Memoização
Programação Dinâmica

Responsáveis:	Diego Brian 14/0136371	Filipe Teixeira 14/0139486
	Paulo Cunha 10/0118577	Pedro Aurélio 14/0158103
	Rodrigo Guimarães 14/0170740	

Sumário

- Compreensão;
- Implementação:
  - Fibonacci;
  - Radares – URI;
  - Sequência Crescente Máxima;
  - Particionamento de Inteiros;
  - Árvore Binária de Pesquisa.
- Referências.

Compreensão

- Terminologia padrão trás confusão:

Compreensão

- Terminologia padrão trás confusão:
  - Programação: refere-se a um planejamento das ações;

Compreensão

- Terminologia padrão trás confusão:
  - Programação: refere-se a um planejamento das ações;
  - Dinâmico: refere-se a processos multiestágios, da física clássica.

Compreensão

- Terminologia padrão trás confusão:
  - Programação: refere-se a um planejamento das ações;
  - Dinâmico: refere-se a processos multiestágios, da física clássica.
- Terminologia direta:

Compreensão

- Terminologia padrão trás confusão:
  - Programação: refere-se a um planejamento das ações;
  - Dinâmico: refere-se a processos multiestágios, da física clássica.
- Terminologia direta:
  - Memoização: armazenamento de dados para consultas futuras.

Compreensão

- Terminologia padrão trás confusão:
  - Programação: refere-se a um planejamento das ações;
  - Dinâmico: refere-se a processos multiestágios, da física clássica.
- Terminologia direta:
  - Memoização: armazenamento de dados para consultas futuras.
    - Facilmente ligado à memorização.

Compreensão

- Relacionado à problemas de otimização;

Compreensão

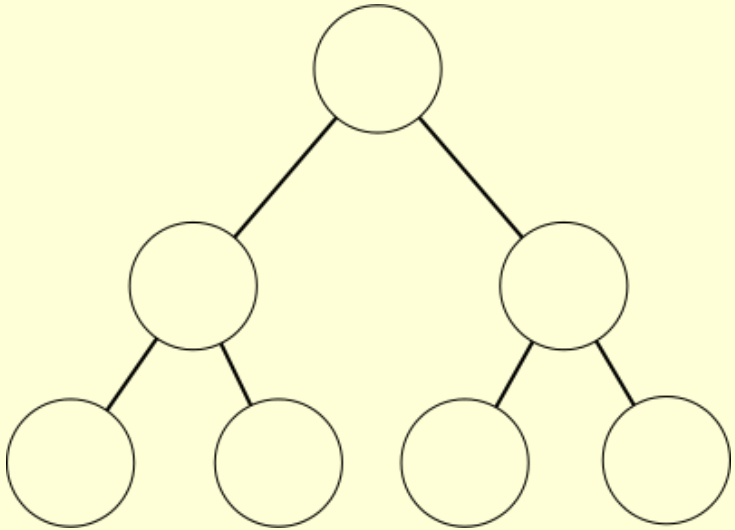
- Relacionado à problemas de otimização;
  - Análise de subproblemas mais simples;

Compreensão

- Relacionado à problemas de otimização;
  - Análise de subproblemas mais simples;
  - Subproblemas interelacionados;

Compreensão

- Relacionado à problemas de otimização;
  - Análise de subproblemas mais simples;
  - Subproblemas interelacionados;
  - Subproblemas com custo imediato e importante.



Compreensão

- Análise para implementação:

Compreensão

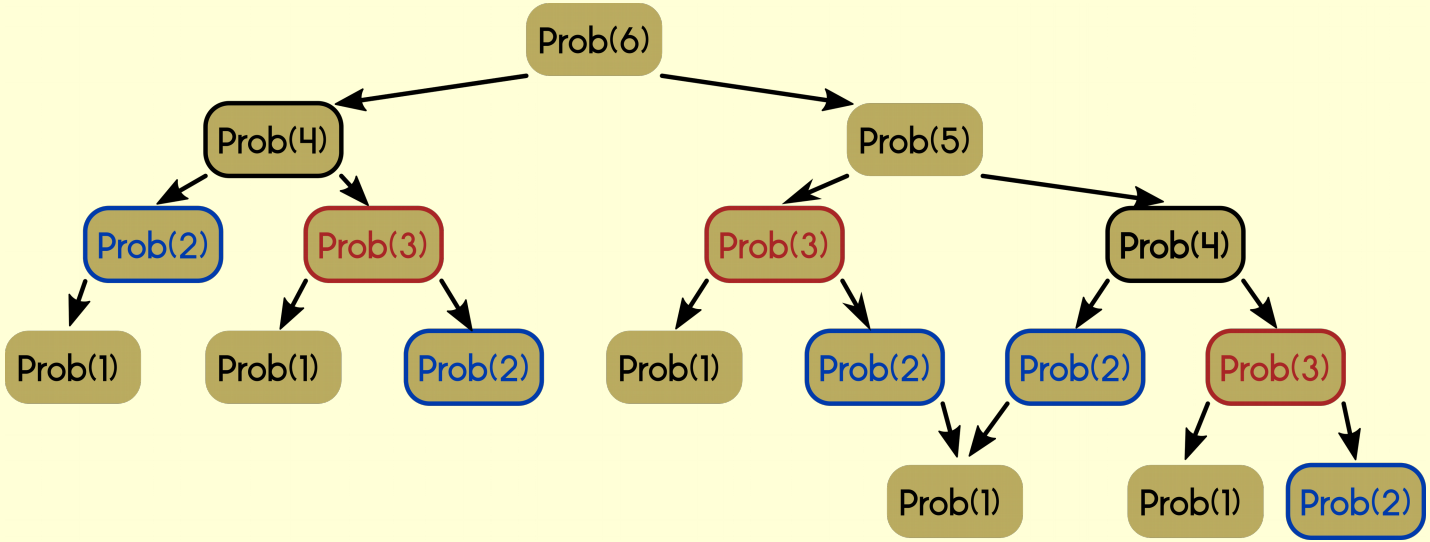
- Análise para implementação:
  - Ao resolver um subproblema, deve-se salvar tal parcial;

Compreensão

- Análise para implementação:
  - Ao resolver um subproblema, deve-se salvar tal parcial;
  - Para o resultado final, consulta-se os resultados parciais;

Compreensão

- Análise para implementação:
  - Ao resolver um subproblema, deve-se salvar tal parcial;
  - Para o resultado final, consulta-se os resultados parciais;
    - Não precisa recalcular.



Compreensão

- Estratégia:

Compreensão

- Estratégia:
  - Caracterização de uma solução ótima;

Compreensão

- Estratégia:
  - Caracterização de uma solução ótima;
    - Definir os subproblemas.

Compreensão

- Estratégia:
  - Caracterização de uma solução ótima;
    - Definir os subproblemas.
  - Identificar uma solução recursiva, não otimizada;

Compreensão

- Estratégia:
  - Caracterização de uma solução ótima;
    - Definir os subproblemas.
  - Identificar uma solução recursiva, não otimizada;
    - Aninhar tais subproblemas por recorrências.

Compreensão

- Estratégia:
  - Caracterização de uma solução ótima;
    - Definir os subproblemas.
  - Identificar uma solução recursiva, não otimizada;
    - Aninhar tais subproblemas por recorrências.
  - Implementar mecanismo de compartilhamento de dados.

Compreensão

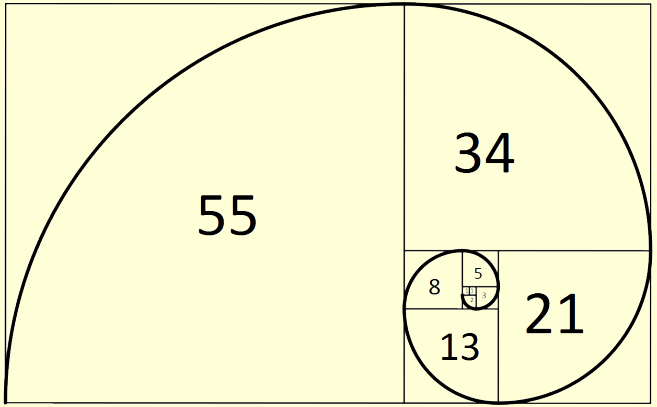
- Estratégia:
  - Caracterização de uma solução ótima;
    - Definir os subproblemas.
  - Identificar uma solução recursiva, não otimizada;
    - Aninhar tais subproblemas por recorrências.
  - Implementar mecanismo de compartilhamento de dados.
    - Criação de uma tabela de resultados parciais.

Implementação

- Para um aprofundamento, têm-se exemplos de implementação:
  - Fibonacci;
  - Radares - URI;
  - Sequência Crescente Máxima;
  - Partição de Inteiros;
  - Árvore Binária de Pesquisa.
- Soluções apresentadas em pseudo-código.

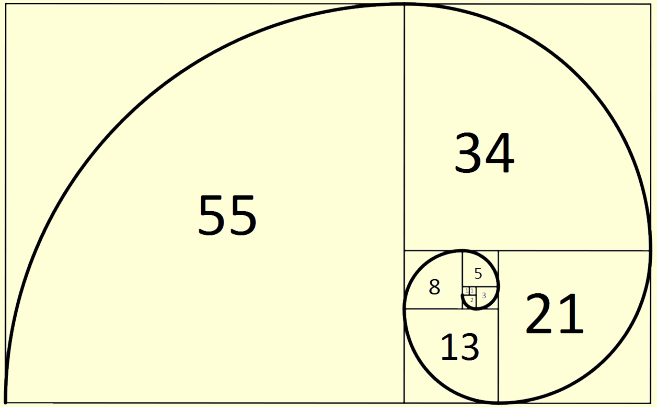
Implementação
Fibonacci

- Algoritmo classicamente conhecido pela forma recursiva;



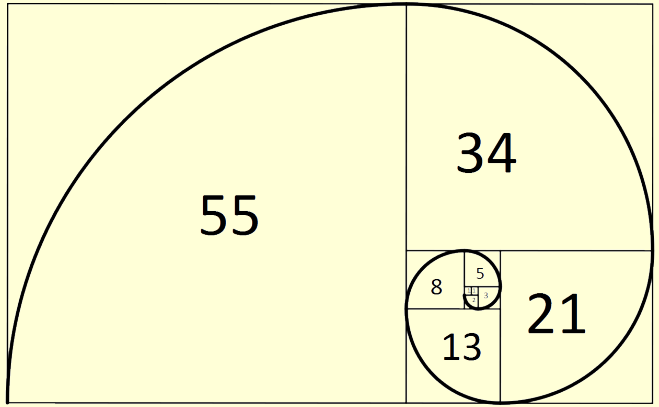
Implementação
Fibonacci

- Algoritmo classicamente conhecido pela forma recursiva;
- Realiza a mesma conta repetidas vezes;



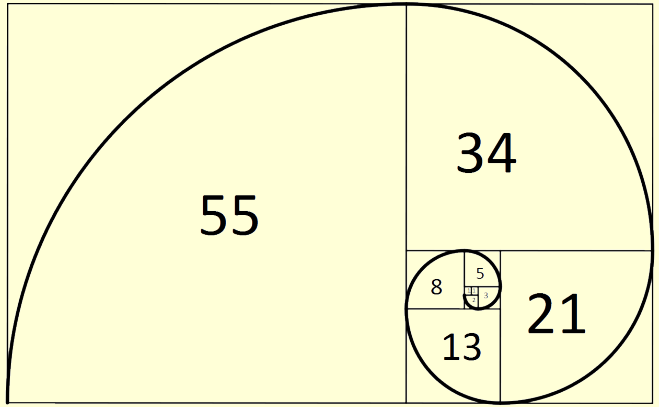
Implementação
Fibonacci

- Algoritmo classicamente conhecido pela forma recursiva;
- Realiza a mesma conta repetidas vezes;
  - Quanto maior o número, maior o retrabalho nos subproblemas;



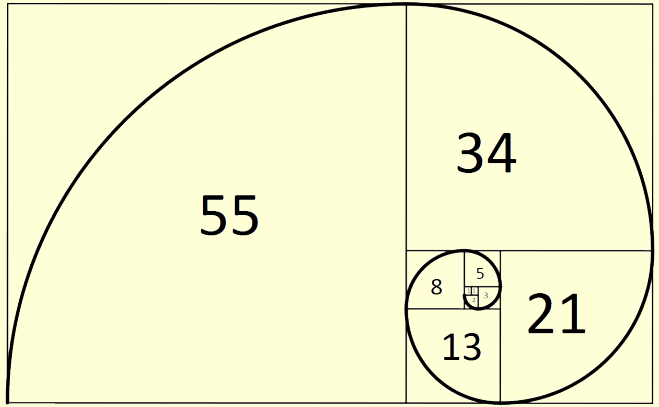
Implementação
Fibonacci

- Algoritmo classicamente conhecido pela forma recursiva;
- Realiza a mesma conta repetidas vezes;
  - Quanto maior o número, maior o retrabalho nos subproblemas;
  - Complexidade exponencial.



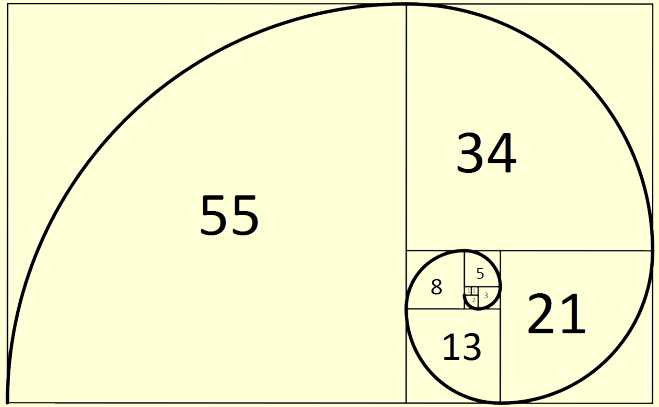
Implementação
Fibonacci

- Algoritmo classicamente conhecido pela forma recursiva;
- Realiza a mesma conta repetidas vezes;
  - Quanto maior o número, maior o retrabalho nos subproblemas;
  - Complexidade exponencial.
- Com memoização os retrabalhos são evitados;



Implementação
Fibonacci

- Algoritmo classicamente conhecido pela forma recursiva;
- Realiza a mesma conta repetidas vezes;
  - Quanto maior o número, maior o retrabalho nos subproblemas;
  - Complexidade exponencial.
- Com memoização os retrabalhos são evitados;
  - Realiza uma consulta no trabalho já realizado.



Implementação
Fibonacci

- Implementação por recursão:

Implementação
Fibonacci

- Implementação por recursão:

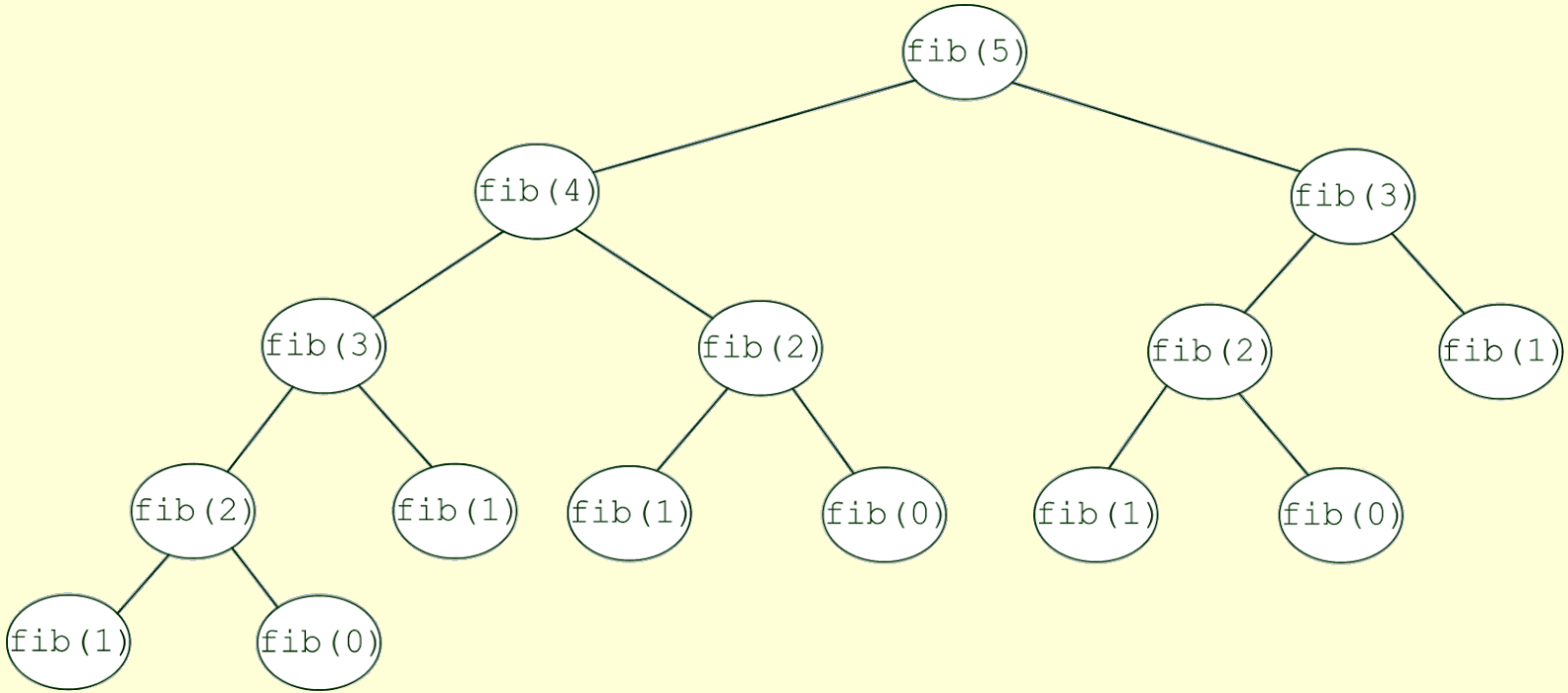
```
fibo_rec (n)
    SE (n < 2)
    |    RETORNA n
    SENÃO
    |    RETORNA fibo_rec( n – 1 ) + fibo_rec( n + 2 )
```

Implementação
Fibonacci

- Árvore de recursão gerada:

Implementação
Fibonacci

- Árvore de recursão gerada:



Implementação
Fibonacci

- Aplicando a memoização:

Implementação
Fibonacci

- Aplicando a memoização:

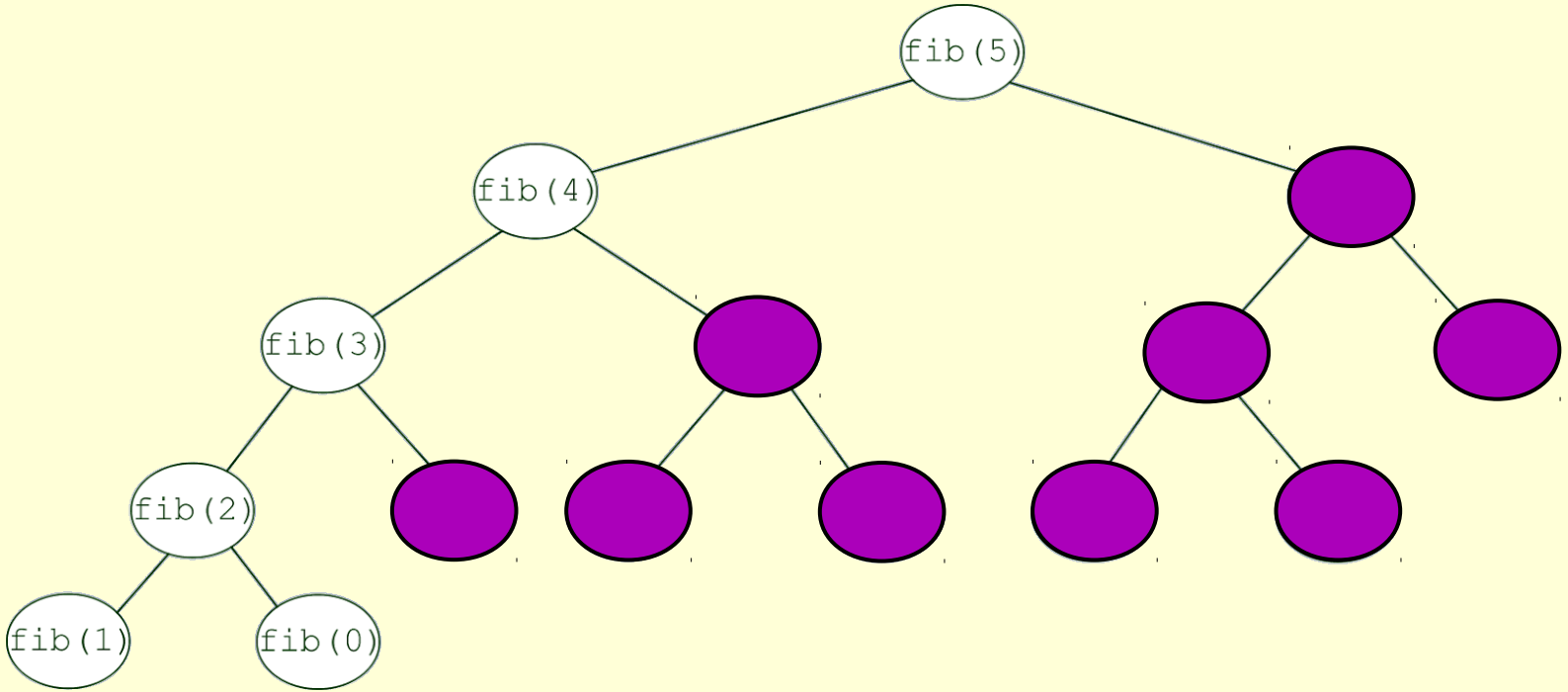
```
fibo_memo( n )
    SE (n < 2)
    |    RETORNA n
    SENÃO
    |    SE fibo_tab[n] é INDEFINIDO
    |    |    fibo_tab[n] = fibo_memo( n – 1 ) + fibo_memo( n + 2 )
         SENÃO
    |    |    RETORNA fibo_tab[n]
```

Implementação
Fibonacci

- Árvore de recursão gerada:

Implementação
Fibonacci

- Árvore de recursão gerada:



Implementação
Radares

- Desafio do URI Online Judge;

Implementação
Radares

- Desafio do URI Online Judge;
- Com a instalação de radares numa cidade, o prefeito quer obter o maior lucro nas cobranças;

Implementação
Radares

- Desafio do URI Online Judge;
- Com a instalação de radares numa cidade, o prefeito quer obter o maior lucro nas cobranças;
- Dados:
  - Quantidades de radares;
  - Posição geográfica dos radares;
  - Lucro associado a cada radar.

Implementação
Radares

```
SomaRadar( atual, comp )
    posFinal = radares[atual]
    precoFinal = preco[atual]

    SE comp < 0
    |    tab[posFinal] = precoFinal
    |    RETORNA
    SE tab[posFinal] é DEFINIDO
    |    RETORNA

    posComp = radares[comp]
    precoComp = preco[comp]
continua...
```

Implementação
Radares

```
continuação...
    SE ( posFinal – posComp ) = 0
    |    tab[posFinal] = MAX( precoFinal, precoComp )
    |    RETORNA somaRadar( atual, comp – 1 )
    SE ( posFinal – posComp ) >= MENOR_DIST
    |    SE tab[posComp] é INDEFINIDO
    |    |    somaRadar( comp, comp – 1 )
    |    tab[posFinal] = MAX( precoFinal, precoFinal + tab[posComp] )
    |    RETORNA
    SENÃO
    |    somaRadar( atual, comp – 1 )
```

Implementação
Sequência Crescente Máxima (SCM)

- Considere A[**N**] uma sequência de números naturais;

Implementação
Sequência Crescente Máxima (SCM)

- Considere A[**N**] uma sequência de números naturais;
- Deve-se extrair a maior subsequência crescente:

Implementação
Sequência Crescente Máxima (SCM)

- Considere A[**N**] uma sequência de números naturais;
- Deve-se extrair a maior subsequência crescente:
  - Subsequência: é o resto da retirada arbitrária de termos de A[**N**];

Implementação
Sequência Crescente Máxima (SCM)

- Considere A[**N**] uma sequência de números naturais;
- Deve-se extrair a maior subsequência crescente:
  - Subsequência: é o resto da retirada arbitrária de termos de A[**N**];
  - Seguimento: é o resto após a retira de **M** termos no início de **P** termos no fim.

Implementação
Sequência Crescente Máxima (SCM)

- Considere A[**N**] uma sequência de números naturais;
- Deve-se extrair a maior subsequência crescente:
  - Subsequência: é o resto da retirada arbitrária de termos de A[**N**];
  - Seguimento: é o resto após a retira de **M** termos no início de **P** termos no fim.
- Considere B[**K**] esta subsequência.

Implementação
Sequência Crescente Máxima (SCM)

- Exemplos:

Implementação
Sequência Crescente Máxima (SCM)

- Exemplos:
  - (1,2,3,4,5,6) é SCM de (1,2,3,9,4,5,6);

Implementação
Sequência Crescente Máxima (SCM)

- Exemplos:
  - (1,2,3,4,5,6) é SCM de (1,2,3,9,4,5,6);
  - (5,6,6,7) é SCM de (9,5,6,3,9,6,4,7);

Implementação
Sequência Crescente Máxima (SCM)

- Exemplos:
  - (1,2,3,4,5,6) é SCM de (1,2,3,9,4,5,6);
  - (5,6,6,7) é SCM de (9,5,6,3,9,6,4,7);
  - (5,6,9) é subsequência maximal de (9,5,6,3,9,6,4,7); não é máxima.

Implementação
Sequência Crescente Máxima (SCM)

- Exemplos:
  - (1,2,3,4,5,6) é SCM de (1,2,3,9,4,5,6);
  - (5,6,6,7) é SCM de (9,5,6,3,9,6,4,7);
  - (5,6,9) é subsequência maximal de (9,5,6,3,9,6,4,7); não é máxima.
    - Maximal: não pode ser ampliada.

Implementação
Sequência Crescente Máxima (SCM)

```
scm ( A, n )
    PARA m = 1 a n
        tab[m] = 1
        PARA i = m – 1 a 1
            SE A[i] <= A[m] E tab[i] + 1 > tab[m]
                tab[m] = tab[i] + 1
    RETORNA tab
```

Implementação
Particionamento de Inteiros

- Dado N inteiro, deve-se determinar o número de maneiras de particionamento de N;

Implementação
Particionamento de Inteiros

- Dado N inteiro, deve-se determinar o número de maneiras de particionamento de N;
- Exemplos:
  - N = 4:
    1) 4;
    2) 3 + 1;
    3) 2 + 2;
    4) 2 + 1 + 1;
    5) 1 + 1 + 1 + 1.

Implementação
Particionamento de Inteiros

- Dado N inteiro, deve-se determinar o número de maneiras de particionamento de N;
- Exemplos:
  - N = 4:
    1) 4;
    2) 3 + 1;
    3) 2 + 2;
    4) 2 + 1 + 1;
    5) 1 + 1 + 1 + 1.
  - N = 6:
    1) 6;
    2) 5 + 1;
    3) 4 + 2;
    4) 4 + 1 + 1;
    5) 3 + 3;
    6) 3 + 2 + 1;
    7) 3 + 1 + 1 +1;
    8) 2 + 2 + 2
    9) 2 + 2 + 1 + 1
    10) 2 + 1 + 1 + 1 + 1
    11) 1 + 1 + 1 + 1 + 1 + 1

Implementação
Particionamento de Inteiros

- Construção, considerando **P** como a maior parcela e **N** o inteiro original:

Implementação
Particionamento de Inteiros

- Construção, considerando **P** como a maior parcela e **N** o inteiro original:
  - Partição = 0, se **N** < 0 ou **P** = 0;

Implementação
Particionamento de Inteiros

- Construção, considerando **P** como a maior parcela e **N** o inteiro original:
  - Partição = 0, se **N** < 0 ou **P** = 0;
  - Partição = 1, se **N** = 0;

Implementação
Particionamento de Inteiros

- Construção, considerando **P** como a maior parcela e **N** o inteiro original:
  - Partição = 0, se **N** < 0 ou **P** = 0;
  - Partição = 1, se **N** = 0;
  - Partição = Partição(**N** – **P**, **P**) + Partição(**N**, **P** – 1), se **N** > 0.

Implementação
Particionamento de Inteiros

```
particionaInt( n, p )
    tab[0][0] = 1
    PARA i = 1 a n
        tab[i][0] = 0
    PARA p = 1 a n
        PARA i = 0 a n
            SE i >= p
                tab[i][p] = tab[i][p – 1] + tab[i – p, p]
            SENÃO
                tab[i][p] = tab[i, p - 1]
```

Implementação
Árvore Binária de Pesquisa

- Realizando repetidas pesquisas, mostra-se frequências:

Implementação
Árvore Binária de Pesquisa

- Realizando repetidas pesquisas, mostra-se frequências:
  - Alguns valores são mais buscados do que outros.

Implementação
Árvore Binária de Pesquisa

- Realizando repetidas pesquisas, mostra-se frequências:
  - Alguns valores são mais buscados do que outros.
- Ter os mais procurados perto da raiz;

Implementação
Árvore Binária de Pesquisa

- Realizando repetidas pesquisas, mostra-se frequências:
  - Alguns valores são mais buscados do que outros.
- Ter os mais procurados perto da raiz;
  - Custo: frequência do valor e a distância até a raiz.

Implementação
Árvore Binária de Pesquisa

- Realizando repetidas pesquisas, mostra-se frequências:
  - Alguns valores são mais buscados do que outros.
- Ter os mais procurados perto da raiz;
  - Custo: frequência do valor e a distância até a raiz.
- Semelhante ao Código de Huffman;

Implementação
Árvore Binária de Pesquisa

- Realizando repetidas pesquisas, mostra-se frequências:
  - Alguns valores são mais buscados do que outros.
- Ter os mais procurados perto da raiz;
  - Custo: frequência do valor e a distância até a raiz.
- Semelhante ao Código de Huffman;
  - Não requer manutenção na ordem dos valores.

Implementação
Árvore Binária de Pesquisa

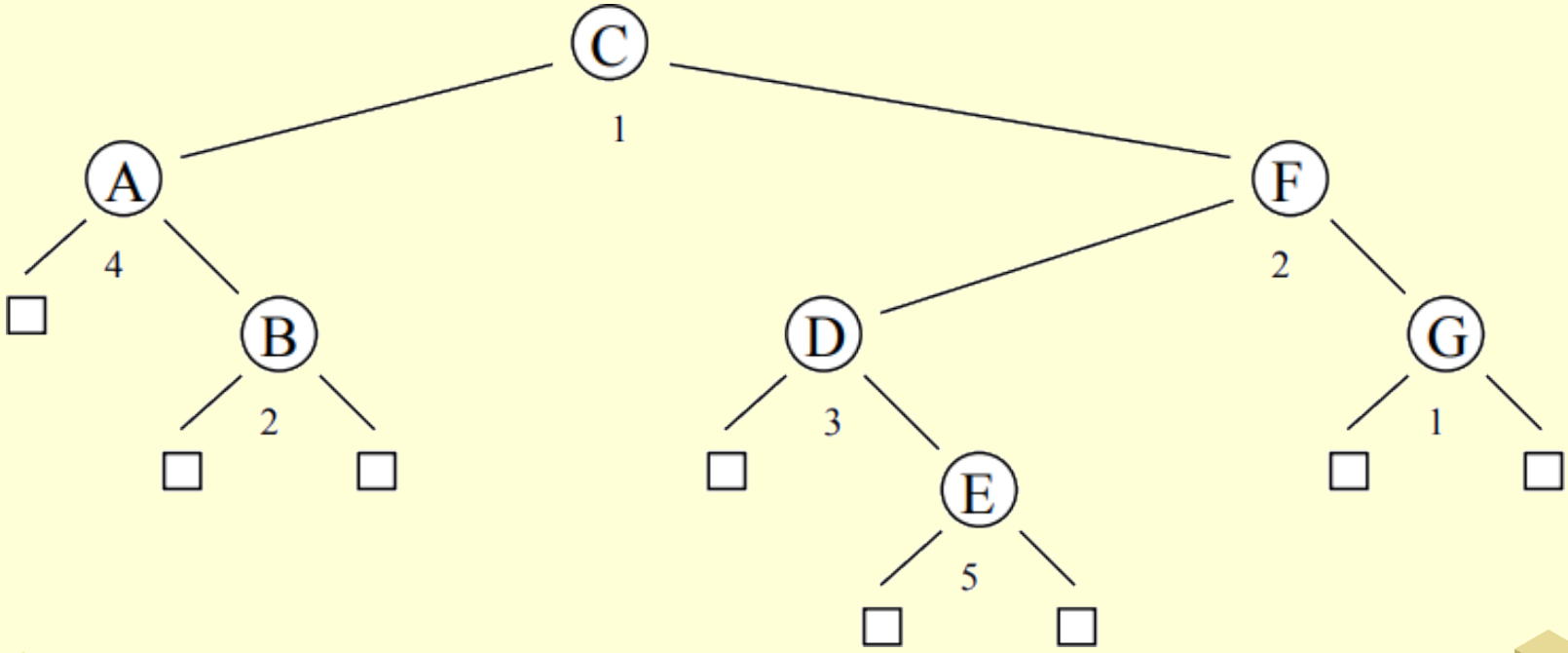
- Realizando repetidas pesquisas, mostra-se frequências:
  - Alguns valores são mais buscados do que outros.
- Ter os mais procurados perto da raiz;
  - Custo: frequência do valor e a distância até a raiz.
- Semelhante ao Código de Huffman;
  - Não requer manutenção na ordem dos valores.
- Semelhante à Ordem de Multiplicação de Matrizes.

Implementação
Árvore Binária de Pesquisa

- Árvore de consideração:

Implementação
Árvore Binária de Pesquisa

- Árvore de consideração:



Implementação
Árvore Binária de Pesquisa

```
PARA i = 0 a N
    PARA j = i + 1 a N + 1
    |   custo[i][j] = MAX_INT
PARA i = 0 a N
    custo[i][i] = freq[i]
PARA i = 1 a N + 1
    custo[i][i – 1] = 0
continua...
```

Implementação
Árvore Binária de Pesquisa

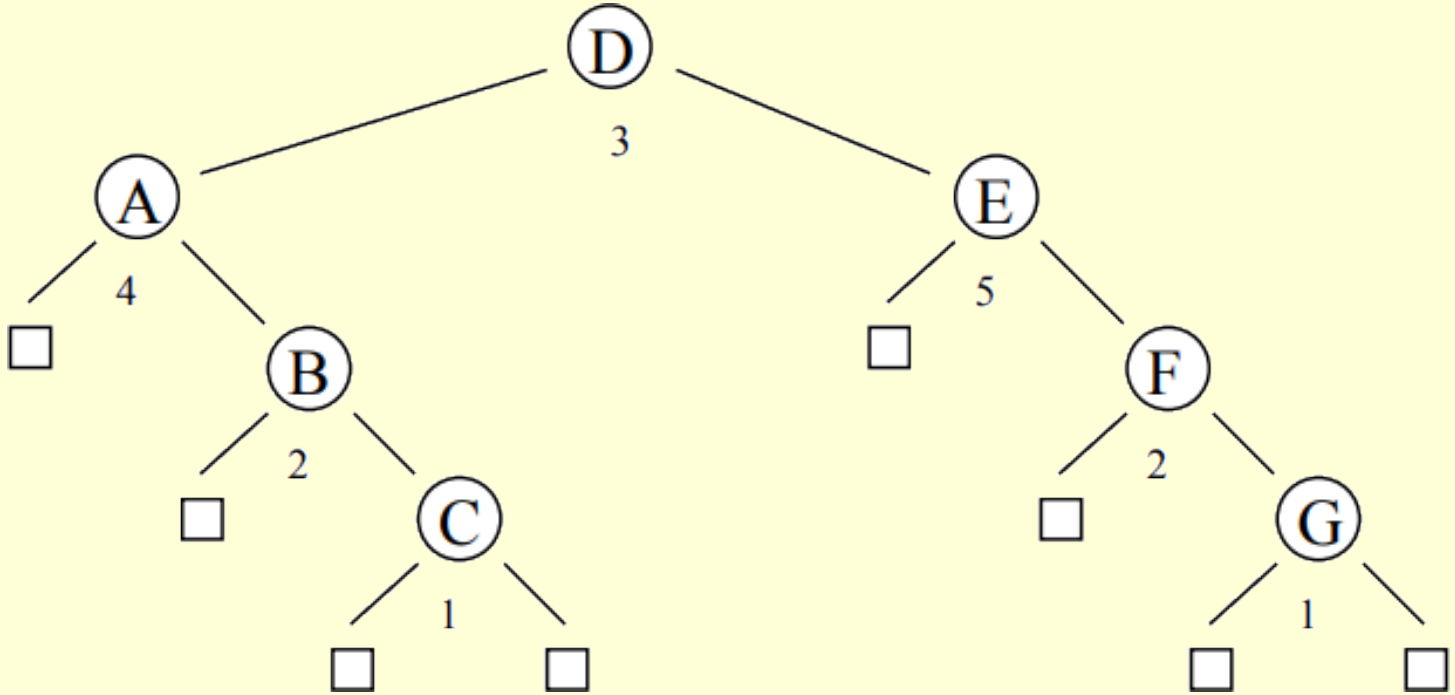
```
continuação...
PARA j = 1 a N – 1
    PARA i = 1 a N – j
    |   PARA k = i a i + j
    |   |   temp = custo[i][k – 1] + custo[k + 1][i + j]
    |   |   SE temp < custo[i][i + j]
    |   |   |   custo[i][i + j] = temp
    |   |   |   melhor[i][i + j] = k
    |   PARA k = i a i + j
    |   |   custo[i][i + j] = custo[i][i + j] + freq[k]
```

Implementação
Árvore Binária de Pesquisa

- Árvore de resultado:

Implementação
Árvore Binária de Pesquisa

- Árvore de resultado:



Referências

1. Quora. What does the "Dynamic" mean in Dynamic Programming? Disponível em: https://www.quora.com/What-does-the-Dynamic-mean-in-Dynamic-Programming/answer/Shai-Simonson-1?share=3af4aa2a&srid=dTB8 ;
2. Arcane Sentiment. Why "dynamic programming"? Disponível em: http://arcanesentiment.blogspot.com/2010/04/why-dynamic-programming.html;
3. UNICAMP. 2 – Introdução à Programação Dinâmica. Disponível em: http://www.dca.fee.unicamp.br/~gomide/courses/IA718/transp/IA718IntroducaoProgramacaoDinamica_2.pdf ;
4. UFJF. Programação Dinâmica. Disponível em: http://www.ufjf.br/epd015/files/2010/06/ProgramacaoDinamica.pdf;
5. IME. Programação Dinâmica. Disponível em: https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/dynamic-programming.html ;

Referências

6. DECOM. Programação Dinâmica. Disponível em: http://www.decom.ufop.br/anderson/2_2012/BCC241/ProgramacaoDinamica.pdf;
7. FEUP. Programação Dinâmica. Disponível em: https://web.fe.up.pt/~gtd/aed2/dinamica.pdf;
8. URI Online Judge. Radares. Disponível em: https://www.urionlinejudge.com.br/judge/pt/problems/view/1689;
9. UERJ. Programação Dinâmica. Disponível em: https://www.ime.uerj.br/~pauloedp/ALGO/Download/ALSLPRDI.pdf;