Monotonic Stacks & Queues

Algoday #2

Maggioros Spiros

Χρησιμότητα Stacks & Queues στο cp

- Έχουμε δει την χρησιμότητα τους σε προβλήματα Γράφων ή Δέντρων(BFS/DFS)
- Υπάρχουν όμως και άλλου τύπου προβλήματα που μπορούμε να χρησιμοποιήσουμε στοίβες ή ουρές , όπως dp προβλήματα.

```
void bfs(int start){
    queue<int> q;
    vector<bool> visited(N , false);
    q.push(start);
    visited[start] = true;
    while(!q.empty()){
        auto current = q.front();
        q.pop();
        for(auto & x : adj[current]){
            if(!visited[x]){
                q.push(x);
                visited[x] = true;
            }
        }
    }
}
```

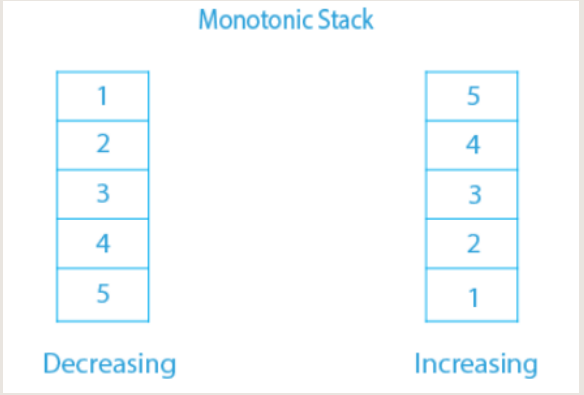
Εύκολος Τρόπος

- Μπορώ απλώς να περάσω τον πίνακα 2 φορές για να βρώ πόσες φορές μια τιμή είναι μικρότερη ή ίση των προηγούμενών τιμών της. Πολυπλοκότητα προφανώς O(n^2).
- Πως μπορούμε να το κάνουμε λίγο καλύτερα?

  Monotonic Stack 1η επιλογή

  Είναι καλύτερο το time complexity? Και

  Γιατι?



Stacks & Queues in DP problems

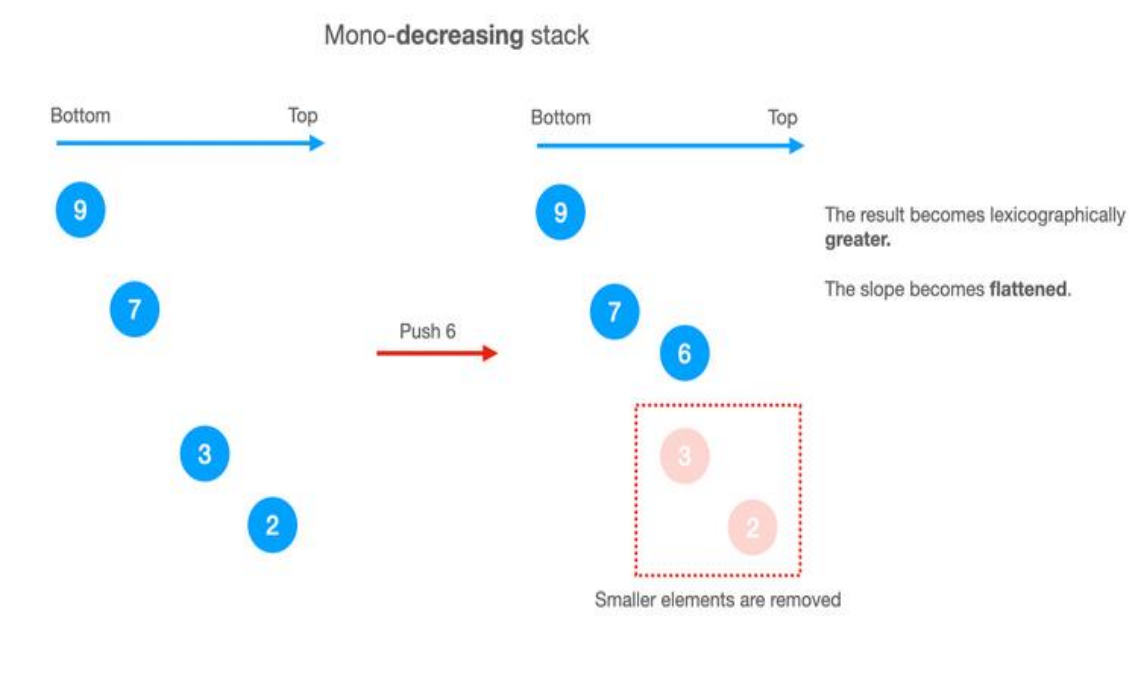
- Π.χ Έστω ότι μας δίνεται ένας πίνακας με τιμές μίας μετοχής,[100 , 80 , 60 , 70 , ...] και το ερώτημα είναι να βρούμε το "span" της μετοχής μια συγκεκριμένη ημέρα(Δηλαδή το μέγιστο αριθμό συνεχόμενων ημερών όπου η τιμή της μετοχής ήταν μικρότερη ή ίση της τιμής εκείνης της μέρας).
- Δηλαδή , για τις τιμές [7,2,1,2] , αν η επόμενη μέρα είχε τιμή 2 , τότε το span θα είναι 4 , γιατι έχουμε 3 τιμές(2,1,2) + 1(σημερινή).

Monotonic Stack: ορισμός

- Υπάρχουν 2 είδη , increasing & decreasing αναλόγως το implementation.
- "σχετικά" καλύτερο time compl.
- Πάλι όμως O(N) time complexity

  Στην χειρότερη περίπτωση , η

  Οποία είναι?



Monotonic Stack στο πρόβλημά μας

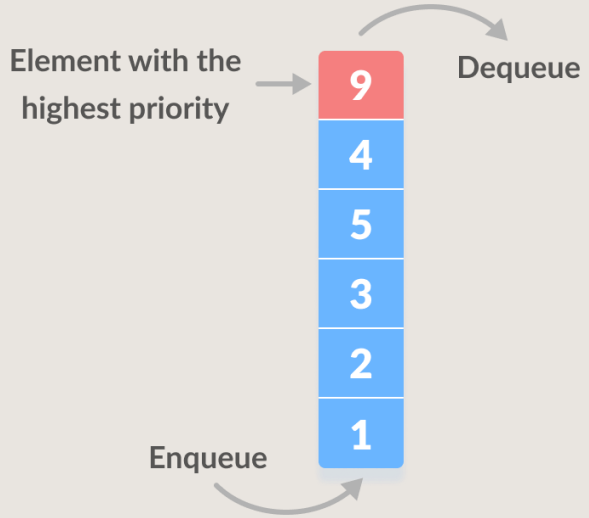
- Η εκφώνηση μπορεί να συντομευτεί σε: Βρες μου την μεγαλύτερη συνεχόμενη αύξουσα ακολουθία από αυτή την μέρα.
- Εξήγηση Βημάτων: Έστω V = [7,2,1,2] , και η σημερινή μέρα έχει τιμή 2.
- Top της στοίβας είναι η προηγούμενη μέρα(2).
- Όσο οι τιμές των ημερών είναι μικρότερες της σημερινής τιμής , pop και αύξησε την τιμή κατά s.top().second.
- Πρόσθεσε την σημερινή τιμή στην στοίβα(push).

Test Cases

- **Input**
- {100, 80, 60, 70, 60, 75, 85}
- **Output**
- [1, 1, 1, 2, 1, 4, 6]

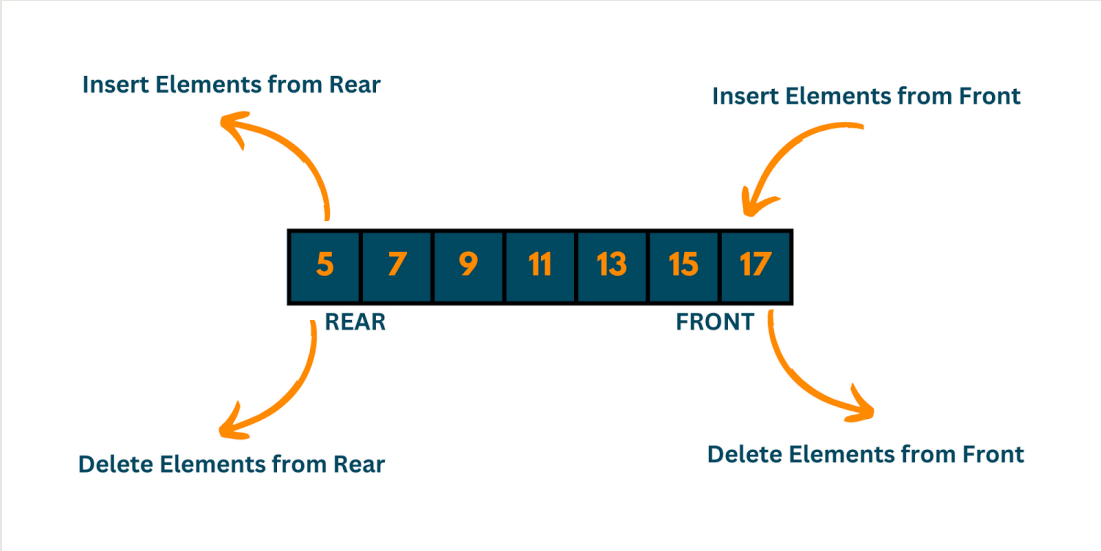
Priority & Monotonic Queues

- Insertion σε O(logn) - για να διατηρείται η μονοτονία.
- Ομοίως μπορώ να τροποποιήσω την ουρά σε αύξουσα ή φθίνουσα.Για min priority queue μπορώ να χρησιμοποιήσω το greater<int>.
- Ίδιες function με την κανονική ουρά(έχω το top , push, pop , κ.ο.κ).
- priority_queue<int> q;



Monotonic Queues

- Συχνά γίνονται implemented με deque<T>(διπλά συνδεδεμένη ουρά).
- Δηλαδή μπορούμε να κάνουμε push_back & push_front και να έχουμε ανά πάσα στιγμή πρόσβαση στο front & back.



Monotonic Queue σε ένα παράδειγμα

- Μας δίνεται σαν input ένας πίνακας , π.χ V = {8,2,4,7} αλλά και ένα limit(int) έστω για το συγκεκριμένο παράδειγμα να είναι limit = 4.
- Μας ζητείται να βρούμε το μεγαλύτερο μή-κενό subarray τ.ω το απόλυτο της διαφοράς του μεγαλύτερου με του μικρότερου στοιχείου του subarray να είναι <= limit.(|max – min| <= limit).
- Για το παράδειγμα μας , η απάντηση είναι 2, μπορούμε να δούμε πως οποιοδήποτε άλλο subarray(μεγέθους > 2) έχει διαφορά > limit = 4.

2 προσεγγίσεις

- Λύση με χρήση Monotonic Queues. Πως το σκέφτηκα?

- Λύση με χρήση Multiset.(Βλ επόμενες διαφάνειες).

Test Cases

- **Input:** nums = [8,2,4,7], limit = 4
- **Output:** 2
- **Input:** nums = [10,1,2,4,7,2], limit = 5
- **Output:** 4
- **Input:** nums = [4,2,2,2,4,4,2,2], limit = 0
- **Output:** 3

Multisets(Άλλος ένας τρόπος)

- Multiset: Implemented σαν Δυαδικό Δέντρο(AVL) στην STL της C++.
- s.find(element): σε O(logn)
- s.erase(element) σε O(logn)
- s.insert(element) σε O(logn)
- Διαφορά με set: επιτρέπει και

Duplicates.

Ερώτηση

Γιατί δεν μας κάνει τίποτα άλλο

Εκτός απο το multiset?