Cobertura de Infraestructura Mínima

Suma de Prefijos + Búsqueda Binaria

Descripción del problema

Un gobierno planea construir K estaciones de servicio en una carretera recta para dar servicio a N ciudades que ya existen a lo largo de esa ruta. Las N ciudades se encuentran en posiciones dadas por las coordenadas a1, a2, ..., aN.

1. Las coordenadas de las ciudades están dadas en orden no decreciente.
2. Las estaciones de servicio se construirán en ubicaciones enteras b1, b2, ..., bK.

El costo de servicio para una ciudad es la distancia hasta la estación de servicio más cercana.

Tu tarea es determinar la distancia mínima máxima de servicio (R) posible. Es decir, encontrar las K ubicaciones de las estaciones (b1, ..., bK) de modo que la ciudad que esté peor servida (la que tenga la mayor distancia a su estación más cercana) se encuentre lo más cerca posible de una estación.

En términos formales: Queremos minimizar el valor R, donde R es igual al máximo, para cada i desde 1 hasta N, de la distancia mínima entre la posición de la ciudad a_i y la posición de la estación b_j, considerando todas las estaciones desde j = 1 hasta K

Entrada

La primera línea contiene dos enteros positivos N y K ($1 \le K \le N \le 10^5$): el número de ciudades y el número de estaciones de servicio a construir.
La segunda línea contiene una secuencia de N enteros a1, a2, ..., aN ($-10^9 \le ai \le 10^9$): las coordenadas de las ciudades.
Todas las coordenadas ai se dan en orden no decreciente.

Salida

Imprima la distancia mínima máxima de servicio R posible, garantizando que R es un entero no negativo.

Ejemplos

Input	Output
3 1 0 4 8	4

Input	Output
5 2 0 2 4 8 10	2

Explicación:

Con dos estaciones (K = 2), se pueden colocar en 2 y en 8.
Las distancias máximas para cada estación son:
Estación 1 en 2: cubre 0, 2 y 4. Distancia máxima: 4 − 2 = 2
Estación 2 en 8: cubre 8 y 10. Distancia máxima: 10 − 8 = 2
La distancia mínima máxima total es 2.

CÓDIGO

```
#include <iostream>
#include <string>
using namespace std;

// Funcion para obtener el valor absoluto (sin llabs)
long long absoluto(long long x) {
  if (x < 0) return -x;
  return x;
}

/*
  Esta función verifica si es posible colocar K estaciones
  de manera que ninguna ciudad quede a más de R distancia
  de su estación más cercana.
*/
bool sePuede(long long* ciudades, int N, int K, long long R) {

  int estacionesUsadas = 1;
  long long primeraCiudad = *ciudades;
```

```
    long long posicionEstacion = primeraCiudad + R;

    for (int i = 0; i < N; i++) {

        long long ciudadActual = *(ciudades + i);

        // Usamos nuestra función absoluto()
        if ( absoluto(ciudadActual - posicionEstacion) > R ) {

            estacionesUsadas++;
            posicionEstacion = ciudadActual + R;

            if (estacionesUsadas > K) {
                return false;
            }
        }
    }

    return true;
}

int main() {

    // Leemos N y K como cadenas para detectar separadores inválidos (por ejemplo comas)
    string sN, sK;
    cout << "Ingrese N (ciudades) y K (estaciones): ";
    if (!(cin >> sN >> sK)) {
        cerr << "Error: no se pudieron leer N y K.\n";
        return 1;
    }
```

```
auto tieneComa = [](const string &s) {
   return s.find(',') != string::npos;
};

if (tieneComa(sN) || tieneComa(sK)) {
   cerr << "Error: no use comas al separar números. Use espacios o saltos de línea.\n";
   return 1;
}

int N = 0, K = 0;
try {
   N = stoi(sN);
   K = stoi(sK);
} catch (...) {
   cerr << "Error: N y K deben ser enteros válidos.\n";
   return 1;
}

if (N <= 0) {
   cerr << "Error: N debe ser mayor que 0.\n";
   return 1;
}

long long* ciudades = new long long[N];

cout << "Ingrese las posiciones de las ciudades, ya ordenadas:\n";

for (int i = 0; i < N; i++) {
   string token;
   if (!(cin >> token)) {
```

```
            cerr << "Error: entrada insuficiente de posiciones (se esperaban " << N << ").\n";
            delete[] ciudades;
            return 1;
        }

        if (tieneComa(token)) {
            cerr << "Error: no use comas en las posiciones. Separe los números con espacios o saltos de línea.\n";
            delete[] ciudades;
            return 1;
        }

        try {
            ciudades[i] = stoll(token);
        } catch (...) {
            cerr << "Error: posición inválida (no es un entero).\n";
            delete[] ciudades;
            return 1;
        }
    }

    long long izquierda = 0;
    long long derecha = 2000000000LL;
    long long mejorRespuesta = 0;

    while (izquierda <= derecha) {

        long long mitad = (izquierda + derecha) / 2;

        if (sePuede(ciudades, N, K, mitad)) {
            mejorRespuesta = mitad;
```

```
            derecha = mitad - 1;
        } else {
            izquierda = mitad + 1;
        }
    }

    cout << "\nLa distancia minima maxima (R) es: " << mejorRespuesta << endl;

    delete[] ciudades;
    return 0;
}
```