Assignment-1

Data Structures and Algorithms
Sorting, Pointers, and Linked Lists

Due Date: 28 Jan, 2025 11:59 PM

Important Notes

- You are only allowed to use **C** for this assignment.
- You are **not allowed** to use the inbuilt `qsort` function in C. You have to implement a sorting algorithm of your choice on your own.
- You are not allowed to copy the exact code given in class. You are allowed to note the logic, but you need to implement the code by yourself.
- Since this assignment is supposed to be a practice for pointers, you are not allowed to use any functions from `<string.h>`. Please implement the required functionality yourself.
- For questions 3 and 4, all functions mentioned must be defined with **exact** function name and function parameters and return type (we would run scripts to check this, so match the case of alphabets also).
- OPERATION names are case sensitive.
- Doubt Document.
- Plagiarism Policy. Please read this before you start the assignment.

TOTAL: [100 PTS]

1 Find the Closest Pair to a Target Sum [15 PTS]

1.1 Problem Statement

You are given an array of size n of distinct integers and a target number. You must find two distinct numbers k_1 and k_2 ($k_1 \leq k_2$) in the array such that:

1. The absolute difference $|\text{target} - (k_1 + k_2)|$ is minimized.
2. If there are multiple such pairs with the same minimum difference, choose the pair where $|k_1 - k_2|$ is the smallest.
3. If there are still multiple such pairs, choose the pair with the larger value of k_1.

1.2 Input Format

- **First line:** An integer n.
- **Second line:** n space-separated integers (the elements of the array).
- **Third line:** A single integer (the target).

Constraints:

- $2 \leq n \leq 10^4$ **[6 PTS]**
- $2 \leq n \leq 10^6$ **[9 PTS]**

1.3 Output Format

Print two space-separated integers k_1 and k_2 (with the condition $\mathbf{k_1} < \mathbf{k_2}$) that represent the chosen pair with the minimum distance to the target according to the tiebreak rule.

1.4 Example Testcase

Sample Input	Sample Output
6 5 2 3 10 6 9 15	6 9

1.5 Explanation

Among all pairs whose sums are closest to 15, $(6, 9)$ achieves a sum of 15, which is at distance 0. Although the pair $(5, 10)$ also sums to 15, we choose $(6, 9)$ because $k_1 = 6$ is larger than 5 in the other pair.

2 Doctor A's Appointment Adventures [20 PTS]

2.1 Problem Statement

Dr. A runs a clinic where all appointments must be exactly L minutes long. Patients request time slots using positive integers (i.e., $\texttt{start}_i$ and $\texttt{end}_i$), but many requests overlap. Your task is to select the maximum possible number of non-overlapping appointments, all of which must be exactly L minutes long. Note that an appointment may begin at the exact moment the previous one ends; in other words, if $\texttt{end}_i = \texttt{start}_{i+1}$, the appointments do not overlap.

2.2 Input Format

- **First line:** Two integers n and L, where n is the number of patients and L is the fixed duration (in minutes) for each appointment.
- **Next n lines:** Each line has two positive integers $\texttt{start}_i$ and $\texttt{end}_i$ indicating the requested start time and end time for an appointment.

Constraints:

- $2 \leq n \leq 10^4,\ 1 \leq L \leq 10^9,\ 1 \leq \texttt{start}_i, \texttt{end}_i \leq 10^9$ **[8 PTS]**
- $2 \leq n \leq 10^6,\ 1 \leq L \leq 10^9,\ 1 \leq \texttt{start}_i, \texttt{end}_i \leq 10^9$ **[12 PTS]**

2.3 Output Format

Print a single integer indicating the maximum number of appointments that can be scheduled without overlap, with the constraint that each appointment is exactly L minutes long.

2.4 Example Testcase

Sample Input	Sample Output
6 30 100 130 115 145 130 160 145 175 160 190 195 225	4

2.5 Explanation

A possible selection is as follows:

1. **100–130**: Selected first since it is the earliest.
2. **115–145**: Overlaps with 100-130, so skip.
3. **130–160**: **Does not overlap**, so select it.
4. **145–175**: Overlaps with 130-160, so skip.
5. **160–190**: Does not overlap with 130-160, so select it.
6. **195–225**: Does not overlap with 160-190, so select it.

Hence, 4 appointments fit without overlap.

3 Pointer Warmup [15 PTS]

Pointers is a very important concept in C programming. Let us begin with a few warm-up questions.

3.1 File Structure

- `functions.h` - Which has the function prototypes.
- `functions.c` - Which implements the functions defined in 3.2.
- `main.c` - Which must use the functions and perform various operations on input and produce output as specified later.

3.2 Functions

- `void reverseString(char* str, int length)`: takes in a pointer to a string and its length and reverses the string in place.
- `char* compressString(char* str, int length)`: takes in a pointer to a string and its length, and returns a pointer to a string that is a compressed version of the original, where each character is followed by the number of times it appears consecutively in the original string. For example, "**aaabbc**" would be compressed to "**a3b2c1**".
  NOTE: The number of times a character occurs continuously can be any number (i.e., it need not be a single digit).
  Important: You are *not* allowed to use any functions from `<string.h>` or the function `sprintf`.
- `int* uniqueElements(int* arr, int length)`: takes in a pointer to an array of integers and its length, and returns a pointer to an array of integers that contains only the unique elements of the original array, without using any library functions except `malloc`, `calloc`, `realloc`, .
  NOTE: The size of the array you are returning must be exactly equal to the number of unique elements.
- `int** transpose(int** matrix, int NumRow, int NumCol)`: takes in a pointer to a matrix of integers and its dimensions and returns a pointer to a matrix that represents the transpose of the original matrix.
  NOTE: The dimensions of the matrix you are returning must be exactly equal to the dimensions of the mathematical transpose.

3.3 Input and Output

The first line contains T, the number of OPERATION's that need to be done. $1 \leq T \leq 100$.

The first line of an OPERATION consists of a string indicating the OPERATION name.

The next few lines contain the required input data according to the table i.e.

Operation name	**Corresponding function**
OPER1	*reverseString*
OPER2	*compressString*
OPER3	*uniqueElements*
OPER4	*transpose*

OPERATION	INPUT FORMAT	CONSTRAINTS	OUTPUT FORMAT
OPER1	First line contains an integer n indicating the length of the string. Second line contains the string	$1 \leq n \leq 10^4$ [**2.5 PTS**]	String (Reversed)
OPER2	First line contains one integer n indicating the length of the string. Second line contains the string	$1 \leq n \leq 10^4$ [**2.5 PTS**] $str[i]$ contains only lowercase alphabet	String (Compressed)
OPER3	First line contains an integer n indicating the length of the array. Second line contains n space separated integers.	$1 \leq n \leq 10^4$ [**5 PTS**] $1 \leq arr[i] \leq 10^4$	All unique elements in a line, space separated and should be in the order they appear first
OPER4	First line contains 2 integers R and C indicating the number of rows and the number of columns, respectively. Following R lines contains C space-separated elements	$1 \leq R, C \leq 10^2$ [**5 PTS**] $1 \leq Mat[i][j] \leq 10^6$	The transposed matrix. Each row should be printed on a new line.

3.4 Example Test Cases

Input	Output
4	
OPER1 10 abcdefghij	jihgfedcba
OPER2 12 aacceeggiiaa	a2c2e2g2i2a2
OPER3 10 1 1 2 4 5 9 9 1 6 8	1 2 4 5 9 6 8
OPER4 3 4 1 2 3 4 2 3 4 5 3 4 5 6	1 2 3 2 3 4 3 4 5 4 5 6

4 Circular Linked List [25 PTS]

In this question, the aim is to create a self-adjusting circular linked list. A self-adjusting list is like a regular list, except that all insertions are performed at the front. When an element is accessed by a *Find* function, it is moved to the front of the list without changing the relative order of the rest of the elements.

4.1 File Structure

- `functions.h` - Which has the function prototypes.
- `functions.c` - Which implements the functions defined in 3.3.
- `main.c` - Which must use the functions and perform various operations on input and produce output as specified later.

4.2 Structs

Each node in the circular linked list needs to have the following attributes:

- `int` Element
- `PtrNode` NextNode

Note:

- You may add additional attributes if they help simplify or optimize your code.
- `PtrNode` represents a pointer to a `Node`; that is, `PtrNode` is equivalent to `Node*`.

4.3 Functions

- `PtrNode Insert(PtrNode Head, int num)`: creates a new node containing the integer `num` and inserts (adds it to the front) it into the circular linked list (identified by `Head`).
- `PtrNode Find(PtrNode Head, int num)`: searches the circular linked list (identified by `Head`) for the node with the integer `num` and then moves the node containing `num` to the front of the circular linked list. The function returns the pointer to the node that contains `num`.(If `num` doesn't exist in the circular linked list, then return a `NULL` pointer).
- `void Print(PtrNode Head)`: prints the elements of the circular linked list in order.

Note: After completion of each operation the linked list must necessarily be circular

4.4 Input and Output

The first line contains T, the number of operations that need to be performed. $1 \leq T \leq 1000$.

The next line consists of a string indicating the OPERATION to be executed.

The next few lines give the data needed for the operation i.e.

Operation name	**Corresponding function**
OPER1	*Insert*
OPER2	*Find*
OPER3	*Print*

OPERATION	INPUT FORMAT	CONSTRAINTS	OUTPUT FORMAT
OPER1	First line contains an integer n that must be inserted into the circular linked list.	$1 \leq n \leq 10^6$ [**10 PTS**]	
OPER2	First line contains an integer n that must be searched in the circular linked list.	$1 \leq n \leq 10^6$ [**10 PTS**]	
OPER3		[**5 PTS**]	Print all the elements in the circular linked list in order.

4.5 Example Test Cases

Input	Output
10	
OPER1 1	
OPER1 2	
OPER1 3	
OPER2 2	
OPER2 3	
OPER1 5	
OPER2 1	
OPER1 9	
OPER1 7	
OPER3	7 9 1 5 3 2

4.6 Explanation

We have:

1. `Insert 1` : Inserts 1. The current state of the circular linked list will be:

$$1$$

2. `Insert 2` : Inserts 2. The current state of the circular linked list will be:

$$2 - 1$$

3. `Insert 3` : Inserts 3. The current state of the circular linked list will be:

$$3 - 2 - 1$$

4. `Find 2` : Finds 2 and re-adjusts it. The current state of the circular linked list will be:

$$2 - 3 - 1$$

5. `Find 3` : Finds 3 and re-adjusts it. The current state of the circular linked list will be:

$$3 - 2 - 1$$

6. `Insert 5` : Inserts 5. The current state of the circular linked list will be:

$$5 - 3 - 2 - 1$$

7. `Find 1` : Finds 1 and re-adjusts it. The current state of the circular linked list will be:

$$1 - 5 - 3 - 2$$

8. `Insert 9` : Inserts 9. The current state of the circular linked list will be:

$$9 - 1 - 5 - 3 - 2$$

9. `Insert 7` : Inserts 7. The current state of the circular linked list will be:

$$7 - 9 - 1 - 5 - 3 - 2$$

10. `Print` : Prints the circular linked list.

5 Sparse Matrices as Linked Lists [25 PTS]

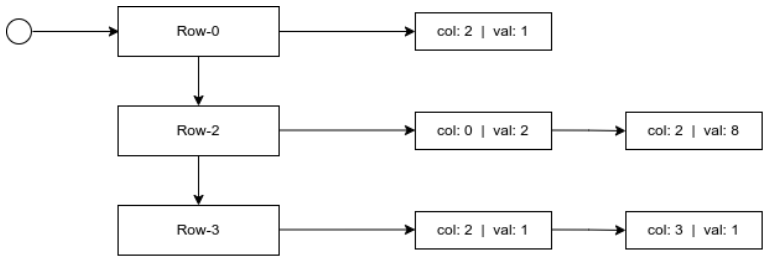
Linked lists are commonly used in the representation of sparse matrices because they can efficiently store and retrieve data without using too much memory. In a sparse matrix, most elements are zero, so storing all of these zeros in memory is inefficient. Instead, only the non-zero elements and their row and column indices are stored.

One way to use linked lists to represent a sparse matrix is to use a linked list for each non-zero row of the matrix. Each node in the lists corresponds to a non-zero element in the row and holds the column index and the value of the element.
Another linked list that stores pointers to the heads of the row lists connect these row lists together.

Example:

$$M = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 8 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Can be represented as



5.1 File Structure

- `functions.h` - Which has the required function prototypes.
- `functions.c` - Which implements the required functions.
- `main.c` - Which must use the functions and perform various operations on input and produce output as specified later.

5.2 Problems:

1. Represent a sparse matrix using linked lists.
2. Find Transpose of a matrix
3. Add two matrices.
4. Multiply 2 Matrices. (This question is only for practice and **will not be graded**, but we strongly suggest you to practice this part, as it will give you more exposure to pointers and linked lists).

5.3 Input and Output

The first line contains a string named OPERATION (ADD, MUL, or TRA) denoting the type of operation to be performed on the matrix/matrices.

Only one OPERATION would be asked per run.

OPR	INPUT FORMAT	CONSTRAINTS
TRA	The second line contains the dimensions of the matrix N and M, followed by K denoting the number of non-zero elements the matrix. (i.e. $N\ M\ K$ for matrix of size $N \times M$) The next K lines contain three integers i, j, val representing the row index, column index, and value of the non-zero elements in the first matrix.	$1 \leq N, M \leq 10^9$ $1 \leq K \leq min(N * M, 10^3)$ [**5 PTS**] $1 \leq K \leq min(N * M, 10^6)$ [**5 PTS**] $0 \leq i < n$, $0 \leq j < m$ $-10^9 \leq val \leq 10^9$,$val \neq 0$
ADD	The second line contains the dimensions of the matrices N and M, followed by K_1, K_2 denoting the number of non-zero elements in each of the matrices. (i.e. $N\ M\ K_1\ K_2$ for matrices of size $N \times M$) (i.e. $N\ M\ K_1\ K_2$ for matrices of size $N \times M$) The next K_1 lines contain three integers i, j, val representing the row index, column index, and value of the non-zero elements in the first matrix. Followed by K_2 lines representing the non-zero elements of the second matrix in the same format	$1 \leq N, M \leq 10^9$ $1 \leq K_1, K_2 \leq min(N * M, 10^3)$ [**10 PTS**] $1 \leq K_1, K_2 \leq min(N * M, 10^6)$ [**5 PTS**] $0 \leq i < n$, $0 \leq j < m$ $-10^9 \leq val \leq 10^9$, $val \neq 0$
MUL	The second line contains the dimensions of the matrices N, M, L, followed by K_1, K_2, denoting the number of non-zero elements in each of the matrices. First matrix dimensions $N \times M$, Second matrix dimensions $M \times L$ (i.e. $N\ M\ L\ \ K_1\ K_2$ for matrices of size $N \times M$) The next K_1 lines contain three integers i, j, val representing the row index, column index, and value of the non-zero elements in the first matrix. Followed by K_2 lines having the non-zero elements in the second matrix in the same format	$1 \leq N, M, L \leq 10^9$ $1 \leq K_1 \leq min(N * M, 10^6)$ $1 \leq K_2 \leq min(M * L, 10^6)$ $0 \leq i < n$, $0 \leq j < m$ $-100 \leq val \leq 100$,$val \neq 0$ max(no of non zero rows of mat1, no of non zero columns of mat2) $\times \max(K_1, K_2)$, $\leq 10^3$ [**Extra**] max(no of non zero rows of mat1, no of non zero columns of mat2) $\times \max(K_1, K_2)$, $\leq 10^6$ [**Extra**]

Example:

ADD
4 3 5 4
0 2 1
2 0 1
2 2 8
3 1 1
3 2 1
1 1 1
2 0 −1
2 2 −8
3 2 4

Corresponds to:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 8 \\ 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & -8 \\ 0 & 0 & 4 \end{bmatrix}$$

Output:
The first line of the output should contain the number of non-zero elements (W) in the resultant matrix after the operation.
The next W lines should contain three spaced integers denoting i, j, val, denoting the row number(0 indexed), column number (0 indexed), and value of non-zero elements in the resultant matrix.

The expected output for the input shown above is:

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

Sample Test Cases:

Input	**Output**
TRA 4 3 4 0 2 1 2 0 1 2 2 8 3 2 1	4 0 2 1 2 0 1 2 2 8 2 3 1
ADD 4 3 5 4 0 2 1 2 0 1 2 2 8 3 1 1 3 2 1 1 1 1 2 0 -1 2 2 -8 3 2 4	4 0 2 1 1 1 1 3 1 1 3 2 5
MUL 3 2 2 3 2 1 0 1 1 1 9 2 1 4 0 1 7 1 0 2	3 1 0 18 1 1 7 2 0 8

NOTE:

- For matrix transpose, though it is easier to produce the output by simply swapping i and j from the input, we recommend you build a linked list representation of the matrix and perform transpose on it. (This may be useful in matrix multiplication)
- As shown in the examples above, you can assume that the input is provided in sorted order of (i, j). The output need not be sorted.

6 Submission Details

- For questions 3 and 4 **OJ** is only used to check the correctness of your code. You are eligible for a full score(for a given function) if and only if the required functions are implemented as instructed. You will be graded on your moodle submission.
- Question 1, 2, 5 will be graded directly as your OJ score, but you still have to submit it in moodle.
- **All solutions submitted in OJ will be considered for plagiarism**
- **Your submission to moodle will also be checked for plagiarism**
- Moodle submission should be of the given format. A *roll_number.zip*, which contains a folder named your roll number.

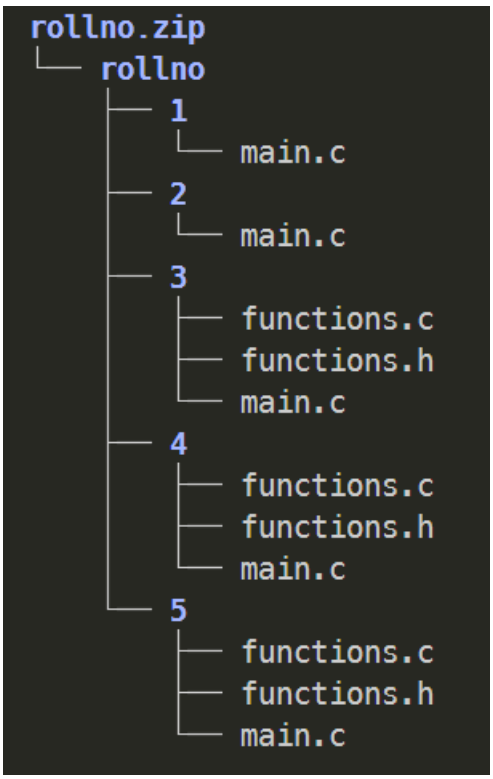


Figure 1: File format

All the best and happy coding!