Algoritmos de busqueda

Por Ariel Parra

[ΓΩ5]

Busqueda lineal (Linear Search)

La búsqueda lineal es un algoritmo simple que recorre secuencialmente todos los elementos de un contenedor y los compara uno por uno con el valor que se desea encontrar. Donde la complejidad sera de **O(n)**

En C++, puedes utilizar la función `find()` de la libraria `<algorithm>`para contenedores STL.

```
auto it = find(stl.begin(), stl.end(), x); // x es el elemento a buscar
auto it = find_if(v.begin(), v.end(), x);
if (it != stl.end()) {
    int elem = *it;
    int pos = distance(v.begin(), it);
} else {
    cout << "No encontrado";
}
```

Otras implementaciones de c++

```
vector<int> v = {1, 2, 3, 4, 5, 6, 7, 8, 9};
vector<int> subseq = {4, 5, 6};

// Buscar la subsecuencia {4, 5, 6} dentro del vector v
auto it = search(v.begin(), v.end(), subseq.begin(), subseq.end());

if (it != v.end()) {
    cout << "Subsecuencia encontrada en la posición: " << distance(v.begin(), it) << endl;
} else {
    cout << "Subsecuencia no encontrada" << endl;
}
```

```cpp
vector<int> v = {1, 1, 1, 2, 3, 4, 4, 4, 5, 6, 7};

// Buscar tres elementos consecutivos con el valor 4
auto it = search_n(v.begin(), v.end(), 3, 4);

if (it != v.end()) {
    cout << "Tres elementos consecutivos con el valor 4 encontrados en la posición: "
     << distance(v.begin(), it) << endl;
} else {
    cout << "No se encontraron tres elementos consecutivos con el valor 4" << endl;
}
```

Busqueda en strings

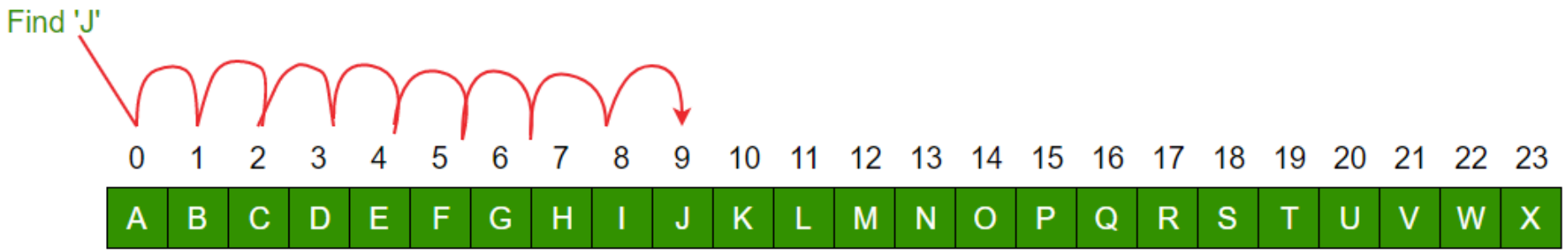
```
string str = "Hola Club Gallos";
string x = "Club"; // Elemento a buscar
auto pos = str.find(elem);
if (pos == string::npos) cout << "No encontrado";
```

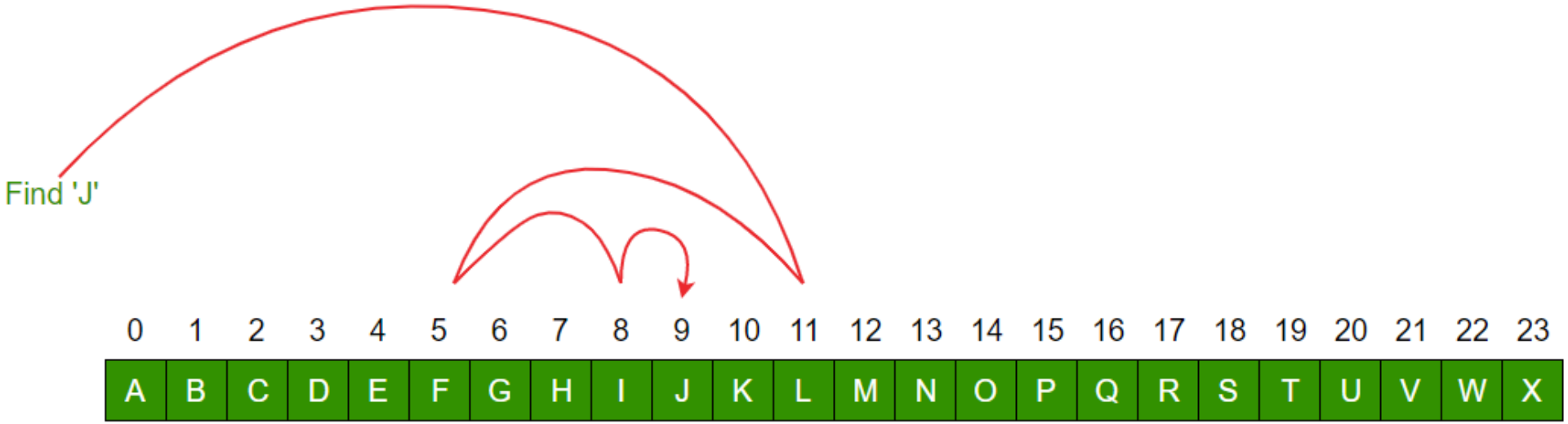
```
int n = 1; // n-ésima ocurrencia
auto pos = 0;
for (int i = 0; i < n; ++i) {
    pos = str.find(x, pos); // Buscar la ocurrencia
    if (pos == string::npos) {
        cout << "No se encontró la " << n << "ª ocurrencia de \"" << x << "\"." << endl;
        return; // Salir si no se encuentra
    }
    pos += x.length(); // posición a la siguiente después de la ocurrencia encontrada
}
cout << "La " << n << "ª ocurrencia de \"" << x << "\" se encuentra en la posición: " << (pos - x.length()) << endl;
```

Busqueda Binaria (Binary Search)

Algoritmo de Búsqueda Binaria es un algoritmo de búsqueda que se utiliza en un arreglo ordenado al dividir repetidamente el intervalo de búsqueda por la mitad (divide y vencerás). La idea de la búsqueda binaria es usar la información de que el arreglo **ya está ordenado** y reducir la complejidad temporal a O(log n).

Comparación entre busqueda lineal y busqueda binaria



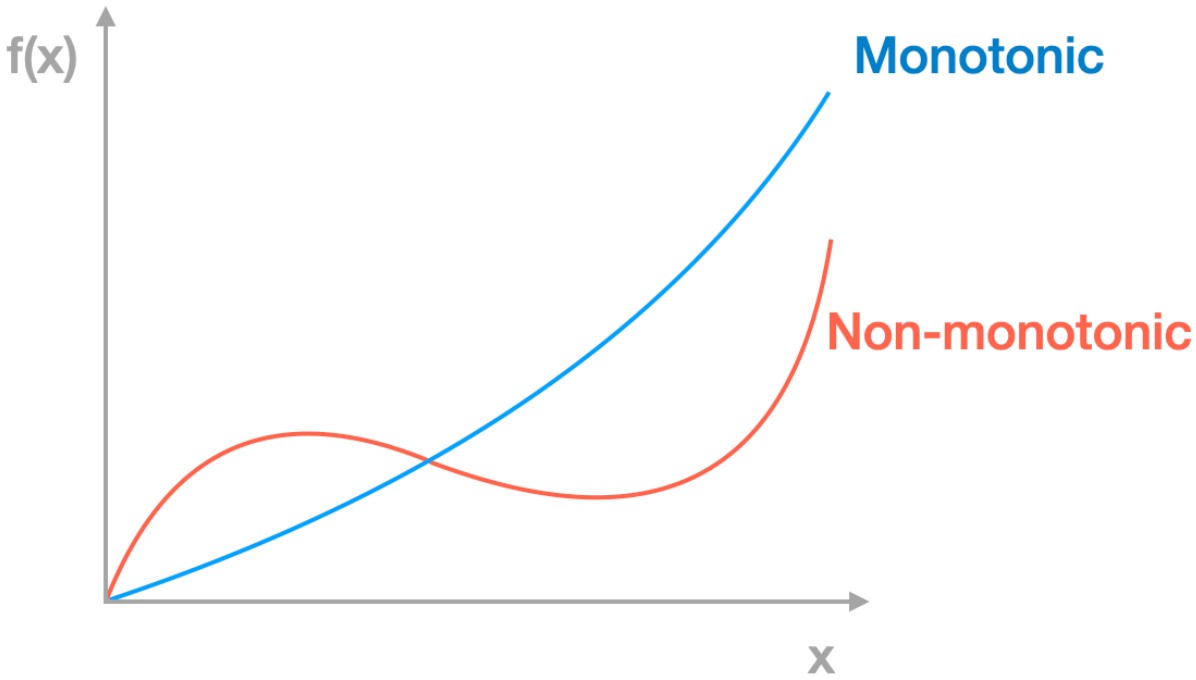


Funciones Monótonas (Búsqueda Binaria)

Una función monótona sigue un **orden particular**, lo que significa que su pendiente es siempre **no negativa** (creciente o constante) o **no positiva** (decreciente o constante).

En términos matemáticos, una función es monótona si, dados dos puntos `x1` y `x2` donde `x1` > `x2`, se cumple que f(`x1`) >= f(`x2`) (no decrecientes) o f(`x1`) <= f(`x2`) (no crecientes).

La **monotonía** es un requisito fundamental para aplicar la búsqueda binaria, ya que esta técnica solo se puede utilizar en un **arreglo ordenado**, que se comporta como una función monótona.



Algoritmo de Busqueda Binaria

```cpp
int binarySearch(vector<int>& v, int left, int right, int x) {
    while (left <= right) {
        int mid = left + (right - left) / 2; //soluciona 3 problemas: min, max y negativos

        if (v[mid] == x)
            return mid; // se encontro

        if (v[mid] < x)
            left = mid + 1;
        else
            right = mid - 1;
    }

  return -1;// no se encontro
}
```

Implentaciones de C++

Estas funciones son tienen una complejidad de tiempo de O(log N), con base de que el hecho de que el contenedor está ordenado.

```
auto it = lower_bound(v.begin(), v.end(), x);
if (it != v.end()) cout << "El primer elemento >= x es: " << *it;
```

```
auto it = upper_bound(v.begin(), v.end(), x);
if (it != v.end())  cout << "El primer elemento > x es: " << *it;
```

```
bool found = binary_search(v.begin(), v.end(), x);
if (found) cout << "El elemento x se encontró en el vector.";
else cout << "El elemento x no se encontró.";
```

Búsqueda Ternaria (Ternary Search)

La **búsqueda ternaria** es una técnica de búsqueda utilizada para determinar el **mínimo o máximo** de una **función unimodal**. La búsqueda ternaria divide el espacio de búsqueda en **tres partes** y luego elimina una de las tres partes para reducir el espacio de búsqueda.

Aunque la **complejidad de tiempo** de la búsqueda ternaria ($2 * \log_3 N$) es mayor que la de la búsqueda binaria ($\log_2 N$), se puede utilizar en varios casos donde la **búsqueda binaria** falla. La **búsqueda binaria** se utiliza para **funciones monótonas**, mientras que la **búsqueda ternaria** se utiliza para **funciones unimodales**. Se utiliza para encontrar el único **máximo o mínimo** de una función **cóncava** o **convexa**.

> Todos los problemas que pueden resolverse con búsqueda binaria también pueden resolverse con búsqueda ternaria, **pero no al contrario**.

Funciones Unimodales

Una **función unimodal** es aquella que tiene un valor m tal que es **monótonamente creciente** para x <= m y **monótonamente decreciente** para x >= m. El valor máximo de la función es f(m) y no existe otro máximo local.

En la **figura (A)**, el gráfico tiene un solo punto máximo y el resto del gráfico desciende desde allí. En la **figura (B)**, el gráfico tiene un único punto mínimo y el resto del gráfico asciende desde allí. Por lo tanto, una función con un **máximo global** o un **mínimo global** se considera una función unimodal.

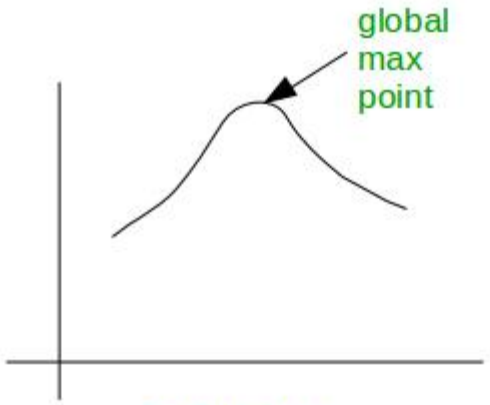


Figure (A)

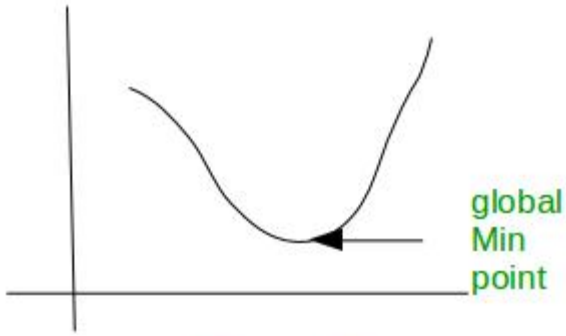


Figure (B)

Algoritmo de Busqueda ternaria para un vector entero

```cpp
int ternarySearch(vector<int>& v, int l, int r, int x) {
    while (r >= l) {
        int mid1 = l + (r - l) / 3;
        int mid2 = r - (r - l) / 3;
        if (v[mid1] == x) return mid1; // x found
        if (v[mid2] == x) return mid2; // x found
        if (x < v[mid1]) {
            r = mid1 - 1;
        } else if (x > v[mid2]) {
            l = mid2 + 1;
        } else {
            l = mid1 + 1;
            r = mid2 - 1;
        }
    }
    return -1;  // x not found
}
```

Algoritmo de Busqueda Ternaria para una función Entera

```
int ternarySearchI(int left, int right, int range) {
    while (right - left > range) {
        int mid1 = left + (right - left) / 3;
        int mid2 = right - (right - left) / 3;

        if (f(mid1) > f(mid2))
            left = mid1;
        else
            right = mid2;
    }
    int ans = INT_MAX;  // "infinito"
    // Busca el mínimo en el rango restante
    for (int i = left; i <= right; ++i)
        ans = min(ans, f(i));
    return ans;
}
```

Algoritmo de Busqueda Ternaria para una función decimal

```
double ternarySearchD(double l, double r, double precision) {
    while ((r - l) > precision) {
        double mid1 = l + (r - l) / 3.0;
        double mid2 = r - (r - l) / 3.0;

        // Aquí usamos a una función f(x) para evaluar los puntos.
        // Puede ser cualquier función convexa o cóncava.
        if (f(mid1) < f(mid2))
            l = mid1;
        else
            r = mid2;
    }

    return (l + (r - l) / 2.0);
}
```

Problemas

- **702C** Cellular Network ⮭
- **1978B** New Bakery ⮭

Referencias

- algo.monster. (s.f.).*Binary Search and Monotonic Function*. Recuperado de https://algo.monster/problems/binary-search-monotonic
- cp-algorithms. (2024). *Ternary Search*. Recuperado de https://cp-algorithms.com/num_methods/ternary_search.html
- GeekForGeeks. (2024). *Binary search*. Recuperado de https://www.geeksforgeeks.org/binary-search/
- Jafari, M. (2018). *Tutorial On Tof (Ternary Search)*. Recuperado de https://codeforces.com/blog/entry/60702
- mrityuanjay8vae. (). *Ternary Search for Competitive Programming*. Recuperado de https://www.geeksforgeeks.org/ternary-search-for-competitive-programming/
- Prakash, S. (2024). *Ternary search*.Recuperado de https://www.geeksforgeeks.org/ternary-search/
- USACO. (s.f.). *Binary Search*. Recuperado de https://usaco.guide/silver/binary-search?lang=cpp