9002 — Aula 24
Algoritmos e Programação de Computadores

Instituto de Engenharia – UFMT

Segundo Semestre de 2014

12 de dezembro de 2014

Roteiro

Revisão

- A idéia é que a solução de um problema pode ser expressa da seguinte forma:
  - Definimos a solução para os casos básicos;
  - Definimos como resolver o problema geral utilizando soluções do mesmo problema só que para casos menores.

Divisão e Conquista

- Dividir o problema em um certo número de subproblemas;
- Conquistar os subproblemas resolvendo-os recursivamente. Se o tamanho do subproblema é suficientemente pequeno (caso base) resolva-o diretamente.
- Combine as soluções dos subproblemas para encontrar uma solução do problema original.

Exponenciação binária

Podemos utilizar esta abordagem para definir a potência um outro algoritmo de exponenciação.
Neste caso, x^n é calculado:

- Caso básico:
  - Se $n = 0$ então $x^n = 1$.
- Caso Geral:
  - Se $n > 0$ e é par, então $x^n = (x^{n/2})^2$.
  - Se $n > 0$ e é ímpar, então $x^n = x(x^{(n-1)/2})^2$.

Note como no caso geral definimos a solução do caso maior em termos de casos menores.

Exponenciação binária

Podemos utilizar esta abordagem para definir a potência um outro algoritmo de exponenciação.
Neste caso, x^n é calculado:

- Caso básico:
  - Se $n = 0$ então $x^n = 1$.
- Caso Geral:
  - Se $n > 0$ e é par, então $x^n = (x^{n/2})^2$.
  - Se $n > 0$ e é ímpar, então $x^n = x(x^{(n-1)/2})^2$.

Note como no caso geral definimos a solução do caso maior em termos de casos menores.

Exponenciação binária

Este algoritmo é mais eficiente do que o clássico. Por que? Quantas chamadas recursivas o algoritmo pode fazer?

```
double power(double a, int n)
{
    double m;
    //Caso base
    if(n == 0)
        return 1;
    if(n == 1)
        return a;

    m = power2(a, n/2);

    if(n % 2)
        return m*m*a;
    else
        return m*m;
}
```

Exponenciação binária

Este algoritmo é mais eficiente do que o clássico. Por que? Quantas chamadas recursivas o algoritmo pode fazer?

```
double power(double a, int n)
{
    double m;
    //Caso base
    if(n == 0)
        return 1;
    if(n == 1)
        return a;

    m = power2(a, n/2);

    if(n % 2)
        return m*m*a;
    else
        return m*m;
}
```

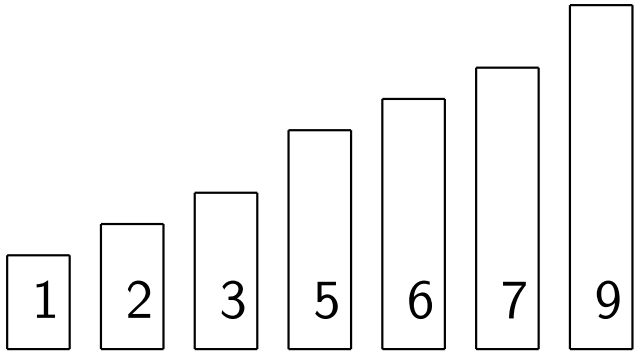
Exponenciação: Algoritmo clássico

```
double power(double a, int n)
{
    if(n == 0) return 1;
    if(n == 1) return a;
    return a*power(a, n-1);
}
```

Exponenciação binária

- Na exponenciação binária, a cada chamada recursiva o valor de n é dividido por 2. Ou seja, a cada chamada recursiva, o valor de n decai para pelo menos a metade.
- Usando divisões inteiras faremos no máximo $\lceil(\log_2 n)\rceil + 1$ chamadas recursivas.
- Enquanto isso, o algoritmo clássico executa faz n multiplicações.

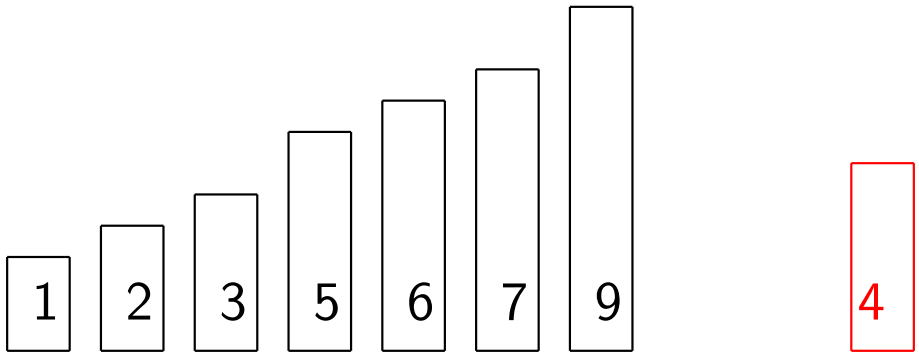
Busca Binária



Problema

Há coleção de elementos em ordem (por exemplo, **1, 2, 3, 5, 6, 7 e 9**) e uma chave de busca (por exemplo, **4**). Deseja-se verificar se algum elemento desta coleção possui o mesmo valor da chave de busca.

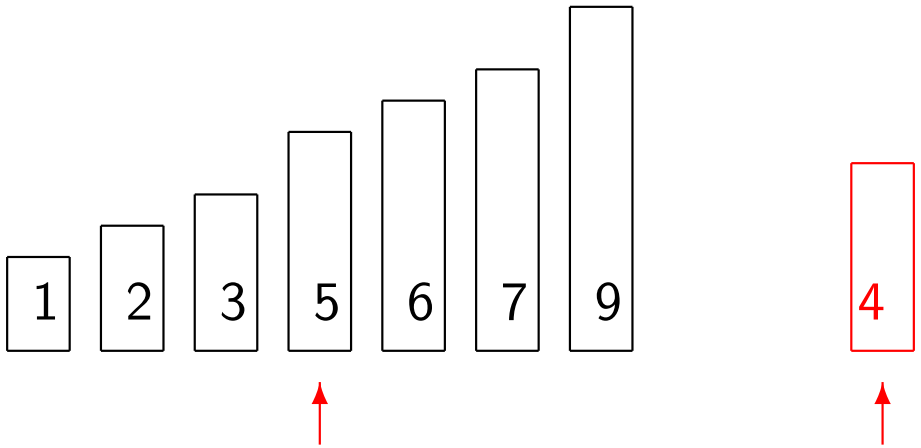
Busca Binária



Ideia

1. Comparar o elemento do meio do vetor com a chave.
2. Redefinir o vetor e buscar recursivamente pela chave.
3. Parar quando há um só elemento (base da recursão).

Busca Binária



Ideia

1. Comparar o elemento do meio do vetor com a chave.
2. Redefinir o vetor e buscar recursivamente pela chave.
3. Parar quando há um só elemento (base da recursão).

Busca Binária

Ideia

1. Comparar o elemento do meio do vetor com a chave.
2. Redefinir o vetor e buscar recursivamente pela chave.
3. Parar quando há um só elemento (base da recursão).

Busca Binária

Ideia

1. Comparar o elemento do meio do vetor com a chave.
2. Redefinir o vetor e buscar recursivamente pela chave.
3. Parar quando há um só elemento (base da recursão).

Busca Binária

Ideia

1. Comparar o elemento do meio do vetor com a chave.
2. Redefinir o vetor e buscar recursivamente pela chave.
3. Parar quando há um só elemento (base da recursão).

Busca Binária

Ideia

1. Comparar o elemento do meio do vetor com a chave.
2. Redefinir o vetor e buscar recursivamente pela chave.
3. Parar quando há um só elemento (base da recursão).

Busca Binária

Ideia

1. Comparar o elemento do meio do vetor com a chave.
2. Redefinir o vetor e buscar recursivamente pela chave.
3. Parar quando há um só elemento (base da recursão).

Soma de um vetor

Vamos implementar!

```
int buscaBinaria(int v[], int inicio, int fim, int chave)
{
    int m = (inicio+fim)/2;

    if(inicio > fim) return 0;
    if(inicio == fim) return  (v[m] == chave);

    if(chave == v[m])
        return 1;
    if(chave < v[m])
        return buscaBinaria(v, inicio, m-1, chave);
    else
        return buscaBinaria(v, m+1, fim, chave);

}
```

Soma de um vetor

Vamos implementar!

```
int buscaBinaria(int v[], int inicio, int fim, int chave)
{
    int m = (inicio+fim)/2;

    if(inicio > fim) return 0;
    if(inicio == fim) return  (v[m] == chave);

    if(chave == v[m])
        return 1;
    if(chave < v[m])
        return buscaBinaria(v, inicio, m-1, chave);
    else
        return buscaBinaria(v, m+1, fim, chave);

}
```

Recursão com várias chamadas

- Não há necessidade da função recursiva ter apenas uma chamada para si própria.
- A função pode fazer várias chamadas para si própria.
- A função pode ainda fazer chamadas recursivas indiretas. Neste caso a função 1, por exemplo, chama uma outra função 2 que por sua vez chama a função 1.

Soma de um vetor

- Dado um vetor, vamos definir $S(i, n)$ como a soma de n elementos a partir da posição i.
- Com isso temos a seguinte definição recursiva para a soma dos elementos de um vetor:
  - Se $n = 1$ então $S(i, n) = v[i]$.
  - Se $n > 1$ então $S(i, n) = S(i, \lceil n/2 \rceil) + S(i + \lceil n/2 \rceil, \lfloor n/2 \rfloor)$.
- Observações
  - $n = \lceil n/2 \rceil + \lfloor n/2 \rfloor$.
  - Dada uma posição i e quantidade $x = \lceil n/2 \rceil$ de elementos incluindo x, a próxima posição a ser considerada será $(i + x)$.
- Para computarmos a soma de todos os elementos de um vetor com n elementos, devemos calcular $S(0, n)$.

Soma de um vetor

- Dado um vetor, vamos definir $S(i, n)$ como a soma de n elementos a partir da posição i.
- Com isso temos a seguinte definição recursiva para a soma dos elementos de um vetor:
  - Se $n = 1$ então $S(i, n) = v[i]$.
  - Se $n > 1$ então $S(i, n) = S(i, \lceil n/2 \rceil) + S(i + \lceil n/2 \rceil, \lfloor n/2 \rfloor)$.
- Observações
  - $n = \lceil n/2 \rceil + \lfloor n/2 \rfloor$.
  - Dada uma posição i e quantidade $x = \lceil n/2 \rceil$ de elementos incluindo x, a próxima posição a ser considerada será $(i + x)$.
- Para computarmos a soma de todos os elementos de um vetor com n elementos, devemos calcular $S(0, n)$.

Soma de um vetor

- Dado um vetor, vamos definir $S(i, n)$ como a soma de n elementos a partir da posição i.
- Com isso temos a seguinte definição recursiva para a soma dos elementos de um vetor:
  - Se $n = 1$ então $S(i, n) = v[i]$.
  - Se $n > 1$ então $S(i, n) = S(i, \lceil n/2 \rceil) + S(i + \lceil n/2 \rceil, \lfloor n/2 \rfloor)$.
- Observações
  - $n = \lceil n/2 \rceil + \lfloor n/2 \rfloor$.
  - Dada uma posição i e quantidade $x = \lceil n/2 \rceil$ de elementos incluindo x, a próxima posição a ser considerada será $(i + x)$.
- Para computarmos a soma de todos os elementos de um vetor com n elementos, devemos calcular $S(0, n)$.

Soma de um vetor

O código recursivo segue abaixo. Basta implementarmos funções para calcular o teto e o chão da divisão de dois números.

```
int soma(int v[], int i, int n){
  if(n == 1)
    return v[i];
  else{
    return soma(v, i, teto(n,2)) +
      soma(v, i+teto(n,2), chao(n,2));
  }
}
```

Soma de um vetor

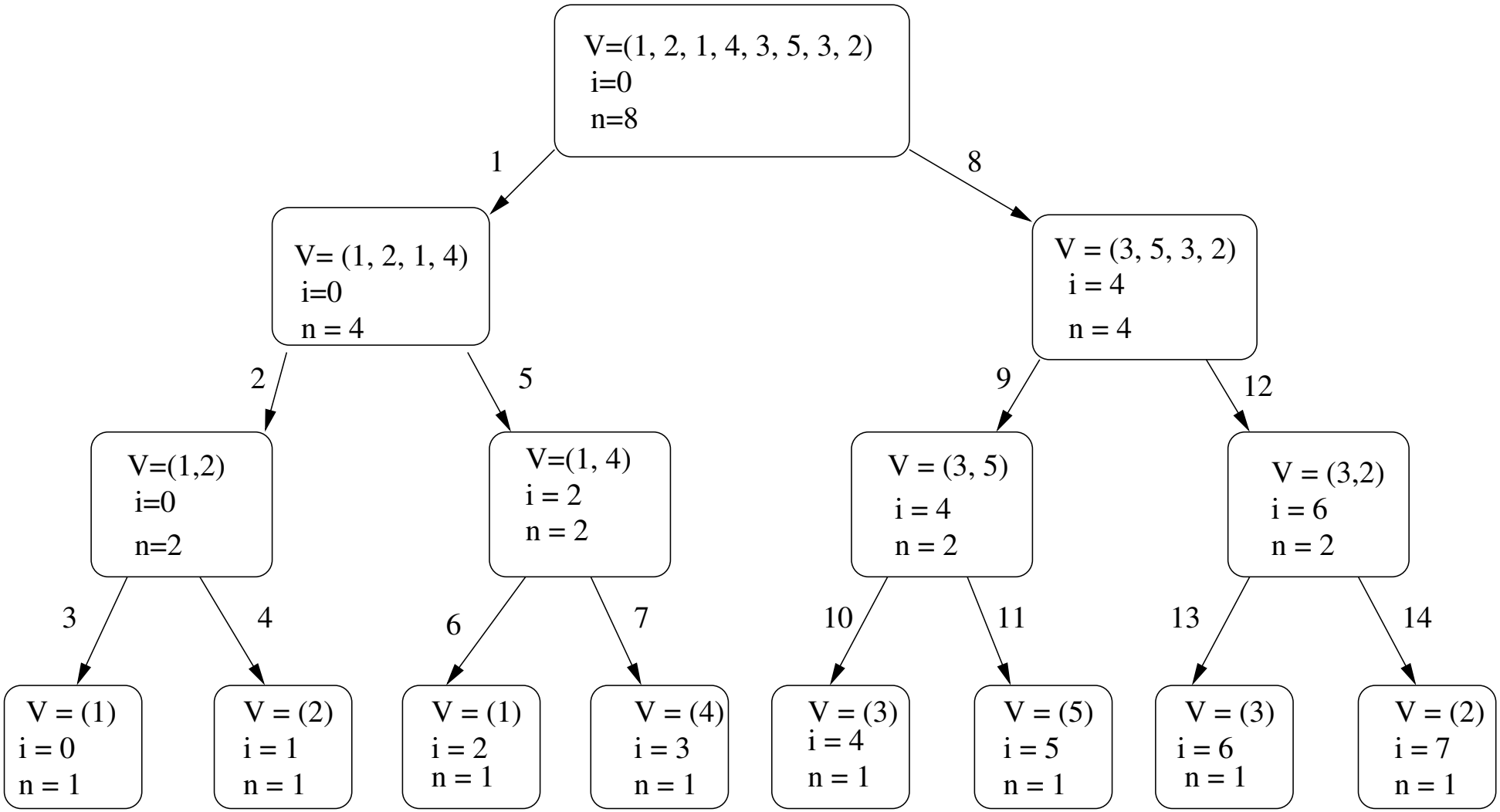
```
int teto(int numerador, int denominador){
  if(numerador % denominador == 0) //se divisão for inteira
    return (numerador/denominador);
  else
    return (numerador/denominador + 1);
}


int chao(int numerador, int denominador){
    return (numerador/denominador);
}
```

Soma de um vetor

- Abaixo temos um exemplo de execução da função para o vetor v=[1, 2, 1, 4, 3, 5, 3, 2].
- Há uma indicação da ordem em que ocorrem as sucessivas chamadas recursivas.
- Em cada balão é apresentado apenas a parte do vetor que está sendo considerada pela função naquele momento.
- A chamada da função deve ser: **somar(v, 0, 8)**.

Soma de um vetor



Exercício

1. Escreva uma função recursiva que encontra o maior (ou um dos maiores) elemento de um vetor.
2. Escreva uma função recursiva que verifica se os elementos de um vetor formam um P.A. de razão 2.