Canadian Computing Competition Senior 2015 - Editorial

Horatiu Stefan Lazu

January 13, 2016

1 CCC 2015 S1 - Zero That Out!

1.1 Explanation

The solution to this problem involves using an abstract data type called a Stack. Stacks are a FILO (First-In-Last-Out) data structure. The methods of interest in a Stack include: push(), pop(), size() and poll(). The push() method adds more numbers to the Stack, pop() removes the top-most item, size() returns the size and poll() returns the top most item but does not remove it. To solve this problem, we simply go through our Stack and push() numbers when the number is not zero, and pop() when it is equal to zero. After we are done, we go linearly through our Stack and add all the sums and return the answer.

1.2 Java Code

```
/* Horatiu Lazu */

import java.io.*;
import java.util.*;
import java.math.*;


class Main{
   public static void main (String [] args){
      new Main();
   }

   public Main(){
      try{
         BufferedReader in;
         in = new BufferedReader (new InputStreamReader (System.in));
         int dictated = Integer.parseInt(in.readLine());
         Stack a = new Stack<Integer>();
         for(int i = 0; i < dictated; i++){
            int num = Integer.parseInt(in.readLine());
            if (num == 0){
               a.pop();
            }
            else{
               a.push(num);
            }
         }
         Integer sum = 0;
         while(!a.isEmpty()){
            sum += (Integer)a.pop();
         }
         System.out.println(sum);
      }
      catch(IOException e){
        System.out.println("IO: General");
      }
    }
  }
```

2 CCC 2015 S2 - Jerseys

2.1 Explanation

This problem consists of efficiently storing the information and ensuring that we can make as many customers satisfied. We begin by declaring an array that will store for every (i+1)th jersey its appropriate size. Let's denote 1 for small, 2 for medium, and 3 for large. In the integer array, the (i)th element will represent the (i+1)th jersey size as an integer. We then go through our requests sequentially, and see if it is possible. We can choose to use a boolean array to see if the jersey was already taken, or we can choose to plurge the other array. We increment our sum variable which is our answer for every possible match. At the end, we simply output our sum variable.

2.2 Java Code

```java
/* Horatiu Lazu */

import java.io.*;
import java.util.*;

public class Main{
   public static void main (String [] args){
      new Main();
   }

   public Main(){
      try{
         BufferedReader in;
         in = new BufferedReader (new InputStreamReader (System.in));
         int numJerseys = Integer.parseInt(in.readLine());
         int customers = Integer.parseInt(in.readLine());
         int [] jersey = new int[numJerseys];
         for(int qq = 0; qq < numJerseys; qq++){
            String input = in.readLine();
            if (input.equals("L")){
               jersey[qq] = 2;
            }
            if (input.equals("M")){
               jersey[qq] = 1;
            }
            if (input.equals("S")){
               jersey[qq] = 0;
            }
         }
         int sum = 0;
         boolean [] used = new boolean[numJerseys];
      for(int i = 0; i < customers; i++){
        StringTokenizer st = new StringTokenizer(in.readLine());
        String sizeS = (st.nextToken());
        int size = -1;;
        if (sizeS.equals("L")){
          size = 2;
        }
        if (sizeS.equals("M")){
          size = 1;
        }
        if (sizeS.equals("S")){
          size = 0;
        }
        int num = Integer.parseInt(st.nextToken());
        if (jersey[num - 1] >= size){
          if (used[num-1] == false){
            sum++;
            used[num-1] = true;
          }
        }
      }
      System.out.println(sum);
    }
    catch(IOException e){
      System.out.println("IO: General");
    }
  }
}
```

3 CCC 2015 S3 - Gates

3.1 Explanation

This problem can be solved in two different methods, one of which uses a greedy method, and another which uses binary trees to enhance the solution.

3.1.1 Solution A: Greedy Method

To solve this problem using greedy, we need to think about the best local decision we can do. Since the planes can only land to gates with a lower number than their number (or equal to), it makes sense to try to dock the plane as close to their maximum dock as possible. As of such, we simply keep a boolean array of the different dock positions (and if they're occupied), and linearly go through and check every position from (i-1) being the highest numbered gate they can reach to zero.

3.1.2 Java Code

```java
/* Horatiu Lazu */

import java.io.*;
import java.util.*;

public class Main{
   public static void main (String [] args){
      new Main();
   }

   public Main(){
      try{
         BufferedReader in;
         in = new BufferedReader (new InputStreamReader (System.in));

      }
      catch(IOException e){
         System.out.println("IO: General");
      }
   }
}
```

3.1.3 Solution B: Binary Trees/Disjoint Sets

Another way of approaching this problem is using Binary Trees/Disjoint Sets. The concept of binary trees is to create a set for every range that is not blocked by an already visited gate. Since finding the head (highest possible gate) takes logarithmic time, then we can reduce the time complexity required to assign the planes

3.1.4 Java Code

```java
/* Horatiu Lazu */

import java.io.*;
import java.util.*;

public class Main{
   public static void main (String [] args){
      new Main();
   }

   public Main(){
      try{
         BufferedReader in;
         in = new BufferedReader (new InputStreamReader (System.in));

      }
      catch(IOException e){
         System.out.println("IO: General");
      }
   }
}
```

4 CCC 2015 S4 - Convex Hull

4.1 Explanation

We identify that this problem is a graph theory question, which has a network consisting of weighed directed edges and every weight affects the hull. The only difference between this and normal shortest path algorithms is that we need to take into account the hull, meaning to take note of the wear. We can solve this problem using Dijkstra's algorithm, which will help find the shortest path, but we will only add nodes to the PriorityQueue if the thickness t is greater than 0.

4.2 Java Code

```java
/* Horatiu Lazu */

import java.io.*;
import java.util.*;
import java.math.*;


class Main{
   public static void main (String [] args){
      new Main();
   }

   public Main(){
      try{
         BufferedReader in;
         in = new BufferedReader (new InputStreamReader (System.in));
         int dictated = Integer.parseInt(in.readLine());
         Stack a = new Stack<Integer>();
         for(int i = 0; i < dictated; i++){
            int num = Integer.parseInt(in.readLine());
            if (num == 0){
               a.pop();
            }
            else{
               a.push(num);
            }
         }
         Integer sum = 0;
         while(!a.isEmpty()){
            sum += (Integer)a.pop();
         }
         System.out.println(sum);

      }
      catch(IOException e){
          System.out.println("IO: General");
        }
      }
    }
```