STL - Practice problems

Data Structure and Algorithms II Laboratory

Fall 2024

Introduction

These are some practice problems to help you get started with learning and implementing various concepts related to C++ Standard Template Library (STL), such as sets, priority queues, and other container types. These problems will guide you through common use cases and will give you a solid foundation for solving real-world problems with C++ STL.

If you wish to dive deeper and explore more advanced topics, there are several great online resources that can help you improve your understanding of these concepts:

- Codeforces STL
- HackerRank C++ STL
- LeetCode Algorithm Problems

We encourage you to explore these platforms for a more in-depth learning experience and to practice solving problems related to C++ STL and algorithms.

Vector and Pair

Problem 1: Sorting Students by Marks

Problem Description: Write a program that reads a list of students' names and marks (as pairs), stores them in a `vector<pair<string, int>>`, and sorts the students by their marks in descending order. If two students have the same marks, sort them alphabetically by their names.

Input Format:

- The first line contains an integer N (number of students).
- The next N lines contain a string (student name) followed by an integer (marks).

Output Format:

- Print N lines, where each line contains the student's name and their marks in sorted order.

Sample Input and Output:

Sample Input	Sample Output
3 John 85 Alice 90 Bob 85	Alice 90 Bob 85 John 85

Explanation:

- The marks are sorted in descending order.
- For students with the same marks, their names are sorted alphabetically.

Problem 2: Storing and Manipulating Coordinates

Problem Description: Given a list of 2D points, store them in a `vector<pair<int, int>>`. Write a program to:

1. Sort the points by x-coordinate in ascending order.
2. If two points have the same x-coordinate, sort by y-coordinate.

Input Format:

- The first line contains an integer N (number of points).
- The next N lines contain two integers representing the x and y coordinates of each point.

Output Format:

- Print N lines, each containing a pair of integers (x, y) in sorted order.

Sample Input and Output:

Sample Input	**Sample Output**

5
3 4
1 2
3 2
5 1
1 3
	1 2
	1 3
	3 2
	3 4
	5 1

Explanation:

- Points are first sorted by their x-coordinate.
- If x-coordinates are equal, points are sorted by their y-coordinate.

Problem 3: Finding Maximum Product

Problem Description: You are given a vector of integers. Find two numbers in the vector that produce the maximum product and return them as a pair.

Input Format:

- The first line contains an integer N (size of the vector).
- The second line contains N integers (elements of the vector).

Output Format:

- Print a single line containing two integers (the pair of numbers producing the maximum product).

Sample Input and Output:

Sample Input	**Sample Output**
6 1 10 2 6 8 3	8 10

Explanation:

- The maximum product is $8 \times 10 = 80$.
- The program identifies these numbers and outputs them as a pair.

Problem 4: Frequency of Elements

Problem Description: Given a vector of integers, count the frequency of each unique number and store the result in a `vector<pair<int, int>>` (number, frequency). Sort the pairs in ascending order of the numbers.

Input Format:

- The first line contains an integer N (size of the vector).
- The second line contains N integers.

Output Format:

- Print each unique number followed by its frequency, sorted by the numbers.

Sample Input and Output:

Sample Input	**Sample Output**
7	2 2
4 2 4 3 2 4 5	3 1
	4 3
	5 1

Explanation:

- The vector contains duplicate elements.
- Frequencies are calculated:
  - 2 appears 2 times.
  - 3 appears 1 time.
  - 4 appears 3 times.
  - 5 appears 1 time.
- The output is sorted by the numbers.

Problem 5: Implement Stack using Vector

Problem Description: Write a program to implement a stack using a `vector`. Your stack should support the following operations:

- **Push**: Add an element to the top of the stack.
- **Pop**: Remove the top element of the stack.
- **Top**: Display the top element of the stack.
- **Empty**: Check if the stack is empty.

Input Format:

- The first line contains an integer Q (number of operations).
- The next Q lines contain operations in one of the following formats:
  - `PUSH X` (where X is an integer to push onto the stack).
  - `POP` (removes the top element).
  - `TOP` (prints the top element).
  - `EMPTY` (prints `YES` if the stack is empty, otherwise `NO`).

Output Format:

- For `TOP`, print the top element.
- For `EMPTY`, print `YES` or `NO`.
- For invalid `POP` or `TOP` operations (when the stack is empty), print `ERROR`.

Sample Input and Output:

Sample Input	Sample Output
6 PUSH 10 PUSH 20 TOP POP TOP EMPTY	20 10 NO

Explanation:

1. Push 10 → Stack: [10]
2. Push 20 → Stack: [10, 20]
3. Top → 20
4. Pop → Stack: [10]
5. Top → 10
6. Empty → NO

Problem 6: Implement Queue using Vector

Problem Description: Write a program to implement a queue using a `vector`. Your queue should support the following operations:

- **Enqueue**: Add an element to the back of the queue.
- **Dequeue**: Remove the element from the front of the queue.
- **Front**: Display the front element of the queue.
- **Empty**: Check if the queue is empty.

Input Format:

- The first line contains an integer Q (number of operations).
- The next Q lines contain operations in one of the following formats:
  - `ENQUEUE X` (where X is an integer to add to the queue).
  - `DEQUEUE` (removes the front element).
  - `FRONT` (prints the front element).
  - `EMPTY` (prints `YES` if the queue is empty, otherwise `NO`).

Output Format:

- For `FRONT`, print the front element.
- For `EMPTY`, print `YES` or `NO`.
- For invalid `DEQUEUE` or `FRONT` operations (when the queue is empty), print `ERROR`.

Sample Input and Output:

Sample Input	**Sample Output**
6 ENQUEUE 5 ENQUEUE 10 FRONT DEQUEUE FRONT EMPTY	5 10 NO

Explanation:

1. Enqueue 5 → Queue: [5]
2. Enqueue 10 → Queue: [5, 10]
3. Front → 5
4. Dequeue → Queue: [10]
5. Front → 10
6. Empty → NO

Stack and Queue

Problem 1: Reverse a String Using a Stack

Problem Description: Write a C++ program that reverses a string using a stack. A stack operates on the Last In First Out (LIFO) principle. The program should take a string input, push each character onto a stack, and then pop the characters to reverse the string.

Input Format:

- A string s containing alphabets and special characters.

Output Format:

- The reversed string.

Sample Input and Output:

Sample Input	**Sample Output**
Hello, World!	!dlroW ,olleH

Explanation:

- Push each character of the string onto a stack.
- Pop the characters off the stack one by one, which gives the string in reverse order.

Problem 2: Check for Balanced Parentheses Using a Stack

Problem Description: Write a C++ program to check if a given expression has balanced parentheses. An expression is considered balanced if:

- Every opening parenthesis (`(`) has a corresponding closing parenthesis (`)`).
- Parentheses are properly nested.

Input Format:

- A string `expression` containing parentheses and other characters.

Output Format:

- `Balanced` if the parentheses are balanced.
- `Not Balanced` if they are not balanced.

Sample Input and Output:

Sample Input	**Sample Output**
`(())`	Balanced
`(()`	Not Balanced

Explanation:

- In the first case, the parentheses are properly balanced.
- In the second case, the parentheses are unbalanced because the opening parenthesis `(` does not have a matching closing parenthesis.

Problem 3: Implement a Simple Queue (FIFO)

Problem Description: Implement a simple queue using C++ STL's `queue`. A queue operates on the First In First Out (FIFO) principle. The program should demonstrate basic queue operations like adding and removing elements.

Input Format:

- A sequence of integers to enqueue into the queue.

Output Format:

- The elements of the queue in the order they are dequeued (FIFO order).

Sample Input and Output:

Sample Input	**Sample Output**
1 2 3 4 5	1 2 3 4 5

Explanation:

- In a queue, the first element added is the first one removed.
- When we dequeue, the numbers come out in the same order they were added.

Problem 4: Reverse the First K Elements of a Queue

Problem Description: Write a program to reverse the first K elements of a queue. For example, if the queue contains $\{1, 2, 3, 4, 5\}$ and $K = 3$, after reversing the first 3 elements, the queue should become $\{3, 2, 1, 4, 5\}$.

Input Format:

- An integer K.
- A queue of integers.

Output Format:

- A queue where the first K elements are reversed and the remaining elements stay in the same order.

Sample Input and Output:

Sample Input	**Sample Output**
Queue: 1 2 3 4 5 K: 3	Queue: 3 2 1 4 5

Explanation:

- Reverse the first K elements using a stack, which reverses their order.
- Append the remaining elements in their original order.

Problem 5: Sort Elements Using a Priority Queue

Problem Description: Write a program that sorts a list of numbers using a priority queue (max-heap). A priority queue allows you to insert elements in any order, but when elements are removed, they are accessed in descending order (highest to lowest).

Input Format:

- A sequence of integers.

Output Format:

- The elements sorted in descending order.

Sample Input and Output:

Sample Input	**Sample Output**
10 3 15 7 8 23 74	74 23 15 10 8 7 3

Explanation:

- A max-heap stores the highest element at the top.
- When elements are removed from the priority queue, they are retrieved in descending order.

Set and Priority Queue

Problem 1: Unique Elements in a Set

Problem Description: You are given a list of integers. Use a `set<int>` to remove duplicates and print the unique integers in ascending order.

Input Format:

- The first line contains an integer N (size of the vector).
- The next N lines contain integers (elements of the vector).

Output Format:

- Print the unique integers in ascending order.

Sample Input and Output:

Sample Input	Sample Output
7	2
4 2 4 3 2 4 5	3
	4
	5

Explanation:

- The `set` removes duplicate values, leaving only the unique integers in sorted order.

Problem 2: Finding the Smallest K Elements

Problem Description: Given a vector of integers, find the smallest K elements in the vector and return them in ascending order. You must use a `priority_queue` to solve this.

Input Format:

- The first line contains two integers N (size of the vector) and K (number of smallest elements to find).
- The second line contains N integers.

Output Format:

- Print the K smallest elements in ascending order.

Sample Input and Output:

Sample Input	**Sample Output**
6 3 6 2 5 3 2 8 1	2 3 6

Explanation:

- The smallest 3 elements are $1, 2, 3$, sorted in ascending order.

Problem 3: Merge Two Sets

Problem Description: You are given two sets of integers. Write a program to merge the sets and output the result in sorted order. Use a `set<int>` to perform this operation.

Input Format:

- The first line contains an integer N (size of the first set).
- The second line contains N integers (elements of the first set).
- The third line contains an integer M (size of the second set).
- The fourth line contains M integers (elements of the second set).

Output Format:

- Print the merged set in sorted order.

Sample Input and Output:

Sample Input	Sample Output

5
1 2 3 4 5
3
2 3 6
	1 2 3 4 5 6

Explanation:

- The sets are merged, and duplicates are removed by the `set`, leaving only unique elements in sorted order.

Problem 4: Top K Elements Using Priority Queue

Problem Description: You are given a vector of integers. Find the top K largest elements in the vector using a `priority_queue`. Return them in descending order.

Input Format:

- The first line contains two integers N (size of the vector) and K (number of largest elements to find).
- The second line contains N integers.

Output Format:

- Print the K largest elements in descending order.

Sample Input and Output:

Sample Input	Sample Output
6 3 1 10 2 6 8 3	10 8 6

Explanation:

- The largest 3 elements are $10, 8, 6$, printed in descending order.

Problem 5: Checking for Subset Using Set

Problem Description: Given two sets of integers, check if the first set is a subset of the second set using a `set<int>`.

Input Format:

- The first line contains an integer N (size of the first set).
- The second line contains N integers (elements of the first set).
- The third line contains an integer M (size of the second set).
- The fourth line contains M integers (elements of the second set).

Output Format:

- Print `YES` if the first set is a subset of the second set. Otherwise, print `NO`.

Sample Input and Output:

Sample Input	Sample Output

3
1 2 3
5
1 2 3 4 5
	YES

Explanation:

- The first set $\{1, 2, 3\}$ is a subset of the second set $\{1, 2, 3, 4, 5\}$, so the output is `YES`.

Problem 6: Find the Median of a Stream of Numbers

Problem Description: Write a program that maintains a running median of a stream of numbers. For each new number added, print the current median. Use a `priority_queue` to maintain the two halves of the numbers.

Input Format:

- The first line contains an integer N (number of integers to process).
- The next N lines contain integers to be added to the stream.

Output Format:

- After each number is added, print the current median.

Sample Input and Output:

Sample Input	Sample Output
5	1
1	1.5
2	1
3	2
4	2

Explanation:

- After adding the first number 1, the median is 1.
- After adding the second number 2, the median is $(1 + 2)/2 = 1.5$.
- After adding the third number 3, the median is 2, and so on.