Data Structures and Algorithms, Spring 2025
Homework 0

TA E-mail: dsa_ta@csie.ntu.edu.tw

Due: 13:00:00, Tuesday, April 15, 2025

Rules and Instructions

Welcome to DSA 2025 from your teaching team with 23 TAs and 2 instructors. Please start the exciting journey by carefully reading the following rules for this homework set. Many rules will also be applied to future homework sets.

- Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.

- Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources (including chatGPT) can be consulted, but not copied from.

- Since everyone needs to write the final solutions alone, there is **absolutely no need to lend your homework solutions and/or source codes to your classmates at any time.** In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

- In Homework 0, the problem set contains 8 problems. All of them are programming problems modified from Judgegirl, which is a judge system designed for Introduction to Computer Programming course in NTU. Special thanks to Prof. Pangfeng Liu for granting us permission to use these problems, and for Prof. Liu's teaching teams in the past for designing these problems.

- For problems in the programming part, you should write your code in C programming language, and then submit the code via the *DSA Judge*:

  https://dsa-2025.csie.org/.

  Each day, you can submit up to 10 times for each problem. If not mentioned in the problem, the judge system will compile your code with

  ```
  gcc [id].c -static -O2 -std=c11
  ```

- For debugging in the programming part, we strongly suggest you produce your own test-cases with programs and/or use tools like `gdb` or compile with flag `-fsanitize=address` to help you solve issues like illegal memory usage. For `gdb`, you are strongly encouraged to use compile flag `-g` to preserve symbols after compiling.

  Note: For students who use IDE (VScode, DevC++...) you can modify the compile command in the settings/preference section. Below are some examples:

  - `gcc [id].c -O2 -std=c11 -fsanitize=address # Works on Unix-like OS`
  - `gcc [id].c -O2 -std=c11 -g # use it with gdb`

- The score of the part that is submitted after the deadline will get some penalties according to the following rule:

$$\textit{Late Score} = \max\left(\left(\frac{5 \times 86400 - \textit{Delay Time}(\textit{sec.})}{5 \times 86400}\right) \times \textit{Original Score}, 0\right)$$

  Please note that the "gold medal" of the class cannot be used on Homework 0 problems.

- The purpose of Homework 0 is to communicate our *expected background* for this class. If you cannot solve Homework 0 within a reasonable amount of time *by yourself*, it is very likely that you cannot solve those more challenging homework sets in the future. Then, you are *strongly encouraged* to not take this class (though you are always welcome to audit the class to learn at your own pace).

- Following the purpose above, the TAs will *not* hint to you how to solve the problems in Homework 0 via emails or during their TA hours (unlike what they will do for other homework sets in the future). They will only clarify any ambiguity in the problem descriptions. If you need discussions on any issues about Homework 0, please go to the Discord channel and discuss the issues with your classmates.

- If you really need email clarifications on the problem descriptions, you are encouraged to include the following tag in the subject of your email: `"[HW0.Px]"`, specifying the problem where you have questions. For example, `"[HW0.P1] Can` *id* `be negative?"`. Adding these tags allows the TAs to track the status of each email and to provide faster responses to you.

Problem 1 - Company (10 pts)

Problem Description

Write a program to determine the relation between two employees in a company with a tree data structure.

- If an employee A can follow the "boss" relation to another employee B, then A is a subordinate of B, and B is a supervisor of A.
- If A and B have a common supervisor C, then A and B are colleague.
- If none of the above is true, then A and B are unrelated.

Now we have a pointer pointing to the boss of an employee, as suggested by the following definition. Note that two different employees may have the same boss, but one employee will only have one boss. The total number of employees is no more than 32. If an employee has no boss, then his/her boss pointer will point to himself/herself.

1.h

```c
#ifndef EMPLOYEE_H
#define EMPLOYEE_H

typedef struct employee {
  char first_name[32];
  char last_name[32];
  struct employee *boss;
} Employee;

int relation(Employee *employee1, Employee *employee2);
#endif
```

Now implement the following function that returns the relation of employee1 to employee2. We need the following definition.

- If their relationship is subordinate, then return 1.
- If their relationship is supervisor, then return 2.
- If their relationship is colleague, then return 3.
- If their relationship is unrelated, then return 4.

Note that you should only upload 1.c and do not output anything to stdout. Otherwise, you may get a `Wrong Answer` or a `Compile Error`. We will use the following command to compile your code:

```
gcc 1.c main.c -static -O2 -std=c11
```

1.c

```c
#include "1.h"

int relation(Employee *employee1, Employee *employee2) {
    //
}
```

main.c

```c
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include "1.h"

typedef struct me {
    int id;
    char first_name[32];
    char last_name[32];
    int boss_id;
} employee;

void readName(employee *e) {
    scanf("%s %s", e->first_name, e->last_name);
}

int nameToIndex(employee *e, employee A[], int n) {
    for (int i = 0; i < n; i++) {
        if (!strcmp(A[i].first_name, e->first_name) &&
            !strcmp(A[i].last_name, e->last_name))
            return i;
    }
    return -1;
}
int main() {
    int n, m;
    employee A[32];
    Employee B[32];
    const char out[4][32] = {"subordinate", "supervisor",
                             "colleague", "unrelated"};
    while (scanf("%d", &n) == 1) {
        for (int i = 0; i < n; i++) {
            scanf("%d", &A[i].id);
            readName(&A[i]);
            scanf("%d", &A[i].boss_id);
        }

        for (int i = 0; i < n; i++) {
            strcpy(B[i].first_name, A[i].first_name);
            strcpy(B[i].last_name, A[i].last_name);
            B[i].boss = NULL;
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (A[i].boss_id == A[j].id)
                    B[i].boss = &B[j];
            }
        }
        scanf("%d", &m);
        employee x, y;
        for (int i = 0; i < m; i++) {
            readName(&x);
            readName(&y);
            int ix = nameToIndex(&x, A, n), iy = nameToIndex(&y, A, n);
            assert(ix != -1);
            assert(iy != -1);
            printf("%d\n", relation(&B[ix], &B[iy]));
        }
    }
    return 0;
}
```

Input

The first line of the input is the number of employees N. The next N lines are the information of the employees. The next line is the number of queries M. Each of the next M lines contains the names of two employees. These two names will be different. Note that you must process a line once you read it.

Output

The output has M lines. Each line has a number which indicates the relation between the two employees.

Constraints

- $2 \leq N \leq 32$
- $1 \leq M \leq 1000$
- All the ids can be saved in `int` without overflowing.
- The length of any name is no more than 31.

Sample Testcases

Sample Input 1

```
6
100 John Smith 200
200 Adam Joshson 300
300 Jane Washington 300
400 Mary Miller 300
500 Eric Page 500
600 James Clark 500
4
John Smith Jane Washington
Jane Washington Adam Joshson
Adam Joshson Mary Miller
Mary Miller James Clark
```

Sample Output 1

```
1
2
3
4
```

Source

This problem is modified from Judge Girl Problem 251.

Problem 2 - Bookshelf (10 pts)

Problem Description

Write a program to simulate a bookshelf. Suppose you have 255 books and a shelf that can hold 8 books. Each of the 255 books is numbered from 1 to 255, and initially, all books are in the bookcase. The 8-book shelf is on the desk and starts empty.

When we want to read a book, we first check if it is already on the shelf. If it is found, we take it out to read and then place it back at the rightmost position of the shelf. If the book is not on the shelf, we retrieve it from the bookcase, read it, and then place it at the rightmost position of the shelf. However, if the shelf is already full (holding 8 books), we remove the leftmost book from the shelf and return it to the bookcase. Then, we shift the remaining books to the left to make room for the newly read book at the rightmost position.

Since the book numbers range up to 255, we can represent each book with 8 bits. Additionally, since a `unsigned long long int` has 8 bytes (64 bits), we can use it to simulate the 8-book shelf.

For convenience, we define the least significant byte of the `unsigned long long int` as the rightmost position on the shelf and the most significant byte as the leftmost position.

For example, if we first read book #1 and then book #2, book #1 will be to the left of book #2. At this point, the variable representing the shelf, when expressed in binary, will be:

```
00000000 00000000 00000000 00000000 00000000 00000000 00000001 00000010
```

Simulate the state of the shelf after a period of time.

Input

The first line contains a number N, which indicates the number of query. Each of the next N lines is given in one of the following formats:

- `1 id` We read a book numbered `id`, and put it onto the bookshelf.
- `2 format` We output what the bookshelf is like at this moment.

Output

For any type 2 operation:

- If the `format` is `1`, print one line containing an **unsigned** decimal number representing the current state of the bookshelf. For example, if the binary representation of the bookshelf is `00000000 00000000 00000000 00000000 00000000 00000000 00000001 00000010`, the output should be `258`.
- If the `format` is `2`, output 8 integers representing the book numbers currently on the shelf from left to right. If a position on the shelf is empty, print `0` for that position.

Constraints

- $1 \le N \le 10^6$
- $1 \le$ `id` ≤ 255
- All the operation will follow the illustrations above.

Sample Testcases

Sample Input 1

```
13
1 130
2 2
1 145
1 65
1 177
1 4
1 200
1 73
1 54
1 177
2 1
1 78
2 2
```

Sample Output 1

```
0 0 0 0 0 0 0 130
9408372585349592753
145 65 4 200 73 54 177 78
```

Source

This problem is modified from Judge Girl Problem 222

Problem 3 - Food Ingredients in Common (10 pts)

Problem Description

In this problem, you are given a list of foods, each with its set of ingredients. Your task is to determine the common ingredients between two different foods.

You will first be given a number of foods, followed by their associated ingredients. Then, you will be asked a series of queries, each asking for the common ingredients between two specific foods. Your goal is to output the common ingredients in dictionary order for each query. If there are no common ingredients, output `nothing`.

Input

- The first line contains an integer n, representing the number of foods.
- The next n lines describe each food. Each line begins with the food's name, followed by an integer i (the number of ingredients), and then the list of i ingredients.
- The next line contains an integer q, representing the number of queries.
- The following q lines each contain two food names, representing a query.

Output

- For each query, output the common ingredients in dictionary order, separated by a space.
- If there are no common ingredients, output `nothing`.

Constraints

- $1 \leq n \leq 100$
- $1 \leq i \leq 10$
- $1 \leq q \leq 10000$
- No repeated ingredients for any single food.
- The name of any ingredient will not be ”nothing”.
- Food and ingredient names consist only of lowercase English letters and have a maximum length of 64 characters.

Sample Testcases

Sample Input 1

```
3
cake 4 butter egg flour sugar
omelet 4 bacon butter egg ham
bread 1 flour
2
cake omelet
omelet bread
```

Sample Output 1

```
butter egg
nothing
```

Source

This problem is modified from Judge Girl Problem 98

Problem 4 - Play with Words (10 pts)

Problem Description

Write a program to play with words. Your program should support the following commands.

- `insert left x`: Insert character `x` at the beginning of a word.
- `insert right x`: Insert character `x` at the end of a word.
- `insert k x`: Insert character `x` as the `k`-th character of this word.
- `delete left`: Delete a character at the beginning of a word.
- `delete right`: Delete a character at the end of a word.
- `delete k`: Delete `k`-th character from the word.

Where `x` is a character other than spaces, and `k` is a number starting from 1. Initially there is nothing in this word, and after the following command the word should be `bbaac`.

Order	Command	Resulting word
0	`insert left a`	`a`
1	`insert left a`	`aa`
2	`insert left b`	`baa`
3	`insert right a`	`baaa`
4	`insert right c`	`baaac`
5	`insert left b`	`bbaaac`
6	`delete right`	`bbaaa`
7	`insert 4 c`	`bbacaa`
8	`delete 5`	`bbaca`
9	`delete 4`	`bbaa`
10	`insert 5 c`	`bbaac`

Input

The input data contains a sequence of commands described above.

Output

After processing the input commands, Your program should find out all of the longest consecutive sequence with the same character from left to right and output the character of each sequence in order, then output the length of the sequences at the end. All data should be separated by a single space.

Constraints

- $0 \leq$ Number of commands ≤ 5000
- `x` $\in$ {`a, b, ..., z`}
- All commands are valid. For example, if your program receives a `delete 5` command, we ensure that the word would have at least 5 characters for now.

Sample Testcases

Sample Input 1

```
insert left a
insert left a
insert left b
insert right a
insert right c
insert left b
delete right
insert 4 c
delete 5
delete 4
insert 5 c
```

Sample Output 1

```
b a 2
```

Sample Input 2

```
insert 1 v
delete left
insert left v
delete right
insert right v
delete 1
insert left g
insert right o
delete right
delete 1
insert 1 o
insert right d
delete 2
```

Sample Output 2

```
o 1
```

Source

This problem is modified from Judge Girl Problem 46

Problem 5 - Friend Cover (10 pts)

Problem Description

We have a set of n people indexed from 0 to $n-1$. Two people can be friends. After knowing their friendship, select a minimum subset of people that if A and B are friends, at least one of them will be in the subset. If multiple minimum subsets exist, select the one with the least person indices in lexicographical order[1]. For example, you should report "1 3 5" instead of "1 15 6", or "1 5 3".

Input

The input contains only one test case. The first line contains the number of people, n, and an integer m. Each of the following m lines contains two people, u_i and v_i, who are friends.

Output

A sequence of indices in lexicographical order and separated with new lines.

Constraints

- $0 \leq n \leq 20$
- $0 \leq m \leq \frac{n(n-1)}{2}$
- $\forall i : u_i \neq v_i$
- $\forall i : 0 \leq u_i, v_i \leq n - 1$

Sample Testcases

Sample Input 1

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

Sample Output 1

```
0
1
3
```

[1] https://en.wikipedia.org/wiki/Lexicographic_order

Hints

You don’t need to do any cut during the recursion.

Source

This problem is modified from Judge Girl Problem 50243

Problem 6 - A Zigzag Array (10 pts)

Problem Description

Prepare a two dimensional zig-zag array using a free buffer and a pointer array. Note that the numbers of columns in rows of a zig-zag array may be different. The ingredients of a zig-zag array are as follow.

- An integer `row` that indicates the number of rows of this two dimensional array.
- An integer array `column[]` that specifies the number of columns in that row.
- An integer buffer, `int buffer[10000];` which will be sufficient to hold all elements.
- An array of integer pointers. `int *array[];` This array has the number of row elements.

Now you need to implement the following function which is defined in the header file `6.h`, so that after calling `prepare_array`, we can use `array[i][j]` to access the elements in this prepared two dimensional zig-zag array.

6.h

```c
#ifndef PREPARE_H
#define PREPARE_H

void prepare_array(int buffer[], int *array[], int row, int column[]);
#endif
```

You should only upload 6.c and do not output anything to stdout. Otherwise, you may get `Wrong Answer` or `Compile Error`. Besides, **you must use buffer to set** *array*, functions like *malloc*, *calloc*, or *realloc* are not allowed.

6.c

```c
#include "6.h"

void prepare_array(int buffer[], int *array[], int row, int column[])
{
  // Implement here
}
```

We will use the following command to compile your code:

```
gcc 6.c main.c -static -O2 -std=c11
```

main.c

```c
#include <stdio.h>
#include "6.h"

int main()
{
  int row;
  int column[50];
  int *array[50];
  int buffer[10000];
  scanf("%d", &row);
  for (int i = 0; i < row; i++)
  {
    scanf("%d", &column[i]);
  }
  prepare_array(buffer, array, row, column);
  for (int i = 0; i < row; i++)
  {
    for (int j = 0; j < column[i]; j++)
    {
      scanf("%d", &array[i][j]);
    }
  }
  for (int i = 0; i < row; i++)
  {
    for (int j = 0; j < column[i]; j++)
    {
      printf("%d ", array[i][j]);
    }
  }
  return 0;
}
```

Constraints

- $1 \leq \texttt{row} \leq 50$
- $1 \leq \texttt{column[i]} \leq 10000$
- $1 \leq \sum \texttt{column[i]} \leq 10000$
- $1 \leq \texttt{array[i][j]} \leq 10000$

Hints

If we properly set `array[i]` to point to a cell in `buffer`, `array[i][j]` will be the `j`-th cell from this location.

Source

This problem is modified from Judge Girl Problem 129

Problem 7 - Mine Field (10 pts)

Problem Description

Write a program to find the mines. There is a nine by nine minefield, in which each cell could have a mine. Now you have a grid N, N_{ij} represents the number of mines of all (up to eight) neighboring cells and the cell itself at row i, column j. Now you need to determine the locations of all mines. For example, if you know the grid N as follows.

```
1 1 2 1 1 1 1 1 0
2 2 3 2 2 3 2 2 0
2 2 3 2 2 3 3 3 1
2 3 2 2 1 2 3 3 2
2 3 1 2 2 3 5 4 3
4 6 3 3 3 4 5 4 3
5 7 4 3 3 4 4 4 3
6 9 6 4 2 2 1 3 3
4 6 4 3 1 1 0 2 2
```

Then, the locations of mines can be represented as a 9×9 grid M as follows. if there is a mine at the cell located at row i, column j, $M_{ij} = 1$; otherwise, $M_{ij} = 0$.

```
0 0 0 0 0 0 0 0 0
0 1 0 1 0 0 1 0 0
0 1 0 0 1 0 1 0 0
0 0 0 0 0 0 0 1 0
1 0 1 0 0 0 0 1 0
1 0 0 0 1 1 1 1 0
1 1 1 0 0 1 0 0 1
1 1 1 0 0 0 0 0 1
1 1 1 0 1 0 0 0 1
```

Note that we do not guarantee that there are solutions. If there is no solution, output `"no solution\n"`. If there are multiple solutions, output the one that has the minimum value, i.e., if you consider the solution as an 81-bit unsigned integer, where bits are from left to right, from top to bottom. Take the minefield above for example, the 81-bit unsigned integer would be `0000000000101...0001`.

Input

There is a 9×9 grid N, where the i-th row contains 9 space-separated integers N_{ij}.

Output

Output grid M, the locations of mines, where the i-th row contains 9 space-separated integers M_{ij}. If there is no solution, output `"no solution\n"`.

Constraints

- $N_{ij} \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$,
- $M_{ij} \in \{0, 1\}$

Sample Testcases

Sample Input 1

```
1 1 2 1 1 1 1 1 0
2 2 3 2 2 3 2 2 0
2 2 3 2 2 3 3 3 1
2 3 2 2 1 2 3 3 2
2 3 1 2 2 3 5 4 3
4 6 3 3 3 4 5 4 3
5 7 4 3 3 4 4 4 3
6 9 6 4 2 2 1 3 3
4 6 4 3 1 1 0 2 2
```

Sample Output 1

```
0 0 0 0 0 0 0 0 0
0 1 0 1 0 0 1 0 0
0 1 0 0 1 0 1 0 0
0 0 0 0 0 0 0 1 0
1 0 1 0 0 0 0 1 0
1 0 0 0 1 1 1 1 0
1 1 1 0 0 1 0 0 1
1 1 1 0 0 0 0 0 1
1 1 1 0 1 0 0 0 1
```

Source

This problem is modified from Judge Girl Problem 248

Problem 8 - Bingo (10 pts)

Problem Description

Write a program to play bingo. A bingo board has m rows and m columns. Each entry has a different number from 1 to $m \times m$. Each number will be called with some sequence. If a player has all the numbers in a row, a column, or a diagonal called he can declare bingo and win the game. Now given the bingo boards of n players, determine who wins the bingo.

Input

The first line of the input has n and m. The second line has the name of the first player. Each of the next m lines has the m numbers of the bingo board of the first player. The next line has the name of the second player, and the next m line have the numbers for the second player, and so on. The next line has $m \times m$ numbers that are called in sequence during the bingo game.

Output

Print the indices of the winners of the game and by which number they won the game. If there are more than one winner, print the player indices one by one according to the order they appear in input.

Constraints

- $1 \leq n \leq 10$
- $1 \leq m \leq 256$
- The length of a name is positive and no more than 64.
- The name consist of only letters.

Sample Testcases

Sample Input 1

```
2 3
Alice
1 2 3
4 5 6
7 8 9
Blob
1 2 3
4 5 6
7 8 9
1 2 4 8 6 3 9 5 7
```

Sample Output 1

```
3 Alice Blob
```

Source

This problem is modified from Judge Girl Problem 99