Aula 23 – Programação Dinâmica

Norton T. Roman & Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2023

Construção Incremental

- No método de construção incremental, temos os seguintes passos:

Construção Incremental

- No método de construção incremental, temos os seguintes passos:
  - Inicialmente, resolvemos o problema para um subconjunto dos elementos da entrada

Construção Incremental

- No método de construção incremental, temos os seguintes passos:
  - Inicialmente, resolvemos o problema para um subconjunto dos elementos da entrada
  - Então adicionamos os demais elementos um a um

Construção Incremental

- No método de construção incremental, temos os seguintes passos:
  - Inicialmente, resolvemos o problema para um subconjunto dos elementos da entrada
  - Então adicionamos os demais elementos um a um
  - Em muitos casos, se os elementos forem adicionados em uma ordem ruim, o algoritmo não será eficiente.

Construção Incremental

- No método de construção incremental, temos os seguintes passos:
  - Inicialmente, resolvemos o problema para um subconjunto dos elementos da entrada
  - Então adicionamos os demais elementos um a um
  - Em muitos casos, se os elementos forem adicionados em uma ordem ruim, o algoritmo não será eficiente.
- Exemplo:
  - Cálculo recursivo de $n!$

Divisão e Conquista

- Na divisão e conquista, o problema principal é decomposto em subproblemas menores

Divisão e Conquista

- Na divisão e conquista, o problema principal é decomposto em subproblemas menores
  - Combinando então as respostas de cada um desses subproblemas

Programação Dinâmica

- Na Programação Dinâmica a solução ótima pode ser obtida a partir da solução ótima obtida previamente de outros subproblemas que, sobrepostos, compõem o problema original.

Programação Dinâmica

- Na Programação Dinâmica a solução ótima pode ser obtida a partir da solução ótima obtida previamente de outros subproblemas que, sobrepostos, compõem o problema original.

Programação Dinâmica

- Na Programação Dinâmica a solução ótima pode ser obtida a partir da solução ótima obtida previamente de outros subproblemas que, sobrepostos, compõem o problema original.
  - A resolução dos subproblemas é calculada e armazenada para ser reutilizada.

Programação Dinâmica

- Na Programação Dinâmica a solução ótima pode ser obtida a partir da solução ótima obtida previamente de outros subproblemas que, sobrepostos, compõem o problema original.
  - A resolução dos subproblemas é calculada e armazenada para ser reutilizada.
- É mais um paradigma de projeto de algoritmos

Programação Dinâmica

- Para aplicarmos Programação Dinâmica, um problema deve apresentar duas características:

Programação Dinâmica

- Para aplicarmos Programação Dinâmica, um problema deve apresentar duas características:
  - **Subestrutura ótima:** ocorre quando uma solução ótima pode ser calculada a partir de soluções ótimas de subproblemas.

Programação Dinâmica

- Para aplicarmos Programação Dinâmica, um problema deve apresentar duas características:
  - **Subestrutura ótima:** ocorre quando uma solução ótima pode ser calculada a partir de soluções ótimas de subproblemas.
  - **Superposição de subproblemas:** ocorre quando o algoritmo reexamina o mesmo (sub)problema diversas vezes.

Programação Dinâmica

Sequência de Fibonacci

Programação Dinâmica

Sequência de Fibonacci

$$Fibonacci(n) = \begin{cases} n & \text{para } n \leq 1 \\ Fibonacci_{n-1} + Fibonacci_{n-2} & \text{caso contrário} \end{cases}$$

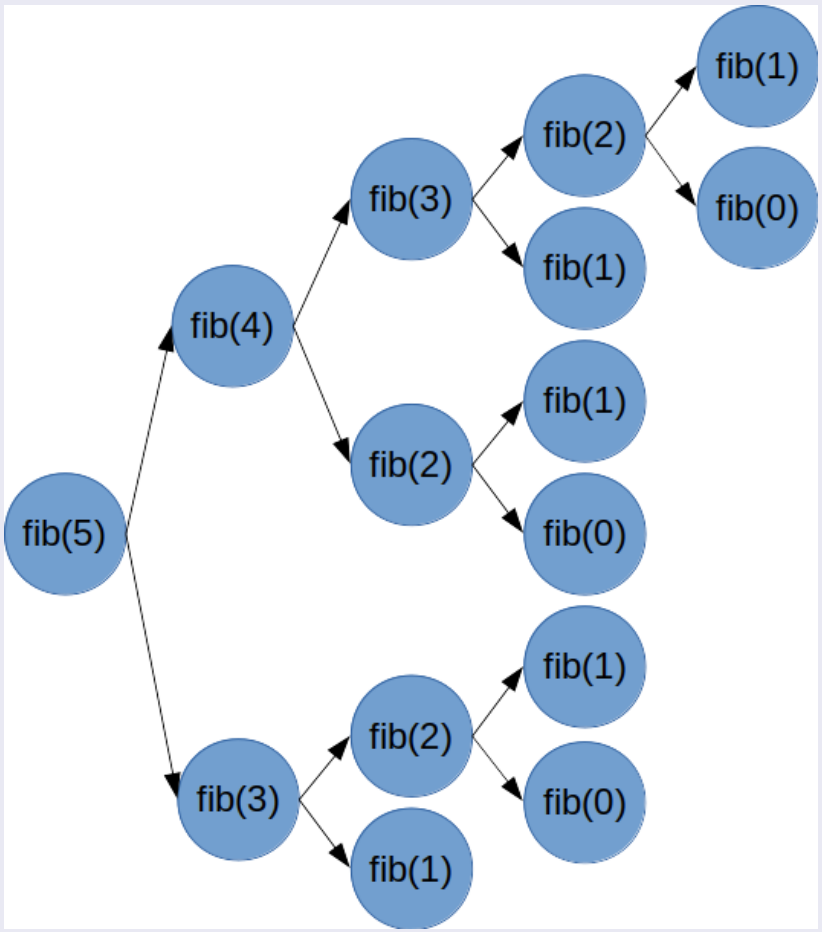
Programação Dinâmica

Código Recursivo (baseado em indução)

```
long fib(int n){
  if (n<=1) return n;
  return fib(n-1) + fib(n-2);
}
```

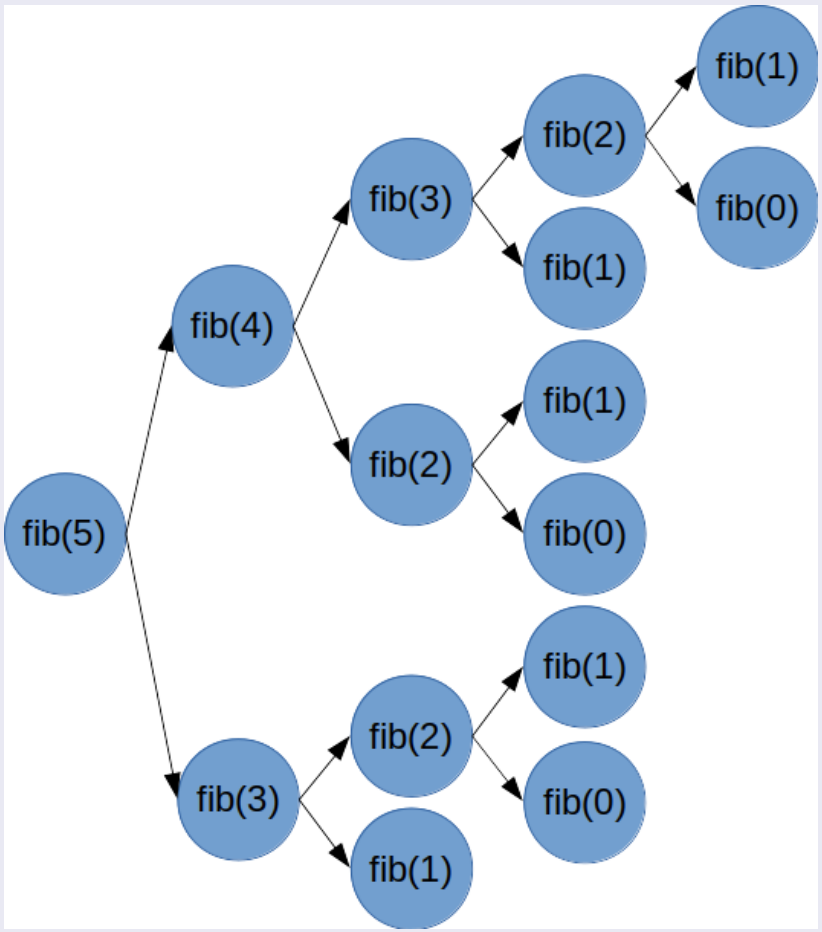
Programação Dinâmica

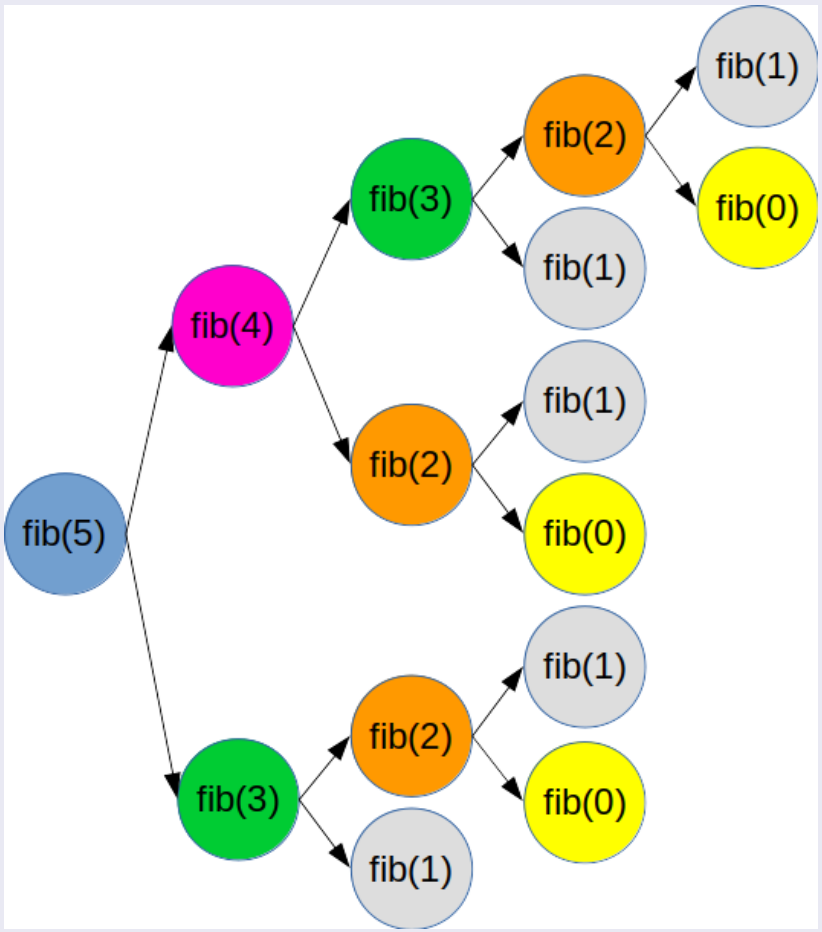
Sequência de Fibonacci



Programação Dinâmica

Sequência de Fibonacci





Sequência de Fibonacci

Código Recursivo (baseado em indução)

```
long fib(int n){
  if (n<=1) return n;
  return fib(n-1) + fib(n-2);
}
```

Sequência de Fibonacci

Código Recursivo (baseado em indução)

```
long fib(int n){
  if (n<=1) return n;
  return fib(n-1) + fib(n-2);
}
```

$$T(n) = \begin{cases} 0 & \text{para } n \leq 1 \\ T(n-1) + T(n-2) + 1 & \text{caso contrário} \end{cases}$$

Sequência de Fibonacci

Código Recursivo (baseado em indução)

```
long fib(int n){
  if (n<=1) return n;
  return fib(n-1) + fib(n-2);
}
```

$$T(n) = \begin{cases} 0 & \text{para } n \leq 1 \\ T(n-1) + T(n-2) + 1 & \text{caso contrário} \end{cases}$$

$$T(n) \approx 1,618^n$$

Sequência de Fibonacci

Programação Dinâmica

```
long fibonacci(int n){
  int i;
  long aux[n];
  if (n<=1) return n;
  aux[0] = 0;
  aux[1] = 1;
  for(i=2;i<n;i++) aux[i] = aux[i-1] + aux[i-2];
  return aux[n-1] + aux[n-2];
}
```

Sequência de Fibonacci

Programação Dinâmica

```
long fibonacci(int n){
  int i;
  long aux[n];
  if (n<=1) return n;
  aux[0] = 0;
  aux[1] = 1;
  for(i=2;i<n;i++) aux[i] = aux[i-1] + aux[i-2];
  return aux[n-1] + aux[n-2];
}
```

Sequência de Fibonacci

Programação Dinâmica

```
long fibonacci(int n){
  int i;
  long aux[n];
  if (n<=1) return n;
  aux[0] = 0;
  aux[1] = 1;
  for(i=2;i<n;i++) aux[i] = aux[i-1] + aux[i-2];
  return aux[n-1] + aux[n-2];
}
```

Sequência de Fibonacci

Programação Dinâmica

```
long fibonacci(int n){
  int i;
  long aux[n];
  if (n<=1) return n;
  aux[0] = 0;
  aux[1] = 1;
  for(i=2;i<n;i++) aux[i] = aux[i-1] + aux[i-2];
  return aux[n-1] + aux[n-2];
}
```

Sequência de Fibonacci

Programação Dinâmica

```
long fibonacci(int n){
  int i;
  long aux[n];
  if (n<=1) return n;
  aux[0] = 0;
  aux[1] = 1;
  for(i=2;i<n;i++) aux[i] = aux[i-1] + aux[i-2];
  return aux[n-1] + aux[n-2];
}
```

Sequência de Fibonacci

Programação Dinâmica

```
long fibonacci(int n){
  int i;
  long aux[n];
  if (n<=1) return n;
  aux[0] = 0;
  aux[1] = 1;
  for(i=2;i<n;i++) aux[i] = aux[i-1] + aux[i-2];
  return aux[n-1] + aux[n-2];
}
```

$$f(n) = n - 2 + 1 = n - 1$$

Sequência de Fibonacci

Programação Dinâmica

```
long fibonacci(int n){
  int i;
  long aux[n];
  if (n<=1) return n;
  aux[0] = 0;
  aux[1] = 1;
  for(i=2;i<n;i++) aux[i] = aux[i-1] + aux[i-2];
  return aux[n-1] + aux[n-2];
}
```

$$f(n) = n - 2 + 1 = n - 1$$

$$f(n) \in \Theta(n)$$

Subsequência de Soma Máxima

Problema

Subsequência de Soma Máxima

Problema

- Dado um arranjo de números inteiros, determine a subsequência cuja soma é máxima **(ou o valor dessa soma)**

Subsequência de Soma Máxima

Problema

- Dado um arranjo de números inteiros, determine a subsequência cuja soma é máxima **(ou o valor dessa soma)**
- Exemplo:

3	13	-10	14	-20	4	15

Subsequência de Soma Máxima

Problema

- Dado um arranjo de números inteiros, determine a subsequência cuja soma é máxima **(ou o valor dessa soma)**
- Exemplo:

3	13	-10	14	-20	4	15

3	13	-10	14	-20	4	15

16

Subsequência de Soma Máxima

Problema

- Dado um arranjo de números inteiros, determine a subsequência cuja soma é máxima **(ou o valor dessa soma)**
- Exemplo:

3	13	-10	14	-20	4	15
3	13	-10	14	-20	4	15
3	13	-10	14	-20	4	15

16

19

Subsequência de Soma Máxima

Problema

- Dado um arranjo de números inteiros, determine a subsequência cuja soma é máxima **(ou o valor dessa soma)**
- Exemplo:

3	13	-10	14	-20	4	15
3	13	-10	14	-20	4	15
3	13	-10	14	-20	4	15
3	13	-10	14	-20	4	15

16

19

20

Subsequência de Soma Máxima

Problema

Subsequência de Soma Máxima

Problema

- Dado um arranjo de **n elementos**, quantas subsequências existem?

Subsequência de Soma Máxima

Problema

- Dado um arranjo de **n elementos**, quantas subsequências existem?
  - Qualquer elemento/posição pode iniciar uma subsequência: $O(n)$

Subsequência de Soma Máxima

Problema

- Dado um arranjo de **n elementos**, quantas subsequências existem?
  - Qualquer elemento/posição pode iniciar uma subsequência: $O(n)$
  - Há até n posições para o fim da sequência ($fim \geq ini$):$O(n)$

Subsequência de Soma Máxima

Problema

- Dado um arranjo de **n elementos**, quantas subsequências existem?
  - Qualquer elemento/posição pode iniciar uma subsequência: $O(n)$
  - Há até n posições para o fim da sequência ($fim \geq ini$):$O(n)$
  - Há da ordem de n^2 subsequências

Subsequência de Soma Máxima

Problema

- Dado um arranjo de **n elementos**, quantas subsequências existem?
  - Qualquer elemento/posição pode iniciar uma subsequência: $O(n)$
  - Há até n posições para o fim da sequência ($fim \geq ini$):$O(n)$
  - Há da ordem de n^2 subsequências
  - Cada subsequência terá de 1 até n elementos: $O(n)$;

Subsequência de Soma Máxima

Problema

- Dado um arranjo de **n elementos**, quantas subsequências existem?
  - Qualquer elemento/posição pode iniciar uma subsequência: $O(n)$
  - Há até n posições para o fim da sequência ($fim \geq ini$):$O(n)$
  - Há da ordem de n^2 subsequências
  - Cada subsequência terá de 1 até n elementos: $O(n)$;
  - Assim, um algoritmo que teste todas as subsequências terá complexidade $O(n^3)$

Subsequência de Soma Máxima

Código Simplista

```
int subSequenciaMaxima(int A[], int n){
  int ini, fim, x, atual, max;
  max = 0;
  for (ini=0; ini<n; ini++){
    for (fim=ini; fim<n; fim++){
      atual = A[ini];
      for(x=ini+1;x<=fim;x++)
        atual += A[x];
      if (atual > max) max = atual;
    }
 }
 return max;
}
```

Subsequência de Soma Máxima

Código Simplista

```
int subSequenciaMaxima(int A[], int n){
  int ini, fim, x, atual, max;
  max = 0;
  for (ini=0; ini<n; ini++){
    for (fim=ini; fim<n; fim++){
      atual = A[ini];
      for(x=ini+1;x<=fim;x++)
        atual += A[x];
      if (atual > max) max = atual;
    }
 }
 return max;
}
```

Subsequência de Soma Máxima

Código Simplista

```
int subSequenciaMaxima(int A[], int n){
  int ini, fim, x, atual, max;
  max = 0;
  for (ini=0; ini<n; ini++){                    n valores
    for (fim=ini; fim<n; fim++){
      atual = A[ini];
      for(x=ini+1;x<=fim;x++)
        atual += A[x];
      if (atual > max) max = atual;
    }
 }
 return max;
}
```

Subsequência de Soma Máxima

Código Simplista

```
int subSequenciaMaxima(int A[], int n){
  int ini, fim, x, atual, max;
  max = 0;
  for (ini=0; ini<n; ini++){                    n valores
    for (fim=ini; fim<n; fim++){              até n valores
      atual = A[ini];
      for(x=ini+1;x<=fim;x++)
        atual += A[x];
      if (atual > max) max = atual;
    }
 }
 return max;
}
```

Subsequência de Soma Máxima

Código Simplista

```
int subSequenciaMaxima(int A[], int n){
  int ini, fim, x, atual, max;
  max = 0;
  for (ini=0; ini<n; ini++){                    n valores
    for (fim=ini; fim<n; fim++){              até n valores
      atual = A[ini];
      for(x=ini+1;x<=fim;x++)               até n-1 valores
        atual += A[x];
      if (atual > max) max = atual;
    }
 }
 return max;
}
```

Subsequência de Soma Máxima

Código Simplista

```
int subSequenciaMaxima(int A[], int n){
  int ini, fim, x, atual, max;
  max = 0;
  for (ini=0; ini<n; ini++){                    n valores
    for (fim=ini; fim<n; fim++){              até n valores
      atual = A[ini];
      for(x=ini+1;x<=fim;x++)               até n-1 valores
        atual += A[x];
      if (atual > max) max = atual;
    }
 }
 return max;
}
```

$O(n^3)$

Subsequência de Soma Máxima

3	13	-10	14	-20	4	15

Cada soma
é feita
múltiplas
vezes:

Programação Dinâmica

Subsequência de Soma Máxima

3	13	-10	14	-20	4	15

Cada soma é feita múltiplas vezes:

3	13	-10	14	-20	4	15
3	13	-10	14	-20	4	15
3	13	-10	14	-20	4	15
3	13	-10	14	-20	4	15
3	13	-10	14	-20	4	15
3	13	-10	14	-20	4	15

3	13	-10	14	-20	4	15
3	13	-10	14	-20	4	15
3	13	-10	14	-20	4	15
3	13	-10	14	-20	4	15
3	13	-10	14	-20	4	15
3	13	-10	14	-20	4	15

Subsequência de Soma Máxima

Arranjo Inicial

3	13	-10	14	-20	4	15

Subsequência de Soma Máxima

Arranjo Inicial

3	13	-10	14	-20	4	15

Arranjo Soma

3	16	6	20	0	4	19

Criamos um arranjo auxiliar com a soma de todos os elementos da posição zero até atual.

Subsequência de Soma Máxima

Arranjo Inicial

3	13	-10	14	-20	4	15

3	13	-10	14	-20	4	15

Arranjo Soma

3	16	6	20	0	4	19

Criamos um arranjo auxiliar com a soma de todos os elementos da posição zero até atual.
Para calcular a soma de uma subsequência qualquer

Programação Dinâmica

Subsequência de Soma Máxima

Arranjo Inicial

3	13	-10	14	-20	4	15

3	13	-10	14	-20	4	15

3	13	-10	14	-20	4	15

Arranjo Soma

3	16	6	20	0	4	19

3	16	6	20	0	4	19

Criamos um arranjo auxiliar com a soma de todos os elementos da posição zero até atual.
Para calcular a soma de uma subsequência qualquer basta pegarmos o valor do arranjo soma correspondente à posição *fim*

Programação Dinâmica

Subsequência de Soma Máxima

Arranjo Inicial

3	13	-10	14	-20	4	15

3	13	-10	14	-20	4	15

3	13	-10	14	-20	4	15

3	13	-10	14	-20	4	15

Arranjo Soma

3	16	6	20	0	4	19

3	16	6	20	0	4	19

3	16	6	20	0	4	19

Criamos um arranjo auxiliar com a soma de todos os elementos da posição zero até atual.

Para calcular a soma de uma subsequência qualquer basta pegarmos o valor do arranjo soma correspondente à posição *fim* e subtrairmos o valor correspondente à posição *início*.

Subsequência de Soma Máxima

Segunda Implementação

```
int subSequenciaMaxima(int A[], int n){
  int ini, fim, x, atual, max;
  max = 0;
  int soma[n];
  soma[0] = A[0];
  for(x=1;x<n;x++) soma[x] = soma[x-1] + A[x];
  for (ini=0; ini<n; ini++){
    for (fim=ini; fim<n; fim++){
      if (ini==0) atual = soma[fim];
      else atual = soma[fim] - soma[ini-1];
      if (atual > max) max = atual;
    }
 }
 return max;
}
```

Subsequência de Soma Máxima

Segunda Implementação

```
int subSequenciaMaxima(int A[], int n){
  int ini, fim, x, atual, max;
  max = 0;
  int soma[n];
  soma[0] = A[0];
  for(x=1;x<n;x++) soma[x] = soma[x-1] + A[x];
  for (ini=0; ini<n; ini++){
    for (fim=ini; fim<n; fim++){
      if (ini==0) atual = soma[fim];
      else atual = soma[fim] - soma[ini-1];
      if (atual > max) max = atual;
    }
 }
 return max;
}
```

Subsequência de Soma Máxima

Segunda Implementação

```
int subSequenciaMaxima(int A[], int n){
  int ini, fim, x, atual, max;
  max = 0;
  int soma[n];
  soma[0] = A[0];
  for(x=1;x<n;x++) soma[x] = soma[x-1] + A[x];    n-1 valores
  for (ini=0; ini<n; ini++){
    for (fim=ini; fim<n; fim++){
      if (ini==0) atual = soma[fim];
      else atual = soma[fim] - soma[ini-1];
      if (atual > max) max = atual;
    }
 }
 return max;
}
```

$O(n)$

Subsequência de Soma Máxima

Segunda Implementação

```
int subSequenciaMaxima(int A[], int n){
  int ini, fim, x, atual, max;
  max = 0;
  int soma[n];
  soma[0] = A[0];
  for(x=1;x<n;x++) soma[x] = soma[x-1] + A[x];     n-1 valores
  for (ini=0; ini<n; ini++){                         n valores
    for (fim=ini; fim<n; fim++){
      if (ini==0) atual = soma[fim];
      else atual = soma[fim] - soma[ini-1];
      if (atual > max) max = atual;
    }
 }
 return max;
}
```

$O(n)$

Subsequência de Soma Máxima

Segunda Implementação

```
int subSequenciaMaxima(int A[], int n){
  int ini, fim, x, atual, max;
  max = 0;
  int soma[n];
  soma[0] = A[0];
  for(x=1;x<n;x++) soma[x] = soma[x-1] + A[x];     n-1 valores
  for (ini=0; ini<n; ini++){                         n valores
    for (fim=ini; fim<n; fim++){                  até n valores
      if (ini==0) atual = soma[fim];
      else atual = soma[fim] - soma[ini-1];
      if (atual > max) max = atual;
    }
 }
 return max;
}
```

$O(n)+O(n^2)$

Subsequência de Soma Máxima

Segunda Implementação

```
int subSequenciaMaxima(int A[], int n){
  int ini, fim, x, atual, max;
  max = 0;
  int soma[n];
  soma[0] = A[0];
  for(x=1;x<n;x++) soma[x] = soma[x-1] + A[x];     n-1 valores
  for (ini=0; ini<n; ini++){                         n valores
    for (fim=ini; fim<n; fim++){                  até n valores
      if (ini==0) atual = soma[fim];
      else atual = soma[fim] - soma[ini-1];
      if (atual > max) max = atual;
    }
 }
 return max;
}
```

$$O(n)+O(n^2) = O(n^2)$$

Subsequência de Soma Máxima

Outra solução

- Existe uma outra solução que mantém apenas dois valores 'memorizados':

Subsequência de Soma Máxima

Outra solução

- Existe uma outra solução que mantém apenas dois valores 'memorizados': o **máximo encontrado** até o momento

Subsequência de Soma Máxima

Outra solução

- Existe uma outra solução que mantém apenas dois valores 'memorizados': o **máximo encontrado** até o momento e a soma da **subsequência atual** (que termina na posição atual)

Subsequência de Soma Máxima

Outra solução

- Existe uma outra solução que mantém apenas dois valores 'memorizados': o **máximo encontrado** até o momento e a soma da **subsequência atual** (que termina na posição atual)
  - Para cada elemento do arranjo, verifica se a soma atual mais este elemento é maior do que o máximo, se sim atualiza o máximo;

Subsequência de Soma Máxima

Outra solução

- Existe uma outra solução que mantém apenas dois valores 'memorizados': o **máximo encontrado** até o momento e a soma da **subsequência atual** (que termina na posição atual)
  - Para cada elemento do arranjo, verifica se a soma atual mais este elemento é maior do que o máximo, se sim atualiza o máximo;
  - Se a soma atual for menor que zero, iguala ela a zero, significando que não há subsequência positiva utilizando o elemento atual que contribua para novas subsequências.

Programação Dinâmica

Subsequência de Soma Máxima

3	13	-10	14	-20	4	15

i: 0
atual: 3
max: 3

Programação Dinâmica

Subsequência de Soma Máxima

3	13	-10	14	-20	4	15

i: 1
atual: 16
max: 16

Programação Dinâmica

Subsequência de Soma Máxima

3	13	-10	14	-20	4	15

i: 2
atual: 6
max: 16

Programação Dinâmica

Subsequência de Soma Máxima

3	13	-10	14	-20	4	15

i: 3
atual: 20
max: 20

Programação Dinâmica

Subsequência de Soma Máxima

3	13	-10	14	-20	4	15

i: 3
atual: 20
max: 20

...

Subsequência de Soma Máxima

Terceira Implementação

```
int subSequenciaSomaMaxima(int A[], int n){
  int max = 0;
  int atual = 0;
  int i;
  for (i=0; i<n; i++){
    atual += A[i];
    if (atual > max) max = atual;
    else if (atual < 0) atual = 0;
  }
  return max;
}
```

Subsequência de Soma Máxima

Terceira Implementação

```
int subSequenciaSomaMaxima(int A[], int n){
  int max = 0;
  int atual = 0;
  int i;
  for (i=0; i<n; i++){
    atual += A[i];
    if (atual > max) max = atual;
    else if (atual < 0) atual = 0;
  }
  return max;
}
```

Subsequência de Soma Máxima

Terceira Implementação

```
int subSequenciaSomaMaxima(int A[], int n){
  int max = 0;
  int atual = 0;
  int i;
  for (i=0; i<n; i++){                                    n valores
    atual += A[i];
    if (atual > max) max = atual;
    else if (atual < 0) atual = 0;
  }
  return max;
}
```

Subsequência de Soma Máxima

Terceira Implementação

```
int subSequenciaSomaMaxima(int A[], int n){
  int max = 0;
  int atual = 0;
  int i;
  for (i=0; i<n; i++){                                    n valores
    atual += A[i];
    if (atual > max) max = atual;
    else if (atual < 0) atual = 0;
  }
  return max;
}
```

$O(n)$

Referências

- Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. Introduction to Algorithms. 2a ed. MIT Press, 2001.
- Gersting, Judith L. Fundamentos Matemáticos para a Ciência da Computação. 3a ed. LTC. 1993.
- Dynamic Programming `https://en.wikipedia.org/wiki/Dynamic_programming`

Aula 23 – Programação Dinâmica

Norton T. Roman & Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2023