Solucions del primer concurs classificatori

Olimpíada Informàtica Catalana 2019

Problema 1: Nombres divisibles per les unitats

La solució esperada consisteix a fer un petit programa que iteri pels nombres entre 20 i 60 i comprovi quants compleixen la propietat, per acabar trobant que el nombre buscat és 17. Podeu trobar un codi que fa això a `pb1.cpp`. Alternativament, com que l'interval és prou petit, es pot fer a mà amb paper i llapis.

Problema 2: Ternes pitagòriques

La manera més ràpida és provar tots els parells (a, b) fins a un nombre raonable com ara 100, per després comprovar si $a^2 + b^2$ és un quadrat o no. Per aquells parells que ho siguin, guardarem en una estructura tipus `std::map` a C++ o `dict` en Python quantes vegades hem trobat aquella c. Podeu trobar aquesta idea a `pb2.cpp`.

Problema 3: Permutació màxima

Aquest problema té dues parts clares. En primer lloc, cal poder iterar sobre totes les permutacions dels dígits. Tant C++ com Python tenen funcions per fer-ho, de forma que no és necessari picar-les un mateix. A C++ podem fer servir `std::next_permutation` i a Python `itertools.permutations`. Un cop fem això, és necessari saber si un determinat nombre és primer, de forma que necessitarem una funció que ens ho calculi. Podeu veure una implementació a `pb3.cpp`.

Problema 4: Ordenant colors

La única dificultat que planteja el problema és ordenar l'entrada. Un cop fet això, només cal imprimir els quadrats de colors com s'indica. Podeu trobar la solució oficial a `pb4.py`.

Problema 5: Distància Manhattan mínima

Per aquest problema la principal dificultat és implementar la distància Manhattan als punts donats, ja que cal anar amb compte amb els ± 1. A part d'això, només cal implementar el que es demana a l'enunciat, com es fa a `pb5.py`.

Problema 6: Tres en ratlla

Tot i no tenir grans idees al darrere, aquest problema és força pesat de picar i és fàcil fer errors, ja que per generar la imatge final cal provar totes les possibles combinacions on es

pot posar la fitxa i, per cadascuna d'elles, cal mirar si qualsevol de les possibles maneres de fer tres en ratlla amb aquella posició queda complerta. A `pb6.py` trobareu la solució oficial.

Problema 7: Eliminàtoria de futbol

Problema força bàsic on només cal fer el que demanen literalment, tenint cura d'implementar les condicions com s'han descrit. Podeu trobar solucions a `pb7.cpp` i `pb7.py`.

Problema 8: Pingüins

Aquesta mena de problemes es poden complicar molt si no se sap com fer-los, però un cop conegut el truc esdevé força senzill. La principal idea es tenir un vector o llista amb les strings a imprimir a cada línia per poder evitar fer múltiples `if`s i acabar amb un codi molt curt. Res millor que un exemple en aquest cas, feu una ullada a `pb8.cpp` i `pb8.py`.

Problema 9: Seqüència Thue-Morse

Hi ha dues maneres convencionals de resoldre aquest problema. En primer lloc, es pot generar d'un sol cop la seqüència amb longitud 10^5, ja que és la posició més gran que es demanarà, i anar simplement contestant cada número mirant aquesta seqüència pregenerada, tal i com es fa a `pb9.cpp`. Alternativament, podem anar generant cada cop la seqüència fins a la mida demanada, com a `pb9.py`. Sobre quina és més ràpida de les dues, dependrà dels casos donats, tot i que la primera és millor en cas pitjor.

Problema 10: El robot passejador

Per aconseguir els 100 punts en aquest problema es poden fer dues coses. Una possible implementació consisteix en guardar les posicions prohibides en un `std::set` o estructura similar que permeti saber si hi ha una restricció eficientment, com es fa a `pb10-1.cpp`. Una altra possible manera és fer servir una matriu per guardar les posicions prohibides, però fixant-se que només cal que vagi de -100 a 100 a les dues dimensions, ja que el robot no pot arribar més lluny. Aquesta segona opció la podeu trobar a `pb10-2.cpp`.

Problema 11: Bonnie i Clide

Aquest problema es resol amb programació dinàmica. En primer lloc, cal observar que el problema és simètric: si la suma total dels diners és s, és equivalent mirar si es pot sumar $s + k$ amb part dels xecs que mirar si es pot sumar $s - k$. Així doncs, podem pensar una recurrència $f(n, x)$ que ens indiqui si podem sumar x amb els n primers xecs:

$$f(n,k) = \begin{cases} \text{True} & \text{si } k = 0 \\ \text{False} & \text{si } k < 0 \text{ o } (n = 0 \text{ i } k > 0) \\ f(n-1,k) \vee f(n-1,k-v_n) & \text{altrament} \end{cases}$$

Només caldrà saber quin és el major $x \leq s$ tal que la resposta sigui afirmativa usant tots els xecs.

Problema 12: Màxima accessibilitat

En primer lloc, és clau fixar-se que els grafs grans estan generats a l'atzar, en cas contrari la solució que plantejarem no seria prou eficient. Així, si generem els components fortament connexos del graf, obtenint el graf dirigit acíclic associat, reduïm força la mida del graf. Ara podem fer una programació dinàmica que compti des de cada vertex comprimit a quants pot arribar de l'original, o bé simplement fer un DFS des de cada vertex comprimit que no tingui arestes entrants. Podeu trobar una solució a `pb12.cpp`.