CS345 : Algorithms II
Semester I, 2020-21, CSE, IIT Kanpur

Assignment 2

Deadline : 11:55 PM, 24 September 2020.

Most Important guidelines

- It is only through the assignments that one learns the most about the algorithms and data structures. You are advised to refrain from searching for a solution on the net or from a notebook or from other fellow students. Remember - **Before cheating the instructor, you are cheating yourself**. The onus of learning from a course lies first on you. So act wisely while working on this assignment.
- **Grading policy:**
  There will be no penalty for submission based on the time of submission as long as the submission is before the deadline.
- Refrain from collaborating with the students of other groups. If any evidence is found that confirms copying, the penalty will be very harsh. Refer to the website at the link: https://cse.iitk.ac.in/pages/AntiCheatingPolicy.html regarding the departmental policy on cheating.

General guidelines

1. There are three problems in this assignment: Difficult, Moderate, and Easy. The difficult one carries 100 marks, the moderate one carries 80 marks, and the easy one carries 60 marks. Attempt **only** one of them.

2. You are strongly discouraged to submit the scanned copy of a handwritten solution. Instead, you should prepare your answer using any text processing software (LaTex, Microsoft word, ...). The final submission should be a single pdf file.

3. You need to justify any claim that you make during the analysis of the algorithm. But you must be formal, concise, and precise. You may use the results proved in the class. But, if you wish to use any homework problem in your solution, you must provide its solution as well.

4. If you are asked to design an algorithm, you may state the algorithm either in plain English or a pseudocode. But it must be formal, complete, unambiguous, and easy to read. You must not submit any code (in C++ or C, python, ...).

5. **Naming the file**:
   The submission file has to be given a name that reflects the information about the assignment number, version attempted (difficult/moderate/esay), and the roll numbers of the 2 students of the group. For example, you should name the file as **D_2_Rollnumber1_Rollnumber2.pdf** if you are submitting the solution for the difficult problem of the 2nd assignment. In a similar manner, the name should be **M_2_Rollnumber1_Rollnumber2.pdf** and **E_2_Rollnumber1_Rollnumber2.pdf** if you are submitting the solution for the moderate problem and the easy problem respectively of the 2nd assignment.

6. **Only one** student of a group has to upload the final submission. Be careful during the submission of an assignment. Once submitted, it can not be re-submitted.

7. Deadline is strict. Make sure you upload the assignment well in time to avoid last minute rush.

8. Contact TA at the email address: ssnair@iitk.ac.in for all queries related to the submission of this assignment. Avoid sending any such queries to the instructor.

Difficult

Dynamic Sequence with Rotation operation

Design and analyse a data structure for a dynamic sequence S that supports the following operations.

1. Insert(S, i, x): Insert a new element with value x at ith place in the sequence S.
2. Delete(S, j): Delete the element present at jth place in the sequence S.
3. Report(S, i): Report ith element of the sequence S.
4. Rotate(S, i, j): Rotate the sequence S starting from ith element to jth element. In other words, the ith element swaps its position with that of the jth element, $(i+1)$th element swaps its position with that of the $(j-1)$th element, and so on.

You must ensure that each of the 4 operations mentioned above is executed in the worst case $O(\log n)$ time by your data structure, where n is the length of the sequence at the time of the operation.

Moderate

Dynamic Sequence with Min and Add together

Design a data structure for a dynamic sequence S of numbers that supports the following operations.

1. Insert(S, i, c): Insert a number with value c at ith place in the sequence S.
2. Delete(S, j): Delete the number present at jth place in the sequence S.
3. Report(S, i): Report the number present at ith place in the sequence S.
4. Min(S, i, j): Report the smallest number among all the numbers at places $i, i+1, \ldots, j$ in the sequence S.
5. Add(S, i, j, x): Add x to each number at places $i, i+1, \ldots, j$ in the sequence S.

You must ensure that each of the 5 operations mentioned above is executed in the worst case $O(\log n)$ time by your data structure, where n is the length of the sequence at the time of the operation.

Easy

Dynamic Sequence on Bits

Design a data structure for a dynamic sequence S of bits that supports the following operations.

1. Insert(S, i, b): Insert a bit with value b at ith place in the sequence S.
2. Delete(S, j): Delete the bit present at jth place in the sequence S.
3. Report(S, i): Report the bit present at ith place in the sequence S.
4. Flip(S, i, j): For each k such that $i \leq k \leq j$, flip kth bit of the sequence S. That is, if the kth bit was 0 previously, it becomes 1 after this operation; it the kth bit was 1 previously, it becomes 0 after this operation.

You must ensure that each of the 4 operations mentioned above is executed in the worst case $O(\log n)$ time by your data structure, where n is the length of the sequence at the time of the operation.

An important note for all the problems of this assignment:
A solution of each of these problems should clearly provide the following details:

1. The fields and their descriptions at each node of the augmented BST.
2. A neat, complete, and formal description of the implementation of each operation stated in the problem. It is perfectly fine if only the changes in the implementation of each operation (with respect to the implementation described in the class) are described. You are encouraged to describe the implementation (or changes) in plain English as well. But try to be formal, complete, and unambiguous. In case you prefer a pseudocode, make sure it is neat, well commented, and unambiguous.
3. It must be shown that each of the fields can be updated efficiently when we delete or insert any element. For this goal, it is sufficient if a recursive formulation of the field of a node in terms of its own fields as well as the fields of its children is described. It can be easily observed that if the field has such a recursive formulation, then it can be maintained efficiently under rotations as well. So there is no need to provide the details of updating a field of a node involved in a rotation if you are able to show a recursive formulation of a field of a node.

There is no need to provide any analysis of the time complexity for each operation if you are submitting the solution of Moderate problem or Easy problem. But, for the Difficult problem, you must also show formally how your implementation achieves $O(\log n)$ time for the Rotate operation.

Sketch of Solutions and Grading Policy

Difficult

Solution

The basic sketch for Rotating the sequence from index i to j in S has 3 major steps:

1. 1. Split S into 3 sequences: S1 = S[1,i-1], S2 = S[i,j] and S3 = S[j+1,n]. This is possible in O(log n) time. This can be obtained by: S1,S2 = Split(S,i) followed by S2,S3=Split(S2,j+1-i)
2. Obtain the reverse of sequence S2.
3. Use Special Union to join the sequences in order S1 , reverse of S2, S3. This is also possible in O(log n). This can be obtained by: S=SpecialUnion(S1,S2) followed by S= SpecialUnion (S,S3).

Since these 3 operations happen sequentially, the final complexity of rotate operation is the worst time of all 3 processes. So, we need to obtain the reverse of a sequence in O(log n) time, since S2 can very well be the whole sequence S. There are 2 approaches to achieve this complexity

Approach 1: Here, we introduce a new field to the node. This is a Boolean field called isReversed. If this value is set at the root node of a subtree, it means that subtree has been reversed While traversing the tree, we have 2 options:

1. Lazily traverse the reversing. When we encounter a node:
   - If isReversed=False, continue traversal with the normal algorithm discussed in class
   - If isReversed=True, we should interchange the left and right subtrees, toggle their isReversed values to keep the reversing of sequence, and set isReversed=False for current node. Then proceed as per the normal algorithm.
2. Keep a parity of isReversed. Initialise the parity to 0 before starting traversal. At a node:
   - If isReversed=False, continue traversal with the normal algorithm discussed in class
   - If isReversed=True, flip the parity and change the traversal algorithm to go left instead of right and vice versa.

In both these methods, you need to make certain changes toexisting methods:

- Modify Insert, Delete and Report to traverse the tree in the new method.
- In case of Delete of non-leaf node, we need to follow this new traversal also in finding the proper successor.
- Split and SpecialUnion should handle the isReversed field of subtrees: either traverse the isReversed value to child nodes while splitting or keep a parity and update accordingly while leftrotation/rightrotation.

  - A detailed pseudocode or unambiguous description of split is required for Split operation with the handling of new field, including detailed analysis of how it finishes in O(log n) time.
  - For SpecialUnion, description needs only the changes required to accommodate new field, and how it doesn't affect the complexity.

- Rotate will have the following steps:
  1. Use modified Split operation to obtain S1,S2,S3 as discussed above.
  2. Toggle isReversed field of S2.
  3. Use modified specialUnion to obtain S as [S1:S2:S3]

Approach 2:
We maintain 2 trees: S and rev. Now the algorithms are:

- Insert(S,i,c): Insert c to i^{th} position of S using algorithm discussed in class. Also, insert c to $(n+1-i)^{th}$ position of Srev, where n is the new size of S after the insert.
- Delete(S,i): Delete node at i^{th} position of S using algorithm discussed in class. Also delete the node at $(n+1-i)^{th}$ position of Srev, where n is the current size of Srev.
- Report(S,i): Standard report algorithm on S
- Rotate (S,i,j): The algorithm will be as follows:
  1. Split S to S1, S2, S3 as discussed above, using i and j as points for split.
  2. Split Srev to Srev1, Srev2, Srev3 similarly, using (n+1-j-i) and (n+1-i) as points for split.
  3. Perform the standard SpecialUnion to obtain S = [S1:Srev2:S3] and Srev=[Srev1:S2:Srev3]

Complexity Analysis

Approach 1

- In Insert, Delete and Report, you need to visit (and maybe modify the isReversed field of) at most h nodes in the traversal, where h is the height of i^{th} node. In a balanced tree of eight h there will be $2^{(O(h))}$ nodes, ie h=O(log n). So, the complexity of all 3 operations remain O(log n).
- For Split and SpecialUnion, you need to explain why the algorithm needs to visit and modify only h number of nodes, where h is the height of the tree. So, worst time complexity is O(log n). In case of split, a detailed analysis is required to prove it doesn't need O($log^2 n$) time.
- For Rotate, we have 2 sequential Split operations, followed by toggling isReversed field of root node of S2 and then 2 sequential SpecialUnion operations. So, the complexity will be 2.O(log n) + O(1) + 2.O(log n) = O(4.logn + 1) = O(log n)

Approach 2

- Insert, Delete, Report, SpecialUnion, Split has no change from that discussed in class
- For Rotate, we need 4 split operations and 4 specialUnion operations. So, the worst time complexity = 4.O(log n) + 4O(log n) = O(8.log n) = O(log n)

Grading policy

Approach 1

- Clearly define all fields in a node and its use [10]
- *Insert* : Pseudocode or clear definition of all changes from standard Insert discussed in class, properly explaining how the rotate field is handled [10]
- *Delete* : Pseudocode or clear definition of all changes from standard delete discussed in class, properly explaining how the rotate field is handled. Also, explain the difference in deletion of non-leaf node [15]
- *Report* : Pseudocode or clear definition of all changes from standard Report discussed in class, properly explaining how the rotate field is handled [10]
- *Split* :
  - Pseudocode or clear and unambiguous definition of split operation with proof of correctness [10]
  - Detailed complexity analysis on how the algorithm finishes in O(log n) time in worst case [10]
- *Union,Balance,LeftRotation/RightRotation* : Explain the changes in the operations for handling the new field, and explain why the complexity remains the same [15]
- *Rotate* :
  - The algorithm : Split, Rotate and Union [10]
  - Detailed analysis of the operations involved and the worst case complexity [10]

Approach 2

- Clearly define all fields in a node and its use, and use of two trees [10]
- *Insert* : Pseudocode or unambiguous definition of how to insert into both trees. [10]
- *Delete* : Pseudocode or unambiguous definition of how to delete from both trees. [10]
- *Correctness of reverse* : Clear proof of correctness of how the reverse of sequence is stored in second tree, and how to obtain reverse of any subsequence [20]
- *Split* :
  - Pseudocode or detailed and unambiguous explanation of algorithm[10]
  - Detiled Complexity analysis to prove O(log n) complexity[10]
- *Rotate* :
  - The algorithm: Splitting and union in both trees with proof of correctness. [15]
  - Detailed analysis of operations included and worst time complexity[15]