Solucions del segon concurs classificatori

Olimpíada Informàtica Catalana 2018

Problema 1: Dos quarts de quatre

A dos quarts de quatre de la matinada, la busca dels minuts fa un angle de 180° respecte el zero. La busca de les hores està exactament entre el 3 i el 4. Com que una volta són 360°, cada hora té 30°, de forma que la busca de les hores fa $(3 + \frac{1}{2}) \cdot 30° = 105°$. Per tant, la diferència són 75°.

Problema 2: Unes sumes i restes

Sigui x el primer terme i y el segon. Si fem $z = x + y$, tenim que la suma dels n primers termes és $\frac{nz}{2}$ si n és parell, i $\frac{(n+1)z}{2} - y$ en cas contrari. Pel cas en qüestió la resposta serà $5000z - y$. Altrament es pot fer un codi que simplement iteri i vagi sumant els termes. Podeu trobar-ne un exemple a `pb2.cpp`, on per simplicitat els dos termes es llegeixen en positiu.

Problema 3: Una vint-i-cinquena part

Solució amb programació

Aquesta és potser la solució més senzilla. Cal fer un programa que pot simplement iterar per tots els nombres i comprovar les diferents propietats que ha de complir el nombre demanat per parar al trobar el quart. Podeu mirar `pb3.cpp` per veure un codi que fa això.

Solució matemàtica

Només cal posar l'enunciat com una equació. Sigui d el nombre de dígits de n, llavors ho podem escriure així:

$$n - 6 \cdot 10^{d-1} = \frac{n}{25} \Rightarrow n = \frac{25 \cdot 10^{d-1}}{4}$$

És fàcil veure que 4 divideix $25 \cdot 10^{d-1} \Leftrightarrow d \geq 3$. Així doncs, ho podem reescriure com

$$n = \frac{25 \cdot 100 \cdot 10^i}{4} = 625 \cdot 10^i, \quad i \geq 0$$

Per tant el quart nombre serà per $i = 3$, que ens dóna 625000.

Problema 4: Tres nombres

Solució amb programació

És fàcil fer un programa que provi totes les possibles ternes dintre d'uns intervals. A `pb4.cpp` es fa fins als intervals $[-100, 100]$ i només dóna dues solucions: $(1, 1, 1)$ i $(-2, -2, -2)$. Amb una mica d'intuïció es pot conjecturar que no hi ha més ternes i per tant la resposta és -8.

Solució matemàtica

Plantegem-ho com un sistema:

$$\begin{aligned} x + yz &= 2 \\ y + xz &= 2 \\ z + xy &= 2 \end{aligned}$$

Amb la primera equació obtenim $x = 2 - yz$ i ho substituïm a la segona:

$$y + 2z - yz^2 = 2 \Rightarrow y = \frac{2(1-z)}{1-z^2} = \frac{2}{1+z}$$

Finalment, substituïm x i y a la tercera equació.

$$\begin{aligned} z + \left(2 - \frac{2}{1+z}z\right)\left(\frac{2}{1+z}\right) &= 2 \\ z + \frac{4}{1+z} - \frac{4z}{(1+z)^2} &= 2 \\ z(1+z)^2 + 4(1+z) - 4z &= 2(1+z)^2 \\ z^3 - 3z + 2 &= 0 \\ (z-1)^2(z+2) &= 0 \end{aligned}$$

I amb això obtenim dues ternes: $(1, 1, 1)$ i $(-2, -2, -2)$. Per les divisions que hem fet cal comprovar també les solucions amb $z = \pm 1$, però es pot veure que no hi ha cap més.

Problema 5: Stop

Problema sense massa complicacions, cal fer simplement el que diu l'enunciat. Podeu veure una possible solució a `pb5.py`.

Problema 6: Distància Manhattan

Un altre problema bastant directe. Presenta la dificultat de tractar els -1 sense errors. Podeu trobar la solució oficial a `pb6.py`.

Problema 7: Cercles creixents

Aquest problema presenta dues dificultats: calcular les dimensions de la imatge i pintar els cercles a les coordenades correctes. L'alçada és el màxim dels diàmetres i l'amplada és la suma d'una progressió aritmètica, que es pot calcular amb fórmula o bé amb un bucle. Respecte a les coordenades, es pot fer també amb fórmula o simplement agafant les del cercle anterior i sumant el diàmetre pertinent. Mireu `pb7.py` per veure aquesta explicació implementada.

Problema 8: Regla 30

Distingim dues parts diferenciades al problema, que són calcular quins quadrats s'han de pintar i pintar-los. La segona part no presenta massa dificultat, però la primera és més delicada.

El més fàcil és fer-se una matriu, generar la primera fila i anar-la omplint de dalt a baix, vigilant no sortir-se pels costats. A l'hora de decidir quins quadrats cal pintar cal seguir les restriccions de l'enunciat, que es poden implementar amb uns pocs condicionals. Una opció més simple d'implementar és veure que si considerem els tres quadrats de la fila anterior com a dígits binaris, el següent quadrat s'ha de pintar si el nombre resultant està entre 1 i 4. A `pb8.py` es fa exactament això.

Problema 9: Parells o senars?

Problema molt directe on només cal fer el que diu l'enunciat, podeu veure una solució a `pb9.cpp`.

Problema 10: Mediana de cinc

Hi ha diferents maneres de resoldre aquest problema, la més fàcil és possiblement llegir els nombres en un vector, ordenar-lo i imprimir el del mig, que és el que es fa a `pb10.cpp`.

Problema 11: Llatí dels porcs

És molt útil per aquest problema fer-se una funció que digui si un caràcter és vocal o no. Tenint això és bastant senzill seguir els passos que explica l'enunciat com fem a `pb11.cpp`.

Problema 12: Quantes preguntes?

La resposta és $\lceil \log_2(n) \rceil$. La idea és que cada cop podem preguntar pel nombre del mig i dividir per la meitat l'interval i si es contemplen els casos límits (quan es travessen les potències de dos) es poden quadrar i arribar a la fórmula donada.

Una altra manera és veure n en binari. La fórmula donada equival al nombre de dígits de n en binari i cada pregunta la podem veure com determinar si el primer dígit no conegut és 0 ó 1, preguntant pel nombre just al límit. El codi no té massa misteri, el podeu trobar a `pb12.cpp`.

Problema 13: ABBA

El problema es resol amb un backtracking que generi les 2^n possibles strings de mida n i comprovi per cadascuna si contenen `ABBA`. Podeu veure una possible implementació a `pb13.cpp`.

És important veure que podem mantenir actualitzat durant el backtracking un booleà que ens digui si `ABBA` ha aparegut de moment o no, per no haver-ho de comprovar al final, tal i com es fa al codi donat. Per aquest problema en particular no serveix massa perquè per molt que detectem la seqüència no podem eliminar casos, però la tècnica és molt útil en certs problemes per optimitzar el codi.

Problema 14: Totxanes

Analitzem què passa en posar una totxana de cada manera. Si la posem vertical, hem posat una columna i el problema és el mateix però amb $n-1$. En canvi, posant-la horitzonal obliguem a posar-ne dues més de la mateixa manera i quedem amb el mateix problema amb $n-3$ columnes. Això ens dóna la següent recurrència:

$$f(n) = f(n-1) + f(n-3)$$

Ara només necessitem els casos bases i es pot veure fàcilment que $f(n) = 1$ si $n \leq 2$. Cal programar ara la recurrència amb programació dinàmica per no repetir càlculs i que el programa sigui prou ràpid. Tot i que la recurrència és força senzilla de programar iterativa, a altres problemes resulta més senzill fer-ho de manera recursiva amb una matriu o vector que ens indica si el resultat ja el tenim calculat. Podeu trobar la solució recursiva amb memoization a `pb14.cpp` i la iterativa a `pb14-b.cpp`.

Problema 15: Arxipèlag

Aquest problema es resol amb un recorregut al graf, ja sigui en amplada (BFS) o en profunditat (DFS). La idea és simplement comptar per a cada component del graf quants nodes conté, ja que això és equivalent a les preguntes que ens fan. A `pb15.cpp` es resol fent servir un DFS per trobar els components.