The problem: Gas Stations

Author: Adrian Panaete

Description of the solution

We notice that a new gas station will be built in the middle of the longest sequence situated between two consecutive gas stations (the house in the middle, if the sequence contains an odd number of houses or the first of the two in the middle if the sequence contains an even number of houses.
To obtain an algorithm in logarithmic time, all the sequences which have, at a certain moment, the maximum length must be processed simultaneously.
We notice that for the construction of any gas station on a sequence of maximum length, the value S will diminish with the same value, so the successive sequence of values S obtained during the process of construction of the gas stations on the sequences with that length will form an arithmetic progression.
It is enough for us to know how many gas stations of each type there are at a certain moment and how many stations of each type there are after all the gas stations have been built on the sequences with the maximum length.
By using this procedure each time we start to build a new set of gas stations, there will be sequences of at most 3 different lengths.
For example, if initially we have N = 57, in time we will obtain the following evolution:
1. One sequence of length 57 transforms in two sequences of length 28
2. The two sequences of length 28 transform in two sequences of length 14 and two of length 13
3. The two sequences of length 14 transform in two sequences of length 7 and in two of length 6 => we have two of length 13, two of 7 and two of 6
4. The two sequences of length 13 transform in 4 of length 6 => we have two of 7 and 6 of 6
5. The two of 7 transform in 4 of 3 => we have 6 of 6, 4 of 3
6. The 6 of 6 transform in 6 of 3 and 6 of 2 => we have 10 of 3, 6 of 2
7. The 10 of 3 transform in 20 of 1 => we have 6 of 2 and 20 of 1
8. The 6 of 2 transform in 6 of 1 => we have 26 of 1
9. The 26 gas stations are built and the whole process is over.

The problem: Tommy's Toy

Authors Cristina Sichim and Adrian Panaete

Description of the solution

We double the sequence and if A<B then we replace A by A+M and we do exactly the same without minding the fact that we have got out of the interval of markings 0 , M-1.
At the end, we are careful to add to each marking i < M the value from i+M.
And we have the answer!
So, we have three stages
Stage 1: at the reading we add 1 and -1 at 4 positions between 0 and 2M-1 (the ones which exceed M-1 are caused by the passage of A in A+M)
Stage 2. We sum partially twice the sequence from 0 to 2M-1
Stage 3. We collect the solution [i + M] to the solution [i] with i from 0 to M-1
The stage 1 has the complexity O(N) and the stages 2 and 3 have complexities O(M)
The final complexity O(N+M) !!!!!

The problem: Options

Author: prof. Marius Nicoli, the Național College "Frații Buzești", Craiova

Description of the solution

A first solution is to calculate the value D[i][j] = the minimum number of steps for reaching the position i,j. The recurrence is D[i][j] = D[i-1][j-1] + D[i-1][j-1] + D[i+1][j-1]. The restriction for the number of columns does not allow this solution to work in the required time (as memory, only the last 2 columns can be used, so there are two vectors). Having the periodical matrix with L rows and K columns, we will have the following pre-calculation:

D[p][i][j] = the minimum number of steps for joining 2^p matrices and for starting from the row i and column 1 of the first one of the p matrices, and for reaching the row j and column k of the last one. For calculating the values D[p][j][j] we use the values D[p-1][i][] and D[p-1][][j], thinking that from the last column of D[p-1][i][] we pass on the first one of D[p-1][][j].

Once calculated the values D, we will first determine M as being the biggest multiple of K smaller than or equal to C and we use the values D[*bit*][][], where *bit* represents positions of a bit of 1 in the writing of M in the basis 2. For the area formed of the

columns from M+1 to C, we can write the dynamics of the first solution.

The problem: KST

Author: Razvan Turturica

Description of the solution

Solution: Adrian Panaete – complexity O(N^2 * logK) 100p

We note d[n][k] = the number of variants in which, having an interval of n consecutive values, we place k of these in the root of a tree and the rest n-k in the sub-trees determined by the k+2 sub-trees KST determined by the fixed values fixate.

OBSERVATIONS:

1. The solution of the problem is d[N][K] where N and K are the read values.
2. If **n>K and k=0** it means, in fact, that they are in a sub-tree that we have just begun to build. The first thing that I have to do is to put K of the n values in the root of the sub-tree. Then, **d[n][0]=d[n][K]**
3. If **n<k** we have too few values to complete the k values which are still necessary in the root, so we will take **d[n][k] = 0;**
4. If **n<K** and **n=k** then we will use all the n values for completing the root, so we will take **d[n][k]=1**

5. If **n<K** and **k=0** then it means that we have just begun to build a new sub-tree KST with fewer than K nodes, so we cannot build more than a leaf, so we will take **d[n][k]=1** The observations treat all the particular cases in which we can say directly how much d[n][k] is.The case that remains is the one in which **n>k and k>0.**

We consider two sub-cases:

1. **The sub-case k = odd**

We fix one of the n values as the middle value between the k values that must be placed in the root. There will remain i smaller values and j larger values than the chosen one and **i+j=n-1.** We must choose k/2 values out of the first i for the root and k/2 values out of the last j for the root. We obtain

d[n][k] = the sum for i+j=n-1 from d[i][k/2]*d[j][k/2]

2. The sub-case k = even

We will fix one of the n values as being the smallest of the k values that we put in the root. There remain i smaller values out of which we do not need to put any one in the root and j larger values out of which we must put k-1 in the root. We obtain

d[n][k] = the sum for i+j=n-1 from d[i][0]*d[i][k-1]

For every fixed k and n the dynamics has the complexity O(n). But n takes all the values from 1 to N, thus another O(N). From an odd k we pass to k/2 and for an even k we pass to k-1 which is odd, therefore we will pass to (k-1)/2. It follows that the number of the values k that we pass through is O(log K).
The final complexity will be O(N * N * log K).

The dynamics works well with memoization, which treats separately the particular cases of n and k and appeals to the necessary values for the general case in dynamics.
The necessary memory would be O(N log K) but for simplifying the implementation, we can declare the matrix of dynamics d[1001][1001] from which we will use N rows and log K columns.

Solution O(N*N*K): In the same way as in the previous solution except for the fact that the passing is made only from k to k-1.

The problem: map

Author: Szabó Zoltan

Description of the solution

The 40-points solution

We use the backtracking technique, trying all the painting options and making sure that two neighbouring countries do not have the same colour.

The 100-points solution

We consider the graph associated to the map, in which the vertices represent countries. There is an edge between two countries if they are neighbours. The maximum degree of a vertex in this graph cannot be higher than 8.

We notice that a vertex with a degree which is smaller than 4 can always be painted irrespective of the colours of its neighbours. Consequently, we will use the following algorithm. All the vertices

with a degree below 4 are introduced into a waiting stack and will be deleted from the graph. Thus, the degrees of other vertices will lower and new vertices will appear with a degree below 4, which we will introduce into the stack in the order of their appearance. This procedure, applied repeatedly, will eliminate all the vertices of the graph.

Finally, the vertices introduced in the stack will be pained in a colour that has not existed till that moment among the colours of the neighbours. This method resembles a topological sorting.

The complexity of the algorithm is O(m*n).

The problem: UpDown

Author: Szabó Zoltan

Description of the solution

For finding the maximal ordered sequence after the first component and decreasingly ordered after the second component, we will sort the sequence in an increasing order after the first component then we will use the algorithm for generating the longest decreasing sub-sequence, after the second component. The sorting can be made with cu complexities O(n) or O(nlogn), the longest increasing sub-sequence is obtained with the complexity O(n logn) or O(n^2). In conclusion, the complexity of the solution is O(n^2) for 75 points and O(nlogn) for 100 points.

The problem: flowers

Author, Gh. Manolache, CNI P. Neamt

Description of the solution

We make 2 DFS for each of the letters of the alphabet. First of all, we will calculate the minimal cost for each sub-tree of the tree, so that if a node is included, all its descendants are included. Secondly, using the values in the first DFS, we will calculate the minimal cost, by taking into consideration each of the nodes as a location which we will bring together. Formally, when we calculate the value for a node x, we will have to take into account the calculated values for each node which is not a descendant of the node x and, also the sum of the quantities.
In conclusion, the minimal solution for a node x is the sum of the solution for each node which is not a descendant of the node x + the value of dp [x].