DAA-Stressful Training
Problem 1132/D

Abel Molina Sánchez C411

Julio 2022

1 Descripción del problema

Descripción completa: `https://codeforces.com/problemset/problem/1132/D`

Problema: n estudiantes con computadoras para un concurso de k minutos de duración. Cada computadora tiene una carga inicial a_i que disminuye cada minuto en una cantidad b_i. Se busca la potencia mínima x de un único cargador que se conectará, máximo a una computadora en cada minuto, incrementando su carga en x unidades. Se busca que ninguna pc tenga carga negativa al principio de un minuto hasta el final de la competencia. En caso de no haber un valor posible para x, devolver -1.

Abstracción: Un experimento de k etapas, con n equipos. Cada equipo con una energía inicial a_i que disminuye en b_i en cada etapa. Determinar, si existe, el menor valor x, tal que, si se utiliza un generador que aumenta en x la energía de a lo sumo un equipo al inicio de una etapa, permita que ningún equipo alcance energía negativa antes del final del experimento.

Entrada:
Línea 1: n y k $1 \leq n \leq 2 * 10^5, 1 \leq k \leq 2 * 10^5$: cantidad de equipos y minutos
Línea 2: n enteros $a_1, a_2, \ldots, a_n$ $1 \leq a_i \leq 10^{12}$: la carga inicial de los equipos
Línea 3: n enteros $b_1, b_2, \ldots, b_n$ $1 \leq b_i \leq 10^7$: el uso de energía de los equipos

Salida:
Línea 1: x tal que $x =$'-1' si no hay solución, o $x \in N$ tal que x es la menor de todas las soluciones posibles.

2 Análisis del problema

El problema consiste, en caso de tener solución, en la búsqueda del valor mínimo que satisfaga las condiciones del mismo. Se puede ver el problema por partes, una, determinar si un deteminado valor es una solución factible, y otra que sea determinar si dicha solución factible es mínima.

Subproblema: Deteminar si x es una solución factible.(Encontrar la potencia de un cargador que garantiza la no descarga de todos los equipos hasta el final.)

Problema: Encontrar la solución factible mínima.

Extraer características del problema
Condición 1: Si en un instante j restando k minutos hay más de k equipos que cumplen que $k > m_i = c_i/b_i$, siendo c_i la carga del equipo i en el instante j, entonces no hay solución.
Demostración: Si un equipo tiene $m_i < k$, entonces, necesita ser cargado antes de que se cumpla el tiempo total. Con lo cual, por el Palomar, si hay más de k equipos por cargarse y k oportunidades de carga, habrá equipos que queden sin cargar, y por tanto, no habrá solución en esa situación.
Condición 2: Para una potencia x, si al comienzo de un minuto j, quedando por lo menos 2 para el final, si hay al menos 2 equipos con $b_i > a_i$, no hay solución.
Demostración: Sean e_1 y e_2 dos equipos tal que $be_1 > ae_1$ y $be_2 > ae_2$. Si en j no se cargan ni e_1 ni e_2, entonces en el siguiente minuto se tiene que $0 > ce_1 = ae_1 - be_1$ y $0 > ce_2 = ae_2 - be_2$ siendo ce_1 y ce_2 las cargas inciales respectivas en dicho minuto ambas negativas, con lo cual, no sería una solución válida. En el otro casos se carga uno de los dos, digamos e_1 sin périda de generalidad. Entonces en el siguiente minuto se tendría $0 > ce_2 = ae_2 - be_2$ un equipo con carga negativa, con lo cual, no sería una respuesta válida.

Condición 3: Para una potencia x, si al comiezo de un minuto, quedando por lo menos 3 para el final, si hay al menos 3 equipos con $b_i \geq a_i$, no hay solución.

Demostración: En el mejor de los casos, tendríamos el menor b_i posible en cada caso, y por tanto $b_i = a_i$. Igualmente, en el mejor escenario se cargaría uno de los equipos en el primer minuto, tendríamos en dicho mejor caso, en el siguiente minutos 2 equipos con $b_i > a'_i = a_i - b_i = 0$, siendo esta la situación vista anteriormente(C1). De la misma forma si no se hubiera elegido para cargar ninguno de los tres en el primer minuto se cumple que hay al menos dos con b_i ¿ a_i(C1).

- Con C1, C2 y C3 tenemos condiciones necesarias para la construcción de una solución factible.

Lema del mínimo: En un minuto j, el equipo más próximo a descargarse es el equipo i, tal que $m_i = a_i/b_i = min\ m_k \forall k$, donde a_i es la carga actual del equipo i en el minuto j.
Demostración: La demostración es directa por razones matemáticas, mientras mayor sea la diferencia del denominador respecto al numerador menor será la razón y en este caso específico, el tiempo que tarde en cubir el primero al segundo.

Lema 1(Final asegurado): En un minuto j arbitrario, si un equipo tiene carga a_i tal que $a_i/b_i >= k - j$, ese equipo nunca se descargará antes del final.
Demostración: Si han transcurrido j minutos de los k totales, entnces, quedan $k - j$ por transcurrir. En cada uno ocurre una descarga, por lo cual habrá una disminución de energía de $(k-j)*b_i$. Luego, el equipo se descargará si $a_i-((k-j)*b_i)$ es menor que 0. Pero como $a_i/b_i >= k - j$, $a_i >= (k - j) * b_i$, con lo cual, no se hará negativo.

Rango de las posibles soluciones: Sea x un valor posible del cargador, $0 \leq x$.(x es no negativo.)
Para buscar una cota superior para x, hay que analizar el posible peor escenario en el que sea posible dar solución.
Este se daría con el equipo más próximo a descargarse en el primer minuto, teniendo la menor carga posible(a_i) y la mayor descarga posible(b_i). Tendríamos $1 - 10^7$, aquí tenemos que con 10^7 garantizamos que este equipo no se descargue, pero habría que cargarlo en cada minuto, propiciendo otras posibles descargas. Luego como la carga de cada equipo no tiene límites, si se busca cargarlo de forma tal que dure directamente hasta el final de la competencia, habrá que multiplicar esta potencia de carga por la cantidad de minutos. En el peor escenario, el concurso dura el máximo posible de minutos, $k = 2 * 10^5$. Entonces, tenemos que si se carga siempre el equipo con $x = 2 * 10^{12}$ se garantiza que no se descargue nunca en el resto del concurso. Luego en un caso donde exista respuesta afirmativa, cualquier otro equipo a cargarse en un minuto posterior, tendrá carga $a_i \geq 0$ y $b_i \leq 10^7$ y $rk < 2 * 10^5$(rk-minutos restantes), con lo cual, $x = 2 * 10^{12}$ será siempre una carga que garantiza la no descarga de todos los equipos siempre que sea posible.
Tenemos que el conjunto de soluciones de ser posibles estas, es: $\{x \in \mathbf{N} | 0 \leq x \leq 2 * 10^{12}\}$.
Pero este rango es independiente de una entrada del problema específica, está centrado en las características generales del problema. Por tanto, este análsis es extrapolable a un caso específico, y de esta forma incluso se pude ampliar y eliminar la restricción al tamaño de entrada especificado. Podemos decir, basado en el análisis anterior que para una instancia específica, la solución va a estar acotada por la cantidad de minutos y el valor máximo de descarga: $k * max(b_i)$.

Lema 2(Propiedad del conjunto de solución del subproblema): Si x es una solución al subproblema: $y > x$ también lo es.
Demostración: Sea x solución afirmativa. Sea $C_x = \{cx_1, cx_2, ..cx_k\}$ una secuencia cualquiera de los valores resultantes de carga en cada minuto, con independencia del equipo. Como x es factible, $cx_i \geq 0\ \forall i$. Luego si se contruye $cy_i = cx_i - x + y$, $cy_i > cx_i \geq 0$ ya que $y - x > 0$. Luego, $y > x$ también es una solución factible.

Por la propiedad del conjunto de soluciones factibles al subproblema podemos ver que el mismo, de forma ordenada, forma un predicado de la forma : "no", "no",..., "no","si","si",....,"si", como respuesta al mismo.

3 Solución:

Primera solución: La primera solución de fuerza bruta, sería, conociendo el rango de definición de la solución y las características de la misma:

```
1. recorrer de forma ordenada el conjunto de posibles soluciones
 1.2 para cada una intentar generar cualquier secuencia válida de operaciones que garanticen la solución.
 1.2.1 si se logra generar una secuencia válida, devolver True, en otro caso False.
2. el primer resultado True, sería la solución, de completarse el recorrido, no habría solución.
```

El hecho de generar una posible secuencia válida de operaciones es hacer las variaciones con repetición de n en k. Donde n es la cantidad de equipos, y k el tiempo que dura el concurso. Hay repetición, ya que un mismo equipo puede cargarse más de una vez durante el concurso. Para mantener la conformación de la variación en un estado posible, hay que comprobar, antes de avanzar, que ningún equipo tiene carga negativa. Sabemos que $VR_{n,k} = n^k$. Tenemos entonces que la solución al subproblema sería $O(n^k)$. Mientras que habría que repetirlo por cada una de los candidatos a solución mientras no haya respuesta afirmativa. Como tenemos que los candidatos están acotados por $k * max(bi))$. Tenemos que la solución de fuerza bruta propuesta al problema $\in O(MAX * n^k)$, donde $MAX = k * max(bi)$

Código: brute_force.py

Podas: Utilizando los condiciones 1 y 2, se puden evitar recursiones innecesarias a la hora de conformar las variaciones, desarrollando solo aquellas que puedan llegar a ser válidas. Aún así no varía la cota del algoritmo.
Código: prune_brute_force.py

Búsqueda Binaria: Teniendo que el conjunto de soluciones posibles al problema conforma un predicado de la forma "no", "no",..., "no","si","si",....,"si" a la pregunta de si x_i es solución factible(Lema 2), y teniendo el conjunto de soluciones acotado inferior y superiormente, es posible utilizar la búsqueda binaria para determinar la respuesta del problema. Para esto, habría que determinar la primera respuesta "si". De no haber, se retorna, "-1".
De esta forma, como los límites del conjunto de solución se puede representar con dos índices y ya está ordenado, realizar la búsqueda binaria va a ser $O(log(MAX))$ donde $MAX = max(b_i) * k$. La solución al problema será $O(log(MAX) * C)$ siendo C la complejidad temporal del algorimto $Check(x)$ para determinar si x respuesta al subproblema.

Algoritmos para $Check(x)$
fuerza bruta: El primer algoritmo para $Check(x)$ sería el visto anteriormente, para un valor x, generar todos las posibles variaciones, siempre que se pueda conformar una variación válida devolver $True$, $False$ en otro caso. $O(n^k)$.
Código: bs_brute_force.py

greedy: La propuesta greedy sería cargar siempre en cada minuto la laptop más próxima a descargarse(o una de ellas), de forma tal que, si se alcanzan los k minutos sin que ninguna llega a tener carga negativa, se devuelve $True$, sino, si alguna alcanza carga negativa se devuelve $False$.

Pseudocódigo:

```
def Check(x, a, k):
    sol = True
    C = {}
    for j in k:
        e = get_min(j)
        if a[e] < 0: # carga de e
            sol = False
        a[e]+=x     #carga e
        C = C + {e}
        update_charge(a,b) #actualiza las cargas
    return sol
```

Ahora hay que demostrar que este enfoque es correcto para determinar una solución al subproblema.

Demostración del greedy: Sea $C = \{c_1, c_2, .., c_k\}$, la secuencia de operaciones de carga obtenida por el greedy. Si el greedy da una respuesta afirmativa, pues, ya se tiene que existe una secuencia de cargas para x para la cual ningún equipo se descarga antes del final, en este caso se tendría C.

Analicemos ahora el caso en el que el greedy da una respuesta negativa, o sea, que para ese valor de carga, no es posible lograr que todos los equipos lleguen hasta el final.

Supongamos que existe alguna secuencia que de respuesta afirmativa al problema. Sea, Af, de todas las posibles secuencias afirmativas, la que tiene un prefijo común mayor con C. $Af \neq C$ porque sino C sería afirmativa. Sea j el primer instante donde difieren C y Af. Por tanto $C_j \neq Af_j$ que significa que en Af se carga un equipo distinto al que se carga en C en el minuto j, sea p el equipo que se carga en C_j.

Existen dos casos, $p \notin Af$ o $p \in Af$. Si $p \notin Af$, significa que, la carga de p es suficiente para acabar la competencia, inclusive en el tiempo j. Pero como el greedy elige al equipo de carga más próxima a agotarse, entonces cualquier equipo en $C_{j'}$ con $j' > j$ tendría también carga para terminar, se cargaría innecesariamente, y por tanto, se tiene que el greedy sería una solución afirmativa si Af es una solución afirmativa, lo cual contradice que el greedy fuera una respuesta negativa.
Si $p \in Af$, p se carga en Af en un tiempo $j' > j$. Sea q el equipo que se carga en Af_j, el momento de descarga de q es mayor o igual al momento de descarga de p, por el modo de selección del greedy. Entonces, si cargar p en el instante j', es una respuesta afirmativa, cargar q en este instante seguiría siendo posible, entonces si hacemos $Af' = Af$ donde $Af'_j = p$ y $Af'_{j'} = q$ es también una solución afirmativa. Luego Af' es una solución afirmativa con prefijo común con C mayor que Af, lo cual es una contradicción con el hecho de que Af fuera la de prefijo común más largo con C.

Luego, si el greedy es una respuesta negativa, no hay respuesta afirmativa.

Una vez demostrada que la estrategia greedy es correcta para el problema, las implementaciones para solucionar el problema son la siguients:

.Analizando el pseudocdigo, vemos como una vez se tiene una respuesta negativa esta no varía, así que, como no es necesario devolver una secuencia, podemos detener el algoritmo en este momento y devolver la respuesta.

Vía 1 $O(kn)$: Recorrer los k minutos y en cada uno seleccionar el mínimo (*Lema del mínimo*). Para seleccionarlo, se recorren los n equipos, $O(n)$. Actualizar la carga de los equipos, haciendo $a_{i,j} = a_{i,j-1} - b_i$ es $O(n)$. Luego, esta implementación del $Check$ sería $O(kn)$. Quedando la solución completa en $O(nk*log(MAX)$, donde $MAX = max(bi)*k$

```
def get_min(j)->int:
    min = k-j  #(lema 1)
    ans = -1
    for i in range(n):
        time_left = a[i]/b[i] # tiempo restante para descargarse
        if time_left < min:
            ans = i
            min = time_left
    return ans

def check(x):
    a_p = [_ for _ in a]
    for j in range(k):
        next = get_min(j)
        if next == -1: # todos tienen carga para terminar (lema 1)
            return True
        elif a_p[next]<0: # el menor ya tenia carga negativa -> no es válido x
            return False
        else:
            a_p[next] += x # se carga el elegido
        for i in range(n): # actualización
            a_p[i]-=b[i]
    return True
```

Código: nk_solution.py

Vía 2 $O(k+n)$: Como los eventos de carga y descarga curren en momentos discretos(minutos $1..k$), es posible determinar el último minuto de carga de cada equipo, el cual viene dado por $m_i = \lfloor \frac{a_i}{b_i} \rfloor$. A su vez, se puede determinar la carga que tendrá en dicho último instante, la cual va a estar dada por $c_i = a_i \% b_i$. Pudiendo computar estos valores, la idea, es reducir el tiempo de ejecución a la hora de obtener el mínimo(próximo equipo a cargar) en un tiempo j. Para esto, se construye un arreglo $next$ de tamaño k y un arreglo c_i de tamaño n, donde $next[i]$ es una lista de los equipos que se descargan definitivamente en el minuto i(o sea, aquellos que $i-1$ es su último instante de carga) y $c_i[e]$ es la carga del equipo e en su último instante de carga. El arreglo $next$ es de tamaño k ya que por el *Lema 1* tenemos que los equipos con $a_i/b_i \geq k$ tienen garantizada la no descarga hasta el final, con lo cual, no es necesario tenerlos en cuenta.

```
def check(x):
    next = [[] for _ in range(k)]
    c_i = [0]*n # c_i guarda la carga del equipo i en el minuto de su descarga
    for i in range(n):
        end_time = a[i]//b[i] + 1 # minuto donde la carga del equipo i se termino
        if end_time < k:
            c_i[i] = a[i]%b[i]
            next[end_time].append(i)
    idx_next = 0 # mantiene la referencia a la primera posicion no vacia en next
    for j in range(k):
        while idx_next < k and len(next[idx_next]) == 0:
            idx_next+=1
        if idx_next <= j: # el ultimo momento de carga del equipo pasó.
            return False
        if idx_next == k: # se llego al final sin nuevas descargas.
            return True
        idx_charger = next[idx_next].pop() # selecciono uno de los mas proximos a descargarse
        c_i[idx_charger]+=x # se carga
```

```
        time_to_end = c_i[idx_charger] // b[idx_charger] # se recalcula el tiempo restante de carga
        c_i[idx_charger] %=  b[idx_charger]
        if idx_next+time_to_end<k: # si supera k, ese equipo ya no se descarga
            next[idx_next+time_to_end].append(idx_charger)
    return True
```

El arreglo *next* se va recorriendo linealmente con un índice de forma tal que este siempre apunte a la primera posición no vacía(al primer instante donde al menos un hay un equipo que se descarga), de esta forma, la operación de obtener el próximo equipo a cargar se puede hacer en $O(1)$ extrayendo del final de $next[indice]$. Luego, recalcular el nuevo tiempo de final para el equipo cargado y su carga en ese momento se hace en $O(1)$, si el tiempo de final supera k, por *lema 1* se tiene que este equipo ya no se descarga por lo cual, no es necesario reasignarlo en una posición dentro de *next*, de otra forma esta reasignación es en $O(1)$(append en una lista.)
La solución del $Check$ queda en $O(n+k)$, y el algoritmo al completo en $O((n+k)log(MAX)$, donde $MAX = max(b_i) * k)$
Código: n_plus_k_solution.py

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.

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

Para validar las soluciones de los algoritmo se utiliza el *checker.py* que permite elegir la implementación a validar y 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 archivos de entrada y salida del test hechos por el generador y para cada uno se computa la solución del algoritmo seleccionado, se comprueba si el valor dado como óptimo coincide.