TALLER PROGCOMP: TRACK BÁSICO BÚSQUEDA BINARIA

Gabriel Carmona Tabja

Universidad Técnica Federico Santa María, Università di Pisa

April 29, 2024

Part I

BUSCAR UN ELEMENTO

PROBLEMA

Problema

Dado una secuencia **ordenada** de elementos de tamaño *n*, se quiere saber el primer elemento (de izquierda a derecha) que cumpla cierta condición.

Problema

Determinar la posición del primer elemento que **NO SEA MENOR** a *x*.

Problema

Determinar la posición del primer elemento que **NO SEA MENOR** a *x*.

1	2	2	3	3	4	5	7

Problema

Determinar la posición del primer elemento que **NO SEA MENOR** a *x*.

1	2	2	3	3	4	5	7
F	F	F	F	F	Т	Т	Т

Problema

Determinar la posición del primer elemento que **NO SEA MENOR** a *x*.

1	2	2	3	3	4	5	7
F	F	F	F	F	Т	Т	Т

Problema

Determinar la posición del primer elemento que **NO SEA MENOR** a *x*.

1	2	2	3	3	4	5	7
F	F	F	F	F	Т	Т	Т

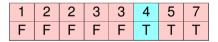
Problema

Determinar la posición del primer elemento que **NO SEA MENOR** a *x*.

1	2	2	3	3	4	5	7
F	F	F	F	F	Т	Т	Т

Problema

Determinar la posición del primer elemento que **NO SEA MENOR** a *x*.



TIPOS DE HACER BÚSQUEDA BINARIA

- ► Buscar el primer *T*
- ► Buscar el último *T*

BUSCAR EL PRIMER T

```
int main() {
   int l; // limite izquierdo
   int r; // limite derecho

while(1 < r) {
   int mid = (1 + r) / 2;
   if(condicion) {
      r = mid;
   } else {
      l = mid + 1;
   }

// l y r deberian ser iguales
   return 0;
}</pre>
```

BUSCAR EL PRIMER T

```
int main() {
   int l; // limite izquierdo
   int r; // limite derecho

while(1 < r) {
   int mid = (1 + r) / 2;
   if(condicion) {
      r = mid;
   } else {
      l = mid + 1;
   }

// l y r deberian ser iguales
   return 0;
}</pre>
```

¿Cuál sería la complejidad?

BUSCAR EL PRIMER T

```
int main() {
   int l; // limite izquierdo
   int r; // limite derecho

while(l < r) {
   int mid = (l + r) / 2;
   if(condicion) {
      r = mid;
   } else {
      l = mid + 1;
   }

// l y r deberian ser iguales
   return 0;
}</pre>
```

¿Cuál sería la complejidad? $O(f(n) \cdot \log_2(r - l + 1))$

BUSCAR EL PRIMER T - CASO LOWER_BOUND

```
bool condicion(vector< int > &nums, int mid, int x) {
     if(nums[mid] < x) return false;</pre>
     return true;
   int main() {
     int x; // numero de la condicion del lower_bound
     vector < int > nums; // se asume que lo llenas eventualmente
     int 1; // limite izquierdo
     int r; // limite derecho
     while (1 < r) {
11
      int mid = (1 + r) / 2;
      if(condicion(nums, mid, x)) {
13
       r = mid:
14
       } else {
15
         1 = mid + 1;
17
18
     // l y r deberian ser iguales
19
     return 0:
20
21
```

BUSCAR EL PRIMER T - CASO LOWER_BOUND

```
bool condicion(vector< int > &nums, int mid, int x) {
     if(nums[mid] < x) return false;</pre>
     return true;
   int main() {
     int x; // numero de la condicion del lower_bound
     vector < int > nums; // se asume que lo llenas eventualmente
     int 1: // limite izquierdo
     int r; // limite derecho
     while (1 < r) {
11
     int mid = (1 + r) / 2;
      if(condicion(nums, mid, x)) {
13
       r = mid:
14
       } else {
15
         1 = mid + 1;
17
18
     // l y r deberian ser iguales
19
     return 0:
20
21
```

¿Cuál sería la complejidad?

BUSCAR EL PRIMER T - CASO LOWER_BOUND

```
bool condicion(vector< int > &nums, int mid, int x) {
     if(nums[mid] < x) return false;</pre>
     return true;
   int main() {
     int x: // numero de la condicion del lower bound
     vector < int > nums; // se asume que lo llenas eventualmente
     int 1: // limite izquierdo
     int r; // limite derecho
     while (1 < r) {
11
     int mid = (1 + r) / 2;
      if(condicion(nums, mid, x)) {
13
       r = mid:
14
       } else {
15
         1 = mid + 1;
17
18
     // l y r deberian ser iguales
19
     return 0:
20
21
```

¿Cuál sería la complejidad? $O(1 \cdot \log_2(n))$

BUSCAR EL ÚLTIMO T

```
int main() {
     int 1; // limite izquierdo
    int r; // limite derecho
     while(1 < r) {
     int mid = (1 + r + 1) / 2;
      if(condicion) {
      1 = mid;
      } else {
8
         r = mid - 1;
10
11
     // l y r deberian ser iguales
     return 0;
13
14
```

BUSCAR EL ÚLTIMO T

```
int main() {
   int l; // limite izquierdo
   int r; // limite derecho

while(1 < r) {
   int mid = (1 + r + 1) / 2;
   if(condicion) {
        l = mid;
        } else {
        r = mid - 1;
        }

// 1 y r deberian ser iguales
   return 0;
}</pre>
```

¿Cuál sería la complejidad?

BUSCAR EL ÚLTIMO T

```
int main() {
   int l; // limite izquierdo
   int r; // limite derecho

while(1 < r) {
   int mid = (1 + r + 1) / 2;
   if(condicion) {
        l = mid;
        } else {
        r = mid - 1;
        }

// 1 y r deberian ser iguales
   return 0;
}</pre>
```

¿Cuál sería la complejidad? $O(f(n) \cdot \log_2(r - l + 1))$

References I