SRM 149

Monday, June 2, 2003

Match summary

SRM 149 was pretty wild with 100 successful challenges, and only 15 successful solutions to what seemed like a pretty easy division 1 medium. The hard problem also caused people some trouble, since it was somewhat novel. It was also was the first problem to return a double though that probably wasn't much of an issue for anyone. **SnapDragon** ended up winning division 1 by a fairly large margin, despite a resubmission on the hard (which he claims still has a bug which was missed by the system tests), and recaptured the all time highest rating. **tomek** continued to look good in his third match with the 2nd highest score. In division 2, an experienced TopCoder, **kmd-10**, won by a wide margin and moved into division 1.

The Problems

FormatAmt Discuss it

Used as: Division Two - Level One:

Value	250
Submission Rate	174 / 202 (86.14%)
Success Rate	134 / 174 (77.01%)
High Score	**simcoen** for 248.13 points

This was probably one of the harder division 2 easy problems that TopCoder has posed. The easiest way to do it was probably to add digits from right to left. First, add the number of cents to a String. Then, if the number of cents is less than 10, you have to add an extra '0'. Next, you add the decimal point. Now, adding the dollars is probably easiest to do three digits at a time. You can use the modulus operator (num % 1000) to get the last 3 digits, but you have to add leading zeroes if the number is less than 100 and there are more digits left to add. Then, you add a comma, divide the number by 1000, to cut off the last 3 digits, and repeat. Last, add the dollar sign and return. That is the approach I'll show in the reference solution here. Another way to do it is to add the dollars all at once, and then insert the commas afterwards. If you really knew your libraries well, there are formatting functions which can help.

```
String s = "."+(cents<10?"0":"")+cents;
do{
   s = (dollars%1000)+s;
   if(dollars/1000 > 0){
      if(dollars%1000<100)s = "0"+s;
      if(dollars%1000<10)s = "0"+s;
   }
   dollars/=1000;
   if(dollars>0)s=","+s;
}while(dollars>0);
return "$"+s;
```

BigBurger Discuss it

Used as: Division Two - Level Two:

Value	500
Submission Rate	152 / 202 (75.25%)
Success Rate	127 / 152 (83.55%)
High Score	**kmd-10** for 484.20 points

Used as: Division One - Level One:

Value	250
Submission Rate	123 / 126 (97.62%)
Success Rate	115 / 123 (93.50%)
High Score	**radeye** for 248.27 points

This was a pretty simple simulation. Since the input is given in order by arrival time, we can loop through all of the customers and serve them one at a time, keeping track of the current time as we go. When we get to a new person, if the current time is less than or equal to the arrival time, we set the current time to the arrival time. Otherwise, we check to see if the difference between the current time and the arrival time is greater than the maximum wait so far, and if it is, we update the maximum wait time. Then, we increase the current time by the amount of time it takes to prepare he food.

```
int t = 0;
int ret = 0;
for(int i = 0; i < arrival.length; i++){
   if(arrival[i] > t)t=arrival[i];
   ret = Math.max(ret,t-arrival[i]);
   t += service[i];
}
return ret;
```

Pricing Discuss it

Used as: Division Two - Level Three:

Value	1000
Submission Rate	55 / 202 (27.23%)
Success Rate	39 / 55 (70.91%)
High Score	**kmd-10** for 934.30 points

The first thing to observe is that the prices changed to each group should be one of the prices in the input. If it weren't, it would be better to increase it a little, since the same people would still be willing to pay. This suggests that we should pick the four prices for the groups, such that each of them is one of the prices from the input. To handle the case where there are less than 4 groups, we simply allow multiple groups to have the same price. So, to pick the 4 group prices, we just have 4 nested for loops. Then, we go through all of the people, and assign them to the group with the highest price that they are willing to pay. If we sort the input first, it makes things a little simpler and faster:

```
   Arrays.sort(price);
   int ret = 0;
   for(int i = 0; i<price.length; i++){
      for(int j = i; j<price.length; j++){
         for(int k = j; k<price.length; k++){
            for(int m = k; m<price.length; m++){
               int rev = 0;
               for(int n = 0; n<price.length; n++){
                  if(price[n]>=price[m]){
                     rev+=price[m];
                  }else if(price[n]>=price[k]){
                     rev+=price[k];
                  }else if(price[n]>=price[j]){
                     rev+=price[j];
                  }else if(price[n]>=price[i]){
                     rev+=price[i];
                  }
               }
               ret=Math.max(ret,rev);
            }
         }
      }
   }
   return ret;
```

MessageMess

Discuss it

Used as: Division One - Level Two:

Value	250
Submission Rate	117 / 202 (92.86%)
Success Rate	15 / 117 (12.82%)
High Score	**ante** for 458.06 points

This problem tripped a lot of people up, mostly because the examples made it look like a brute force approach might work. But, as experienced TopCoders will tell you, medium level problems usually aren't that simple. To solve this problem in time required a little bit of simple dynamic programming. You should have a String[] to keep track of valid restored substrings. That is, element i of the String[] represents a valid restoration of the characters from 0 to i, if there is one. If there is no valid restoration, or if there is more than one valid restoration, the String[] should have "IMPOSSIBLE!", or "AMBIGUOUS!", respectively. So, how do we do this now? Well, if there is a valid restoration for a substring, then there is a last word in that restoration. So, if we have a valid restoration for characters 0 to i, then we check each word to see if adding that word to the restoration from 0 to i is consistent with the input. If it is, then we update the restoration from 0 to j, where j is the index of the character where the added word ends. Once we have this figured out, its pretty simple to handle the cases where the return is "AMBIGUOUS!" or "IMPOSSIBLE!". I should note that another way to solve this problem is to use memoized recursion, which involves a recursive function that determines whether a substring from character 0 to character i can be restored or not. Here is dgoodman's (the writer) code:

```
        int len = mess.length();
        int[] ways = new int[len+1];
        String[] ans = new String[len+1];
        ways[0]=1;
        ans[0]="";
        for(int i=0; i<len; i++){
            if(ways[i]>0){
                for(int id = 0; id<dic.length; id++){
                    String word = dic[id]; int n = i + word.length();
                    if(n>len) continue;
                    if(!word.equals(mess.substring(i,n))) continue;
                    if(ways[n]>0) ways[n]=2; else ways[n]=ways[i];
                    if(ways[n]==1) ans[n] = ans[i] + " " + word;
                }
            }
        }
        if(ways[len]>1) return "AMBIGUOUS!";
        if(ways[len]<1) return "IMPOSSIBLE!";
        return ans[mess.length()].trim();
```

GForce Discuss it

Used as: Division One - Level Three:

Value	250
Submission Rate	15 / 202 (11.90%)
Success Rate	4 / 15 (26.67%)
High Score	**SnapDragon** for 529.16 points

There are a couple of ways to do this. One of them involves some searching, and the other involves solving some equations explicitly. I'll discuss the searching solution first, and then the faster solution that dispenses with any search.

Before we get into the solution though, lets talk a little about the calculus behind the problem. First, let's define a function avgForce(n) representing the average force starting at n. avgForce(n) is the same as the area under the curve from n to n+period, which is the same as the integral from n to n+period. Now, the integral from n to n+period is the sum of the integrals of all the pieces involved, and each of these integrals is a quadratic, since the integral of a linear equation is a quadratic.

So, one way to solve this problem is to break the function into a number of segments, each of which is continuous (doesn't have any of the points from the input in it) and then search that segment, which is a quadratic, for the maxima. To find the segments, we take all of the points in the input, and all of the point in the input and subtract period from them, and sort them. Each of the intervals between two adjacent points in the sorted list is one of the above segments. To search this segment for a maxima, we can use the trinary search discussed in the round tables a while ago (seen in SnapDragon's solution). There are a number of iterative approaches that would also work, including gradient ascent and Newton's method. Depending on which approach we use, we might also have to check all the endpoints separately. The last task remaining is to write a function to calculate the avgForce(n) function. Since the function isn't continuous, the simplest way to compute it is one piece at a time. We can try to do some calculus to solve this, and it isn't that tricky, but we can use some simple geometry instead. Each section in the integral is a rectangle with a triangle on top of it. So, if we just find the dimensions of the rectangles and triangles the areas follow. Using the slopes of the line segments, this is pretty simple.

Now, the solution without search. All of the previous observations apply, but here we also note that, at a maxima, the derivative (if it is defined) is zero. So, we can find the derivative of the quadratic, and then solve for when it is equal to zero. Finding the quadratic explicitly and solving it is more work than we need though. Instead, we observe that, unless one of the endpoints is one of the points in the input (when the derivative is not defined), the forces at the endpoints are equal. If they weren't we could slide the window one way or the other, and increase the total force. To see this, observe that when sliding the window some small amount, the force changes by that small amount times the difference between the forces. So, given a starting point, we want to slide the window to the point where the two forces are equal. If you work out the math, you'll find that this distance is the difference in the forces divided by the difference in the slopes. Once you have the point where the two forces are equal you check the force this produces, and also check all of windows with an endpoint at one of the points in the input. Here is the writer's solution, which uses this approach:

```
public class GForce{
    int n; int[] f, time; double[] whole,slope;
    //solution interval must either start at start, end at end, or be an
    //interval whose endhts are equal to each other with left slope >= right slope
    public double avgAccel(int period, int[] zf,int[] zt){
        n = zf.length; f=zf; time=zt;
        whole = new double[n];  //[0] is gbg, [1] is 0..1  [n-1] is last segment
        slope = new double[n];
        for(int i=1; i<n; i++) whole[i] = (time[i]-time[i-1])*(f[i]+f[i-1])*.5;
        for(int i=1; i<n; i++) slope[i] = (f[i]-f[i-1])*1.0/(time[i]-time[i-1]);

        int iFin = 1; // segment containing time[n-1]-period
        for(int i=1; time[i]+period < time[n-1]; i++) iFin = i+1;
        double max = area(time[n-1]-period,iFin,time[n-1],n-1);
        for(int i=1; i<n; i++) for(int j=i; j<n; j++){ //seg i to seg j?
            double t1 = Math.max(time[i-1], time[j-1]-period); //start
            double t2 = Math.min(time[i], time[j]-period);    //end
            if(t1>t2) continue;
       double oldmax = max;
            max = Math.max( max, area(t1,i,t1+period,j) );  //equal slopes?
            if(max>oldmax)System.out.println("improve "+t1+" =>"+max);
            max = Math.max( max, eqHt(i,j, t1,t2, period) );
        }
        return max/period;
    }
    double eqHt(int i, int j, double t1, double t2, int period){
        if(slope[i]<= slope[j]) return -1.0;
        double tstar = t1 + (getY(j,t1+period) - getY(i,t1))/(slope[i]-slope[j]);
        if(tstar<t1 || tstar>t2) return -1.0;
        return area(tstar,i,tstar+period,j);
    }
    double getY(int i, double t){ return f[i-1] + slope[i]*(t-time[i-1]); }

    double area(double ti, int i, double tj, int j){//segi, time ti to segj,timetj
        double ai = (time[i]-ti)  *.5*(f[i]  +getY(i,ti));
        double aj = (tj-time[j-1])*.5*(f[j-1]+getY(j,tj));
        double sum = ai+aj; if(i==j) sum -= whole[i];
        for(int k=i+1; k<j; k++) sum += whole[k];
        return sum;
    }
}
```

By **lbackstrom**
TopCoder Member