Projeto de algoritmos: uma visão prática

Aula 02

Rafael Melo
Instituto de Computação
Universidade Federal da Bahia

Programação dinâmica

Motivação

- Técnica poderosa para resolver problemas de otimização
  - Encontrar uma solução com valor ótimo
  - Minimização ou maximização
- Define soluções ótimas para problemas baseadas nas soluções ótimas para subproblemas
  - evita recálculos de subproblemas em comum
- Utiliza tabulação (ou memorização)

Passos da programação dinâmica

- Caracterize a estrutura de uma solução ótima.
- Defina uma recorrência para o cálculo de uma solução ótima considerando a otimalidade dos subproblemas.
- Compute o valor de uma solução ótima de forma *bottom-up*.
  - Definindo uma ordem do cálculo dos subproblemas
- Construa a solução ótima utilizando a informação computada.

Programação dinâmica é difícil de entender?

- Programação dinâmica parece complicada de se entender em um primeiro momento
- Uma vez compreendida, torna-se razoavelmente fácil de ser aplicada.
- Programação dinâmica parece mágica até que você tenha visto exemplos o suficiente

Problema da mochila 0-1

Problema da mochila 0-1

- **Entrada:** um conjunto $I = \{1, \ldots, n\}$ de items, uma utilidade c_i e um peso a_i para cada $i \in I$, e uma capacidade b da mochila.
- **Saída:** um subconjunto $I' \subseteq I$ dos items que maximize a soma total das utilidades de forma que a soma dos pesos não ultrapasse a capacidade da mochila.

Exemplo

I = {1, 2, 3, 4}

c = (10, 6, 15, 9)

a = (2, 1, 3, 2)

b = 3

- **Algumas possíveis soluções:**

  I' = {1}, utilidade: 10, peso: 2

  I' = {1, 2}, utilidade: 16, peso: 3

  I' = {3}, utilidade: 15, peso: 3

- **Representação das soluções:**

  x = (1, 0, 0, 0)

  x = (1, 1, 0, 0)

  x = (0, 0, 1, 0)

Formulação matemática do problema

$$
\begin{aligned}
P : z = \quad & Max \quad \sum_{j=1}^{n} c_j x_j \\
& s.\ a : \quad \sum_{j=1}^{n} a_j x_j \leq b \\
& \qquad\quad x_j \in \{0,1\},\ j = 1, \ldots, n
\end{aligned}
$$

Definição dos subproblemas

- $P_k(\lambda)$: problema da mochila restrita aos k primeiros itens e uma mochila de capacidade λ
- $f_k(\lambda)$: valor da solução ótima para o problema $P_k(\lambda)$

Formulação matemática dos subproblemas

$$
\begin{aligned}
P_k(\lambda) : f_k(\lambda) = \quad & Max \quad \sum_{j=1}^{k} c_j x_j \\
& s.\ a: \quad \sum_{j=1}^{k} a_j x_j \leq \lambda \\
& \qquad\quad x_j \in \{0,1\},\ j = 1, \ldots, k
\end{aligned}
$$

Definição dos subproblemas

- Problema original da mochila
  - $P_n(b)$, com valor ótimo $z = f_n(b)$
- Pergunta:
  - Como calcular $f_k(\lambda)$ recursivamente baseado nos valores $f_s(u)$ para $s \leq k$ e $u \leq \lambda$?

Determinando a recorrência

- O que é possível dizer sobre a solução ótima x^* para $P_k(\lambda)$ com valor $f_k(\lambda)$?
  - Duas possibilidades: $\boldsymbol{x_k^* = 0}$ ou $\boldsymbol{x_k^* = 1}$
- **Caso 1:** $x_k^* = 0$
  - solução ótima satisfaz $f_k(\lambda) = f_{k-1}(\lambda)$
- **Caso 2:** $x_k^* = 1$
  - solução ótima satisfaz $f_k(\lambda) = c_k + f_{k-1}(\lambda - a_k)$

Determinando a recorrência

- Combinando Caso 1 e Caso 2, temos:
  - $f_k(\lambda) = max \{ f_{k-1}(\lambda) , c_k + f_{k-1}(\lambda - a_k) \}$
- Casos base:
  - $f_0(\lambda) = 0$ para $\lambda \geq 0$

Obtenção da solução ótima

- Podemos manter um indicador $p_k(\lambda)$
  - $p_k(\lambda) = 0$, se $f_k(\lambda) = f_{k-1}(\lambda)$,
  - $p_k(\lambda) = 1$, caso contrário.
- Obtendo a solução:
  - se $p_n(b)=0$, então como $f_n(b)=f_{n-1}(b)$, definimos $x_n^* = 0$ e continuamos o processo com o valor $f_{n-1}(b)$;
  - se $p_n(b)=1$, então $f_n(b) = c_n + f_{n-1}(b - a_n)$, definimos $x_n^* = 1$ e repetimos este procedimento para $f_{n-1}(b - a_n)$;
  - após n iterações, obteremos a solução ótima.

Algoritmo

```python
def PD_mochilabinaria(utilidades, pesos, capacidade):
    n = len(utilidades)

    f = [[0 for u in range(capacidade + 1)] for i in range(n + 1)]
    p = [[0 for u in range(capacidade + 1)] for i in range(n + 1)]

    for i in range(1, n + 1):
        for w in range(1, capacidade + 1):
            if pesos[i - 1] > w:
                f[i][w] = f[i - 1][w]
                p[i][w] = 0
            else:
                case1 = f[i - 1][w]
                case2 = utilidades[i - 1] + f[i - 1][w - pesos[i - 1]]
                if case1 >= case2:
                    f[i][w] = case1
                    p[i][w] = 0
                else:
                    f[i][w] = case2
                    p[i][w] = 1

    valor_otimo = f[n][capacidade]
    solucao_otima = [0] * (n + 1)

    for i in range(1, n + 1):
        for w in range(1, capacidade + 1):
            print(p[i][w], end=" ")
        print()

    for i in range(n, 0, -1):
        if p[i][capacidade] == 1:
            solucao_otima[i] = 1
            capacidade = capacidade - pesos[i - 1]

    del solucao_otima[0]

    return valor_otimo, solucao_otima
```

Exemplo execução

- c = (10, 7, 25, 24) a = (2, 1, 6, 5) b = 7

Exemplo execução

- c = (10, 7, 25, 24) a = (2, 1, 6, 5) b = 7

λ	f_0	f_1	f_2	f_3	f_4	p_1	p_2	p_3	p_4
0	0								
1	0								
2	0								
3	0								
4	0								
5	0								
6	0								
7	0								

Exemplo execução

- c = (10, 7, 25, 24) a = (2, 1, 6, 5) b = 7

λ	f_0	f_1	f_2	f_3	f_4	p_1	p_2	p_3	p_4
0	0	0				0			
1	0	0				0			
2	0	10				1			
3	0	10				1			
4	0	10				1			
5	0	10				1			
6	0	10				1			
7	0	10				1			

Exemplo execução

- c = (10, 7, 25, 24) a = (2, 1, 6, 5) b = 7

λ	f_0	f_1	f_2	f_3	f_4	p_1	p_2	p_3	p_4
0	0	0	0			0	0		
1	0	0	7			0	1		
2	0	10	10			1	0		
3	0	10	17			1	1		
4	0	10	17			1	1		
5	0	10	17			1	1		
6	0	10	17			1	1		
7	0	10	17			1	1		

Exemplo execução

- c = (10, 7, 25, 24) a = (2, 1, 6, 5) b = 7

λ	f_0	f_1	f_2	f_3	f_4	p_1	p_2	p_3	p_4
0	0	0	0	0		0	0	0	
1	0	0	7	7		0	1	0	
2	0	10	10	10		1	0	0	
3	0	10	17	17		1	1	0	
4	0	10	17	17		1	1	0	
5	0	10	17	17		1	1	0	
6	0	10	17	25		1	1	1	
7	0	10	17	32		1	1	1	

Exemplo execução

- c = (10, 7, 25, 24) a = (2, 1, 6, 5) b = 7

λ	f_0	f_1	f_2	f_3	f_4	p_1	p_2	p_3	p_4
0	0	0	0	0	0	0	0	0	0
1	0	0	7	7	7	0	1	0	0
2	0	10	10	10	10	1	0	0	0
3	0	10	17	17	17	1	1	0	0
4	0	10	17	17	17	1	1	0	0
5	0	10	17	17	24	1	1	0	1
6	0	10	17	25	31	1	1	1	1
7	0	10	17	32	34	1	1	1	1

Exemplo execução

- c = (10, 7, 25, 24) a = (2, 1, 6, 5) b = 7

λ	f_0	f_1	f_2	f_3	f_4	p_1	p_2	p_3	p_4
0	0	0	0	0	0	0	0	0	0
1	0	0	7	7	7	0	1	0	0
2	0	10	10	10	10	**1**	**0**	**0**	0
3	0	10	17	17	17	1	1	0	0
4	0	10	17	17	17	1	1	0	0
5	0	10	17	17	24	1	1	0	1
6	0	10	17	25	31	1	1	1	1
7	0	10	17	32	34	1	1	1	**1**

Tempo de execução

- Calculando o número de operações necessárias para obtermos $z = f_n(b)$, verificamos que o cálculo de $f_k(\lambda)$ para $\lambda = 0, 1, \ldots, b$ e $k = 1, \ldots, n$ necessita de um número constante de operações.
- O tempo de execução do algoritmo é $\Theta(nb)$, sendo portanto pseudo-polinomial.
- O algoritmo tem tempo de execução polinomial se $b \in O(\log n)$, já que b pode ser representado em notação binária com k bits, ou seja, $b \leq 2^k$.

Atividades práticas

Contato:
rafael.melo@ufba.br
rafaelmelo.ufba@gmail.com