Programação Dinâmica

Tópicos Avançados de Programação

Humberto Longo

Instituto de Informática
Universidade Federal de Goiás

Bacharelado em Ciência da Computação, 2007

Problemas de Programação Dinâmica

Estrutura dos problemas

- Geralmente possuem estruturas recursivas.
  - Sub-problemas devem ser resolvidos para que a solução final possa ser alcançada.
- Em geral aplicados a problemas de otimização.
  - Buscam uma solução ótima.

Por que não recursão?

- Se a estrutura do problema é recursiva, por que um algoritmo recursivo não seria uma boa solução?
  - Algoritmos puramente recursivos trabalharão mais que o necessário resolvendo os sub-problemas comuns mais de uma vez (repetição de cálculos).
- Uma boa estratégia seria ter uma tabela para:
  - Guardar os resultados dos sub-problemas que já foram resolvidos.
  - Fazer consultas na tabela para encontrar os resultados de tais sub-problemas.

Definição (Programação Dinâmica)

Técnica de programação que visa resolver problemas que exigem que diversos cálculos de sub-problemas sejam refeitos. Para isso, os resultados parciais são armazenados, não sendo necessário recalculá-los.

Exemplo de uso

Exemplo (Distância de Edição)

- Problema de se comparar duas palavras (*strings*) para saber o custo necessário para transformar uma na outra.
- Normalmente é usado o algoritmo de distância de edição, que calcula a melhor forma de resolver o problema utilizando programação dinâmica.

Exemplo de uso

Exemplo (Distância de Edição)

- Dadas duas palavras t e p, define-se a distância de edição $D[t, p]$ entre elas como o custo total mínimo necessário para transformar t em p ou vice-versa.
- Para calcular tal custo, define-se custos de operações de edição de uma palavra, que podem ser:
  1. Substituição (*Replacement*).
  2. Inserção (*Insertion*).
  3. Remoção (*Deletion*).
  4. Casamento (*Match*).
- A operação de casamento não é contada na distância de edição, ou seja, custo 0.
- Assim, valores menores indicam menor distância de edição.

Exemplo de uso

Exemplo (Distância de Edição)

- Algoritmo:
  - Dados duas palavras (A e B) e os custos de inserção, remoção e substituição de elementos.
  - Calcular o custo para transformar a palavra A na palavra B através do cálculo do custo de transformação de subcadeias de A em subcadeias de B.
  - Armazenar os resultados em uma tabela.

Exemplo de uso

Exemplo (Distância de Edição)

- O tamanho da matriz é $(M+1) \times (N+1)$, onde M e N são os tamanhos das cadeias A e B respectivamente.
- Cada linha da tabela representa uma subcadeia de A e cada coluna representa uma subcadeia de B.
  - A linha i representa a cadeia com os i primeiros caracteres de A (cadeia vazia para $i = 0$).
  - A coluna j representa a cadeia com os j primeiros caracteres de B.
  - A posição $[i][j]$ da tabela representa o custo de transformar a subcadeia com i caracteres de A na subcadeia com j caracteres de B.

Exemplo de uso

Exemplo (Distância de Edição)

- Matriz é iniciada com 0 na posição $[0][0]$.
- Às demais posições da linha 0 é adicionado o custo de uma inserção ao elemento da esquerda.
- Às demais posições da coluna 0 é adicionado o custo de remoção ao elemento de cima.

		P	A	I
	0	1	2	3
C	1			
A	2			
S	3			
A	4			

Exemplo de uso

Exemplo (Distância de Edição)

- Para as demais posições $[i][j]$, o algoritmo verifica se é melhor:

1. Transformar a subcadeia i na $j-1$ e inserir um elemento;
   $cas \rightarrow pa$: transforma cas em p e insere a no fim da subcadeia.
2. Transformar a subcadeia $i-1$ na j e remover um elemento;
   $cas \rightarrow pa$: transforma ca em pa e depois remove o último caractere (s).
3. Transformar a subcadeia $i-1$ na $j-1$ e substituir um elemento (caso o elemento $A[i]$ seja diferente do $B[j]$);
   $cas \rightarrow pa$: transforma ca em p e depois substitui o último caractere (s) por a.

Exemplo de uso

Exemplo (Distância de Edição)

- Custo para transformar $CASA$ em PAI:

Inicialização

		P	A	I
	0	1	2	3
C	1			
A	2			
S	3			
A	4			

⇒

Substituição

		P	A	I
	0	1	2	3
C	1	1		
A	2			
S	3			
A	4			

⇒

Substituição

		P	A	I
	0	1	2	3
C	1	1	2	
A	2			
S	3			
A	4			

⇒

Substituição

		P	A	I
	0	1	2	3
C	1	1	2	3
A	2			
S	3			
A	4			

⇒

Substituição

		P	A	I
	0	1	2	3
C	1	1	2	3
A	2			
S	3			
A	4			

⇒

Substituição

		P	A	I
	0	1	2	3
C	1	1	2	3
A	2	2		
S	3			
A	4			

⇒

Casamento

		P	A	I
	0	1	2	3
C	1	1	2	3
A	2	2	1	
S	3			
A	4			

⇒

Inserção

		P	A	I
	0	1	2	3
C	1	1	2	3
A	2	2	1	2
S	3			
A	4			

⇒

Substituição

		P	A	I
	0	1	2	3
C	1	1	2	3
A	2	2	1	2
S	3	3		
A	4			

⇒

Exemplo de uso

Exemplo (Distância de Edição)

- Custo para transformar $CASA$ em PAI:

Remoção

		P	A	I
	0	1	2	3
C	1	1	2	3
A	2	2	1	2
S	3	3	2	
A	4			

$\Rightarrow$

Substituição

		P	A	I
	0	1	2	3
C	1	1	2	3
A	2	2	1	2
S	3	3	2	2
A	4			

$\ldots$

Remoção

		P	A	I
	0	1	2	3
C	1	1	2	3
A	2	2	1	2
S	3	3	2	2
A	4	4	3	3

- Custo no pseudo-código:

R Remoção,
I Inserção,
S Substituição,
0 Casamento.

Exemplo de uso

Exemplo (Distância de Edição)

Dist(A, B)

Entrada: Cadeias A, B
Saída: Inteiro

```
1  m ← |A|;
2  n ← |B|;
3  M[0][0] ← 0;
4  para i ← 1 até m faça
5  |  M[i][0] ← M[i − 1][0] + R;
6  para j ← 1 até n faça
7  |  M[0][j] ← M[0][j − 1] + I;
8  para i ← 1 até m faça
9  |  para j ← 1 até n faça
10 |  |  se A[i] = B[j] então
11 |  |  |  custoExtra ← 0;
12 |  |  senão
13 |  |  |  custoExtra ← S;
   |  |
   |  |  M[i][j] ← min ( M[i − 1][j]     + R,
   |  |                 M[i][j − 1]     + I,
14 |  |                 M[i − 1][j − 1] + custoExtra );
15 retorna M[m][n];
```

Exemplo de uso

Exemplo (Problema da Mochila)

- Busca calcular a melhor maneira de se armazenar em um compartimento qualquer muitos itens com valores agregados, de forma que o somatório dos valores de tais itens seja o máximo possível dentre as possibilidades de combinação de itens.

- Matriz bidimensional (M), de ordem $(N+1) \times (C+1)$, onde:

  N: Número de itens;

  C: Capacidade da mochila.

Exemplo de uso

Exemplo (Problema da Mochila)

- Idéia do algoritmo:
  1. Começar com uma mochila de capacidade 1 e descobrir o valor máximo possível para esta mochila.
  2. Passar para uma mochila de capacidade 2 e aproveitar as informações da mochila de capacidade atual - 1 (nesse caso, 1) para descobrir o valor máximo para uma mochila de capacidade 2.
  3. Repetir esse procedimento até capacidade atual ser igual a C. Além da capacidade ir crescendo, também vai se adicionando novos itens no processo.

Exemplo de uso

Exemplo (Problema da Mochila)

- Idéia do algoritmo:
  1. Percorrendo a tabela por linha (por item), e para a coluna j atual (capacidade atual), analisar se o item atual i (linha atual) cabe na mochila de capacidade j.
  2. Se couber é preciso escolher o maior valor entre:
     - 2.1 Valor na mochila de mesma capacidade j que não tinha esse item (essa informação estará em $M[i-1][j]$).
     - 2.2 Soma do valor do item com o valor na mochila de capacidade ($j-$ peso do item i) e que não tinha esse item ($M[i-1][j-$ peso do item $i]$).
  3. Se não couber, o maior valor para essa mochila será o valor da mochila de mesma capacidade, mas que não tem esse item ($M[i][j] = M[i-1][j]$).

Exemplo de uso

Exemplo (Problema da Mochila)

Knapsack(N, C)

Entrada: Número de produtos (N) e Capacidade da mochila (C)
Saída: Valor máximo para a capacidade C

```
1  M[0][0] ← 0;
2  para i ← 1 até N faça
3      M[i][0] ← 0;
4  para j ← 1 até C faça
5      M[0][j] ← 0;
6  para i ← 1 até N faça
7      para j ← 1 até C faça
8          se itens[i].peso ≤ j então
9              M[i][j] ← max( M[i − 1][j],
                              M[i − 1][j − itens[i].peso] + itens[i].valor );
10         senão
11             M[i][j] ← M[i − 1][j];
12 retorna M[N][C];
```

Exemplo de uso

Exemplo (Problema da Mochila)

Item	Peso	Valor
1	4	2
2	2	1
3	1	3
4	2	4
5	2	1