A. Requirements

Code (100%)

You can write your code in Java, Python, C, or C++. The *time limit* may vary among different languages, depending on the performance of the language. Your code must be a complete excutable program instead of only a function. We guarantee test data strictly compliance with the requirements in the description, and you do not need to deal with cases where the input data is invalid.

No AI Assistance or Plagiarism: All code must be your own. The use of AI tools (e.g., ChatGPT, GitHub Copilot) or copying from external sources or peers is **strictly forbidden**.

Violations of the plagiarism rules will result in 0 points or even **failure** of this course.

Libraries in this assignment:

- For C/C++, you can only include standard library.
- For Java, you can only `import java.util.*`
- For Python, you can only import standard library. In other words, you cannot import libraries such as `numpy`.

We provide an example problem to illustrate the information above better.

B. Example Problem: A + B Problem

Description

Given 2 integers A and B, compute and print A + B

Input

Two integers in one line: A, and B

Output

One integer: A + B

Sample Input 1

```
1 2
```

Sample Output 1

```
3
```

Problem Scale & Subtasks

For 100% of the test cases, $0 \leq A, B \leq 10^6$

Solutions

Java

```java
import java.util.*;

public class Example {
    public static void main(String[] args) {
        int a, b;
        Scanner scanner = new Scanner(System.in);
        a = scanner.nextInt();
        b = scanner.nextInt();
        scanner.close();
```

```
        System.out.println(a + b);
    }
}
```

Python

```
AB = input().split()
A, B = int(AB[0]), int(AB[1])
print(A + B)
```

C

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int A, B;
  scanf("%d%d", &A, &B);
  printf("%d\n", A + B);
  return 0;
}
```

C++

```
#include <iostream>

int main(int argc, char *argv[])
{
  int A, B;
  std::cin>> A >> B;
  std::cout<< A + B << std::endl;
  return 0;
}
```

C. Submission

After finishing this assignment, you are required to submit your code to the Online Judge System (OJ), and upload your `.zip` package of your code files to BlackBoard.

C.1 Online Judge

Once you have completed one problem, you can submit your code on the page on the Online Judge platform (`oj.cuhk.edu.cn`, campus only) to gain marks for the code part. You can submit your solution of one problem for **no more than 80 times**.

After you have submitted your program, OJ will test your program on all test cases and give you a grade. The grade of your latest submission will be regarded as the final grade of the corresponding problem. Each problem is tested on multiple test cases of different difficulty. You will get a part of the score even if your algorithm is not the best.

Note: The program running time may vary on different machines. Please refer to the result of the online judge system. OJ will show the time and memory limits for different languages on the corresponding problem page.

If you have other questions about the online judge system, please refer to OJ wiki (campus network only). If this cannot help you, feel free to contact us.

C.2 BlackBoard

You are required to upload your **source codes** to the BlackBoard platform. You need to name your files according to the following rules and compress them into `A4_<Student ID>.zip` :

```
A4_<Student ID>.zip
|-- A4_P1_<Student ID>.java/py/c/cpp
|-- A4_P2_<Student ID>.java/py/c/cpp
|-- A4_P3_<Student ID>.java/py/c/cpp
```

For Java users, **you don't need to consider the consistency of class name and file name**.

For example, suppose your ID is 123456789, and your problem 1 and 2 is written in `Python`, problem 3 is written in `Java` then the following contents should be included in your submitted `A4_123456789.zip`:

```
A4_123456789.zip
|-- A4_P1_123456789.py
|-- A4_P3_123456789.py
|-- A4_P3_123456789.java
```

C.3 Late Submissions

Submissions after May.6 2025 23:59:00(UTC+8) would be considered as LATE.

The LATE submission page will open after deadline on OJ.

Submisson time = max{latest submisson time for every problem, BlackBoard submisson time}

There will be penalties for late submission:

- 0–24 hours after deadline: final score = your score×0.8
- 24–72 hours after deadline: final score = your score×0.5
- 72+ hours after deadline: final score = your score×0

FAQs

Q: My program passes samples on my computer, but not get AC on OJ.

A: Refer to OJ Wiki Q&A

Authors

If you have questions for the problems below, please contact:

- Prof. LI, Wenye: wyli@cuhk.edu.cn

CSC3100 Data Structures Spring 2025

Programming Assignment 4

Prof. LI, Wenye: wyli@cuhk.edu.cn

Due: May.6 2025 23:59:00

Assignment Link: https://oj.cuhk.edu.cn/d/csc3100_2025_spring/homework/6809eee89e91f80b1d303f91

Announcement

- There are three programs in the assignment. Students may choose to finish two of them.
- The maximum score of this assignment is limited to 100 marks, even if the marks he/she earned has exceeded 100.

Problem 1 – Binary Search Tree (50 marks)

Write a program that reads a sequence of distinct input integers, then builds a binary search tree, and outputs the pre-order, in-order and post-order traversals of the binary search tree.

Consider the following input:

Input Format

A sequence of distinct input integers, representing the value of each node.

Output Format

- There are 3 output lines in total.
- Line 1 is the in-order traversal result, with numbers separated by a space.
- Line 2 is the pre-order traversal result, with numbers separated by a space.
- Line 3 is the post-order traversal result, with numbers separated by a space.

Sample Input 1

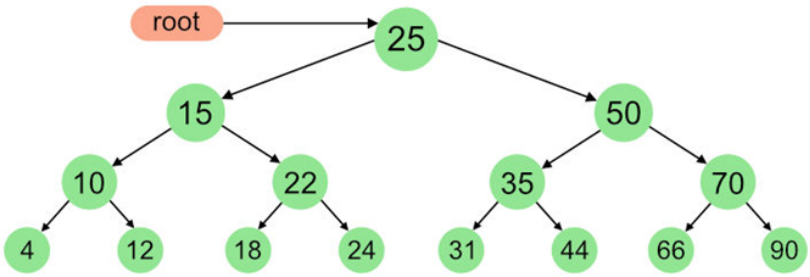
```
25 15 50 35 44 70 31 66 90 10 4 12 22 18 24
```

Based on the order of the input numbers, the corresponding binary search tree and the traversal orders are shown as follows.

InOrder(root) visits nodes in the following order:
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

A Post-order traversal visits nodes in the following order:
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



Therefore, your program is expected to output the following:

Sample Output 1

```
4 10 12 15 18 22 24 25 31 35 44 50 66 70 90
25 15 10 4 12 22 18 24 50 35 31 44 70 66 90
4 12 10 18 24 22 15 31 44 35 66 90 70 50 25
```

Problem Scale & Subtasks

- In the test, the number of input integers is between 2 and 1000.
- The total number of nodes is less than 1000.
- The input sequence may produce a binary search tree of any structure, which may not be complete or perfect as in this example.

Program 2 — Huffman Tree (50 marks)

Description

This assignment implements the Huffman tree, which is a powerful algorithm widely in data compression.
Write a code which reads one line of string input (encoded with ASCII code) from console and then compresses the input using Huffman tree method. The program will output the number of bits in the compressed result as a string of "0"s and "1"s.

Input Format

The input string consists of ASCII letters with possible values from 32 to 126. (ASCII letters are case-sensitive. That is, "A" and "a" are different.)

Sample Input 1

```
SUSIE SAYS IT IS EASY,
```

After constructing a Huffman tree, one possible encoding dictionary will be:

```
32:00
44:01110
65:010
69:1111
73:110
83:10
84:0110
85:01111
89:1110
```

The dictionary shows that the space character will encoded as "00" (note that 32 is the ASCII code of the space character. 44 is the ASCII code of the comma character. . .)

Therefore, the corresponding compression result is:

```
10011111011011110010010111010001100110001101000111101010111001110
```

And the desired output will be the number of bits in the compression result, i.e., the total number of "0"s and "1"s.

Sample Output 1

```
65
```

Problem Scale & Subtasks

- The maximum length of the input string is one million.

Program 3 — Puzzle-8 Problem (60 marks)

Description

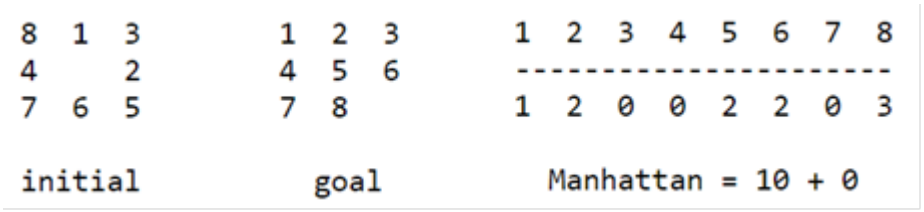
The 8-puzzle problem is a puzzle invented in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. Your goal is to rearrange the blocks so that they are in order. You are permitted to slide blocks horizontally or vertically into the blank square. The following shows a sequence of legal moves from an initial board position to the goal position.

```
   1  3       1     3       1  2  3       1  2  3       1  2  3
4  2  5   =>  4  2  5   =>  4     5   =>  4  5     =>  4  5  6
7  8  6       7  8  6       7  8  6       7  8  6       7  8

initial                                                 goal
```

An algorithmic solution to the problem is based on the classical A* search algorithm. We define a state of the game to be the board position, the number of moves made to reach the board position, and the previous state. First, insert the initial state (the initial board, 0 moves, and a null previous state) into a priority queue. Then, delete from the priority queue the state with the minimum priority, and insert onto the priority queue all neighboring states (those that can be reached in one move). Repeat this procedure until the state dequeued is the goal state. The success of this approach hinges on the choice of priority function for a state. We consider the following priority function:

- **Manhattan priority function.** The sum of the distances (sum of the vertical and horizontal distance) from the blocks to their goal positions, plus the number of moves made so far to get to the state.



There is a key observation here: to solve the puzzle from a given state on the priority queue, the total number of moves we need to make (including those already made) is at least its priority, using the Manhattan priority function. (This is true because each block must move its Manhattan distance from its goal position. Note that we do not count the blank tile when computing the Manhattan priorities.) Consequently, as soon as we dequeue a state, we have not only discovered a sequence of moves from the initial board to the board associated with the state, but one that makes the fewest number of moves.

Write a program that reads the initial board from an input file (file name specified by the user) and outputs a file (file name specified by the user) with a sequence of board positions that solves the puzzle in the fewest number of moves.

Input Format

The input file will consist of the 3-by-3 initial board position. The input format uses "0" to represent the blank square.

Output Format

Several lines,

- Line 1 is the fewest number of moves to solve the puzzle, followed by an empty line.
- Line 3-6 is the initial board as given in the input, followed by an empty line.
- Line 7-9 is the board after the first move, followed by an empty line.
- ...
- Line xx-yy is the objective board of the puzzle that has been solved.

Sample Input 1

```
0 1 3
4 2 5
7 8 6
```

Sample Output 1

```
4

0 1 3
4 2 5
7 8 6

1 0 3
4 2 5
7 8 6
```

```
1 2 3
4 0 5
7 8 6

1 2 3
4 5 0
7 8 6

1 2 3
4 5 6
7 8 0
```