Universidad de las Fuerza Armadas

Nombre: Madellyn Tasipanta

NRC: 29852

Ejercicios de búsqueda Binaria

Ejercicio 1: Energía de Monstruos

Descripción: Estás en un juego con n monstruos en posiciones pos[i] en una línea. Tienes un ataque que inflige daño d al monstruo objetivo y d/2 (división entera) a monstruos adyacentes. Cada monstruo tiene health[i] puntos de vida. Encuentra el daño mínimo d necesario para poder derrotar a todos los monstruos atacando óptimamente.

Input:

- Primera línea: n (número de monstruos)
- Segunda línea: n enteros con las posiciones (ordenadas)
- Tercera línea: n enteros con los puntos de vida

Output:

- Daño mínimo necesario

Ejemplo

```
Input:
3
1 3 5
5 8 6

Output:
8
```

Código

```
#include <bits/stdc++.h>
using namespace std;

int n;
vector<long long> pos, health;

bool canDefeat(long long d) {
    vector<long long> damage(n, 0);

    for (int i = 0; i < n; i++) {
```

```
            long long currentDamage = damage[i];

            if (currentDamage >= health[i]) {
                continue;
            }

            long long needed = health[i] - currentDamage;
            long long attacks = (needed + d - 1) / d;

            damage[i] += attacks * d;

            if (i > 0) {
                damage[i - 1] += attacks * (d / 2);
            }
            if (i < n - 1) {
                damage[i + 1] += attacks * (d / 2);
            }
        }

        for (int i = 0; i < n; i++) {
            if (damage[i] < health[i]) {
                return false;
            }
        }

        return true;
    }

    int main() {
        ios_base::sync_with_stdio(false);
        cin.tie(NULL);

        cin >> n;

        pos.resize(n);
        health.resize(n);

        for (int i = 0; i < n; i++) {
            cin >> pos[i];
        }

        for (int i = 0; i < n; i++) {
            cin >> health[i];
```

```
    }

    long long left = 1, right = *max_element(health.begin(), health.end()) * 2;
    long long answer = right;

    while (left <= right) {
        long long mid = left + (right - left) / 2;

        if (canDefeat(mid)) {
            answer = mid;
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    cout << answer << endl;

    return 0;
}
```

Ejercicio 2: Pintar Tablero

Descripción: Tienes un tablero de n x m y k pintores. Cada pintor puede pintar una fila o una columna completas en una unidad de tiempo. Los pintores trabajan simultáneamente. Debes pintar al menos p casillas del tablero. Encuentra el tiempo mínimo necesario, sabiendo que una casilla pintada múltiples veces cuenta solo como una.

Input:

- Primera línea: n, m (dimensiones), k (pintores) y p (casillas objetivo)
- Segunda línea: estrategia óptima permitida (filas o columnas primeros)

Output:

- Tiempo mínimo para pintar al menos p casillas

Ejemplo

```
Input:
5 4 3 15

Output:
2
```

Código

```
#include <bits/stdc++.h>
using namespace std;

long long n, m, k, p;

long long calculateCells(long long rows, long long cols) {
    return min(rows, n) * m + min(cols, m) * n - min(rows, n) * min(cols, m);
}

bool canPaint(long long t) {
    long long maxCells = 0;

    for (long long i = 0; i <= k; i++) {
        long long rowPainters = i;
        long long colPainters = k - i;

        long long rowsPainted = min(n, rowPainters * t);
        long long colsPainted = min(m, colPainters * t);

        long long cells = calculateCells(rowsPainted, colsPainted);
        maxCells = max(maxCells, cells);
    }

    return maxCells >= p;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    cin >> n >> m >> k >> p;

    long long left = 0, right = max(n, m);
    long long answer = right;

    while (left <= right) {
        long long mid = left + (right - left) / 2;

        if (canPaint(mid)) {
            answer = mid;
            right = mid - 1;
        } else {
```

```
            left = mid + 1;
        }
    }

    cout << answer << endl;

    return 0;
}
```

Ejercicios con Hash

Ejercicio 2: Anagramas en Ventana

Descripción: Dado un string s de longitud n y un string patrón p de longitud m, encuentra todas las posiciones iniciales de las subsecuencias de s que son anagramas de p. Un anagrama es una permutación de las letras.

Input:

- Primera línea: string s
- Segunda línea: string p

Output:

- Primera línea: número de anagramas encontrados
- Segunda línea: posiciones iniciales (0-indexed) en orden ascendente

Ejemplo

```
Input:
cbaebabacd
abc

Output:
2
0 6
```

Código

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
```

```
string s, p;
cin >> s >> p;

int n = s.length();
int m = p.length();

if (m > n) {
   cout << 0 << endl;
   return 0;
}

map<char, int> patternFreq, windowFreq;

for (char c : p) {
   patternFreq[c]++;
}

vector<int> result;

for (int i = 0; i < m; i++) {
   windowFreq[s[i]]++;
}

if (windowFreq == patternFreq) {
   result.push_back(0);
}

for (int i = m; i < n; i++) {
   windowFreq[s[i]]++;

   windowFreq[s[i - m]]--;
   if (windowFreq[s[i - m]] == 0) {
      windowFreq.erase(s[i - m]);
   }

   if (windowFreq == patternFreq) {
      result.push_back(i - m + 1);
   }
}

cout << result.size() << endl;
if (result.size() > 0) {
```

```
    for (int i = 0; i < result.size(); i++) {
        if (i > 0) cout << " ";
        cout << result[i];
    }
    cout << endl;
  }

  return 0;
}
```

Ejercicio 2: Pares con Diferencia K

Descripción: Dado un arreglo de n enteros distintos arr[i] y un entero k, encuentra el número de pares (i, j) donde i < j y |arr[i] - arr[j]| = k.

Input:

- Primera línea: n (tamaño) y k (diferencia)
- Segunda línea: n enteros distintos del arreglo

Output:

- Número de pares con diferencia k

Ejemplo

```
Input:
5 2
1 5 3 4 2

Output:
3
```

Código

```
#include <bits/stdc++.h>
using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);

  int n, k;
  cin >> n >> k;
```

```
    vector<int> arr(n);
    unordered_set<int> elements;

    for (int i = 0; i < n; i++) {
        cin >> arr[i];
        elements.insert(arr[i]);
    }

    int count = 0;

    for (int i = 0; i < n; i++) {
        if (elements.count(arr[i] + k)) {
            count++;
        }
    }

    cout << count << endl;

    return 0;
}
```