Técnica de 2 Punteros

Lautaro Lasorsa

BootCamp 2021

Contenidos

Contenidos

Contenidos

Problema

Enunciado

Dado un vector de longitud N que contiene valores de 0 y 1, quiero conocer el rango de mayor longitud tal que la cantidad de 1s en ese rango sea menor o igual a X (X>0)

Problema

Enunciado

Dado un vector de longitud N que contiene valores de 0 y 1, quiero conocer el rango de mayor longitud tal que la cantidad de 1s en ese rango sea menor o igual a X (X>0)

Caso de ejemplo

Vector : 0 0 0 1 0 1 0 1 0 0 0
X = 2
Respuestas posibles: [0, 6], [4, 10] (longitud: 7)

Posibles soluciones

- $O(N^3)$: Probando todos los posibles rangos de forma independiente.

Al final de la presentación se agregan las implementaciones de las soluciones alternativas de este problema.

Posibles soluciones

- $O(N^3)$: Probando todos los posibles rangos de forma independiente.
- $O(N^2)$: Probando todos los posibles rangos pero utilizando un acumulador o una tabla aditiva.

Al final de la presentación se agregan las implementaciones de las soluciones alternativas de este problema.

Posibles soluciones

- $O(N^3)$: Probando todos los posibles rangos de forma independiente.
- $O(N^2)$: Probando todos los posibles rangos pero utilizando un acumulador o una tabla aditiva.
- $O(N * log(N))$: Utilizando una tabla aditiva y búsqueda binaria.

Al final de la presentación se agregan las implementaciones de las soluciones alternativas de este problema.

Posibles soluciones

- $O(N^3)$: Probando todos los posibles rangos de forma independiente.
- $O(N^2)$: Probando todos los posibles rangos pero utilizando un acumulador o una tabla aditiva.
- $O(N * log(N))$: Utilizando una tabla aditiva y búsqueda binaria.
- $O(N)$: Utilizando la técnica de 2 punteros.

Al final de la presentación se agregan las implementaciones de las soluciones alternativas de este problema.

Observaciones Clave

- Si un rango tiene X 1s o menos, cualquier subrango del mismo también cumplirá esta propiedad.

Observaciones Clave

- Si un rango tiene X 1s o menos, cualquier subrango del mismo también cumplirá esta propiedad.
- Como quiero el rango más largo posible, si puedo agrandar un rango sin excederme de la cantidad de 1s permitida siempre nos conviene hacerlo.

Observaciones Clave

- Si un rango tiene X 1s o menos, cualquier subrango del mismo también cumplirá esta propiedad.
- Como quiero el rango más largo posible, si puedo agrandar un rango sin excederme de la cantidad de 1s permitida siempre nos conviene hacerlo.
- En base a las 2 observaciones anteriores, concluyo que solo me interesan los rangos maximales (aquellos que no puedo agrandar sin excederme de la cantidad de 1s permitida)

Contenidos

Introducción

Técnica

La técnica de los 2 punteros consiste en tener, justamente, 2 índices o punteros que van a ir avanzando y recorriendo el vector. En este caso lo que vamos a representar con ellos es el rango que estamos considerando actualmente.

Introducción

Técnica

La técnica de los 2 punteros consiste en tener, justamente, 2 índices o punteros que van a ir avanzando y recorriendo el vector. En este caso lo que vamos a representar con ellos es el rango que estamos considerando actualmente.

Convención

Para aplicar está técnica en este caso la mejor convención a adoptar es la [l,r). Es decir, que los índices l y r describen un rango que empieza en el elemento l e incluye a ese elemento y termina en el elemento r pero no incluye ese elemento. Notar que si l=r el rango es vacío, y que la longitud del rango es simplemente r-l.

Desarrollo (1)

Situación Inicial

Inicialmente vamos a considerar el rango vacío que empieza en 0, es decir l = r = 0. Antes de seguir, notemos que pedir que haya X 1s o menos es igual a pedir que la suma de los valores del rango sea igual o menor a X. Además, tendre un contador de la suma en el rango actual, cont, y uno para almacenar la respuesta, res, y ambos inician en 0. Puedo, además, guarda el rango de esa longitud en contadores lres y rres.

Desarrollo (2)

Iteraciones

En cada paso, voy a considerar si cont+v[r]>X. Si es el caso, haré cont = cont-v[l], l++. Sino, haré cont = cont+v[r],r++. Posteriormente, consideraré este rango actual y si r-l>res, haré res = r-l, lres = l, rres = r.

Desarrollo (2)

Iteraciones

En cada paso, voy a considerar si cont+v[r]>X. Si es el caso, haré cont = cont-v[l], l++. Sino, haré cont = cont+v[r],r++. Posteriormente, consideraré este rango actual y si r-l>res, haré res = r-l, lres = l, rres = r.

Finalización

El algoritmo termina cuando r = N, y entonces la respuesta final es el contenido de res, lres y rres.

Implementación

Implementación de la solución en C++

```
1  void Problema(vector<int> & v, int X, int & res, int & lres,
       int & rres){
2      res = 0, lres = -1, rres = -1;
3      int l = 0, r = 0, cont = 0, N = int(v.size());
4      while(r<N){
5          if(v[r]+cont>X) cont-=v[l],l++;
6          else{
7              cont+=v[r],r++;
8              if(r-l>res) res = r-l, lres = l, rres = r;
9          }
10     }
11 }
```

Caso borde

Pregunta

Originalmente el problema lo planteamos con X>0, es decir que un rango de un solo elemento siempre cumplía la restricción. Pero, ¿y si X = 0?.

Caso borde

Pregunta

Originalmente el problema lo planteamos con X>0, es decir que un rango de un solo elemento siempre cumplía la restricción. Pero, ¿y si X = 0?.

Respuesta

La idea e implementación antes expuestas resuelven también este caso, dado que si v[i] = 1, y estamos en la situación l = r = i, cont = 0, se dará la siguiente secuencia.

Se pudo avanzar a la posición i+1 sin considerar en ningún momento el rango que contenía a la posición i.

Caso borde

Pregunta

Originalmente el problema lo planteamos con X>0, es decir que un rango de un solo elemento siempre cumplía la restricción. Pero, ¿y si X = 0?.

Respuesta

La idea e implementación antes expuestas resuelven también este caso, dado que si v[i] = 1, y estamos en la situación l = r = i, cont = 0, se dará la siguiente secuencia.

1. l = i, r = i, cont = 0

Se pudo avanzar a la posición i+1 sin considerar en ningún momento el rango que contenía a la posición i.

Caso borde

Pregunta

Originalmente el problema lo planteamos con X>0, es decir que un rango de un solo elemento siempre cumplía la restricción. Pero, ¿y si X = 0?.

Respuesta

La idea e implementación antes expuestas resuelven también este caso, dado que si v[i] = 1, y estamos en la situación l = r = i, cont = 0, se dará la siguiente secuencia.

1. l = i, r = i, cont = 0
2. l = i+1, r = i, cont = -1

Se pudo avanzar a la posición i+1 sin considerar en ningún momento el rango que contenía a la posición i.

Caso borde

Pregunta

Originalmente el problema lo planteamos con X>0, es decir que un rango de un solo elemento siempre cumplía la restricción. Pero, ¿y si X = 0?.

Respuesta

La idea e implementación antes expuestas resuelven también este caso, dado que si v[i] = 1, y estamos en la situación l = r = i, cont = 0, se dará la siguiente secuencia.

1. l = i, r = i, cont = 0
2. l = i+1, r = i, cont = -1
3. l = i+1, r = i+1, cont = 0

Se pudo avanzar a la posición i+1 sin considerar en ningún momento el rango que contenía a la posición i.

Para practicar

Notar que este problema es el mismo que el problema *Días Feriados* del Certamen Provincial de la OIA 2017 tomado en el Nivel 2.
Pueden acceder al mismo y enviar sus soluciones en:
`http://juez.oia.unsam.edu.ar/#/task/feriados/statement`

Contenidos

Contenidos

Objetivo

El objetivo de esta sección es plantear qué forma tienen en general los problemas que podemos resolver con la técnica de 2 punteros.

Características

- Tenemos un vector V, y una condición P(V,L,R) que puede o no ser cumplida por el rango [L,R) del vector V

Características

- Tenemos un vector V, y una condición P(V,L,R) que puede o no ser cumplida por el rango [L,R) del vector V
- Si P(V,L,R) se cumple, se cumplen también P(V,L+1,R) y P(V,L,R-1). Es decir, cualquier subrango de [L,R) cumple P.

Características

- Tenemos un vector V, y una condición P(V,L,R) que puede o no ser cumplida por el rango [L,R) del vector V
- Si P(V,L,R) se cumple, se cumplen también P(V,L+1,R) y P(V,L,R-1). Es decir, cualquier subrango de [L,R) cumple P.
- Por tanto, podemos hablar de los rangos maximales que cumplen P, rangos [L,R) tales que se cumple P(V,L,R) pero no P(V,L-1,R) ni P(V,L,R+1), y puntualmente de rangos maximales por izquierda, tales que vale P(V,L,R) pero no P(V,L-1,R).

Características

- Tenemos un vector V, y una condición P(V,L,R) que puede o no ser cumplida por el rango [L,R) del vector V
- Si P(V,L,R) se cumple, se cumplen también P(V,L+1,R) y P(V,L,R-1). Es decir, cualquier subrango de [L,R) cumple P.
- Por tanto, podemos hablar de los rangos maximales que cumplen P, rangos [L,R) tales que se cumple P(V,L,R) pero no P(V,L-1,R) ni P(V,L,R+1), y puntualmente de rangos maximales por izquierda, tales que vale P(V,L,R) pero no P(V,L-1,R).
- Tenemos una operación Op(V,L,R) que queremos aplicarla a todos los rangos maximales por izquierda para obtener la respuesta.

Aplicar al problema ya visto

- El vector lo da el enunciado, y nuestra condición P(V,L,R) es que la cantidad de 1s en [L,R) sea menor o igual que X.

Aplicar al problema ya visto

- El vector lo da el enunciado, y nuestra condición P(V,L,R) es que la cantidad de 1s en [L,R) sea menor o igual que X.
- Lógicamente, si hay hasta X 1s en [L,R), un subrango no va a tener más 1s.

Aplicar al problema ya visto

- El vector lo da el enunciado, y nuestra condición P(V,L,R) es que la cantidad de 1s en [L,R) sea menor o igual que X.
- Lógicamente, si hay hasta X 1s en [L,R), un subrango no va a tener más 1s.
- En este caso la operación Op(V,L,R) lo que hace es:

```
1 if(r-l>res) res = r-l, lres = l, rres = r;
```

Un estado fácil de actualizar

Algo importante a tener en cuenta es que durante la ejecución del algoritmo vamos actualizando un estado que describe el rango [L,R) que estamos viendo actualmente. Con este estado queremos hacer 4 operaciones:

- Cuando avanzamos R, debemos ser capaces de agregar un elemento al final.

Cada una de estas 4 operaciones puede realizarse hasta N veces, por lo cuál la complejidad de las mismas determina la complejidad total del programa.

Un estado fácil de actualizar

Algo importante a tener en cuenta es que durante la ejecución del algoritmo vamos actualizando un estado que describe el rango [L,R) que estamos viendo actualmente. Con este estado queremos hacer 4 operaciones:

- Cuando avanzamos R, debemos ser capaces de agregar un elemento al final.
- Cuando avanzamos L, debemos ser capaces de eliminar un elemento del principio.

Cada una de estas 4 operaciones puede realizarse hasta N veces, por lo cuál la complejidad de las mismas determina la complejidad total del programa.

Un estado fácil de actualizar

Algo importante a tener en cuenta es que durante la ejecución del algoritmo vamos actualizando un estado que describe el rango [L,R) que estamos viendo actualmente. Con este estado queremos hacer 4 operaciones:

- Cuando avanzamos R, debemos ser capaces de agregar un elemento al final.
- Cuando avanzamos L, debemos ser capaces de eliminar un elemento del principio.
- Ver si se cumple la condición P para dicho rango.

Cada una de estas 4 operaciones puede realizarse hasta N veces, por lo cuál la complejidad de las mismas determina la complejidad total del programa.

Un estado fácil de actualizar

Algo importante a tener en cuenta es que durante la ejecución del algoritmo vamos actualizando un estado que describe el rango [L,R) que estamos viendo actualmente. Con este estado queremos hacer 4 operaciones:

- Cuando avanzamos R, debemos ser capaces de agregar un elemento al final.
- Cuando avanzamos L, debemos ser capaces de eliminar un elemento del principio.
- Ver si se cumple la condición P para dicho rango.
- Aplicar la operación Op al rango [L,R)

Cada una de estas 4 operaciones puede realizarse hasta N veces, por lo cuál la complejidad de las mismas determina la complejidad total del programa.

El estado en este problema

En nuestro problema nuestro estado es muy simple, es un entero cont que cuenta la cantidad de 1s en el rango, o lo que es lo mismo la suma del rango. Ahora, veamos como hacemos cada una de las operaciones:

- Avanzar R es simple, hacemos cont+=v[r].

El estado en este problema

En nuestro problema nuestro estado es muy simple, es un entero cont que cuenta la cantidad de 1s en el rango, o lo que es lo mismo la suma del rango. Ahora, veamos como hacemos cada una de las operaciones:

- Avanzar R es simple, hacemos cont+=v[r].
- Avanzar L es igual de simple, hacemos cont-=v[l];

El estado en este problema

En nuestro problema nuestro estado es muy simple, es un entero cont que cuenta la cantidad de 1s en el rango, o lo que es lo mismo la suma del rango. Ahora, veamos como hacemos cada una de las operaciones:

- Avanzar R es simple, hacemos cont+=v[r].
- Avanzar L es igual de simple, hacemos cont-=v[l];
- Ver si se cumple P es sencillo también, es cont<=X. Como en el código queremos verlo antes de avanzar R, vemos si cont+v[r]>X (y en ese caso, como no se cumple P, avanzamos L)

El estado en este problema

En nuestro problema nuestro estado es muy simple, es un entero cont que cuenta la cantidad de 1s en el rango, o lo que es lo mismo la suma del rango. Ahora, veamos como hacemos cada una de las operaciones:

- Avanzar R es simple, hacemos cont+=v[r].
- Avanzar L es igual de simple, hacemos cont-=v[l];
- Ver si se cumple P es sencillo también, es cont<=X. Como en el código queremos verlo antes de avanzar R, vemos si cont+v[r]>X (y en ese caso, como no se cumple P, avanzamos L)
- En este problema el Op es O(1) dado que es tomar el más largo entre 2 rangos (la respuesta hasta ahora y el que estamos procesando actualmente)

Contenidos

Contenidos

Unstable Strings

Enunciado

Dado un string s formado por 0, 1 y ?, me interesa contar la cantidad de substrings (rangos del string) en los que puedo cambiar los ?s por 0s o 1s (de forma independiente cada ?) de tal forma que la secuencia resultante sea alternada (es decir: 0101010... o 1010101...).
$1 \leq |s| \leq 200{,}000$

Para practicar:
`https://codeforces.com/problemset/problem/1535/C`

Unstable Strings

Enunciado

Dado un string s formado por 0, 1 y ?, me interesa contar la cantidad de substrings (rangos del string) en los que puedo cambiar los ?s por 0s o 1s (de forma independiente cada ?) de tal forma que la secuencia resultante sea alternada (es decir: 0101010... o 1010101...).
$1 \leq |s| \leq 200{,}000$

Ejemplos

Para practicar:
`https://codeforces.com/problemset/problem/1535/C`

Unstable Strings

Enunciado

Dado un string s formado por 0, 1 y ?, me interesa contar la cantidad de substrings (rangos del string) en los que puedo cambiar los ?s por 0s o 1s (de forma independiente cada ?) de tal forma que la secuencia resultante sea alternada (es decir: 0101010... o 1010101...).
$1 \leq |s| \leq 200{,}000$

Ejemplos

- Si s = 0?10, res = 8

Para practicar:
`https://codeforces.com/problemset/problem/1535/C`

Unstable Strings

Enunciado

Dado un string s formado por 0, 1 y ?, me interesa contar la cantidad de substrings (rangos del string) en los que puedo cambiar los ?s por 0s o 1s (de forma independiente cada ?) de tal forma que la secuencia resultante sea alternada (es decir: 0101010... o 1010101...).
$1 \leq |s| \leq 200{,}000$

Ejemplos

- Si s = 0?10, res = 8
- Si s = ???, res = 6

Para practicar:
`https://codeforces.com/problemset/problem/1535/C`

Unstable Strings

Enunciado

Dado un string s formado por 0, 1 y ?, me interesa contar la cantidad de substrings (rangos del string) en los que puedo cambiar los ?s por 0s o 1s (de forma independiente cada ?) de tal forma que la secuencia resultante sea alternada (es decir: 0101010... o 1010101...).
$1 \leq |s| \leq 200{,}000$

Ejemplos

- Si s = 0?10, res = 8
- Si s = ???, res = 6
- Si s = ?10??1100, res = 25

Para practicar:
`https://codeforces.com/problemset/problem/1535/C`

Observaciones clave: Rangos

- Si en un rango hay un 0 en una posición par y uno en una posición impar, el rango no puede ser válido (idem con los 1s)

Observaciones clave: Rangos

- Si en un rango hay un 0 en una posición par y uno en una posición impar, el rango no puede ser válido (idem con los 1s)
- Si en un rango hay un 0 en una posición par y un 1 en otra posición par (o un 0 en una impar y un 1 en otra impar) el rango no puede ser válido.

Observaciones clave: Rangos

- Si en un rango hay un 0 en una posición par y uno en una posición impar, el rango no puede ser válido (idem con los 1s)
- Si en un rango hay un 0 en una posición par y un 1 en otra posición par (o un 0 en una impar y un 1 en otra impar) el rango no puede ser válido.
- Por lo visto anteriormente, si un rango es invalido cualquiera que lo contenga lo es también.

Observaciones clave: Rangos

- Si en un rango hay un 0 en una posición par y uno en una posición impar, el rango no puede ser válido (idem con los 1s)
- Si en un rango hay un 0 en una posición par y un 1 en otra posición par (o un 0 en una impar y un 1 en otra impar) el rango no puede ser válido.
- Por lo visto anteriormente, si un rango es invalido cualquiera que lo contenga lo es también.
- Si no se dan ninguna de las 2 condiciones descritas anteriormente, el rango puede ser válido. Puntualmente, no nos interesan las posiciones de los ?s.

Observaciones clave: Estado

- Para ver que se cumplan las 2 condiciones anteriores, alcanzar con guardar la cantidad de 0s en posiciones pares, la cantidad de 0s en posiciones impares, la cantidad de 1s en posiciones pares y la cantidad de 1s en posiciones impares.

Observaciones clave: Estado

- Para ver que se cumplan las 2 condiciones anteriores, alcanzar con guardar la cantidad de 0s en posiciones pares, la cantidad de 0s en posiciones impares, la cantidad de 1s en posiciones pares y la cantidad de 1s en posiciones impares.
- Al avanzar R, es decir al agregar un elemento, solo necesito aumentar en 1 el valor que corresponda (o ninguno si es un ?)

Observaciones clave: Estado

- Para ver que se cumplan las 2 condiciones anteriores, alcanzar con guardar la cantidad de 0s en posiciones pares, la cantidad de 0s en posiciones impares, la cantidad de 1s en posiciones pares y la cantidad de 1s en posiciones impares.
- Al avanzar R, es decir al agregar un elemento, solo necesito aumentar en 1 el valor que corresponda (o ninguno si es un ?)
- Al avanzar L, es decir al sacar un elemento, debo reducir en 1 el valor que corresponda (o ninguno si es un ?)

Observaciones clave: Estado

- Para ver que se cumplan las 2 condiciones anteriores, alcanzar con guardar la cantidad de 0s en posiciones pares, la cantidad de 0s en posiciones impares, la cantidad de 1s en posiciones pares y la cantidad de 1s en posiciones impares.
- Al avanzar R, es decir al agregar un elemento, solo necesito aumentar en 1 el valor que corresponda (o ninguno si es un ?)
- Al avanzar L, es decir al sacar un elemento, debo reducir en 1 el valor que corresponda (o ninguno si es un ?)
- P se cumple si y solo si:
  $(0_{par} == 0 \ o \ 0_{impar} == 0) \ y \ (1_{par} == 0 \ o \ 1_{impar} == 0) \ y \ (0_{par} == 0 \ o \ 1_{par} == 0) \ y \ (0_{impar} == 0 \ o \ 1_{impar} == 0)$

Observaciones clave: Estado

- Para ver que se cumplan las 2 condiciones anteriores, alcanzar con guardar la cantidad de 0s en posiciones pares, la cantidad de 0s en posiciones impares, la cantidad de 1s en posiciones pares y la cantidad de 1s en posiciones impares.
- Al avanzar R, es decir al agregar un elemento, solo necesito aumentar en 1 el valor que corresponda (o ninguno si es un ?)
- Al avanzar L, es decir al sacar un elemento, debo reducir en 1 el valor que corresponda (o ninguno si es un ?)
- P se cumple si y solo si:
  $(0_{par} == 0 \ o \ 0_{impar} == 0) \ y \ (1_{par} == 0 \ o \ 1_{impar} == 0) \ y \ (0_{par} == 0 \ o \ 1_{par} == 0) \ y \ (0_{impar} == 0 \ o \ 1_{impar} == 0)$
- Esta información es fácilmente comprobable en una matriz ps[2][2] donde el primer índice indica el valor y el segundo la paridad de la posición.

La operación

- Todos los rangos que procesamos tienen extremos R distintos.

La operación

- Todos los rangos que procesamos tienen extremos R distintos.
- Para cada rango que procesamos, es el máximo rango válido que termina en esa posición R.

La operación

- Todos los rangos que procesamos tienen extremos R distintos.
- Para cada rango que procesamos, es el máximo rango válido que termina en esa posición R.
- Por tanto, todos los otros rangos que terminan en el mismo R son los sufijos de este.

La operación

- Todos los rangos que procesamos tienen extremos R distintos.
- Para cada rango que procesamos, es el máximo rango válido que termina en esa posición R.
- Por tanto, todos los otros rangos que terminan en el mismo R son los sufijos de este.
- Si tiene longitud X = R-L, tiene X sufijos no vacíos. Por tanto, para cada rango que proceso quiero agregar a la respuesta R-L.

Implementación: Estado

```
1  struct estado{
2      int ps[2][2];
3      estado(){
4          memset(ps,0,sizeof(ps));
5      }
6      bool P(){ return (ps[0][0]==0 or ps[0][1]==0) and (ps
           [1][0]==0 or ps[1][1]==0) and (ps[0][0]==0 or ps
           [1][0]==0) and (ps[0][1]==0 or ps[1][1]==0);
7      }
8      void Cambiar(char c, int p, int d){
9          if(c!='?') ps[c-'0'][p%2]+=d;
10     }
11 };
```

Implementación: Algoritmo

```
1  string s;cin>>s;
2  int n = s.size(), l = 0, r = 0;
3  estado est;
4  long long res = 0;
5  while(r<n){
6      est.Cambiar(s[r],r,1);
7      if(est.P()){
8          r++;
9          res += r-l;
10         }else{
11         est.Cambiar(s[r],r,-1);
12         est.Cambiar(s[l],l,-1);
13         l++;
14         }
15     }
16 cout<<res<<endl;
```

Contenidos

Array

Enunciado

Dado un vector de enteros V y un entero K, queremos encontrar un rango [L,R] (notar: inclusive - inclusive) que sea minimal por inclusión tal que haya exactamente K números distintos en ese rango. Si no existe devolver [-1,-1]
$1 \leq N = |V|, K \leq 100{,}000$

Para practicar:
`https://codeforces.com/problemset/problem/224/B`

Array

Enunciado

Dado un vector de enteros V y un entero K, queremos encontrar un rango [L,R] (notar: inclusive - inclusive) que sea minimal por inclusión tal que haya exactamente K números distintos en ese rango. Si no existe devolver [-1,-1]
$1 \leq N = |V|, K \leq 100{,}000$

Ejemplos

Para practicar:
`https://codeforces.com/problemset/problem/224/B`

Array

Enunciado

Dado un vector de enteros V y un entero K, queremos encontrar un rango [L,R] (notar: inclusive - inclusive) que sea minimal por inclusión tal que haya exactamente K números distintos en ese rango. Si no existe devolver [-1,-1]
$1 \leq N = |V|, K \leq 100{,}000$

Ejemplos

- V = {1, 2, 2, 3}, K = 2, res = [1,2] // Indexa en base 1

Para practicar:
`https://codeforces.com/problemset/problem/224/B`

Array

Enunciado

Dado un vector de enteros V y un entero K, queremos encontrar un rango [L,R] (notar: inclusive - inclusive) que sea minimal por inclusión tal que haya exactamente K números distintos en ese rango. Si no existe devolver [-1,-1]
$1 \leq N = |V|, K \leq 100{,}000$

Ejemplos

- V = {1, 2, 2, 3}, K = 2, res = [1,2] // Indexa en base 1
- V = {1, 1, 2, 2, 3, 3, 4, 5}, K = 3, res = [2,5]

Para practicar:
`https://codeforces.com/problemset/problem/224/B`

Array

Enunciado

Dado un vector de enteros V y un entero K, queremos encontrar un rango [L,R] (notar: inclusive - inclusive) que sea minimal por inclusión tal que haya exactamente K números distintos en ese rango. Si no existe devolver [-1,-1]
$1 \leq N = |V|, K \leq 100{,}000$

Ejemplos

- V = {1, 2, 2, 3}, K = 2, res = [1,2] // Indexa en base 1
- V = {1, 1, 2, 2, 3, 3, 4, 5}, K = 3, res = [2,5]
- V = {4, 7, 7, 4, 7, 4, 7}, K = 4, res = [-1,-1]

Para practicar:
`https://codeforces.com/problemset/problem/224/B`

Observaciones Clave

- Si en un rango [L,R) (notar inclusive - exclusive) hay menos de K elementos distintos, la única forma de aumentar la cantidad de elementos distintos es agrandando el rango. Esto lo hacemos avanzando R. (Pasamos a evaluar el rango [L,R+1))

Observaciones Clave

- Si en un rango [L,R) (notar inclusive - exclusive) hay menos de K elementos distintos, la única forma de aumentar la cantidad de elementos distintos es agrandando el rango. Esto lo hacemos avanzando R. (Pasamos a evaluar el rango [L,R+1))
- Si en un rango [L,R) hay más de K elementos distintos, la única forma de reducir la cantidad de elementos distintos es achicar el rango. Esto lo haremos avanzando L (Pasamos a evaluar el rango [L+1,R))

Operaciones Posibles

Hay 2 operaciones posibles que podemos usar para resolver este problema, siendo ambas válidas:

- Si el rango tiene exactamente K elementos distintos, lo que nos interesa ver es si es minimal. Para esto, una posibilidad es considerar la siguiente operación:
  Inicialmente, L_{res} y R_{res} son -1, y cuando hacemos Op(V,L,R) decimos: Si $L_{res} == -1$ *or* $(L_{res} \leq L$ *and* $R \leq R_{res})$ entonces $L_{res} = L$, $R_{res} = R$

Operaciones Posibles

Hay 2 operaciones posibles que podemos usar para resolver este problema, siendo ambas válidas:

- Si el rango tiene exactamente K elementos distintos, lo que nos interesa ver es si es minimal. Para esto, una posibilidad es considerar la siguiente operación:
  Inicialmente, L_{res} y R_{res} son -1, y cuando hacemos Op(V,L,R) decimos: Si $L_{res} == -1$ *or* $(L_{res} \leq L$ *and* $R \leq R_{res})$ entonces $L_{res} = L$, $R_{res} = R$
- Otra observación que podemos hacer es notar que el rango de menor longitud que cumpla la propiedad va a ser minimal (dado que si hubiera uno dentro de este que también cumple la propiedad, tendría menor longitud), y por tanto podemos buscar simplemente el rango con K elementos más corto.

¿Cómo mantener el estado?

Lo que nos interesa es la cantidad de números distintos en el rango, pero para poder mantener esto debemos poder mantener también la cantidad de apariciones de cada número en el rango.
Para esto podemos utilizar un map<int,int> de C++, que guardará la cantidad de apariciones en el rango. Lógicamente, la cantidad de distintos es la cantidad de elementos cuyo valor sea distinto de 0.

Implementación: Estado

```
1  struct estado{
2    map<int,int> apps;
3    int cdif = 0;
4    void Agregar(int c){
5        apps[c]++;
6        if(apps[c]==1) cdif++;
7    }
8    void Sacar(int c){
9        apps[c]--;
10       if(apps[c]==0) cdif--;
11   }
12 };
```

Implementación: Algoritmo

```
1  int n,k, lres = -2, rres = -1, l = 0, r = 0; cin>>n>>k;
2  vector<int> a(n); estado est;
3  for(int i = 0;i<n;i++) cin>>a[i];
4  while(r<n or est.cdif>=k){
5      if(est.cdif<k and r<n){
6          est.Agregar(a[r]);
7          r++;}
8      else{
9          if(est.cdif==k)
10             if(lres==-2 or (lres<=l and r<=rres) )
11                 lres = l, rres = r;
12         est.Sacar(a[l]);
13         l++;}
14 }
15 cout<<lres+1<<" "<<rres<<endl; /// Notar que paso de base 0 a
       base 1 y de [) a [].
```

Contenidos

To Add or Not to Add

Enunciado

Dado un vector A de enteros, y un número K, quiero realizar hasta K veces la operación que incrementa en 1 algún elemento de A.
Mi objetivo es maximizar la cantidad de apariciones del número que más se repite luego de realizar estas operaciones, y que este número sea el menor posible entre los que pueden alcanzar esa cantidad de repeticiones.
La respuesta es primero la cantidad de repeticiones y luego el número que las alcanza.
$1 \leq N = |A| \leq 100{,}000, 0 \leq K \leq 10^9$

To Add or Not to Add

Enunciado

Dado un vector A de enteros, y un número K, quiero realizar hasta K veces la operación que incrementa en 1 algún elemento de A.
Mi objetivo es maximizar la cantidad de apariciones del número que más se repite luego de realizar estas operaciones, y que este número sea el menor posible entre los que pueden alcanzar esa cantidad de repeticiones.
La respuesta es primero la cantidad de repeticiones y luego el número que las alcanza.
$1 \leq N = |A| \leq 100{,}000, 0 \leq K \leq 10^9$

Ejemplos

To Add or Not to Add

Enunciado

Dado un vector A de enteros, y un número K, quiero realizar hasta K veces la operación que incrementa en 1 algún elemento de A.
Mi objetivo es maximizar la cantidad de apariciones del número que más se repite luego de realizar estas operaciones, y que este número sea el menor posible entre los que pueden alcanzar esa cantidad de repeticiones.
La respuesta es primero la cantidad de repeticiones y luego el número que las alcanza.
$1 \leq N = |A| \leq 100{,}000, 0 \leq K \leq 10^9$

Ejemplos

- Si A = {6, 3, 4, 0, 2} y K = 3, res = 3, 4

To Add or Not to Add

Enunciado

Dado un vector A de enteros, y un número K, quiero realizar hasta K veces la operación que incrementa en 1 algún elemento de A.
Mi objetivo es maximizar la cantidad de apariciones del número que más se repite luego de realizar estas operaciones, y que este número sea el menor posible entre los que pueden alcanzar esa cantidad de repeticiones.
La respuesta es primero la cantidad de repeticiones y luego el número que las alcanza.
$1 \leq N = |A| \leq 100{,}000, 0 \leq K \leq 10^9$

Ejemplos

- Si A = {6, 3, 4, 0, 2} y K = 3, res = 3, 4
- Si A = {5, 5, 5} y K = 4, res = 3, 5

To Add or Not to Add

Enunciado

Dado un vector A de enteros, y un número K, quiero realizar hasta K veces la operación que incrementa en 1 algún elemento de A.
Mi objetivo es maximizar la cantidad de apariciones del número que más se repite luego de realizar estas operaciones, y que este número sea el menor posible entre los que pueden alcanzar esa cantidad de repeticiones.
La respuesta es primero la cantidad de repeticiones y luego el número que las alcanza.
$1 \leq N = |A| \leq 100{,}000, 0 \leq K \leq 10^9$

Ejemplos

- Si A = {6, 3, 4, 0, 2} y K = 3, res = 3, 4
- Si A = {5, 5, 5} y K = 4, res = 3, 5
- Si A = {3, 1, 2, 2, 1} y K = 3, res = 4, 2

Para practicar

Para practicar

`https://codeforces.com/problemset/problem/231/C`

Observaciones Clave: Orden

- En este caso es muy importante notar que si cambiamos el orden del vector A, y en particular si lo ordenamos de menor a mayor, la respuesta no cambia. Por tanto, vamos a ordenar A y de ahora en adelante asumirlo como un vector ordenado de menor a mayor.

Observaciones Clave: Orden

- En este caso es muy importante notar que si cambiamos el orden del vector A, y en particular si lo ordenamos de menor a mayor, la respuesta no cambia. Por tanto, vamos a ordenar A y de ahora en adelante asumirlo como un vector ordenado de menor a mayor.
- Si elijo un valor X e intento maximizar sus apariciones, me conviene siempre elegir los elementos de A más cercanos a X (que no lo pasan) e incrementarlos.

Observaciones Clave: Orden

- En este caso es muy importante notar que si cambiamos el orden del vector A, y en particular si lo ordenamos de menor a mayor, la respuesta no cambia. Por tanto, vamos a ordenar A y de ahora en adelante asumirlo como un vector ordenado de menor a mayor.
- Si elijo un valor X e intento maximizar sus apariciones, me conviene siempre elegir los elementos de A más cercanos a X (que no lo pasan) e incrementarlos.
- Siempre me conviene elegir un X que sea a su vez un elemento de A.

Observaciones Clave: Orden

- En este caso es muy importante notar que si cambiamos el orden del vector A, y en particular si lo ordenamos de menor a mayor, la respuesta no cambia. Por tanto, vamos a ordenar A y de ahora en adelante asumirlo como un vector ordenado de menor a mayor.
- Si elijo un valor X e intento maximizar sus apariciones, me conviene siempre elegir los elementos de A más cercanos a X (que no lo pasan) e incrementarlos.
- Siempre me conviene elegir un X que sea a su vez un elemento de A.
- Por tanto, lo que puedo hacer es tomar un rango [L,R) y tratar de hacer que todos los elementos del rango sean iguales a A[R-1] aplicándoles la operación.

Observaciones Clave: Orden

- En este caso es muy importante notar que si cambiamos el orden del vector A, y en particular si lo ordenamos de menor a mayor, la respuesta no cambia. Por tanto, vamos a ordenar A y de ahora en adelante asumirlo como un vector ordenado de menor a mayor.
- Si elijo un valor X e intento maximizar sus apariciones, me conviene siempre elegir los elementos de A más cercanos a X (que no lo pasan) e incrementarlos.
- Siempre me conviene elegir un X que sea a su vez un elemento de A.
- Por tanto, lo que puedo hacer es tomar un rango [L,R) y tratar de hacer que todos los elementos del rango sean iguales a A[R-1] aplicándoles la operación.
- Notar que esto nos permite una solución alternativa con Búsqueda Binaria y Tabla Aditiva, cuya implementación esta al final de esta presentación.

Entonces... ¿Cuáles son los rangos?

En esta transformación del problema, los rangos [L,R) serán como dijimos antes los rangos del vector A ordenado que mediante la operación de aumentar en 1 algún elemento de A haremos que sean todos iguales a A[R-1].

La condición P(A,L,R)

- Notemos que si tengo el rango [L,R) lo que voy a intentar es que todos los elementos sean iguales a A[R-1]

La condición P(A,L,R)

- Notemos que si tengo el rango [L,R) lo que voy a intentar es que todos los elementos sean iguales a A[R-1]
- Para hacer que A[L] = A[R-1], tengo que hacer A[R-1]-A[L] operaciones, para hacer que A[L+1]=A[R-1] tengo que hacer A[R-1]-A[L+1] operaciones y así...

La condición P(A,L,R)

- Notemos que si tengo el rango [L,R) lo que voy a intentar es que todos los elementos sean iguales a A[R-1]
- Para hacer que A[L] = A[R-1], tengo que hacer A[R-1]-A[L] operaciones, para hacer que A[L+1]=A[R-1] tengo que hacer A[R-1]-A[L+1] operaciones y así...
- Es decir, tengo que hacer (A[R-1]-A[L])+(A[R-1]-A[L+1])+(A[R-1]-A[L+2])+...+(A[R-1]-A[R-2]) = $\sum_{i=L}^{R-1}(A[R-1]-A[i])$ operaciones

La condición P(A,L,R)

- Notemos que si tengo el rango [L,R) lo que voy a intentar es que todos los elementos sean iguales a A[R-1]
- Para hacer que A[L] = A[R-1], tengo que hacer A[R-1]-A[L] operaciones, para hacer que A[L+1]=A[R-1] tengo que hacer A[R-1]-A[L+1] operaciones y así...
- Es decir, tengo que hacer (A[R-1]-A[L])+(A[R-1]-A[L+1])+(A[R-1]-A[L+2])+...+(A[R-1]-A[R-2]) = $\sum_{i=L}^{R-1}(A[R-1]-A[i])$ operaciones
- Notar que esto es igual a $(R-L)*A[R-1]-\sum_{i=L}^{R-1}A[i]$

La condición P(A,L,R)

- Notemos que si tengo el rango [L,R) lo que voy a intentar es que todos los elementos sean iguales a A[R-1]
- Para hacer que A[L] = A[R-1], tengo que hacer A[R-1]-A[L] operaciones, para hacer que A[L+1]=A[R-1] tengo que hacer A[R-1]-A[L+1] operaciones y así...
- Es decir, tengo que hacer (A[R-1]-A[L])+(A[R-1]-A[L+1])+(A[R-1]-A[L+2])+...+(A[R-1]-A[R-2]) = $\sum_{i=L}^{R-1}(A[R-1]-A[i])$ operaciones
- Notar que esto es igual a $(R-L)*A[R-1]-\sum_{i=L}^{R-1}A[i]$
- Finalmente, entonces, mi condición es $P(A,L,R) \equiv (R-L)*A[R-1]-\sum_{i=L}^{R-1}A[i] \leq K$

El estado y la operación

- Por lo anterior, en nuestro estado lo único que debemos conservar es la cantidad de elementos y su suma.

El estado y la operación

- Por lo anterior, en nuestro estado lo único que debemos conservar es la cantidad de elementos y su suma.
- Para evaluar si se cumple P, tengo R y puedo hacer $cant * A[R-1] - \sum_{i=L}^{R-1} A[i]$ *vs* K (la suma la tengo precomputada así que no necesito calcularla cada vez)

El estado y la operación

- Por lo anterior, en nuestro estado lo único que debemos conservar es la cantidad de elementos y su suma.
- Para evaluar si se cumple P, tengo R y puedo hacer $cant * A[R-1] - \sum_{i=L}^{R-1} A[i]$ *vs* K (la suma la tengo precomputada así que no necesito calcularla cada vez)
- Para nuestra operación lo único que haremos es quedarnos con el mayor de los rangos evaluados, y si hay empate el que termina primero.

El estado y la operación

- Por lo anterior, en nuestro estado lo único que debemos conservar es la cantidad de elementos y su suma.
- Para evaluar si se cumple P, tengo R y puedo hacer $cant * A[R-1] - \sum_{i=L}^{R-1} A[i]$ *vs* K (la suma la tengo precomputada así que no necesito calcularla cada vez)
- Para nuestra operación lo único que haremos es quedarnos con el mayor de los rangos evaluados, y si hay empate el que termina primero.
- Notar que todas estas operaciones son $O(1)$, por tanto la parte de 2 punteros es $O(N)$. Sin embargo, por el ordenamiento del principio la complejidad queda $O(N * log(N))$

Implementación: Estado

```
1  typedef long long intl;
2  struct estado{
3      intl tot = 0, cant = 0;
4      void Agregar(int x){
5          cant++;
6          tot+=x;
7      }
8      void Sacar(int x){
9          cant--;
10         tot-=x;
11     }
12     bool P(intl X, intl K){
13         return (X*cant-tot)<=K;
14     }
15 };
```

Implementación: Algoritmo

```
1  int N, K, res1 = -1, res2 = 0, l = 0, r = 0;
2  cin>>N>>K;
3  vector<int> A(N); estado est;
4  for(int i = 0;i < N;i++) cin>>A[i];
5  sort(A.begin(),A.end());
6  while(r<N){
7      est.Agregar(A[r]);
8      if(est.P(A[r],K)){
9          if(r-l+1>res1) res1 = r-l+1, res2 = A[r];
10         r++;
11     }else{ est.Sacar(A[r]); est.Sacar(A[l]); l++; }
12 }
13 cout<<res1<<" "<<res2<<endl;
```

Contenidos

Otros casos de 2 punteros

Aunque durante esta lección vimos la técnica de 2 punteros siempre aplica a analizar rangos de un vector, lo cierto es que existen otras situaciones donde aparecen los 2 punteros, como por ejemplo:

- 2 punteros que empiezan cada uno en un extremo del vector y que avancen hacia el medio (cada uno en la dirección contraria a la que apunta originalmente)

Lo que tienen en común estas técnicas es que en todos los casos tengo 2 punteros que van avanzando a lo largo de una serie de iteraciones. La clave es que cada iterador solo puede avanzar/retroceder en una dirección.

Otros casos de 2 punteros

Aunque durante esta lección vimos la técnica de 2 punteros siempre aplica a analizar rangos de un vector, lo cierto es que existen otras situaciones donde aparecen los 2 punteros, como por ejemplo:

- 2 punteros que empiezan cada uno en un extremo del vector y que avancen hacia el medio (cada uno en la dirección contraria a la que apunta originalmente)
- Puede haber, también, 2 punteros en 2 vectores o estructuras distintas. En este caso también lo podemos pensar como que eliminamos elementos de las estructuras a medida que los procesamos.

Lo que tienen en común estas técnicas es que en todos los casos tengo 2 punteros que van avanzando a lo largo de una serie de iteraciones. La clave es que cada iterador solo puede avanzar/retroceder en una dirección.

Contenidos

Three Parts of the Array

Enunciado

Dado un vector de enteros V, con |V| = N, quiero dividirlo en 3 partes consecutivas, tales que la primera y la tercera sumen lo mismo y dicha suma sea el mayor número posible.

Dicho más formalmente, quiero i, j tales que i<j y $\sum_{k=0}^{i} A[k] = \sum_{k=j}^{N-1} A[k]$ y se maximice $\sum_{k=0}^{i} A[k]$

Mi objetivo es obtener dicha suma.

$1 \leq N \leq 200{,}000$

Para practicar:

`https://codeforces.com/problemset/problem/1006/C`

Three Parts of the Array

Enunciado

Dado un vector de enteros V, con |V| = N, quiero dividirlo en 3 partes consecutivas, tales que la primera y la tercera sumen lo mismo y dicha suma sea el mayor número posible.

Dicho más formalmente, quiero i, j tales que i<j y $\sum_{k=0}^{i} A[k] = \sum_{k=j}^{N-1} A[k]$ y se maximice $\sum_{k=0}^{i} A[k]$

Mi objetivo es obtener dicha suma.

$1 \leq N \leq 200{,}000$

Ejemplos

Para practicar:

`https://codeforces.com/problemset/problem/1006/C`

Three Parts of the Array

Enunciado

Dado un vector de enteros V, con |V| = N, quiero dividirlo en 3 partes consecutivas, tales que la primera y la tercera sumen lo mismo y dicha suma sea el mayor número posible.
Dicho más formalmente, quiero i, j tales que i<j y $\sum_{k=0}^{i} A[k] = \sum_{k=j}^{N-1} A[k]$ y se maximice $\sum_{k=0}^{i} A[k]$
Mi objetivo es obtener dicha suma.
$1 \leq N \leq 200{,}000$

Ejemplos

- Si V = {1, 3, 1, 1, 4}, res = 5

Para practicar:
`https://codeforces.com/problemset/problem/1006/C`

Three Parts of the Array

Enunciado

Dado un vector de enteros V, con |V| = N, quiero dividirlo en 3 partes consecutivas, tales que la primera y la tercera sumen lo mismo y dicha suma sea el mayor número posible.

Dicho más formalmente, quiero i, j tales que i<j y $\sum_{k=0}^{i} A[k] = \sum_{k=j}^{N-1} A[k]$ y se maximice $\sum_{k=0}^{i} A[k]$

Mi objetivo es obtener dicha suma.

$1 \leq N \leq 200{,}000$

Ejemplos

- Si V = {1, 3, 1, 1, 4}, res = 5
- Si V = {1, 3, 2, 1, 4}, res = 4

Para practicar:

`https://codeforces.com/problemset/problem/1006/C`

Three Parts of the Array

Enunciado

Dado un vector de enteros V, con |V| = N, quiero dividirlo en 3 partes consecutivas, tales que la primera y la tercera sumen lo mismo y dicha suma sea el mayor número posible.

Dicho más formalmente, quiero i, j tales que i<j y $\sum_{k=0}^{i} A[k] = \sum_{k=j}^{N-1} A[k]$ y se maximice $\sum_{k=0}^{i} A[k]$

Mi objetivo es obtener dicha suma.

$1 \leq N \leq 200{,}000$

Ejemplos

- Si V = {1, 3, 1, 1, 4}, res = 5
- Si V = {1, 3, 2, 1, 4}, res = 4
- Si V = {4, 1, 2}, res = 0 // Rango vacio.

Para practicar:

`https://codeforces.com/problemset/problem/1006/C`

Aparición de los 2 punteros

Lo interesante es que en el propio enunciado nos hablan de índices y sumas de rangos. Por tanto podemos:

- Empezar con i = 0 y j = N-1, y un candidato a respuesta, res = 0.

Notar que en este caso el estado es tan sencillo que no es necesario separarlo en una estructura aparte. Aunque podríamos hacerlo si quisiéramos

Aparición de los 2 punteros

Lo interesante es que en el propio enunciado nos hablan de índices y sumas de rangos. Por tanto podemos:

- Empezar con i = 0 y j = N-1, y un candidato a respuesta, res = 0.
- En cada paso, considerar $sum_1 = \sum_0^{i-1}$ y $sum_2 = \sum_{j+1}^{N-1}$. Por lo que hay 3 posibilidades.

Notar que en este caso el estado es tan sencillo que no es necesario separarlo en una estructura aparte. Aunque podríamos hacerlo si quisiéramos

Aparición de los 2 punteros

Lo interesante es que en el propio enunciado nos hablan de índices y sumas de rangos. Por tanto podemos:

- Empezar con i = 0 y j = N-1, y un candidato a respuesta, res = 0.
- En cada paso, considerar $sum_1 = \sum_0^{i-1}$ y $sum_2 = \sum_{j+1}^{N-1}$. Por lo que hay 3 posibilidades.
- Si $sum_1 > sum_2$ debo avanzar el índice j, que es hacer j = j-1

Notar que en este caso el estado es tan sencillo que no es necesario separarlo en una estructura aparte. Aunque podríamos hacerlo si quisiéramos

Aparición de los 2 punteros

Lo interesante es que en el propio enunciado nos hablan de índices y sumas de rangos. Por tanto podemos:

- Empezar con i = 0 y j = N-1, y un candidato a respuesta, res = 0.
- En cada paso, considerar $sum_1 = \sum_0^{i-1}$ y $sum_2 = \sum_{j+1}^{N-1}$. Por lo que hay 3 posibilidades.
- Si $sum_1 > sum_2$ debo avanzar el índice j, que es hacer j = j-1
- Si $sum_1 < sum_2$ debo avanzar el índice i, que es hacer i = i+1

Notar que en este caso el estado es tan sencillo que no es necesario separarlo en una estructura aparte. Aunque podríamos hacerlo si quisiéramos

Aparición de los 2 punteros

Lo interesante es que en el propio enunciado nos hablan de índices y sumas de rangos. Por tanto podemos:

- Empezar con i = 0 y j = N-1, y un candidato a respuesta, res = 0.
- En cada paso, considerar $sum_1 = \sum_0^{i-1}$ y $sum_2 = \sum_{j+1}^{N-1}$. Por lo que hay 3 posibilidades.
- Si $sum_1 > sum_2$ debo avanzar el índice j, que es hacer j = j-1
- Si $sum_1 < sum_2$ debo avanzar el índice i, que es hacer i = i+1
- Si $sum_1 = sum_2$ tengo un valor candidato a ser la respuesta, por tanto $res = max(sum_1, res)$, y luego avanzo ambos índices.

Notar que en este caso el estado es tan sencillo que no es necesario separarlo en una estructura aparte. Aunque podríamos hacerlo si quisiéramos

Implementación: Algoritmo

```
int N; cin>>N;
vector<int> V(N);
int i = 0, j = N-1;
long long res = 0, sum1 = 0, sum2 = 0;
for(int k = 0;k<N;k++) cin>>V[k];
while(i<=j+1){
    if(sum1<sum2) sum1+=V[i], i++;
    else if(sum1>sum2)sum2+=V[j],j--;
    else{
        res = max(res,sum1);
        sum1+=V[i],i++;
        sum2+=V[j],j--;
    }
}
cout<<res<<endl;
```

Contenidos

Eugeny and Play List

Enunciado

Tengo N canciones, cada una dura t_i segundos y es reproducida c_i veces. Tengo M preguntas $v_i (v_i < v_{i+1})$ que preguntan el número de canción reproducida en ese momento.
$1 \leq N, M \leq 100{,}000$

Para practicar:
`https://codeforces.com/problemset/problem/302/B`

Eugeny and Play List

Enunciado

Tengo N canciones, cada una dura t_i segundos y es reproducida c_i veces. Tengo M preguntas $v_i (v_i < v_{i+1})$ que preguntan el número de canción reproducida en ese momento.
$1 \leq N, M \leq 100{,}000$

Ejemplos

Para practicar:
`https://codeforces.com/problemset/problem/302/B`

Eugeny and Play List

Enunciado

Tengo N canciones, cada una dura t_i segundos y es reproducida c_i veces. Tengo M preguntas $v_i (v_i < v_{i+1})$ que preguntan el número de canción reproducida en ese momento.
$1 \leq N, M \leq 100{,}000$

Ejemplos

- t = {2} ; c = {8} ; v = {1, 16} ; res = {1, 1}

Para practicar:
`https://codeforces.com/problemset/problem/302/B`

Eugeny and Play List

Enunciado

Tengo N canciones, cada una dura t_i segundos y es reproducida c_i veces. Tengo M preguntas $v_i (v_i < v_{i+1})$ que preguntan el número de canción reproducida en ese momento.
$1 \leq N, M \leq 100{,}000$

Ejemplos

- t = {2} ; c = {8} ; v = {1, 16} ; res = {1, 1}
- t = {2, 1, 1, 2} ; c = {1, 2, 1, 2} ; v = {1, 2, 3, 4, 5, 6, 7, 8, 9} ; res = {1, 1, 2, 2, 3, 4, 4, 4, 4}

Para practicar:
`https://codeforces.com/problemset/problem/302/B`

Primeras observaciones

- Como no nos importa en cuál de las reproducciones de la canción estamos, sino sólo cuál canción es. Podemos considerar un solo vector A, tal que $A_i = t_i * c_i$

Primeras observaciones

- Como no nos importa en cuál de las reproducciones de la canción estamos, sino sólo cuál canción es. Podemos considerar un solo vector A, tal que $A_i = t_i * c_i$
- A su vez, como nos interesa el tiempo transcurrido, nos interesa la suma del prefijo. Llamemos $suma_i = \sum_{k=0}^{i} A[k]$

Primeras observaciones

- Como no nos importa en cuál de las reproducciones de la canción estamos, sino sólo cuál canción es. Podemos considerar un solo vector A, tal que $A_i = t_i * c_i$
- A su vez, como nos interesa el tiempo transcurrido, nos interesa la suma del prefijo. Llamemos $suma_i = \sum_{k=0}^{i} A[k]$
- Para la pregunta v_j, la respuesta es i tal que $suma_{i-1} < v_j \leq suma_i$ (Como indexamos en base 1, podemos tomar que $suma_0 = 0$)

Iteración

- Empezamos nuestra iteración con i = 0, j = 0, suma = 0 y luego vemos en cada paso.

Iteración

- Empezamos nuestra iteración con i = 0, j = 0, suma = 0 y luego vemos en cada paso.
- Si suma $\geq$ v[j], hacemos res[j] = i, j++.

Iteración

- Empezamos nuestra iteración con i = 0, j = 0, suma = 0 y luego vemos en cada paso.
- Si suma $\geq$ v[j], hacemos res[j] = i, j++.
- Si en cambio suma<v[j], hacemos suma+=A[i],i++.

Iteración

- Empezamos nuestra iteración con i = 0, j = 0, suma = 0 y luego vemos en cada paso.
- Si suma $\geq$ v[j], hacemos res[j] = i, j++.
- Si en cambio suma<v[j], hacemos suma+=A[i],i++.
- Finalizamos cuando j = M.

Iteración

- Empezamos nuestra iteración con i = 0, j = 0, suma = 0 y luego vemos en cada paso.
- Si suma $\geq$ v[j], hacemos res[j] = i, j++.
- Si en cambio suma<v[j], hacemos suma+=A[i],i++.
- Finalizamos cuando j = M.
- Notar que esta solución es O(N), también existe una solución $O(N * log(N))$ usando Búsqueda binaria. Esta se agrega al final de la presentación. Esta tiene la ventaja de que puede recibir las preguntas en cualquier orden y online.

Implementación: Algoritmo (1)

```
1  int N,M, i = 0, j = 0, suma = 0; cin>>N>>M;
2  vector<int> A(N), v(M), res(M);
3  for(int k = 0;k<N;k++){
4      int c,t; cin>>c>>t;
5      A[k] = c*t;
6  }
7  for(int k = 0;k<M;k++) cin>>v[k];
8  while(j<M){
9      if(suma<v[j]) suma+=A[i], i++;
10     else res[j] = i,j++;
11 }
12 for(int k = 0;k<M;k++) cout<<res[k]<<"\n";
```

Contenidos

Contenidos

$O(N^3)$

Fuerza Bruta

```
1 void Problema(vector<int> & v, int X, int & res, int & lres,
      int & rres){
2     res = 0;
3     for(int l = 0;l<int(v.size());l++)
4         for(int r = l;r<=int(v.size());r++){
5             int cont = 0;
6             for(int k = l;k<r;k++) cont+= v[k];
7             if(cont<=X and (r-l)>res) res = r-l, lres = l,
                  rres = r;
8         }
9 }
```

$O(N^2)$

Con tabla aditiva

```
1  void Problema(vector<int> & v, int X, int & res, int & lres,
       int & rres){
2      int N = int(v.size());
3      vector<int> add(N+1,0);
4      for(int i = 1;i<=N;i++) add[i] = add[i-1]+v[i-1];
5      res = 0;
6      for(int l = 0; l<N;l++)
7          for(int r = l;r<=N;r++){
8              if(add[r]-add[l]<=X and (r-l)>res){
9                  res = (r-l), lres = l, rres = r;
10             }
11         }
12 }
```

$O(N * log(N))$

Con Búsqueda binaria

```
1  void Problema(vector<int> & v, int X, int & res, int & lres,
       int & rres){
2      int N = int(v.size()); res = 0;
3      vector<int> add(N+1,0);
4      for(int i = 1;i<=N;i++) add[i] = add[i-1]+v[i-1];
5      for(int l = 0; l<N;l++){
6          int ra = l, rb = N+1;
7          while(rb-ra>1){
8              int m = (ra+rb)/2;
9              if(v[m]-v[l]>X) rb = m;
10             else ra = m;}
11         if(ra-l>res) res = (ra-l), lres = l, rres = ra;
12     }}
```

Contenidos

$O(N * log(N))$

Con Búsqueda binaria

```
1  int n,k, res1 = -1, res2 = 0; cin>>n>>k;
2  vector<long long> a(n), pre(n+1);
3  for(int i = 0;i<n;i++) cin>>a[i];
4  sort(a.begin(),a.end());
5  for(int i = 0;i<n;i++) pre[i+1] = pre[i]+a[i];
6  for(int i = 0;i<n;i++){
7      int l = -1, r = i;
8      while(r-l>1){
9          int m = (l+r)/2;
10         long long sum = (i-m)*a[i]-(pre[i]-pre[m]);
11         if(sum<=k) r = m;
12         else l = m;}
13     if(i-r+1>res1) res1 = i-r+1, res2 = a[i];
14 }  cout<<res1<<" "<<res2<<endl;
```

Contenidos

$O(N * log(N))$

Con Búsqueda binaria

```
1  int N,M; cin>>N>>M;
2  vector<int> A(N+1), v(M), res(M);
3  for(int k = 0;k<N;k++){
4      int c,t; cin>>c>>t;
5      A[k+1] = A[k]+c*t;
6  }
7  for(int k = 0;k<M;k++) {
8      cin>>v[k];
9      cout<<lower_bound(A.begin(),A.end(),v[k])-A.begin()<<"\n";
10     /* Notar que uso la funcion de la std lower_bound para
11     la Busqueda binaria */
12 }
```