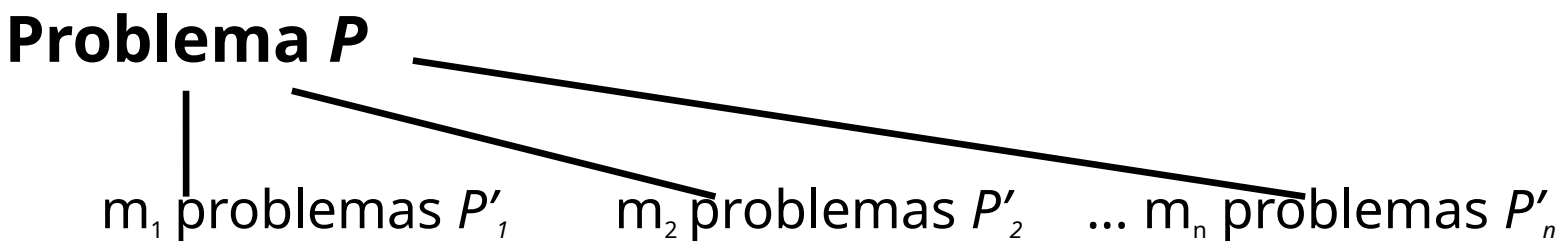
Grafos: Programação Dinâmica- Todos os Menores Caminhos

Prof. André Britto

Programação Dinâmica



- Resolvendo $P'_1, P'_2, \ldots, P'_n$ resolve P.
- P'_i $1 \quad i \quad n$ é de natureza igual ou semelhante a P.
- Tamanho de P'_i é menor que P.

Programação Dinâmica

- **Ideia:** resolver P'_i a primeira vez em que for considerado; nas $m_i - 1$ outras vezes consultar uma tabela onde está armazenada a solução de P'_i.

Programação Dinâmica

Aplicação da técnica exige:

- decomposição de P em P'_i $1 \quad i \quad n$ seja de natureza simples.

- tabela para armazenamento de P'_i deve ser definida de modo a tornar simples o acesso a seus resultados.

Programação Dinâmica

Exemplo Clássico:

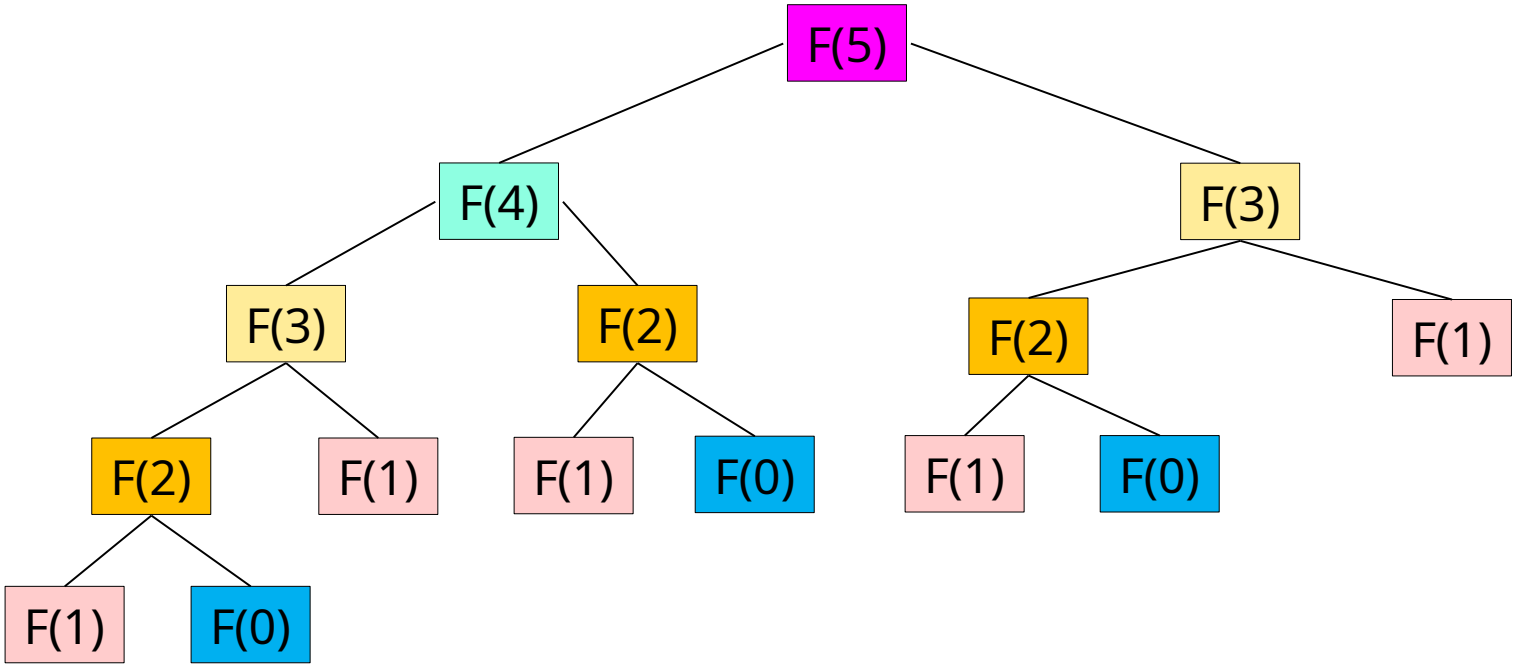
Sequência de Fibonacci *F(n)*

- Formulação recursiva:

  Se $n \quad 1$ então $F(n) = n$

  senão $F(n) = F(n-1) + F(n-2)$

Fibonacci: Árvore de Recursão



Vários problemas iguais são resolvidos mais de uma vez na formulação recursiva.
A formulação recursiva leva a um algoritmo de complexidade exponencial!

Programação Dinâmica

Eliminação da redundância de solução de problemas parciais pela introdução de uma estrutura de armazenamento destas soluções.

Na primeira vez que o problema aparece, ele é calculado, nas demais vezes consulta-se a solução armazenada.

Para a sequência de Fibonacci, a estrutura pode ser um vetor de *n+1* elementos.

n	*0*	*1*	*2*	*3*	*4*	*5*	*6*
F[n]	*0*	*1*	*1*	*2*	*3*	*5*	*8*

Programação Dinâmica

```
Algoritmo Fibonacci(n, F);
{entrada: inteiro n
 saída : sequência de Fibonacci no vetor F}
 inicio
   para i = 0, 1, ..., n faça
      se i   1 então F[i] := i
      senão F[i] := F[i-1] + F[i-2];
  fim
```

- Observe que agora um vetor é usado e o cálculo dos valores da sequência se dá por consulta aos valores armazenados em F.
- Complexidade?

Programação Dinâmica

```
Algoritmo Fibonacci(n, F);
{entrada: inteiro n
 saída : sequência de Fibonacci no vetor F}
 inicio
   para i = 0, 1, ..., n faça
      se i   1 então F[i] := i
      senão F[i] := F[i-1] + F[i-2];
  fim
```

- Observe que agora um vetor é usado e o cálculo dos valores da sequência se dá por consulta aos valores armazenados em F.
- Complexidade? ***O(n)***

Programação Dinâmica

Aplicações da técnica

- Problema de Determinar todos os caminhos mais curtos entre vértices de um grafo.

Todos os caminhos mais curtos

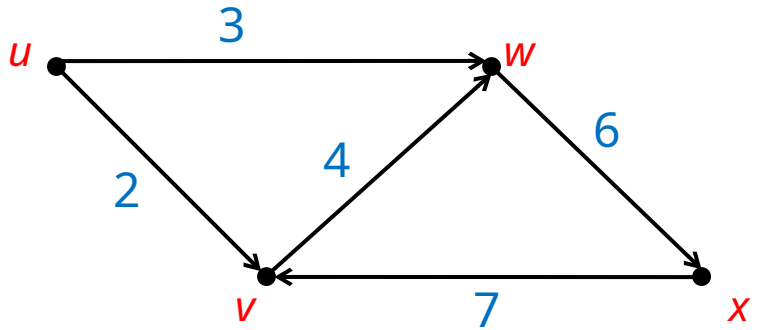
- Problema: Dado um grafo, com peso nas arestas, *G* (direcionado ou não) encontrar o caminho de menor tamanho entre todos os pares de vértices de *G.*

- Peso nas arestas :
  - considerar inteiro não negativo
  - reflete o tamanho (distância) do caminho entre dois vértices, extremos da aresta.

Todos os caminhos mais curtos

- Simplificação: vamos na realidade determinar os comprimentos desses caminhos curtos, ao invés do caminho explicitamente.

- Suposição: dígrafo



(u,w) *nova aresta adicionada*

(u,w)
(u,w,x)
} novos caminhos mais curtos

Todos os caminhos mais curtos

- 1ª Solução: indução no número de arestas.

  Temos todos os caminhos mais curtos para um grafo com *m-1* arestas. O que acontece ao unirmos a *m-ésima* aresta, digamos *(u,v)* ?

  - Complexidade:

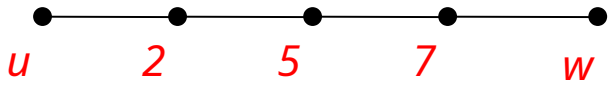
    $O(mn^2)$ $O(n^4)$

Todos os caminhos mais curtos:

- 2ª Solução: indução no tipo do caminho
  - vértices e arestas fixos.
  - rotular os vértices dos caminhos com rótulos de *1* a *n*.

Um caminho de *u* a *w* é chamado um *k-caminho* se excetuando-se *u* e *w,* o rótulo mais alto dos vértices no caminho for *k*. Em particular, um *0-caminho* é uma aresta(não há outros vértices no caminho além de *u* e *w).*

Ex.:



7-caminho

Todos os caminhos mais curtos

- Hipótese de Indução:

  Sabemos os comprimentos dos caminhos mais curtos entre todos os pares de vértices tais que **somente *k* caminhos**, para algum ***0 k < m***, são considerados

  - O que acontece para caminhos com $k = m$, ou seja, $k < m+1$?

Todos os caminhos mais curtos

- Base de Indução: $m = 1$
- Hipótese de indução: vale para todos os k caminhos com 0 $k < m$.
- Caso geral: Estender para $k < m+1$ considerar todos os *m-caminhos*.

  - V_m vértice rotulado com m.
  - Qualquer *m-caminho* mais curto inclui V_m exatamente uma vez.

Todos os caminhos mais curtos

- P *k-caminho* mais curto ($k < m$) entre u e V_m.
- Q *j-caminho* mais curto ($j < m$) entre V_m e w.
- $P \cdot Q$ *m-caminho* mais curto entre u e w ?

  $(u,\ldots,V_m,\ldots,w)$

- Hipótese de Indução: Sabemos comprimento P e Q.

  ⇩

  Soma = comprimento de $P \cdot Q$

- Para a solução verifica-se se esse *m-caminho* é menor que o *(m-1)-caminho* entre u e w.

Algoritmo de Floyd-Warshall

```
algoritmo CaminhosMaisCurtos(matComp)

{dados: matrix n x n de comprimentos dos caminhos, matComp;
        matComp[x,y] é o peso da aresta (x,y), se ela
        existir e ∞ no caso contrário.
        matComp[x,x] = 0 para  x
}
{saída:
      matComp contém os comprimentos dos caminhos mais
      curtos entre os pares de vértices do grafo.
}
```

Algoritmo de Floyd-Warshall

```
início
   para v := 1,2,...,n faça
      para u := 1,2,...,n faça
         para w := 1,2,...,n faça
            Se matComp[u,v] + matComp[v,w] <
               matComp[u,w] então
                     matComp[u,w] := matComp[u,v] +
                                     matComp[v,w];
fim
```

Algoritmo de Floyd-Warshall

```
início
   para v := 1,2,...,n faça (Sequência de indução)
      para u := 1,2,...,n faça
         para w := 1,2,...,n faça
            Se matComp[u,v] + matComp[v,w] <
               matComp[u,w] então
                     matComp[u,w] := matComp[u,v] +
                                     matComp[v,w];
fim
```

m-caminhos entre *u* e *w*

Algoritmo de Floyd-Warshall

```
início
   para v := 1,2,...,n faça (Sequência de indução)
      para u := 1,2,...,n faça
         para w := 1,2,...,n faça
            Se matComp[u,v] + matComp[v,w] <
               matComp[u,w] então
                     matComp[u,w] := matComp[u,v] +
                                     matComp[v,w];
fim
```

m-caminhos entre *u* e *w*

- Programação Dinâmica ?

Algoritmo de Floyd-Warshall: Complexidade

Complexidade:

```
início
 para v := 1,2,...,n faça
   para u := 1,2,...,n faça
     para w := 1,2,...,n faça
       Se matComp[u,v] + matComp[v,w] <
          matComp[u,w] então
             matComp[u,w] := matComp[u,v]+
                             matComp[v,w];
fim
```

Algoritmo de Floyd-Warshall: Complexidade

Complexidade:

```
início
 para v := 1,2,...,n faça
   para u := 1,2,...,n faça
     para w := 1,2,...,n faça
       Se matComp[u,v] + matComp[v,w] <
          matComp[u,w] então
             matComp[u,w] := matComp[u,v]+
                             matComp[v,w];
fim
```

$O(|V|)$

Algoritmo de Floyd-Warshall: Complexidade

Complexidade:

```
início
 para v := 1,2,...,n faça
   para u := 1,2,...,n faça
     para w := 1,2,...,n faça
       Se matComp[u,v] + matComp[v,w] <
          matComp[u,w] então
             matComp[u,w] := matComp[u,v]+
                             matComp[v,w];
fim
```

$O(|V|)$ (laço interno)

$O(|V|)$ (laço intermediário)

Algoritmo de Floyd-Warshall: Complexidade

Complexidade:

```
início
 para v := 1,2,...,n faça
   para u := 1,2,...,n faça
     para w := 1,2,...,n faça
       Se matComp[u,v] + matComp[v,w] <
          matComp[u,w] então
             matComp[u,w] := matComp[u,v]+
                             matComp[v,w];
fim
```

$O(|V|)$ $O(|V|)$ $O(|V|)$

Algoritmo de Floyd-Warshall: Complexidade

Complexidade: $O(|V|^3)$

```
início
 para v := 1,2,...,n faça
   para u := 1,2,...,n faça
     para w := 1,2,...,n faça
       Se matComp[u,v] + matComp[v,w] <
          matComp[u,w] então
             matComp[u,w] := matComp[u,v]+
                             matComp[v,w];
fim
```

$O(|V|)$ $O(|V|)$ $O(|V|)$

Referências

- Seção 5.4 do Szwarcfiter, J. L., *Grafos e Algoritmos Computacionais,* Ed. Campus, 1983.

- Seção 25.2 do Cormen, *Introduction to Algorithms,* MIT Press, 2009.

- Seção 3.9 do Jungnickel, D., *Graphs, Networks and Algorithms,* Springer, 2007.