DAA-Employment
Problem 1214/F

Abel Molina Sánchez C411

Julio 2022

1 Descripción del problema

Descripción completa: `https://codeforces.com/problemset/problem/1214/F`

Se tienen m ciudades, n puestos de trabajos ubicados en algunas de esas ciudades y n candidatos a ocupar dichos puestos de trabajo, cada uno de los cuales vive en una de las m ciudades. Las ciudades están numeradas de $1 - m$ y tienen disposición circular, la ciudad i es adyacente a la $i + i$ para $i \in 0...m - 1$, la ciudad m es adyacente a 1.
Se quiere ubicar a cada trabajador en un puesto de trabajo de forma tal que se minice la distancia total que separa las casas del trabajo.

Entrada:

Línea 1: m, n $(1 \leq m \leq 10^9, 1 \leq n \leq 200000)$: cantidad de ciudades y cantidad de vacantes.

Línea 2: n enteros $a_1, a_2, a_3, \ldots, a_n (1 \leq a_i \leq m)$: ciudades donde se encuentran las vacantes.

Línea 3: n enteros $b_1, b_2, b_3, \ldots, b_n (1 \leq b_i \leq m)$: ciudades donde viven los candidatos.

Salida:

Línea 1: La distancia total mínima.

Línea 2: n enteros diferentes de 1 a n. El i-ésimo de ellos debería ser el índice de candidato que debería trabajar en el i-ésimo lugar de trabajo.

2 Análisis del problema

Se tiene un problema de optimización, donde se busca minizar un valor, en este caso la distancia total.

Hay que hacer una distribución de los de los candidatos entre las plazas de tal forma que esto ocurra, teniendo en cuenta que la disposición del arreglo de ciudades es circular.

Distancia: Sean x, y dos puntos dentro del arreglo de ciudades, como la distancia es simétrica, digamos $x < y$, la distancia mínima $d(x, y)$ va a estar dada por:

$min(d1, d2)$: $d1 = y - x$ y $d2 = x + m - y$

Sentido: En el círculo de ciudades, vamos a decir que un punto x' está a la derecha de x si la distancia de x' a x es menor por la derecha, viceversa con la izquierda.

3 Enfoques. Soluciones

Fuerza bruta: Como tenemos n candidatos y n plazas ya definidas, la finalidad del problema es buscar la distribución óptima, y esto se puede lograr generando todas las distribuciones de candidatos en vacantes, calculando la distancia total en la misma, y quedándonos con la menor. De esta forma se explora todo el universo de soluciones. Como en este caso, tenemos que son n candidatos distintos a distribuir en n puestos distintos, sin repetición, entonces tenemos que todas las posibles distribuciones son las Permutaciones de n. Generar las $n!$ permutaciones hace que sea solo posible trabajar en

casos con n pequeño(10).
Código: brute_force.py

Idea2: Ordenar las vacantes y candidatos por orden de ciudad, y emparejar a cada candidato con la vacante libre a menor distancia.
En un análisis de casos, esta idea no resultó ser válida para el problema.

Contraejemplo:

ciudades: 1-10
vacante en: 4,4,7,7,9
candidatos en: 2,2,3,5,7,8

Siguiento este cálculo la distancia total daría 11, mientras que en un análisis de la distribución de puntos podemos encontrar rápidamente una cuya distancia total es $9 < 11$ y que no sigue esta estructura.

Solución: Esta solución se basa en la siguiente idea que tiene que ser demostrada: para cada juego de puntos hay una distribución óptima donde los candidatos ubicados en sus puestos de trabajo siguen el orden relativo a su ordenación por el número de ciudad. O sea, que para cada candidato, si su plaza es la plaza i, la del candidato siguiente es la plaza $(i+1)modn$ siguiendo el sentido de ordenación(derecha, arreglo circular). Demostrando esta idea, para obtener el óptimo solo sería necesario encontrar la posición del primer candidato dentro de esta distribución, ya que el resto de candidatos c_i estarán ubicados en las vacantes $v[(i+x)modn]$ siendo x la posición del primero.

Teorema Si se ordenan los arreglos de candidatos y plazas por su número de ciudad, de menor a mayor(sentido derecha), existe una solución óptima donde el orden relativo de los candidatos se mantiene. De forma tal que el $(i+1)modn$ candidato trabaja en la plaza contigua del candidato i en el sentido de ordenación.

Demostración: Sea Opt una distribución óptima de candidatos y plazas, donde tenemos al menos dos candidatos y plazas. Si hubiera solo un par, es directo que tiene el orden relativo consigo mismo. Entonces, supongamos que tenemos al menos dos candidatos tal que no siguen el orden planteado. Si lo siguen, es directo.
Sean x y x' tal que $x' > x$ (a la derecha de x), y que x trabaja en la plaza y y x' en la plaza y' , $y' < y$(a la izquierda de y). En este punto, planteamos la siguiente relación de distancias entre los puntos involucrados.
Sean las distancias:
$d = x <-->x'$, $d > 0$, porque $x' \neq x$
$e = y' <-->y$,
$p = x <-->y$
$q = x' <-->y'$
$p^* = x <-->y'$
$q^* = x' <-->x$

La distancia acumulada que aporta el par al óptimo será $p + q$. Hay que demostrar que en cualquier situación, $p^* + q^* \leq p + q$, donde $p^* + q^*$ será la distancia acumulada que aporta este par en caso de hacer un swap entre los candidatos de y' , y.

Vamos a hacer una casuística sobre los 5 posibles escenarios de distribución de los puntos:

1) $y'..y...x...x'$

$$q = q^* + e \tag{1}$$
$$q^* = q - e \tag{2}$$
$$p^* = p + e \tag{3}$$
$$p^* + q^* = p + q + e - e \tag{4}$$
$$p^* + q^* = p + q \tag{5}$$

2) $y'..x...y...x'$

$$p^* = e - p \tag{6}$$
$$q^* = q - e \tag{7}$$
$$p^* + q^* = q - p + e - e \tag{8}$$
$$p^* + q^* = q - p \tag{9}$$
$$p^* + q^* \leq q \tag{10}$$
$$p^* + q^* \leq q + p \tag{11}$$

3) $x..y'...y...x'$

$$q = q^* + e \tag{12}$$
$$p = p^* + e \tag{13}$$
$$p + q = p^* + q^* + 2e \tag{14}$$
$$p + q - 2e = p^* + q^* \tag{15}$$
$$p + q - 2e \geq p^* + q^* \tag{16}$$
$$p + q \geq p^* + q^* \tag{17}$$

4) $x..y'...x'...y$

$$p = p^* + e \tag{18}$$
$$e = q + q^* \tag{19}$$
$$p + e = p^* + q^* + e + q \tag{20}$$
$$p = p^* + q^* + q \tag{21}$$
$$p - q = p^* + q^* \tag{22}$$
$$p \geq p^* + q^* \tag{23}$$
$$p + q \geq p^* + q^* \tag{24}$$
$$\tag{25}$$

5) $x..x'...y'...y$

$$p = p^* + e \tag{26}$$
$$q = q^* - e \tag{27}$$
$$p + q = p^* + q^* + e - e \tag{28}$$
$$p + q = p^* + q^* \tag{29}$$
$$\tag{30}$$

Si fueran iguales $x = x'$ es directo que un cambio relativo entre sus vacantes no afecta la distancia del óptimo.

Tenemos que en todos los escenarios, hacer un cambio entre los candidatos para y y y', o lo que es lo mismo, colocarlos en el orden relativo, no aumenta la distancia del óptimo, la mantiene, o la disminuye. No se explicita que x, x', y, y' sean contiguos porque están abarcados, pero en el escenario en que no los fueran, entonces, habría al menos un x'' intermedio, que, o bien ya está en el orden relativo con ambos, o se encuentra en estos casos con x o x', haciendo según convenga $x = x''$ o $x' = x''$ se seguen los mismo pasos y se construye el orden planteado.
Con lo cual, existe un óptimo para cada juego de puntos que sigue la distribución planteada.

El algoritmo hace lo siguiente:

```
1. Se ordenan los candidatos y las vacantes por ciudad.
2. Se itera por las vacantes y asigna el primer candidato.
3. Se completan el resto de vacantes calculando la distancia acumulada.
  3.1 Si disminuye la distancia, se actualiza y se guarda la distribución.
4. Devuelve la distancia y la distribución final.
```

Código: solution.py

Demostración de correctitud: El algoritmo haya el óptimo ya que itera por todas las posibles distribuciones de candidatos por vacantes que siguen el orden planteado, quedándose de ellas con la de menor distancia acumulada. Por el teorema anterior se demostró que existe un óptimo que sigue esta distribución, este tiene que ser el de menor distancia acumulada, ya que es óptimo.
Igualmente, podemos ver que, sea *Opt* una distribucón óptima para el problema y sea C la distribución hayada por el algoritmo anterior, si *Opt* es distinto de C, existe una secuencia finita de pasos para convertir *Opt* en C que es la discrita en la demostración anterior.

Complejidad temporal: La ordenación es $O(nlogn)$ y luego iterar por cada una de las vacantes para el primer candidato es $O(n)$ y en cada una se completa la distribución de los n candidatos con lo cual, tenemos $O(n^2)$, quendando la complejidad temporal del algoritmo en $O(n^2)$.

4 Generador y tester

En la carpeta *.test/* se encuentran 2 subcarpetas *in/, out/*, cada una contiene repectivamente, las entradas generadas y las salidas obtenidas por el algoritmo de fuerza bruta.
También se incluyen algunos casos tomados del tester del codeforce, estos se identifican por su nombre: $t < num > cf$.

El generador de casos *test/test_generator.py*, genera juegos de datos de entrada de forma completamente aleatoria, siempre dentro de las restricciones del problema, o permite ingresar cotas manualmente. La cantidad de candidatos está restringida a 10, por el carácter factorial del algoritmo de fuerza bruta con que se generan las soluciones.

El generador coloca los archivos *.in* y *.out* en sus carpetas correspondientes.

Para validar las soluciones de los algoritmos, se utiliza el *checker.py* que da dos opciones para los casos de prueba:
1) Correr un caso específico o
2) correr todos los casos.
La estrategia seguida para validar las soluciones fue la siguiente: Se leen los archivo de entrada y salida del test hecho por el generador y el archivo de solución, se comprueba si el valor dado como óptimo coincide, de hacerlo, se pasa a comprobar la distribución dada. Si esta coincide con la del generador, se acepta, sino, se comprueba que el valor óptimo de esa distribución coincide con el valor óptimo real.

Apuntes

. El script *all_optimal_bf.py* se utilizó para visualizar todas las soluciones óptimas de un problema en busca de similitudes.

. Para el problema abordado hay soluciones que tienen una complejidad temporal inferior, al menos de $O(nlogn)$ basándose en la idea expresada. Algunas están presentes en el codeforce, y aunque las analicé, no logré comprenderlas con la profunidad requerida como para defenderlas como propia.