Busca Binária

Binary Search

Oficinas de Programação Competitiva

30 de novembro de 2025

Introdução

Motivação

Problema Clássico
Como você encontra uma palavra no dicionário?

Motivação

Problema Clássico
Como você encontra uma palavra no dicionário?

- Você abre o dicionário **no meio**
- Verifica se a palavra está antes ou depois
- Repete o processo na metade correta

Motivação

Problema Clássico
Como você encontra uma palavra no dicionário?

- Você abre o dicionário **no meio**
- Verifica se a palavra está antes ou depois
- Repete o processo na metade correta

Isso é Busca Binária

O que é Busca Binária?

Busca Binária é um algoritmo eficiente para encontrar um elemento em uma lista ordenada.

O que é Busca Binária?

Busca Binária é um algoritmo eficiente para encontrar um elemento em uma lista ordenada.

Conceitos

- Espaço de busca
- Atualização do espaço de busca

O que é Busca Binária?

Busca Binária é um algoritmo eficiente para encontrar um elemento em uma lista ordenada.

Conceitos

- Espaço de busca
- Atualização do espaço de busca

Complexidade
A cada iteração, diminuimos metade do espaço de busca, resultando em complexidade $O(\log n)$.

Implementação Clássica

Problema

LeetCode X
Dado um array a de tamanho n, responda q perguntas do tipo: dado um número x, elemento x está em a ou não.

Restrições: $n, q \leq 10^5$ e $a_i \leq 10^9$, $1 \leq i \leq n$

Problema

LeetCode X
Dado um array a de tamanho n, responda q perguntas do tipo: dado um número x, elemento x está em a ou não.

Restrições: $n, q \leq 10^5$ e $a_i \leq 10^9$, $1 \leq i \leq n$

- Usamos as variáveis l e r como indicadores do espaço de busca.
- Enquanto o tamanho do espaço de busca $(r - l + 1)$ é maior que 1, comparamos o elemento do meio do espaço de busca com x.
- Atualizamos o espaço de busca.

Implementação Clássica

```
1  int l = 0;
2  int r = n-1;
3  while (l <= r) {
4      int m = (r + l) / 2;
5      if (m < x) {
6          l = m + 1;
7      } else {
8          r = m;
9      }
10 }
11 if (a[l] == x) {
12     cout << "YES\n";
13 } else {
14     cout << "NO\n";
15 }
```

Detalhes de Implementação

Cuidados

- `m = (r + l) / 2`: arredonda para baixo
- Se atualizarmos o espaço de busca com `l = m`, teríamos um loop infinito.
- Para arredondar para cima: `m = (r + l + 1) / 2`

Funções da STL

Busca Binária na STL

Por que usar funções prontas?
Os casos de borda da implementação clássica podem ser chatos.

As funções prontas são:

- Menos propenso a erros (casos de borda, loops infinitos)
- Código mais limpo e legível
- Testado e otimizado

Busca Binária na STL

Por que usar funções prontas?
Os casos de borda da implementação clássica podem ser chatos.

As funções prontas são:

- Menos propenso a erros (casos de borda, loops infinitos)
- Código mais limpo e legível
- Testado e otimizado

As funções

`lower_bound` Primeiro elemento $\geq$ valor

`upper_bound` Primeiro elemento $>$ valor

Revisão de iteradores?

Iteradores são objetos que apontam para elementos em um container (como um `vector`, etc.) e permitem percorrê-los.

Revisão de iteradores?

Iteradores são objetos que apontam para elementos em um container (como um `vector`, etc.) e permitem percorrê-los.

Analogia
Pense em um iterador como um **ponteiro inteligente** que sabe navegar pelo vetor.

Revisão de iteradores?

Iteradores são objetos que apontam para elementos em um container (como um `vector`, etc.) e permitem percorrê-los.

Analogia
Pense em um iterador como um **ponteiro inteligente** que sabe navegar pelo vetor.

Por que usar?

- Funções da STL (como `lower_bound`) retornam iteradores

Operações com Iteradores

```
1 vector<int> v = {10, 20, 30, 40, 50};
```

Operações com Iteradores

```
vector<int> v = {10, 20, 30, 40, 50};
```

1. Obter Iteradores

```
auto inicio = v.begin();   // aponta para v[0]
auto fim = v.end();        // aponta DEPOIS do ultimo
```

Operações com Iteradores

```
vector<int> v = {10, 20, 30, 40, 50};
```

1. Obter Iteradores

```
auto inicio = v.begin();   // aponta para v[0]
auto fim = v.end();        // aponta DEPOIS do ultimo
```

Importante

`v.end()` **NÃO** aponta para o último elemento, mas para a posição **depois** do último! É usado como limite superior em loops.

Operações com Iteradores

```
1 vector<int> v = {10, 20, 30, 40, 50};
2 auto it = v.begin();
```

Operações com Iteradores

```
vector<int> v = {10, 20, 30, 40, 50};
auto it = v.begin();
```

2. Dereferenciar (acessar o valor)

```
cout << *it << "\n";        // 10 (valor apontado)
```

Operações com Iteradores

```cpp
vector<int> v = {10, 20, 30, 40, 50};
auto it = v.begin();
```

2. Dereferenciar (acessar o valor)

```cpp
cout << *it << "\n";       // 10 (valor apontado)
```

3. Deslocar

```cpp
it++;              // avanca 1 posicao
it += 2;           // avanca 2 posicoes
it--;              // volta 1 posicao
cout << *it << "\n";  // depende dos deslocamentos
```

Operações com Iteradores

```
vector<int> v = {10, 20, 30, 40, 50};
//                0   1   2   3   4
```

4. Obter Índice

```
auto it = v.begin() + 2;  // aponta para v[2]
int indice = it - v.begin();
cout << indice << "\n";   // 2

// Verificar o valor
cout << *it << "\n";      // 30
cout << v[indice] << "\n"; // 30 (mesmo elemento)
```

Operações com Iteradores

```
vector<int> v = {10, 20, 30, 40, 50};
//                0   1   2   3   4
```

4. Obter Índice

```
auto it = v.begin() + 2;  // aponta para v[2]
int indice = it - v.begin();
cout << indice << "\n";   // 2

// Verificar o valor
cout << *it << "\n";      // 30
cout << v[indice] << "\n"; // 30 (mesmo elemento)
```

Relação

`it - v.begin()` nos dá o índice do elemento no vetor.

lower_bound e upper_bound

```
1 vector<int> v = {1, 2, 4, 4, 4, 7, 9};
2 //               0  1  2  3  4  5  6
```

lower_bound e upper_bound

```cpp
vector<int> v = {1, 2, 4, 4, 4, 7, 9};
//               0  1  2  3  4  5  6
```

```cpp
// lower_bound: primeiro >= valor
auto it1 = lower_bound(v.begin(), v.end(), 4);
cout << *it1 << "\n";          // 4
cout << it1 - v.begin() << "\n";  // 2 (posicao)
```

lower_bound e upper_bound

```
vector<int> v = {1, 2, 4, 4, 4, 7, 9};
//               0  1  2  3  4  5  6
```

```
// lower_bound: primeiro >= valor
auto it1 = lower_bound(v.begin(), v.end(), 4);
cout << *it1 << "\n";          // 4
cout << it1 - v.begin() << "\n";  // 2 (posicao)
```

```
// upper_bound: primeiro > valor
auto it2 = upper_bound(v.begin(), v.end(), 4);
cout << *it2 << "\n";          // 7
cout << it2 - v.begin() << "\n";  // 5 (posicao)
```

Cuidados com Iteradores

Runtime Error: Dereferenciar end()

```
1 vector<int> v = {1, 2, 3};
2 auto it = lower_bound(v.begin(), v.end(), 10);
3
4 // ERRADO! it == v.end()
5 cout << *it << "\n";  // Runtime Error!
```

Cuidados com Iteradores

Runtime Error: Dereferenciar end()

```
1 vector<int> v = {1, 2, 3};
2 auto it = lower_bound(v.begin(), v.end(), 10);
3
4 // ERRADO! it == v.end()
5 cout << *it << "\n";  // Runtime Error!
```

Forma Correta

```
1 if (it != v.end()) {
2     cout << *it << "\n";  // Seguro
3 } else {
4     cout << "Elemento nao encontrado\n";
5 }
```

Obtendo Outras Comparações

Desejado	**Como obter**
Primeiro $\geq x$	`lower_bound(x)`
Primeiro $> x$	`upper_bound(x)`
Último $< x$	`lower_bound(x) - 1`
Último $\leq x$	`upper_bound(x) - 1`

Obtendo Outras Comparações

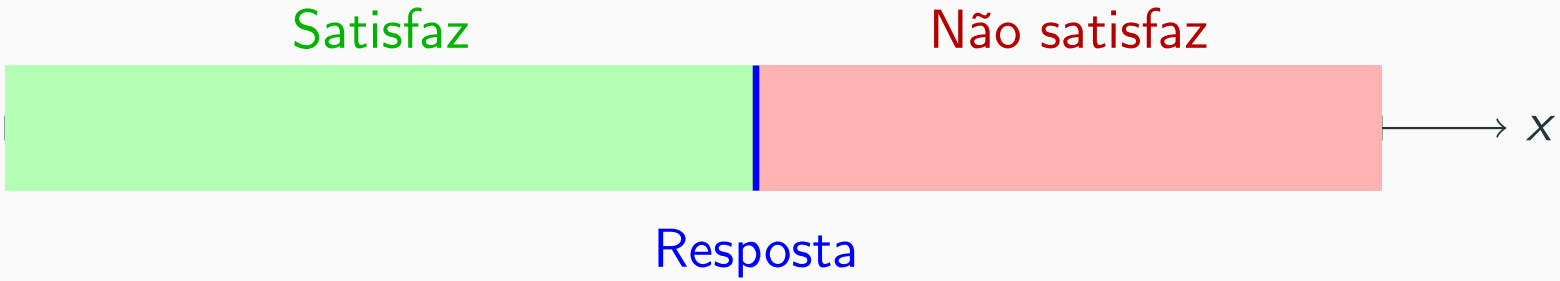
Desejado	Como obter
Primeiro $\geq x$	`lower_bound(x)`
Primeiro $> x$	`upper_bound(x)`
Último $< x$	`lower_bound(x) - 1`
Último $\leq x$	`upper_bound(x) - 1`

Atenção

Ao fazer `it - 1`, verifique se `it != v.begin()`!

Busca Binária na Resposta

Quando podemos usar busca binária?



- Queremos encontrar a **fronteira** entre as duas regiões
- A busca binária nos permite fazer isso em $O(\log n)$ verificações

Expandindo o Conceito

Insight Importante
Busca binária não funciona apenas em arrays ordenados!

Expandindo o Conceito

Insight Importante
Busca binária não funciona apenas em arrays ordenados!

Definition
Podemos usar busca binária em qualquer **função monotônica**:

- Se $f(x)$ é verdadeiro, então $f(x+1)$ também é (ou vice-versa)
- Queremos encontrar a fronteira entre verdadeiro e falso

Expandindo o Conceito

Insight Importante
Busca binária não funciona apenas em arrays ordenados!

Definition
Podemos usar busca binária em qualquer **função monotônica**:

- Se $f(x)$ é verdadeiro, então $f(x+1)$ também é (ou vice-versa)
- Queremos encontrar a fronteira entre verdadeiro e falso

Aplicação
Transformar problemas de **otimização** em problemas de **decisão**:

- Otimização: "Qual o máximo/mínimo valor possível?"
- Decisão: "É possível obter valor x?"

Problema: CSES 1620 - Factory Machines

Enunciado
Você tem n máquinas. A máquina i produz um produto em k_i segundos.

Qual o tempo mínimo para produzir t produtos?

Problema: CSES 1620 - Factory Machines

Enunciado
Você tem n máquinas. A máquina i produz um produto em k_i segundos.

Qual o tempo mínimo para produzir t produtos?

Example
Entrada: $n = 3$, $t = 7$, tempos $= [3, 2, 5]$

Em 8 segundos:

- Máquina 1: $\lfloor 8/3 \rfloor = 2$ produtos
- Máquina 2: $\lfloor 8/2 \rfloor = 4$ produtos
- Máquina 3: $\lfloor 8/5 \rfloor = 1$ produto
- Total: $2 + 4 + 1 = 7$ produtos

Resposta: 8 segundos

Estratégia de Solução

1. **Problema de Otimização:**
   "Qual o tempo mínimo para produzir t produtos?"

Estratégia de Solução

1. **Problema de Otimização:**
   "Qual o tempo mínimo para produzir t produtos?"
2. **Transformar em Decisão:**
   "É possível produzir t produtos em x segundos?"

Estratégia de Solução

1. **Problema de Otimização:**
   "Qual o tempo mínimo para produzir t produtos?"
2. **Transformar em Decisão:**
   "É possível produzir t produtos em x segundos?"
3. **Observar Monotonicidade:**
   Se é possível em x segundos, também é possível em $x + 1$ segundos.

Estratégia de Solução

1. **Problema de Otimização:**
   "Qual o tempo mínimo para produzir t produtos?"
2. **Transformar em Decisão:**
   "É possível produzir t produtos em x segundos?"
3. **Observar Monotonicidade:**
   Se é possível em x segundos, também é possível em $x + 1$ segundos.
4. **Busca Binária:**
   Buscar o menor x que torna a resposta verdadeira.

Implementação - CSES 1620

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

int t, n;
vector<int> k;

bool check(long long x) {
    long long products = 0;
    for (int i = 0; i < n; i++) {
        products += x / k[i];
    }
    return products >= t;
};
int main() {
    cin >> n >> t;
    k = vector<int>(n);
    for (int i = 0; i < n; i++) {
        cin >> k[i];
    }
    // continua...
```

Implementação - CSES 1620 (cont.)

```
1       // ... continuacao
2       long long l = 1;
3       long long r = (long long)*min_element(k.begin(), k.end()
            )*t;
4       while (l <= r) {
5           long long m = (r + l) / 2;
6           if (check(m)) {
7               r = m-1;
8           } else {
9               l = m+1;
10          }
11      }
12      cout << l << "\n";
13  }
```

Implementação - CSES 1620 (cont.)

```
1       // ... continuacao
2       long long l = 1;
3       long long r = (long long)*min_element(k.begin(), k.end()
            )*t;
4       while (l <= r) {
5           long long m = (r + l) / 2;
6           if (check(m)) {
7               r = m-1;
8           } else {
9               l = m+1;
10          }
11      }
12      cout << l << "\n";
13  }
```

Cuidado!
long long é muito importante para esse problema!

Quando Usar Busca Binária na Resposta?

Sinais para Identificar

- Problema pede "máximo/mínimo valor"
- Existe uma função monotônica relacionada à resposta
- Verificar se um valor específico funciona é "fácil"
- O espaço de busca é muito grande para força bruta

Quando Usar Busca Binária na Resposta?

Sinais para Identificar

- Problema pede "máximo/mínimo valor"
- Existe uma função monotônica relacionada à resposta
- Verificar se um valor específico funciona é "fácil"
- O espaço de busca é muito grande para força bruta

Palavras-chave Comuns

- "Minimize o máximo..."
- "Maximize o mínimo..."
- "Qual o menor tempo para..."
- "Qual a maior distância possível..."

Dicas de Implementação

1. Use `long long` quando necessário (overflow!)
2. Cuidado com a condição de parada: `while (l <= r)`
3. Verifique se iteradores são válidos antes de dereferenciar
4. Sempre verifique a monotonicidade da função

Perguntas?