Busca em Memória Principal

Estrutura de Dados e Algoritmos – Ciência da Computação



Prof. Daniel Saad Nogueira Nunes

IFB – Instituto Federal de Brasília, Campus Taguatinga



Sumário

- Introdução
- 2 Busca Binária
- Análise



Sumário

Introdução



Introdução

- Dada uma sequência de entrada, como buscar um determinado elemento?
- Podemos utilizar um laço de repetição.



Busca Sequencial



Busca Sequencial

- O algoritmo de busca sequencial funciona.
- No pior caso ele tem que varrer o vetor inteiro.
- Tempo $\Theta(n)$.
- Temos como fazer melhor?



Busca Sequencial Otimizada

- Se a entrada já estiver ordenada, podemos melhor a busca sequencial.
- Assim que encontramos um elemento que é maior do que a chave, sabemos que ela não estará na sequência, pois a entrada está ordenada.



Busca Sequencial

```
/** Pré-requisitos, v está ordenado **/
    int busca_sequencial_otimizada(int *v, size_t n, int key) {
      int i;
      for (i = 0; i < n; i++) {
        if (key < v[i]) {
          break;
        } else if (v[i] == key) {
          return i;
10
      return -1;
11
12
```



Busca Sequencial Otimizada

- Conseguimos salvar algum tempo caso detectemos que a chave é menor do que algum elemento durante a varredura na busca sequencial otimizada.
- Contudo, no pior caso, ainda temos que varrer o vetor inteiro.
- Como tirar mais vantagem do fato da entrada estar ordenada?



Busca Sequencial Otimizada

- Conseguimos salvar algum tempo caso detectemos que a chave é menor do que algum elemento durante a varredura na busca sequencial otimizada.
- Contudo, no pior caso, ainda temos que varrer o vetor inteiro.
- Como tirar mais vantagem do fato da entrada estar ordenada?
- Busca Binária!



Sumário



- Levando em consideração que o vetor está ordenado, podemos efetuar a busca binária.
- Ela funciona da seguinte forma. Suponha que a sequência V[0,n-1].
- Inicialmente calcula-se o ponto médio da sequência: $m \leftarrow \lfloor \frac{n}{2} \rfloor$.
- \bullet Se a chave corresponde à $V[m], {\tt ent} \tilde{\tt ao}$ encontramos a chave no ponto médio.



- ullet Se a chave não é igual ao elemento V[m], temos duas opções.

 - ② A chave é maior do que V[m].
- No primeiro caso, sabemos que se a chave se encontra em V, ela deve estar no intervalo V[0,m-1].
- No segundo, concluímos que se a chave está em V, ela se encontra em V[m+1,n-1].
- Isso é permitido pois sabemos que o vetor está ordenado. Logo, sabemos que todos os elementos à esquerda de V[m] são menores ou iguais à V[m]. Simetricamente, todos os elementos à direita de V[m] são maiores ou iguais a V[m].



- Caso a chave não corresponda ao elemento V[m], continuamos a busca no subvetor à esquerda de V[m] ou à direita de V[m] utilizando a mesma estratégia!
- Descartamos metade dos elementos com uma única comparação!

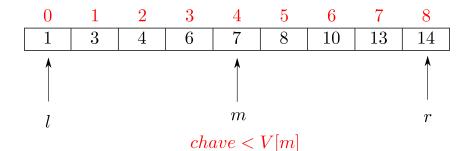


$$chave = 4$$

0	1	2	3	4	5	6	7	8
1	3	4	6	7	8	10	13	14

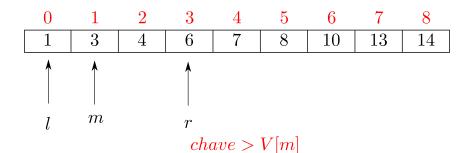


$$chave = 4$$



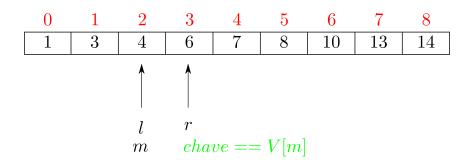


$$chave = 4$$





$$chave = 4$$



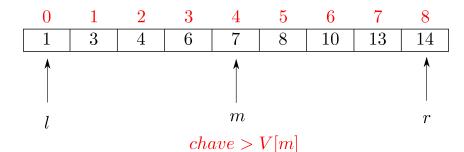


$$chave = 11$$

0	1	2	3	4	5	6	7	8
1	3	4	6	7	8	10	13	14

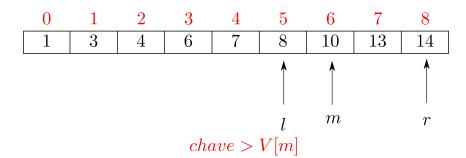


$$chave = 11$$



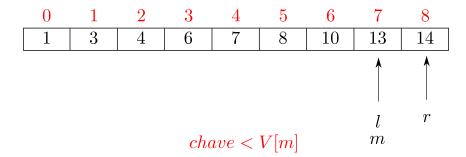


$$chave = 11$$





$$chave = 11$$





Exemplo

figuras/busca-binaria-2-5.pdf





Busca Binária: Recursão

- Também é possível implementar a busca binária recursivamente.
- Casos base: subvetor vazio ou chave encontrada.



Busca Binária: Recursão

```
int busca_binaria_rec(int *v, size_t n, int key) {
   return busca_binaria_rec_helper(v, 0, n - 1, key);
}
```



Busca Binária: Recursão

```
int busca_binaria_rec_helper(int *v, int 1, int r, int key) {
1
      if (1 > r) /**Caso base, vetor vazio**/
        return -1;
3
      int mid = 1 + (r - 1) / 2;
      /**Caso base, a chave é iqual ao elemento central**/
      if (key == v[mid])
        return mid;
      if (key < v[mid]) /**Recursão na metade inferior**/
        return busca_binaria_rec_helper(v, 1, mid - 1, key);
      else /**Recursão na metade superior**/
10
        return busca_binaria_rec_helper(v, mid + 1, r, key)
11
12
```



Sumário

Análise



Análise

- No pior caso, a busca binária descarta metade dos elementos a cada comparação.
- ullet $\Theta(\lg n)$ passos.

Número de Comparações					
\overline{n}	Busca Sequencial	Busca Binária			
2^1	$2^1 - 1$	2			
2^2	$2^2 - 1$	3			
2^3	$2^3 - 1$	4			
2^4	$2^4 - 1$	5			
2^{10}	$2^{10} - 1$	11			
2^{20}	$2^{20}-1$	21			
2^{30}	$2^{30}-1$	31			