Programação Dinâmica

Prof. João Gluz – 2017/02

Programação Dinâmica - Introdução

- Números de Fibonacci
- Sequencia de números muito famosa definida por Leonardo Fibonacci
- Vai nos permitir identificar porque a programação dinâmica é útil
- 0,1,1,2,3,5,8,13,21,34,...

Números de Fibonacci

$F(0) = 0$
$F(1) = 1$
$F(n) = F(n-1) + F(n-2)$

Programação Dinâmica - Introdução

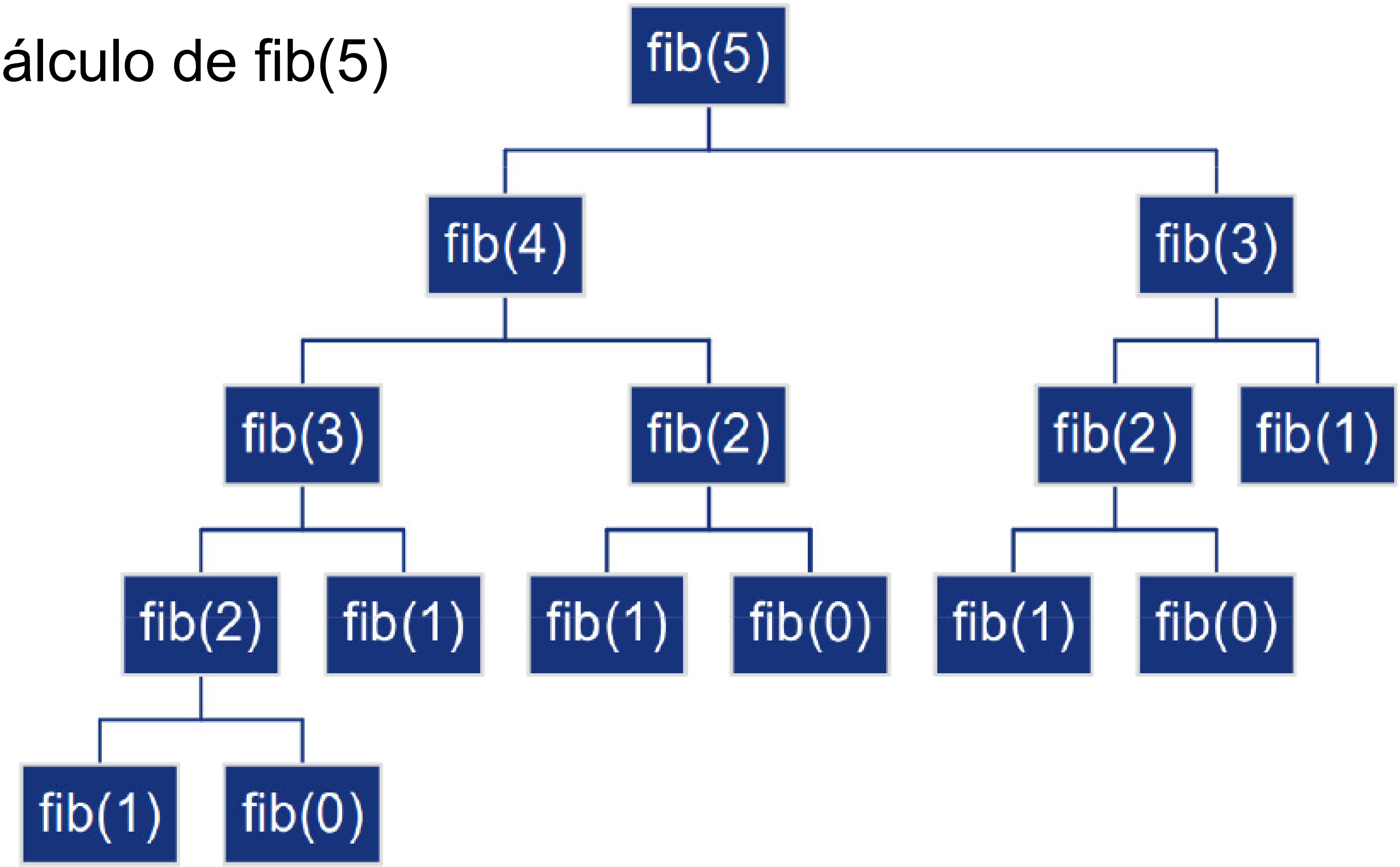
- Implementação elementar diretamente baseada na definição:

fib(**int** n)

1. **if** n=0 ou n=1 **then**
2. **return** n
3. **else**
4. **return** *fib*(n – 1) + *fib*(n - 2)

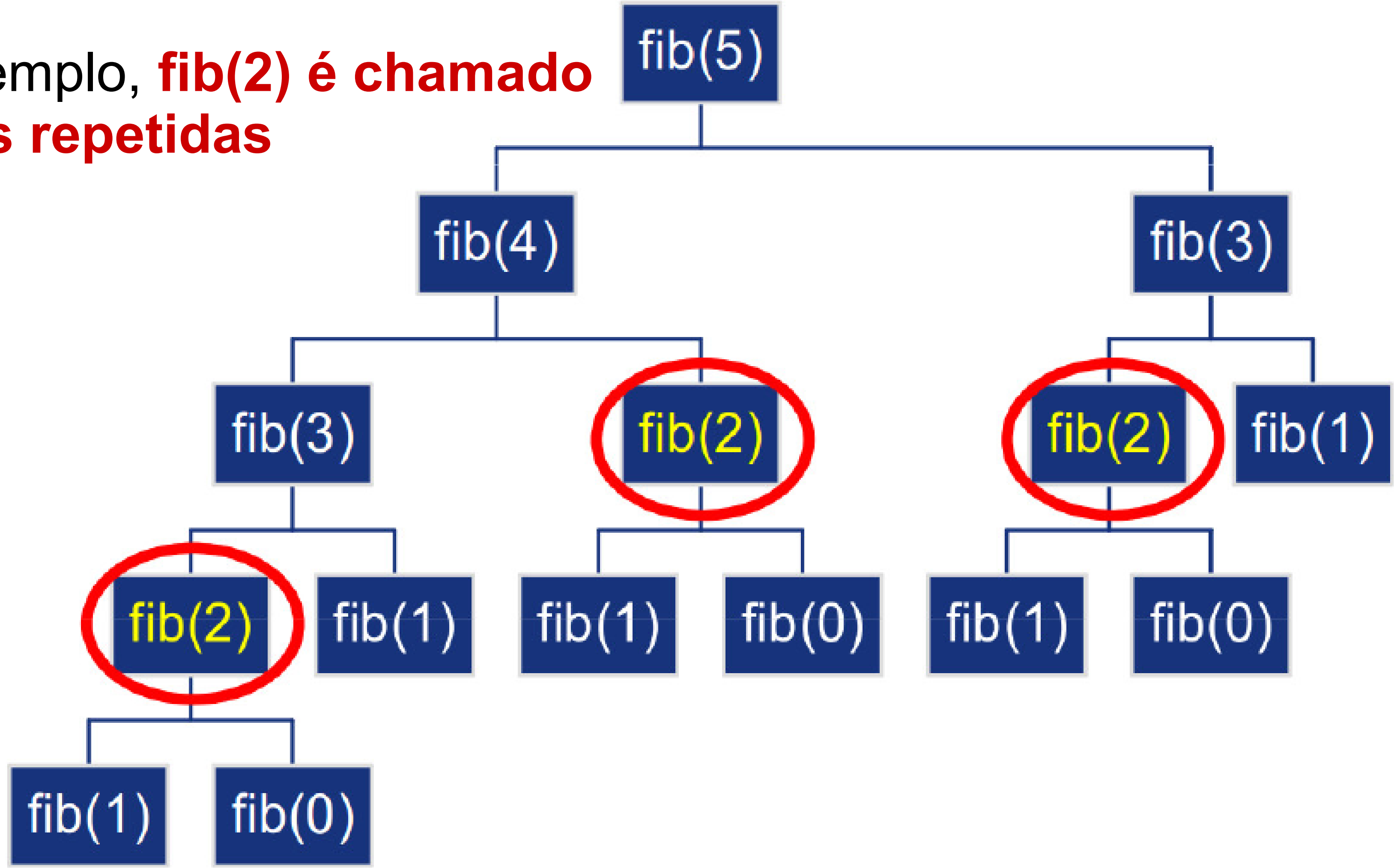
Programação Dinâmica - Introdução

- Pontos negativos da implementação recursiva da função fib diretamente baseada na definição recursiva
- Complexidade exponencial $O(2^n)$
- Cálculo de fib(5)



Programação Dinâmica - Introdução

- Porém existe espaço para melhoras significativas
- Existem muitas repetições nas chamadas
- Por exemplo, **fib(2) é chamado 3 vezes repetidas**



Programação Dinâmica - Introdução

- Como melhorar: usar a memória para manter e recuperar resultados previamente calculados
- Na série de Fibonacci, isso pode ser feito por uma versão iterativa do algoritmos que começa do zero e mantém em memória os dois últimos números calculados da sequência (que são aqueles necessários para construir o próximo número)
- *fibiter*(int in)

1. **if** n=0 ou n=1 **then**
2.   **return** n
3. **else**
4.   **int** f; **int** f1 ← 1; **int** f2 ← 2
5.   **for** i ← 2 **to** n **do**
6.   f ← f1 + f2
7.   f2 ← f1; f1 ← f
8.   **return** f

Programação Dinâmica – Definição

- **Programação Dinâmica:**
- Uma técnica de projeto de algoritmos normalmente usada em problemas de otimização que é baseada em guardar os resultados de subproblemas em vez de os recalcular.
- Clássica troca de espaço por tempo
- As vezes é denominada de programação tabular pelo uso de tabelas para armazenar as soluções pré-calculadas
  - Problema de Otimização: busca encontrar a "melhor" solução entre todas as soluções possveis, segundo um determinado critério (função objetivo). Geralmente busca descobrir um máximo ou mínimo dessa função

Programação Dinâmica – Idéias

- Idéias importantes que formam a base da Programação Dinâmica:
  - Dividir de um problema em subproblemas do mesmo tipo
  - Calcular o mesmo subproblema apenas uma vez
- Essas idéias podem ser aplicadas em variados tipos de problemas

Programação Dinâmica – Características

- Quais são, então, as características que um problema deve apresentar para poder ser resolvido com PD?
- Ter soluções com **subestrutura ótima**
- Ter **subproblemas coincidentes**
- Subestrutura Ótima:
  - Já vimos isso antes no caso dos algoritmos gulosos: uma solução tem subestrutura ótima quando a solução ótima de um problema contém nela própria soluções ótimas para subproblemas do mesmo tipo
- Subproblemas Coincidentes
  - Quando um *espaço de subproblemas* é "pequeno", isto e, não são muitos os subproblemas a resolver pois muitos deles são exatamente iguais uns aos outros.

Programação Dinâmica - Exemplo

- **Pirâmide de Numeros**
- Problema "clássico" das Olimpíadas Internacionais de Informática de 1994
- Calcular o caminho, que começa no topo da pirâmide e termina na base, com maior soma. Em cada passo pode-se ir diagonalmente para baixo e para a esquerda ou para baixo e para a direita.

```
        7
      3   8
    8   1   0
  2   7   4   4
4   5   2   6   5
```

Programação Dinâmica – Exemplo

- **Pirâmide dos Números**
  - Caminhos possíveis

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

Soma = **21**

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

Soma = **30**

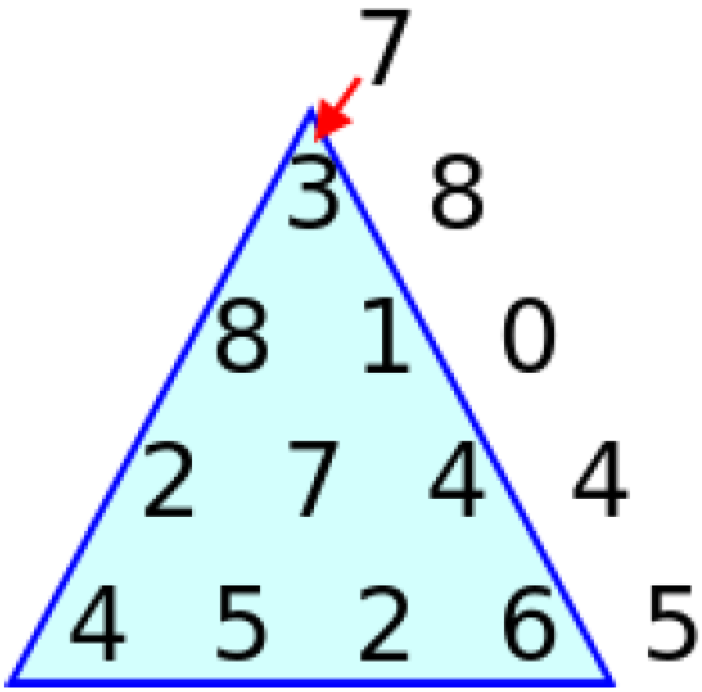
  - Restrições: todos os números são inteiros entre 0 e 99 e o número de linhas do triângulo é no máximo 100.

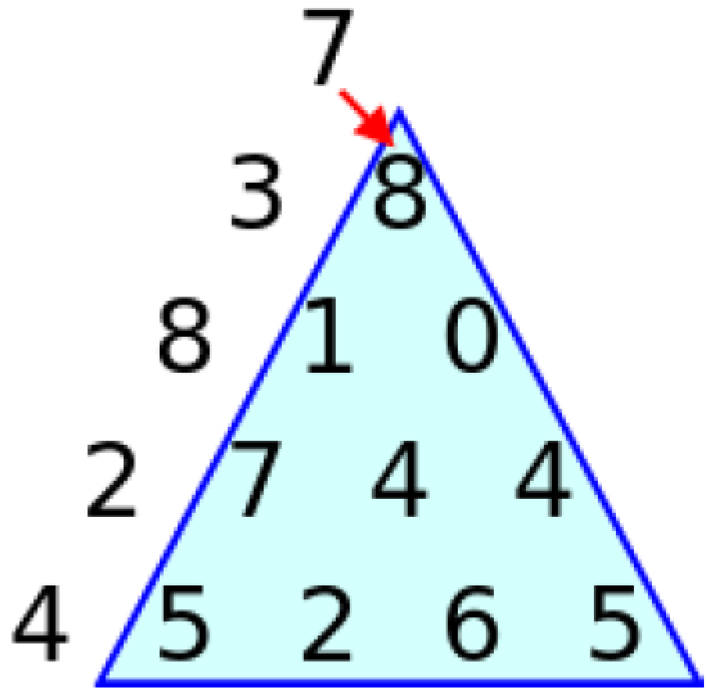
Programação Dinâmica – Exemplo

- **Pirâmide de Números**
- Como resolver o problema?
  - Ideia: Pesquisa Exaustiva (Força Bruta) - avaliar todos os caminhos possíveis e ver qual o melhor.
- Mas quanto tempo demora isto? Quantos caminhos existem?
  - Em cada linha há duas decisões: esquerda ou direita
  - Se *n* é a altura da pirâmide, um caminho tem *n* - 1 decisões!
  - Então, existem então 2^{n-1} caminhos diferentes
- Um programa para calcular todos os caminhos tem portanto complexidade exponencial $O(2^n)$:
  - $2^{99} \sim 6{,}34 * 10^{29}$ (633825300114114700748351602688)

Programação Dinâmica – Exemplo

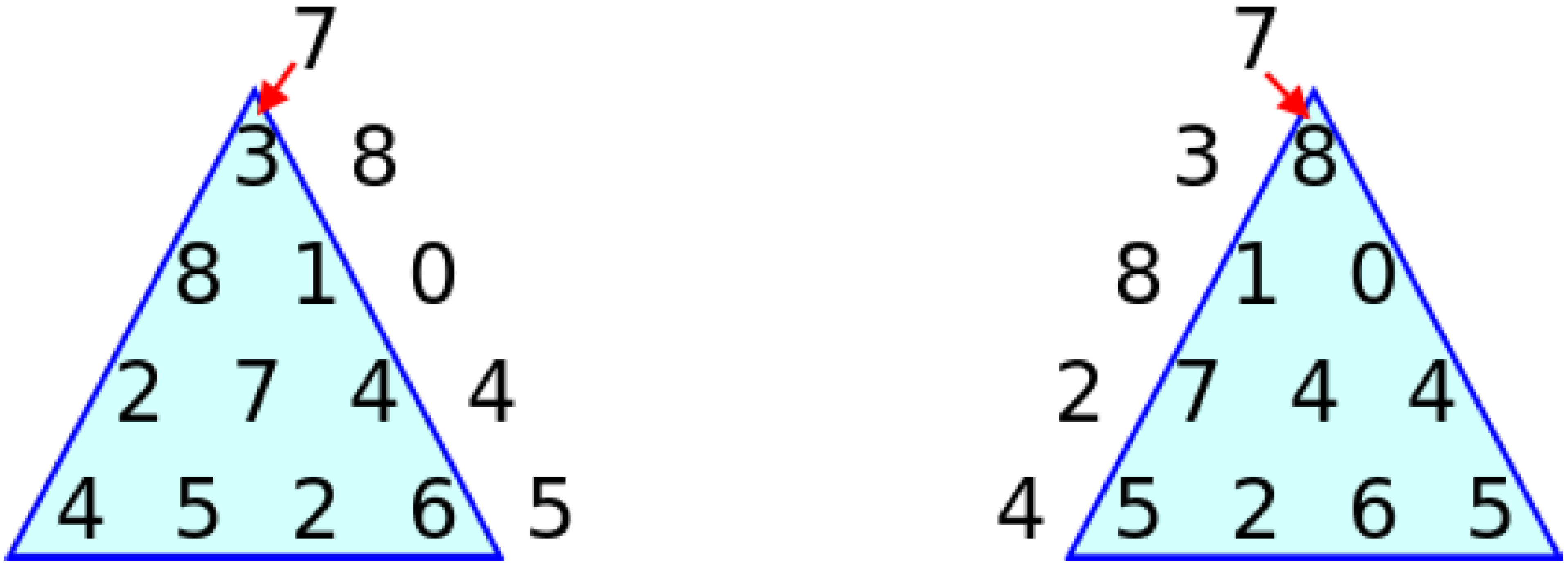
- **Pirâmide de Números**
- Quando estamos no topo da pirâmide, temos duas decisões possíveis (esquerda ou direita):
- Em cada um dos casos temos de ter em conta todas os caminhos das respectivas subpirâmides.





Programação Dinâmica – Exemplo

- **Pirâmide de Números**
- Mas o que nos interessa saber sobre estas subpirâmides?
  - Apenas interessa o valor da sua melhor rota interna - que é uma **instância mais pequena do mesmo problema**!
- Para o exemplo, a solução é 7 mais o máximo entre o valor do melhor caminho de cada uma das subpirâmides.



Programação Dinâmica – Exemplo

- **Pirâmide de Números**
- Pelo fato do problema da pirâmide poder ser reduzido a duas instâncias menores do mesmo problema isso indica que este problema pode ser resolvido recursivamente
- Uma pirâmide de n níveis pode ser representada por uma matriz diagonal onde P[i][j] armazena o j-ésimo número da i-ésima linha, para i<n e j<=i
- A função PiraMax(n, P, i, j) calcula o melhor (maior) valor que se consegue da posição i,j em uma pirâmide P com n níveis

	1	2	3	4	5
1	7				
2	3	8			
3	8	1	0		
4	2	7	4	4	
5	4	5	2	6	5

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

Programação Dinâmica – Exemplo

- **Pirâmide de Números – Primeira Versão**

- **int** PiraMax(**int** n, int P[][], **int** i, **int** j)

```
if i = n then
    return P[i][j]
else
    return P[i][j] + max(PiraMax(n,P, i+1, j),
                         PiraMax(n,P, i+1, j+1))
```

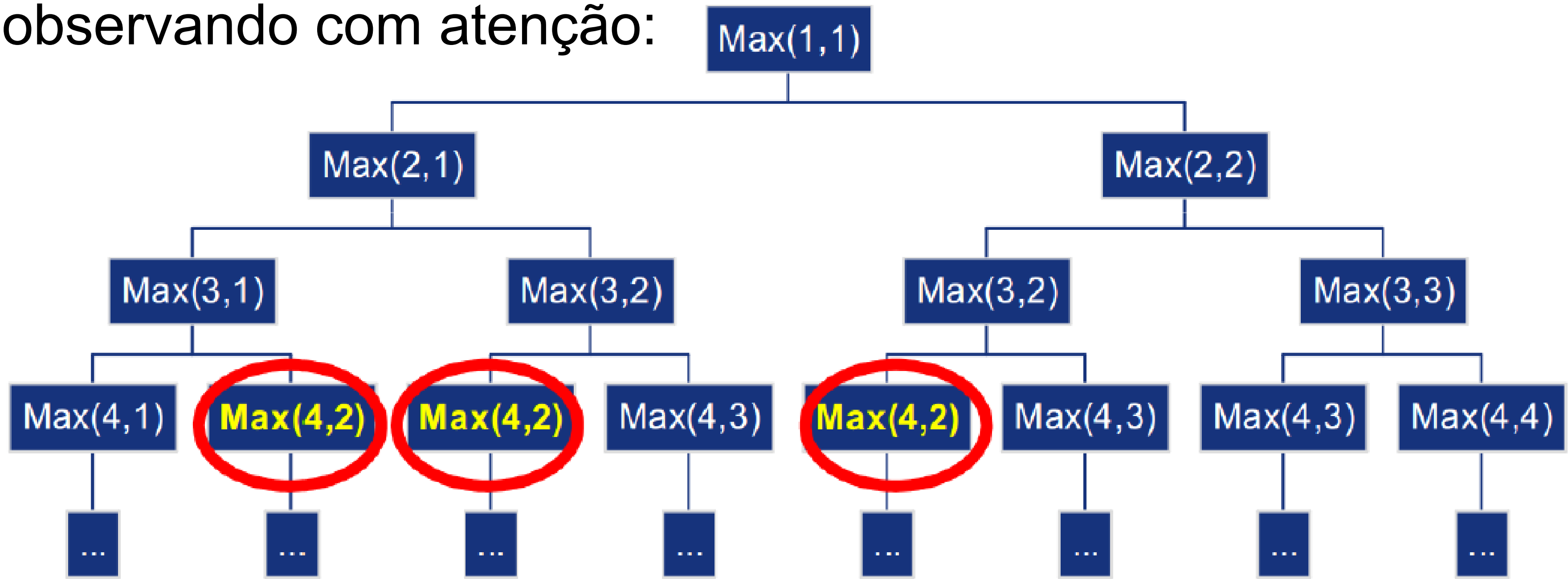
- Para resolver o problema basta chamar PiraMax(1, 1)

	1	**2**	**3**	**4**	**5**
1	7				
2	3	8			
3	8	1	0		
4	2	7	4	4	
5	4	5	2	6	5

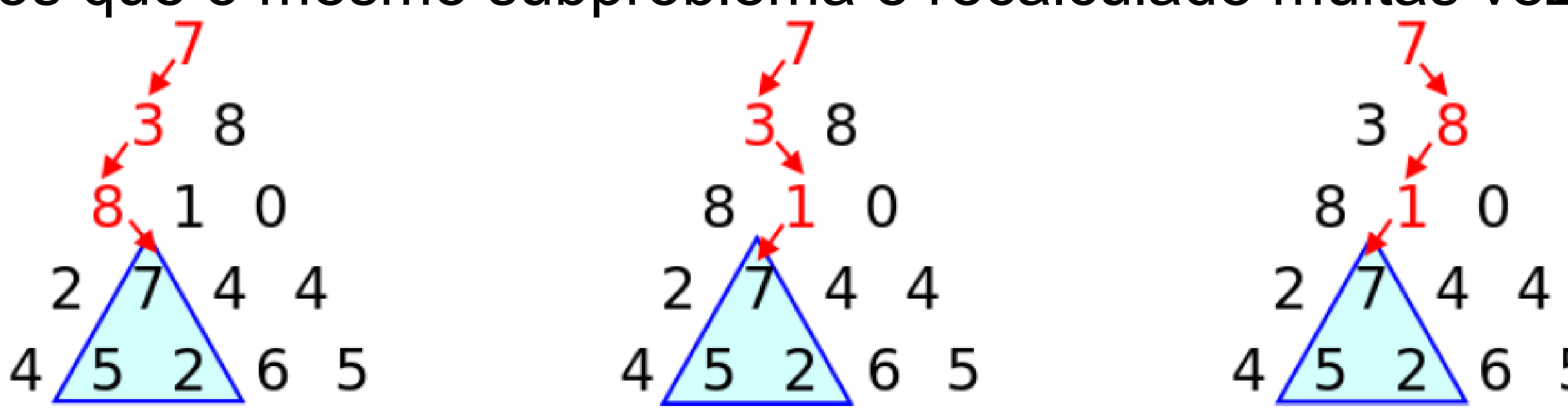
```
        7
      3   8
    8   1   0
  2   7   4   4
4   5   2   6   5
```

Programação Dinâmica – Exemplo

- **Pirâmide de Números – Primeira Versão**
- Versão elementar tem crescimento exponencial, porém observando com atenção:

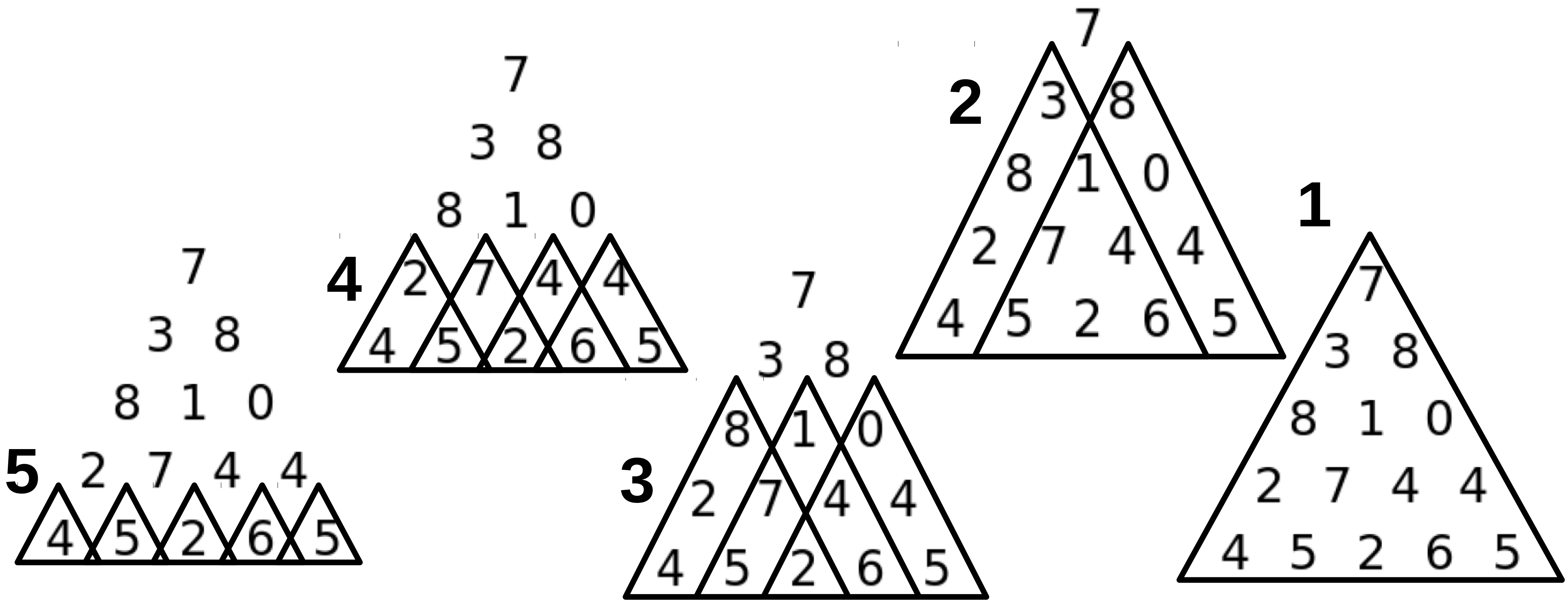


- Vemos que o mesmo subproblema é recalculado muitas vezes:



Programação Dinâmica – Exemplo

- Exemplo de Subproblemas Coincidentes
- No problema das pirâmides, para um determinada instância do problema, existem apenas **$n + (n-1) + ... + 1 < n^2$** subproblemas diferentes porque, como já vimos, muitos subproblemas são coincidentes



Programação Dinâmica – Exemplo

- Portanto no caso do problema das pirâmides os subproblemas na verdade crescem polinomialmente e não exponencialmente como sugere a análise inicial
- Porém, da mesma forma que no caso da subestrutura ótima deve-se ter cuidado, porque essa característica nem sempre acontece ou as vezes não é útil:
  - Mesmo com subproblemas coincidentes, ainda assim sobram muitos subproblemas a resolver (por exemplo, subproblemas não coincidentes restantes ainda podem ter crescimento exponencial)
  - Ou simplesmente não existem subproblemas coincidentes.

Programação Dinâmica - Metodologia

- Se um problema apresenta as duas características de subestrutura ótima e subproblemas coincidentes há uma boa chance de que a PD se pode aplicar.
  - Que passos deve-se seguir então para resolver um problema com PD?
- Guia para resolver com PD

1) **Caracterizar a solução ótima do problema**
2) **Definir recursivamente a solução ótima, em função de soluções ótimas de subproblemas**
3) **Obter as soluções de todos os subproblemas**: o armazenamento das soluções pode seguir a estratégia "de trás para a frente" (*bottom-up*) ou usar memoização (*top-down*)
4) **Reconstruir a solução ótima, baseada nos cálculos efetuados** (opcional - apenas se for necessário)

Programação Dinâmica – Metodologia

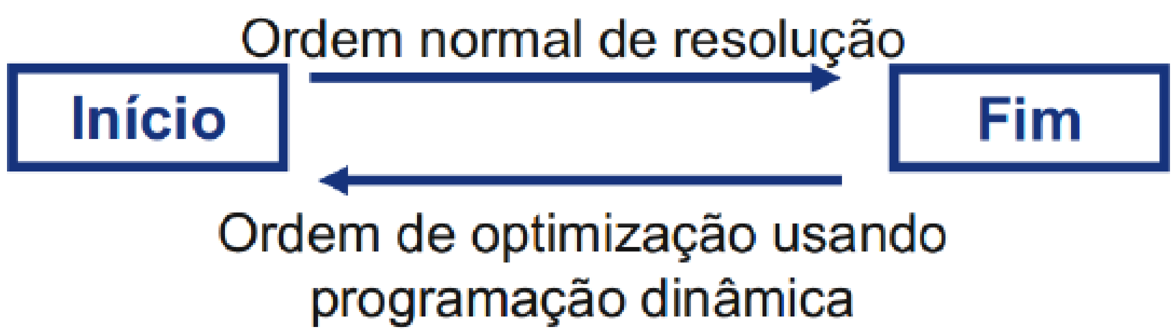
- **1) Caracterizar a solução ótima do problema**
- Compreender bem o problema
- Verificar se um algoritmo que obtenha todas as soluções usando força bruta não é suficiente
- Tentar generalizar o problema (é preciso prática para perceber como generalizar da maneira correta)
- Procurar dividir o problema em subproblemas do mesmo tipo (divisão e conquista)
- Verificar se o problema tem solução com subestrutura ótima
- Verificar se existem subproblemas coincidentes

Programação Dinâmica – Metodologia

- **2) Definir recursivamente a solução ótima, em função de soluções ótimas de subproblemas**
- Definir recursivamente o valor da solução ótima, com rigor e exatidão, a partir de subproblemas mais pequenos do mesmo tipo
- Imaginar sempre que os valores das soluções ótimas ja estão disponíveis quando precisar deles
- Não é necessário codificar: basta definir a recursão de forma rigorosa e precisa (matematicamente)

Programação Dinâmica – Metodologia

- **3) Calcular as soluções de todos os subproblemas: "de tras para a frente" (*bottom-up*)**
- Descobrir a ordem em que os subproblemas dependem uns dos outros começando dos subproblemas mais pequenos até chegar ao problema global (isso é a abordagem bottom-up) e codificar, usando uma tabela para registrar as soluções já calculadas.
- Normalmente esta ordem é inversa em relação à ordem normal da função recursiva que resolve o problema



Programação Dinâmica – Metodologia

- **3) Calcular as soluções de todos os subproblemas usando memoização (*top-down*)**
- Existe uma técnica, conhecida em inglês como "memoization" e traduzida ao português como memoização (note que não é o mesmo que memorização), que permite resolver o problema pela ordem normal ("top-down") da função recursiva
- Com a memoização pode-se usar a função recursiva obtida diretamente a partir da definição da solução e ir mantendo uma tabela com os resultados dos subproblemas.
- A idéia da memoização é simples: dá primeira vez que se necessita de um valor ele deve ser calculado e guardado na tabela, porém partir daí basta obter o resultado já calculado armazenado na tabela

Programação Dinâmica – Metodologia

- **4) Reconstruir a solução ótima, baseada nos cálculos efetuados**
- Pode ou não ser um requisito do problema
- Duas alternativas:
  - A solução pode ser construída diretamente a partir da tabela dos sub-problemas, ou
  - Deve-se criar uma nova tabela que guarda as decisões em cada etapa

Programação Dinâmica – Exemplo

- **Voltando à Pirâmide dos Números**
- Como o problema tem subestrutura ótima e subproblemas coincidentes pode-se usar a PD
- Ou seja, a saída para otimizar o algoritmo é reaproveitar o que já foi calculado, ou seja:
  - **Só calcular uma vez o mesmo subproblema**
- Ideia: criar uma tabela para armazenar os valores já calculados de cada subproblema
  - Matriz M[i][j]
- O valor armazenado em uma dada posição i,j já foi calculado se não é negativo (lembre-se que os valores da pirâmide ficam entre 0 e 99)
- Assim no início vamos assumir que os valores armazenados em M[i][j] são todos negativos,

Programação Dinâmica – Exemplo

- **Pirâmide de Números – Versão Top-Down com memoização**
- **int** PiraMaxPDTD(**int** n, **int** P[][], **int** M[][], **int** i, **int** j)

```
if M[i][j] ≥ 0 then
      return M[i][j]
if i = n then
      M[i][j] ← P[i][j]
      return M[i][j]
else
      M[i][j] ← P[i][j] + max(PiraMaxPDTD(n,P, i+1, j),
                              PiraMaxPDTD(n,P, i+1, j+1))
      return M[i][j]
```

Programação Dinâmica – Exemplos

- Qual a complexidade de PiraMaxPDTP(n,P,M,i,j) em relação ao número de níveis n?
- A aplicação direta de equações de recorrência não é possível por conta da técnica de memoização
  - Não dá pra "simplesmente" criar uma equação de recorrência a partir do código e então resolver essa equação para obter a ordem de complexidade
- Para calcular a complexidade é importante entender que a **memoização** funciona efetivamente como um **corte** ou **limite** para a quantidade de recursões que serão possíveis no algoritmo

Programação Dinâmica – Exemplos

- Qual a complexidade de PiraMaxPDTP(n,P,M,i,j) em relação ao número de níveis n?
- O ponto chave para se identificar qual a complexidade do algoritmo acima é entender que pela estrutura dos subproblemas coincidentes da pirâmide dos números, somente $\mathbf{n^2}$ **chamadas recursivas** serão feitas pelo algoritmo, em todos os outros casos a resposta será dada diretamente pela tabela M[i][j] ou por P[i][j]
- O teste da linha 1 da técnica de memoização corta (ou limita) o número de recursões para apenas $\mathbf{n^2}$
- Se o tempo de execução das etapas não recursivas for uma constante c, então $T(n) = c*\mathbf{n^2}$ e portanto

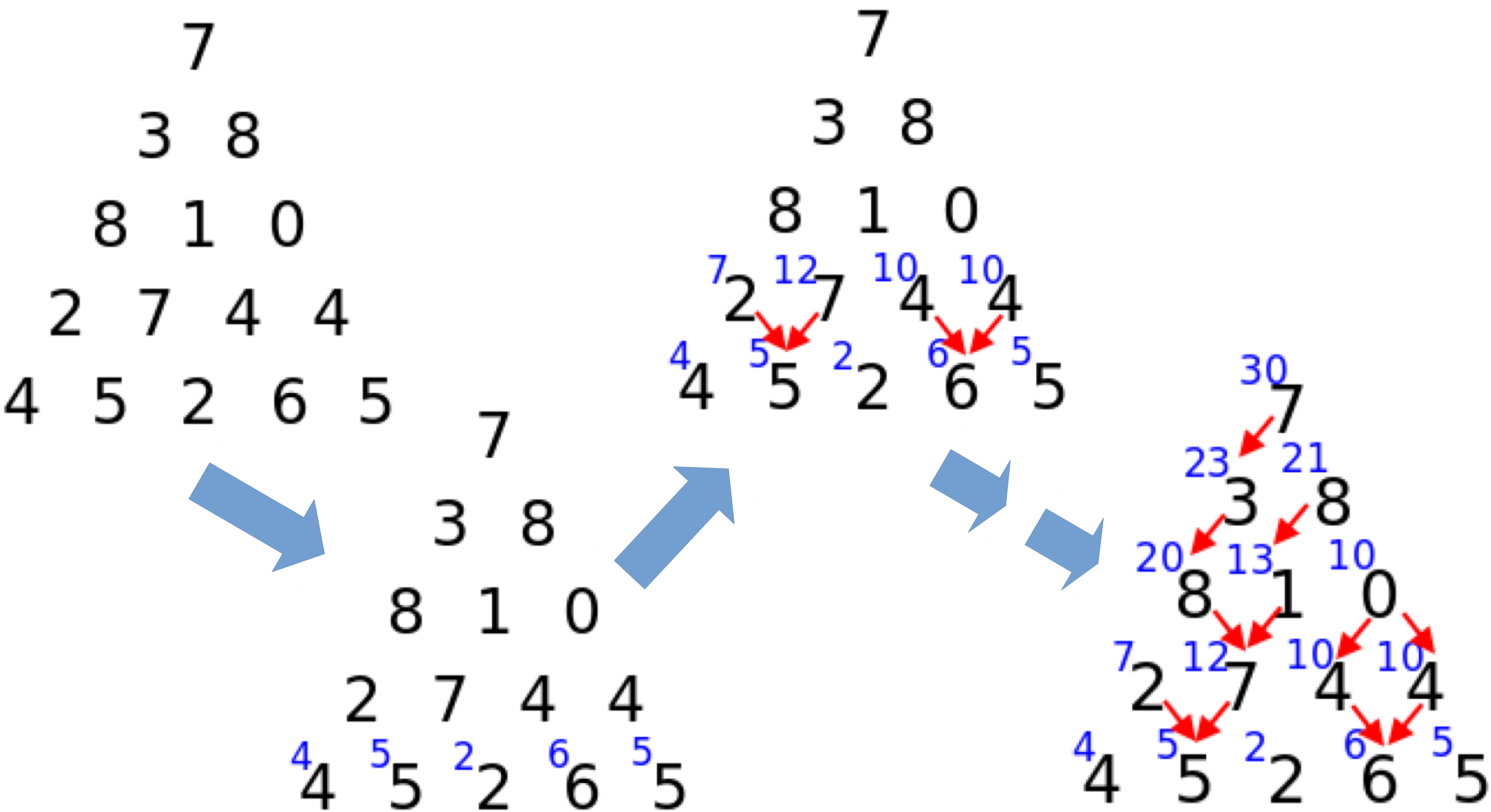
  $T(n) = \mathbf{O}(\mathbf{n^2})$

Programação Dinâmica – Exemplo

- **Pirâmide dos Números – Versão Bottom-Up**
- Abordagem de trás pra frente (*bottom-up*) para construir o algoritmo
- Nesse caso precisa-se descobrir se existe uma ordem para preencher a tabela de modo que quando necessitamos de um valor ele já está na tabela?
- Sim essa ordem existe:
  - deve-se começar do fim (da base) da pirâmide (e não do topo como feito na primeira versão recursiva) para registrar os valores já calculados na ordem correta

Programação Dinâmica – Exemplo

- **Pirâmide dos Números**
- Começando da base da pirâmide



Programação Dinâmica – Exemplo

- **Pirâmide dos Números – Versão Bottom-Up**
- Uma pequena otimização adicional:
  - Tendo em conta que começamos a registrar os máximos da base da pirâmide para cima, pode-se utilizar a própria matriz P[i][j] para registrar esses máximos, ao invés de criar uma nova matriz M[i][j]
- **int** PiraMaxPDBU(**int** n, **int** P[][])
  1. **int** i,j
  2. **for** i ← n-1 **downto** 1 **do**
  3.   **for** j ← 1 **to** i **do**
  4.   P[i][j] ← P[i][j] + max(P[i+1][j], P[i+1][j+1])
  5. **return** P[1][1]
- A solução calculada fica armazenada em P[1][1] após a execução do algoritmo

Programação Dinâmica – Exemplo

- **Pirâmide dos Números – Versão Bottom-Up**
- Note que a implementação de PiraMaxPDBU é iterativa e o tempo necessário para resolver o problema agora cresce de forma polinomial ($O(n^2)$)
- Isso é uma boa solução para o problema ($99^2 = 9801$)
- No caso da pirâmide dos números a solução bottom-up é melhor por evitar o overhead das chamadas recursivas e por usar menos memória
  - A complexidade de espaço de PiraMaxPDTP é **O**(n) enquanto que a complexidade de espaço de PiraMaxPDBU é **O**(1)
- Em alguns caso, o uso de memoização recursiva pode ser melhor, particularmente se nem todos os subproblemas possíveis tiverem que ser resolvidos para um problema particular