Solucions Concurs Classificatori OICat 2021

Olimpíada Informàtica Catalana

Problema C1: Periodisme

Autor: Víctor Martín

Solució 1: A mesura que anem llegint els valors, n'acumulem el màxim i la suma, amb la qual obtenim la mitjana, i comprovem que no superi aquest màxim.

Solució 2: Encara més senzill: és fàcil veure que la resposta és negativa si i només si tots els valors són iguals. Per comprovar-ho, és suficient amb veure que tots són iguals al primer.

Repte: Seríeu capaços de fer una solució d'"una sola línia", és a dir, una solució d'aquesta mena?

```
import sys

for line in sys.stdin:
    print(...)
```

Únicament podeu afegir codi dins del parèntesi de `print()`, i podeu usar els `import`s que volgueu. No és del tot fàcil, i potser us caldrà usar algunes eines de Python (per exemple, les eines de manipulació d'strings). Podeu veure una possible solució d'aquest repte al GitHub, tot i que no és immediata d'entendre.

Problema Q: Nombres alegres

Autor: Xavier Povill

Una manera de resoldre'l és anar provant tots el enters des d'[1] i comprovar que siguin alegres (òbviament, n és alegre si i només si $100n+25$ és un quadrat

[1] 1

perfecte). Per comprovar si un enter m és un quadrat perfecte, una manera simple de fer-ho és comprovar si es compleix $\lfloor\sqrt{m}\rfloor^2 = m$. Com a suggerència extra, es poden accelerar els càlculs començant pel 404 i avançant de 404 en 404, alhora que s'evita comprovar una de les condicions.

Repte: De manera semblant al repte anterior, podríeu fer una solució d'una sola línia (ignorant `imports`)? (I no, no val fer `print` de la resposta.)

Problema C2: Nombres xulos

Autor: Salvador Roura

Per a cada nombre n de l'entrada, mantenim un comptador de quants divisors primers d'un sol dígit (2, 3, 5 i 7) té. Per a cadascun d'aquests primers p, dividim n per p tants cops com puguem (és a dir, sempre i quan aquesta divisió sigui entera), i d'aquesta manera p deixarà de ser un factor primer de n. Si n és xulo, al final haurem comptat tres divisors primers d'un sol dígit, i la n ens haurà quedat sense cap altre factor primer, cosa que implicarà acabar amb $n = 1$.

Problema G1: Rectangle contenidor mínim

Autor: Víctor Martín

Mentre llegim la llista de punts, els anem guardant i anem actualitzant les dimensions del rectangle. Per fer-ho, usem quatre variables (x i y mínimes i màximes del rectangle). Els mínims els podem inicialitzar amb un nombre prou gran (almenys $n - 1$), i els màxims amb un de prou petit (com a màxim 0). Així garantim que en llegir el primer punt el rectangle tindrà dimensions 1×1 i contindrà només aquest primer punt, i ens evitem haver de considerar casos particulars. Un cop tenim les dimensions del rectangle el dibuixem, i a continuació dibuixem tots els punts.

Problema C3: Canvi quasi-mínim

Autor: Salvador Roura

[2] Aquesta nota a peu de pàgina representa un exponent, i també l'aclariment següent: $\lfloor x \rfloor$ representa el màxim enter més petit o igual que x.

De les monedes només ens interessa la suma c dels seus valors. A partir de c volem saber quin seria el canvi mínim. Per aconseguir-ho, fem un bucle començant per la moneda de valor més gran (200 cèntims) i acabant amb la de valor més petit (1 cèntim). Per a cadascuna, n'usem tantes com puguem sense passar-nos de la quantitat que ens queda, i actualitzem aquesta quantitat. Si el canvi mínim usa com a molt una moneda menys que el canvi donat, la resposta serà afirmativa.

Problema C4: Magatzem

Autor: Salvador Roura

Solució 1: Podem resoldre aquest problema amb maps de C++. El més important que cal saber és que l'element mínim d'un map `M` està a `M.begin()`, i l'element màxim a una posició abans que `M.end()` (fer això requereix l'ús d'iteradors).

Solució 2: Si algú no sabés com es pot obtenir el màxim, com a solució alternativa es podrien fer servir dos maps: un de normal per obtenir el mínim, i un amb el signe de tots els elements canviat per obtenir el màxim.

Solució 3: També es podia resoldre el problema usant multisets, seguint esquemes similars a les dues solucions anteriors.

Problema G2: Suma de cubs

Autor: Víctor Martín

Podem resoldre el problema dibuixant franja per franja. A cada iteració, comencem dibuixant el quadrat central, guardant-nos les seves coordenades i el seu color corresponent. Per a cada franja, fem un bucle per dibuixar la resta de quadrats a partir de les coordenades i del color del quadrat central. Un petit truc per evitar dibuixar els rectangles dels extrems de les franges parells és senzillament posar `Blue` com a color de fons de la imatge.

Problema C5: Suma de productes

Autor: Salvador Roura

Sigui L la llista de nombres, i $M_L(i)$ el valor màxim que podem aconseguir agafant tots els elements fins a l'i-èssim (començant a $i = 0$). L'objectiu del

problema és per tant calcular $M_L(n-1)$. Clarament, tenim que si $i \geq 3$:

$$M_L(i) = \max(M_L(i-1) + L_i, M_L(i-2) + L_{i-1}L_i, M_L(i-3) + L_{i-2}L_{i-1}L_i),$$

on L_i és l'i-èssim element de la llista. Podem actualitzar aquests valors $M_L(i)$ amb una programació dinàmica senzilla (calculant els L_i de petit a gran). Per solucionar la resta de casos (és a dir, per a $i < 3$), podem fer dues coses.

Primera opció: Clarament, $M_L(0) = L_0, M_L(1) = \max(L_0 + L_1, L_0L_1)$, i $M_L(2) = \max(M_L(1) + L_2, L_0 + L_1L_2, L_0L_1L_2)$.

Segona opció: Observem que si a qualsevol llista li afegim tants zeros per l'esquerra com vulguem, el resultat final serà exactament el mateix. Per tant, prenem la llista L', que és la llista que obtenim després d'afegir tres zeros a l'esquerra d'L. Així, $M_{L'}(0) = M_{L'}(1) = M_{L'}(2) = 0$, i per a $i \geq 3$, $M_{L'}(i) = M_L(i-3)$ vindrà actualitzada per la fórmula donada anteriorment. Per tant, la solució en aquest cas és $M_{L'}(n+2)$.

Problema C6: Codificació irreversible

Autor: Víctor Martín

Podem solucionar aquest problema amb un backtracking. Començant des de l'inici de la codificació, a cada pas, intentem usar el caràcter següent i avancem una posició. També, si podem, usem els dos caràcters següents: els dos dígits següents han d'existir (és a dir, no ens sortim de la codificació per la dreta), i han de tenir un valor no més gran que 26. Si en un pas el proper dígit és 0, no podem seguir explorant perquè la nostra cerca no ens donaria cap paraula vàlida. Cada cop que arribem al final de la codificació, escrivim la paraula que haguem construït recursivament.

Problema G3: Laberint foradat

Autor: Max Balsells

Solució 1 (BFS): Normalment, quan pensem un "laberint" com un graf, el graf que fem servir té com a nodes el conjunt de posicions del laberint, i com a arestes els parells de posicions adjacents que no són obstacles. Ara bé, per a aquest problema, necessitem alguna cosa una mica més complexa.

Usarem un graf $G = (V, E)$ dirigit, on el conjunt de vèrtexs V seran les ternes (x, y, k), que representaran la posició (x, y) havent gastat k blocs (i per

tant V tindrà $nm(c+1)$ elements). El conjunt d'arestes E serà el següent: Per a tot parell de posicions adjacents que no siguin obstacles (x, y) i (x', y'), tindrem arestes de (x, y, k) a (x', y', k) per a tot $k \in \{0, \dots c\}$ si (x', y') no té un forat, i si en té, tindrem arestes de (x, y, k) a $(x', y', k+1)$ per a tot $k \in \{0, \dots c-1\}$.

Sigui (e_x, e_y) la posició de la casella d'entrada i (s_x, s_y) la de sortida. Volem per tant trobar la distància mínima al graf G des de $(e_x, e_y, 0)$ (la casella d'entrada usant 0 blocs) fins (s_x, s_y, k), per tot k. D'entre tots aquests vèrtexs corresponents a la sortida, a més, volem quedar-nos amb el que tingui menor k, és a dir, el que usi menys blocs.

Podem trobar aquesta distància mínima fent un BFS des de $(e_x, e_y, 0)$, guardant també un pare de cada vèrtex (diem que el pare d'un vèrtex v és un vertex u pel qual hi ha un camí de distància mínima fins v on el penúltim vèrtex d'aquest camí (és a dir, l'anterior a v) és u). L'enunciat del problema assegura que els pares del vèrtexs que estiguin al camí òptim són únics.

D'entre els vèrtexs (s_x, s_y, k) que hàgim aconseguit visitar, agafarem el que tingui menor k, és a dir, el que hagi fet servir menor nombre de blocs. Anomenem k_0 a aquest valor.

Sigui $v_0 = (s_x, s_y, k_0)$. Per un vèrtex u del camí òptim, sigui $p(u)$ el seu pare. Prenem ara $v_1 = p(v_0)$, $v_2 = p(v_1)$, ...

Fent això tindrem $v_d = (e_x, e_y, 0)$, on d és la distància mínima entre $(e_x, e_y, 0)$ i (s_x, s_y, k_0).

Ara $v_d, v_{d_1}, \dots, v_1, v_0$ serà el camí òptim. Recorrent-lo i anant acolorint els forats que ens trobem en l'ordre pertinent solucionarem el problema.

Al GitHub hi trobareu un codi ben explicat. Allà hi podeu veure com implementar el BFS sense haver d'usar el graf G explícitament.

Solució 2 (Dijkstra, basada en la solució d'en Pau Martí Biosca): Considerem el graf habitual representat pel laberint (com hem dit prèviament, el graf que té com a nodes el conjunt de posicions del laberint, i com a arestes els parells de posicions adjacents que no són obstacles). Considerem que és dirigit encara que per a cada aresta (u, v) hi hagi una aresta (v, u).

Volem anar des de l'entrada fins a la sortida minimitzant (k, d), on k és el nombre de blocs usats i d és la distància. Aquí, direm que $(k, d) < (k', d')$ si i només si o bé $k < k'$ o bé $k = k'$ i $d < d'$. En altres paraules, d'entre els camins que minimitzen el nombre de blocs emprats volem minimitzar-ne la distància.

Podem fer això de dues maneres.

Primera opció: Posem els següents pesos a les arestes del graf: Si l'aresta acaba en una casella que no té un forat, el seu pes serà $(0, 1)$, i si acaba en una que en té, el seu pes serà $(1, 1)$. Sumarem aquests pesos de la manera habitual: $(a, b) + (c, d) = (a + c, b + d)$. Per tant, un camí de pes total (k, d) voldrà dir que passa per k forats i que té longitud d.

Segona opció: Seguint l'opció anterior, fem que les arestes tinguin pes 1 en el primer cas, i pes $M + 1$ en el segon. Aquí M serà un enter prou gran com per assegurar que serà major que la distància entre l'entrada i la sortida. (amb $M \geq mn$ és suficient, així que podem agafar $M = 10^4$). Fent això, el pes (k, d) de la opció anterior és equivalent al pes $Mk + d$, i al ser M prou gran, $(k, d) < (k', d')$ resulta equivalent a $Mk + d < Mk' + d'$.

Ara que hem assignat pesos a les arestes i que hem vist que el nostre camí òptim serà el que tingui cost mínim, podem usar l'algoritme de Dijkstra per resoldre el problema.

Usarem una priority queue. Una manera de resoldre el problema, seguint la primera opció, serà utilitzar tuples (k, d, x, y) indicant que podem anar fins (x, y) amb k blocs i distància d (equivalent, seguint la segona opció, tuples (w, x, y), on w serà el cost del camí que portem). Fixeu-vos que posant les tuples en aquest ordre, la priority queue minimitzarà (k, d) en la primera opció (i w en la segona). Per a cada casella, a més, cal que us guardeu el cost mínim amb el que hi podeu arribar, i un pare (en aquest cas, una casella des d'on podeu venir per arribar amb aquest cost mínim).

A partir d'aquí, iniciem el Dijkstra des de la casella d'entrada, on a cada iteració intentem visitar les caselles adjacents que no siguin obstacles i que milloren el cost, actualitzant el cost i el pare de la casella adjacent en qüestió quan ho aconseguim, i parem quan arribem a la casella de sortida.

Un cop hi arribem, similarment a com ho fem a l'altre solució, podem seguir els pares fins arribar a l'entrada per reconstruir el camí de cost mínim, i podem anar recorrent aquest camí alhora que anem dibuixant (i sobreescrivint en el dibuix) les caselles del camí òptim i els forats amb el seu color pertinent.

Us donem també un parell d'alternatives per resoldre el problema seguint l'esquema d'aquesta solució. La primera, implementar la priority queue "a mà" (simplement un array on a cada iteració del Dijkstra hi busqueu l'element mínim i l'esborreu). La segona, guardar els camins mínims parcials (és a dir, que acaben un vèrtex qualsevol) en els estats del Dijkstra, aprofitant que el mapa era petit. Fer les dues coses alhora fàcilment pot comportar un time limit.