] = integers[1][1]; // start point
14
15     // start dp
16
17     /* modify code within this part */
18
19     for (int i = 1; i <= n; i++)
20         for (int j = 1; j <= n; j++)
21             for (int k = 1; k <= n; k++)
22                 for (int l = 1; l <= n; l++){
23                     dp[i][j][k][l] = max(dp[i-1][j][k-1][l], dp[i][j-1][k-1][l],
24                                          dp[i-1][j][k][l-1], dp[i][j-1][k][l-1])
25                                      + integers[i][j] + integers[k][l];
26                     if(i==k&&j==l) dp[i][j][k][l]-=integers[i][j];
27                 }
28
29     /* modify code within this part */
30
31     cout << dp[n][n][n][n];
32
33 }
```

5.3 How to save memory? (12 points)

Bob takes a look at Alice'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: Let $dp[l][i][k]$ represents the total sum that the length of current path is l, with the first path reaches $(i, l - i)$, the second path reaches $(k, l - k)$.

Now, we enumerate l from 1 to $2n$, i from 1 to n, k from 1 to n, with the dp equation
`dp[l][i][k] = max(dp[l-1][i-1][k-1], dp[l-1][i][k-1], dp[l-1][i-1][k], dp[l-1][i][k])`
`+ integers[i][l-i] + integers[k][l-k];` Of course the situation that (i==k) needs
to be considered like above.

5.4   Does optimization end here? (12 points)

Looking at the modification proposed by Bob, Alice 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: Since in every iteration we only need the value of $dp[l-1][][]$, we can reduce the array to be $2 \times n \times n$. Namely $dp[1][i][k]$ represents all the states that the length are odd and $dp[2][i][k]$ represents all the states that the length are even. Now the dp equation can be rewritten into:

```
dp[l%2][i][k] = max(dp[(l-1)%2][i-1][k-1], dp[(l-1)%2][i][k-1], dp[(l-1)%2][i-1][k],
dp[(l-1)%2][i][k]) + integers[i][l-i] + integers[k][l-k];
```

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

Reference

Assignment 4, VE281, FA2021, UMJI-SJTU
Assignment 5, VE281, FA2021, UMJI-SJTU
Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne, Princeton University