M24-CS1.304 Data Structures and Algorithms for Problem Solving

Assignment 1

Deadline : 16 Sept 2024

Important Points:

1. **The assignment has 4 questions where each question has one or two parts. You need to do ANY THREE among the four questions. Each of the first three questions weigh 20 marks each. The last (fourth question) on Seam Carving has a bonus of 2 marks weighing 22 marks.**
2. **Only C++ is allowed. (Unless specified otherwise)**
3. **Directory structure to be followed for submission:**

   2024201001_A1

   ├── 2024201001_A1_Q1a.cpp

   ├── 2024201001_A1_Q1b.cpp

   ├── 2024201001_A1_Q2.cpp

   ├── 2024201001_A1_Q3a.cpp

   ├── 2024201001_A1_Q3b.cpp

   └── 2024201001_A1_Q4.cpp

   …. And so on

   Replace your roll number in place of 2024201001

3. **Submission Format:** Follow the above mentioned directory structure and zip the `RollNo_A1` folder and submit `RollNo_A1.zip` on Moodle.

Note: All submissions which are not in the specified format will be awarded 0 in the assignment.

4. **C++ STL (including vectors) is not allowed** for any of the questions unless specified otherwise in the question. So `#include <bits/stdc++.h>` is not allowed.
5. You can ask queries by posting on Moodle.
6. **Late Submission Rule:** Deadlines for assignments are final and will not be extended. Late submissions cost 5% loss of marks for each late day upto 6 days. (i.e, 5%, 10%,...). Submissions beyond 6 days from the deadline shall receive 0 marks.

NOTE: In case of plagiarism in any of the questions, the students involved will be awarded -10 as the final marks of Assignment 1.

1. Deque [20 marks]

a. [15 marks]

Problem Statement:

Implement Deque

What is deque?

- Deque is the same as dynamic arrays with the ability to resize itself automatically when an element is inserted, with their storage being handled automatically by the container.
- They support insertion and deletion from both ends in amortized constant time.
- Inserting and erasing in the middle is linear in time.

Operations : The C++ standard specifies that a legal (i.e., standard-conforming) implementation of deque must satisfy the following performance requirements: (consider the data type as T)

1. deque() - initialize an empty deque. Time complexity: O(1)

2. deque(n) - initialize a deque of length n with all values as default value of T. Time complexity: O(n)

3. deque(n, x) - Initialize a deque of length n with all values as x. Time complexity: O(n)

4. void push_back(x) - append data x at the end. Time complexity: **constant amortized time**

5. void pop_back() - erase data at the end. Time complexity: **constant amortized time**

6. void push_front(x) - append data x at the beginning. Time complexity: **constant amortized time**

7. void pop_front() - erase an element from the beginning. Time complexity: **constant amortized time**

8. T front() - returns the first element(value) in the deque. If the first element is not present, return the default value of T. Time complexity: O(1)

9. T back() - returns the last element(value) in the deque. If the last element is not present, return the default value. Time complexity: O(1)

10. T D[n] - returns the nth element of the deque. You need to overload the [ ] operator. If nth element is not present return default value of T. Time complexity: O(1)

11. bool empty() - returns true if deque is empty else returns false. Time complexity: O(1)

12. int size() - returns the current size of deque. Time complexity: O(1)

13. void resize(n) - change the size dynamically to new size n. Time complexity: O(n)

◦ If the new size n is greater than the current size of the deque, then insert new elements with the default value of T at the end of the queue.

◦ If the new size n is smaller than the current size, then keep n elements from the beginning of the deque.

14. void resize(n, d) - change the size dynamically to new size n. Time complexity: O(n)

◦ If the new size n is greater than the current size of the deque, then insert new elements with value d at the end of the queue.

◦ If the new size n is smaller than the current size, then keep n elements from the beginning of the deque.

15. void reserve(n) : change the capacity of deque to n, if n > current capacity; otherwise do nothing. Time complexity: O(n)

16. void shrink_to_fit() - reduce the capacity of the deque to current size. Time Complexity: O(size())

17. void clear() - remove all elements of deque. Time complexity: O(n)

18. int capacity() - return the current capacity of deque. Time complexity: O(1)

Note :

1. Your deque should be generic type i.e. it should be datatype independent and can support primitive data types like integer, float, string, etc. **Hint:** Use template in C++ (link)
2. For 1, 2 & 3 You can either define a constructor for the class or initialize the class object using void return type functions.
3. C++ STL is **not allowed** (including vectors, design your own if required)
4. D[0] - element at index 0 (i.e. first element from the front),
   D[1] - element at index 1 (i.e. second element from the front),
   D[-1] - element at last index (i.e. first element from the back),
   D[-2] - element at second last index (i.e. second element from the back)
5. Size of the deque is the number of elements currently present in your deque.
6. Capacity of the deque is the number of elements your deque can accommodate with currently held memory.
7. During Operation 1 both size and capacity of the deque should be set to zero.
8. If size is equal to capacity and a new element is inserted, then the capacity is doubled, unless capacity is zero, then it will become one.
9. If you have doubts about deciding the new capacity in any of the operations, refer to the behavior of the member functions of STL vector containers.

B.[5 marks]

Problem Statement:

Using the deque implemented above, you are supposed to implement a randomized queue. Implement the following operations:

1. **void enqueue(x):** add the item to the randomized queue
2. **T dequeue():** remove and return a random item
3. **T sample():** return a random sample (but do not remove it)

The above operations must be done in **constant amortized time**. Randomness of fetched items will be checked.

Randomized queue: A *randomized queue* is similar to a stack or queue, except that the item removed is chosen uniformly at random from items in the data structure.

Constant amortized time: Refer this [link]

Note: each element in the randomized queue must have an equal probability of being selected. **Only these three operations will be checked in this part of the question.**

Input Format:

Design an infinitely running menu-driven main function. Each Time the user inputs an integer corresponding to the serial number of the operation listed above. Then, take necessary arguments related to the selected operation and execute the respective method. Finally, the program must exit with status code 0, when 0 is provided as a choice.

2. Trie Harder [20 marks]

Problem Statement:

Design an efficient spell checker using the ***Trie*** data structure which supports the functionalities mentioned below.
1. Spell Check: Check if the input string is present in the dictionary.
2. Autocomplete: Find all the words in the dictionary which begin with the given input.
3. Autocorrect: Find all the words in the dictionary which are at an edit distance(Levenshtein distance) of at most 3 from the given input.

Input Format:

- First line will contain two space separated integers **n**, **q** which represents the number of words in the dictionary and the number of queries to be processed respectively.
- Next **n** lines will contain a single string **s** which represents a word in the dictionary.
- Next **q** lines will contain two space separated values, First one will be an integer $\mathbf{a_i}$ and second will be a string $\mathbf{t_i}$.
    - $\mathbf{a_i}$ = 1 means Spell Check operation needs to be done on $\mathbf{t_i}$.
    - $\mathbf{a_i}$ = 2 means Autcomplete operation needs to be done on $\mathbf{t_i}$.
    - $\mathbf{a_i}$ = 3 means Autocorrect operation needs to be done on $\mathbf{t_i}$.
- Both strings **s** and **t** consist of lowercase English letters.

Output Format:

For each query print the result in a new line.
- Spell check: Print '1' if string is present in the dictionary, otherwise '0'.
- Autocomplete & Autocorrect: Print the number of words in the first line. The following lines will be the set of words in lexicographical order

Constraints:

1 <= n <= 1000
1 <= q <= 1000
1 <= len(s) <= 100 1 <= len(ti) <= 110

Sample Input:

10 4

consider

filters

filers

entitled

tilers

litter

dames

filling

grasses

fitter

1 litter

1 dame

2 con

3 filter

Sample Output:

1
0
1

consider

5

filers

filters

fitter

litter

tilers

Note:

1. Only trie should be used for storing the words in the dictionary.
2. You are allowed to use vector for this problem

3. Priority Queue [20 marks]

a. [15 marks]

Problem Statement: Implement priority queue

What is a priority queue?

- A priority queue is a data structure that maintains a collection of elements,
each associated with a priority or value. Elements are stored in a way that allows the retrieval of the element with the highest (or lowest) priority quickly.
- Unlike deques or lists, a priority queue doesn't maintain the elements in any specific order based on their values, except for ensuring that the highest (or lowest) priority element is readily accessible.
Operations : The C++ standard specifies that a legal (i.e., standard-conforming) implementation of priority queue must satisfy the following performance requirements.

1. priority_queue() - initialize an empty priority queue. Time complexity: O(1)
2. int size() - returns the current size of the priority queue. Time complexity: O(1)
3. void push(int el) - insert an element *el* in the priority queue. Time complexity: O(log(size()))
4. int top() - returns the top (highest or lowest priority) element in the priority queue. Time complexity: O(1)
5. void pop() - remove the top element of the priority queue. Time complexity: O(log(size()))
6. bool empty() - returns true if the priority queue is empty else returns false. Time complexity: O(1)

b. [5 marks]

Problem Statement : *David's Bakery* has introduced a new promotion for students. Under this promotion, he monitors his daily sales, and if the sales on a specific day is greater than or equal to the **combined sum of the median sale for a trailing number of *d* days** and **the median sale from the first day of the promotion**, (follow through the example included below to understand better) then he offers free *'Cheeeeeese Maggi'* to the first five customers on the following day. However, David will only provide this free offer if he has transaction data for at least the trailing number of *d* prior days.

Given the number of trailing days *d* and the daily sales record for David's Bakery over a period of *n* days, can you determine how many days david will offer free maggi to students during this entire period of *n* days?

Input Format :

- The first line contains two space-separated integers *n* and *d*, the number of days of daily sales data, and the number of trailing days respectively.
- The second line contains *n* space-separated non-negative integers where each integer *i* denotes *sales[i]*.

Output Format :

- *int:* the total number of days david will offer free maggi to students.

Constraints :

- 1<=n<=2*10^5
- 1<=d<n
- 1 <= sales[i] <= 10^5

a. Sample input:

8 5
2 3 4 2 1 6 3 6

Sample output:

2

Explanation:

For the first five days, david will not offer any free maggi to students because he didn't have sufficient sales data.

On the sixth day, David has d = 5 days worth of trailing sales data, which is {2, 3, 4, 2, 1}. The median sale of the preceding five days is 2, and he also has the entire sales data from the beginning, which remains {2, 3, 4, 2, 1}, with a median sale value of 2 as well. Since, David's sales on the 6th day is greater than the combined sum, David offers free Maggi to students for the 6th day.

On the seventh day, David has d = 5 days worth of trailing sales data, which is {3, 4, 2, 1, 6}. The median sale of the preceding five days is 3, and he also has the entire sales data from the beginning, which is {2, 3, 4, 2, 1, 6}, with a median sale value of (2 + 3) / 2 = 2.5 . Since, David's sales on the 7th day is less than the combined sum, he will not offer free Maggi to students for the 7th day.

On the eighth day, David has d = 5 days worth of trailing sales data, which is {4, 2, 1, 6, 3}. The median sale of the preceding five days is 3, and he also has the entire sales data from the beginning, which is {2, 3, 4, 2, 1, 6, 3}, with a median sale value of 3 as well. Since, David's sales on the 8th day is equal to the combined sum, David offers free Maggi to students for the 8th day.

Therefore, in total david will offer free maggi for 2 days (6th and 8th day).

Note :

1. You need to implement priority queue for Integer data type only.
2. You need to implement both minimum and maximum priority queue (Implementation details are up to you).

4. Seam Carving [20 + 2(bonus) marks]

Problem Statement:

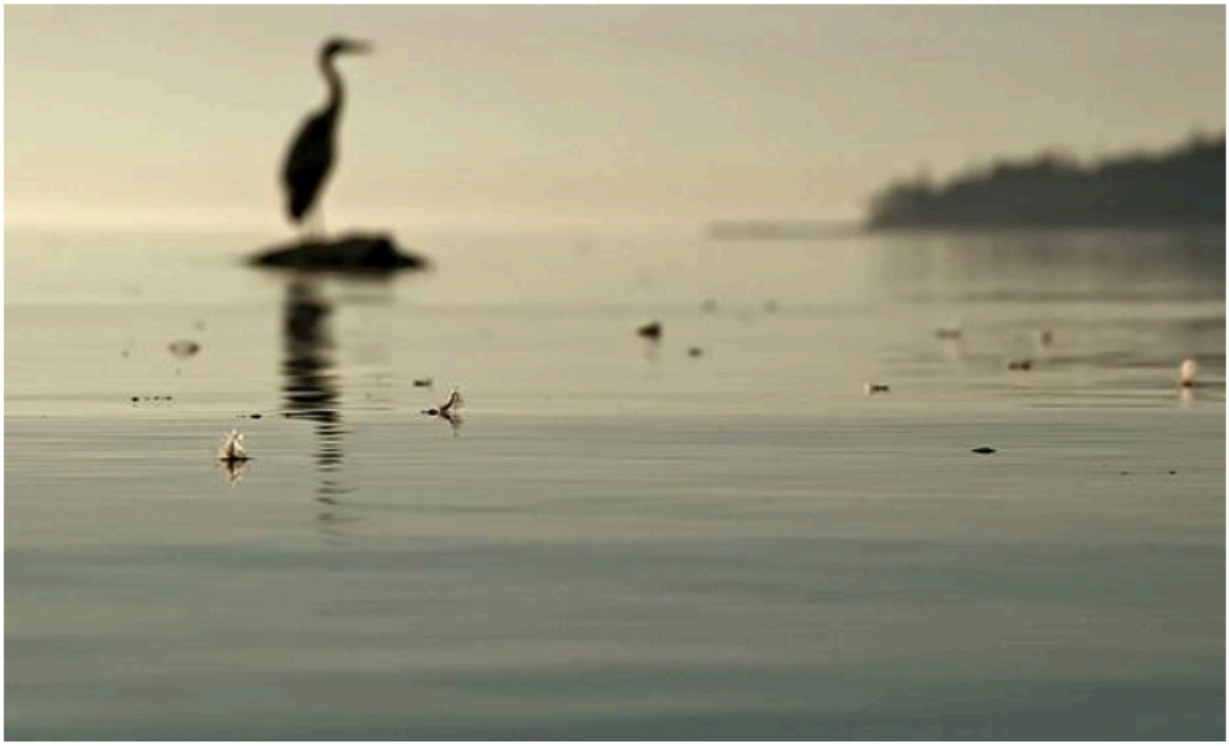
Apply seam carving content aware image-resizing algorithm on a given image. Take the height and width (in pixels) of the output image as inputs from the user.

What is Seam Carving?

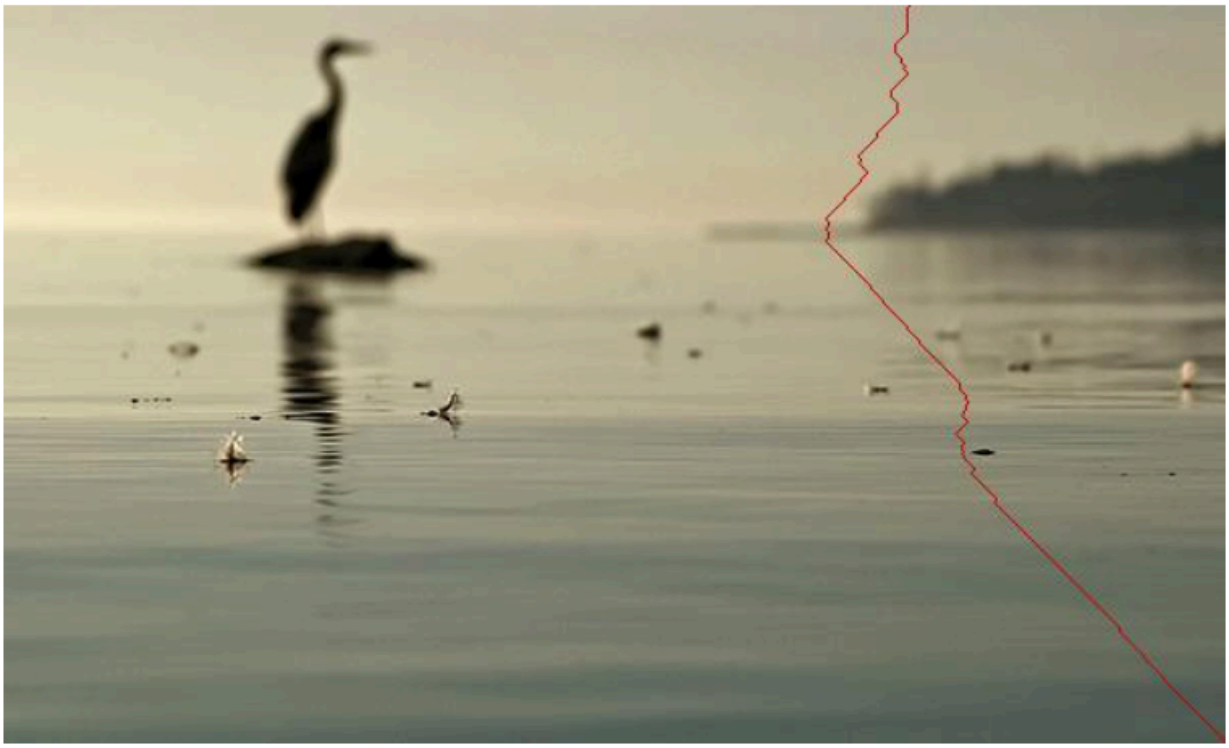
- Seam-carving is a content-aware image resizing technique where the image is reduced in size by one pixel of height (or width) at a time.
- A vertical seam in an image is a path of pixels connected from the top to the bottom with one pixel in each row.
- A horizontal seam is a path of pixels connected from the left to the right with one pixel in each column.

Steps:

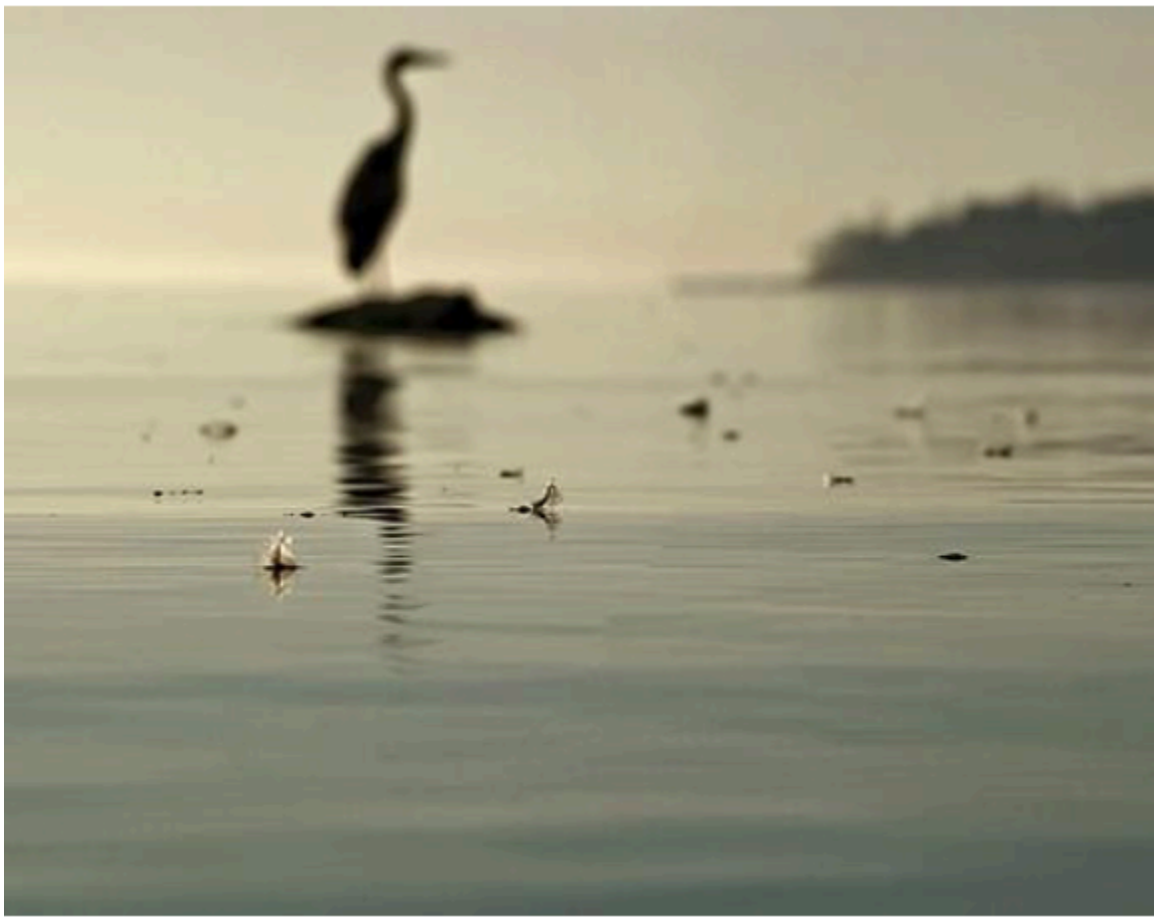
- **Energy Calculation:** Each pixel has some RGB values. Calculate energy for each pixel. For example, you can use a dual-gradient energy function, but you are free to use any energy function of your choice. Also, you can refer to this link for details.
- **Seam Identification:** Identify the lowest energy seam.
- **Seam Removal:** Remove the lowest energy seam.



Original image



Lowest energy seam



Resized image using seam carving

Program Flow:

1. Extract individual pixel's RGB values from the sample image. Load the RGB values in a 3D matrix (Height x Width x 3).
2. Apply seam carving algorithm.
3. Generate sample image output using the RGB values for resized image (New_Height x New_Width x 3).

Submission Format:

- Only C++ is allowed

Evaluation parameters:

Accuracy and Performance of the code.

Note:

1. You should use opencv to extract RGB values of each pixel and to generate images back from RGB values.
2. To install the OpenCV CPP libraries, you can refer to this document.
3. You need to **implement your own** seam carving algorithm.
4. C++ STL is **not allowed** (including vectors, design your own if required).
5. Images are provided in the sample_input folder, alongside assignment pdf.

NOTE: In case of plagiarism in any of the questions, the students involved will be awarded -10 as the final marks of Assignment 1.