Projeto e Análise de Algoritmos I

Pedro Hokama

Fontes

- [*clrs*] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest e Clifford Stein.
- [*timr*] Algorithms Illuminated Series, Tim Roughgarden
- Desmistificando Algoritmos, Thomas H. Cormen.
- Algoritmos, Sanjoy Dasgupta, Christos Papadimitriou e Umesh Vazirani
- Stanford Algorithms
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V`
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBsLsZl7A5HEK6`
- Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
- Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420

Qualquer erro é de minha responsabilidade.

Problema da Mochila

Dado uma coleção I de n itens. Cada item $i \in I$ tem:

- Um valor v_i (não negativo)
- Um peso w_i (não negativo e inteiro)

Além disso também é dado uma capacidade W não negativa e inteira. Queremos encontrar um subconjunto $S \subseteq I$ cujo peso não ultrapasse W, ou seja,

$$\sum_{i \in S} w_i \leq W$$

e que maximiza

$$\sum_{i \in S} v_i$$

Esse problema aparece em vários contextos, basicamente qualquer problema em que temos uma quantidade limitada de recursos e queremos maximizar a eficiência.

Desenvolvendo um Algoritmo de Programação Dinâmica

- Vamos tentar pensar em como uma solução ótima pode ser formada usando a solução de subproblemas menores.
- O objetivo é encontrar recorrência que descreva esse fato.
- A ideia é partir de uma solução ótima e tentar disseca-la.
- Seja S uma solução para o problema da mochila com itens I e capacidade W.
- Assuma alguma ordenação dos itens
- Caso 1: o item n (último) não está em S. Logo S também é uma solução para o problema que considera só os $n-1$ primeiros itens.
- Caso 2: Suponha que $n \in S$. Então o que eu posso dizer sobre $S - \{n\}$?
  1. É solução para os $n-1$ primeiros itens e mochila com capacidade W
  2. É solução para os $n-1$ primeiros itens e mochila com capacidade $W - v_n$
  3. É solução para os $n-1$ primeiros itens e mochila com capacidade $W - w_n$
  4. Pode não ser viável

- **Notação:** Seja $V_{i,x}$ o valor da melhor solução que:
  1. só usa os i primeiros itens
  2. tem tamanho total $\leq x$
  3. Passo 1: encontra uma recorrência

Para $i \in I$ e para qualquer x,

$$V_{i,x} = \max \begin{cases} V_{(i-1),x} \\ v_i + V_{(i-1),(x-w_i)} \end{cases} \qquad \text{(somente se } w_i \leq x\text{)}$$

- Passo 2: Agora precisamos identificar os subproblemas.
- Variamos por todos os prefixos de itens $\{1, 2, \ldots, i\}$
- Variamos todas as capacidades residuais possíveis $x \in \{0, 1, 2, \ldots, W\}$
- Passo 3: Usar a recorrência para revolver todos os subproblemas (preencher tabela)
- Seja A um vetor bidimensional tal que $A[i, x]$ vai guardar o valor de $V_{i,x}$

Algoritmo 1: Knap(I, W)

Entrada: Um conjunto de itens I e uma capacidade W
Saída: O valor da solução ótima

```
1 A[0, x] = 0 para todo x;
2 para i = 1, 2, ..., n faça
3     para x = 0, 1, 2, ..., W faça
4         A[i, x] = max{A[i − 1, x]; A[i − 1, x − w_i] + v_i};
5 devolva A[n, W];
```


- O tempo de execução é $O(nW)$

Exemplo

- Um exemplo com 4 itens e Capacidade 6
- $v_1 = 3,\ w_1 = 4$
- $v_2 = 2,\ w_2 = 3$
- $v_3 = 4,\ w_3 = 2$
- $v_4 = 4,\ w_4 = 3$

Algoritmo 2: Knap(I, W)

```
1 A[0, x] = 0 para todo x;
2 para i = 1, 2, ..., n faça
3     para x = 0, 1, 2, ..., W faça
4         A[i, x] =
5             max { A[i − 1, x],
                  { A[i − 1, x − w_i] + v_i
6 devolva A[n, W];
```

6					
5					
4					
3					
2					
1					
x = 0					
i =	0	1	2	3	4

6	0	3	3	7	8
5	0	3	3	6	8
4	0	3	3	4	4
3	0	0	2	4	4
2	0	0	0	4	4
1	0	0	0	0	0
x = 0	0	0	0	0	0
i =	0	1	2	3	4

Problema do Alinhamento de Sequências

Sejam $X = x_1x_2 \dots x_m$ e $Y = y_1y_2 \dots y_n$ duas sequências sobre um alfabeto Σ. Deseja-se formar X' e Y' inserindo lacunas (-) em X e/ou Y até que $|X'| = |Y'| = L$, de modo a minimizar a penalidade total

$$C(X', Y') = \sum_{k=1}^{L} c(X'_k, Y'_k),$$

onde

$$c(a, b) = \begin{cases} 0, & \text{se } a = b \in \Sigma, \\ \beta, & \text{se } a \neq b \in \Sigma, \\ \alpha, & \text{se } a = \text{- ou } b = \text{-}. \end{cases}$$

Considere as sequências $X = \text{ACGT}$ e $Y = \text{AGA}$, com penalidade de lacuna $\alpha = 2$ e de mismatch $\beta = 1$.
Um possível alinhamento seria:

$$\begin{array}{lcccc} X': & A & C & G & T \\ Y': & A & G & A & - \end{array}$$

com custo total:

$$C = c(A, A) + c(C, G) + c(G, A) + c(T, -) = 0 + 1 + 1 + 2 = 4.$$

Entretanto um alinhamento melhor é:

$$\begin{array}{lcccc} X': & A & C & G & T \\ Y': & A & - & G & A \end{array}$$

O custo total é:

$$C(X', Y') = c(A, A)+c(C, -)+c(G, G)+c(T, A) = 0+2+0+1 = 3.$$

Logo, o custo mínimo de alinhamento é 3.

Subestrutura Ótima do Problema de Alinhamento

Sejam $X = x_1x_2 \dots x_m$ e $Y = y_1y_2 \dots y_n$. O alinhamento ótimo de X e Y possui a seguinte propriedade:

$$D(i,j) = \text{custo mínimo de alinhar } x_1x_2 \dots x_i \text{ com } y_1y_2 \dots y_j.$$

Então, se conhecemos a solução ótima dos prefixos menores, podemos construir a solução ótima de prefixos maiores. Em particular, o alinhamento ótimo de X e Y termina com uma das três situações:

1. x_i alinhado com y_j;
2. x_i alinhado com uma lacuna;
3. y_j alinhado com uma lacuna.

Essa decomposição garante a **subestrutura ótima**.

Algoritmo 3: Alinhamento-Seq(X, Y, α, β)

```
1 D[0,0] = 0;
2 para i = 1,2,...,m faça
3     D[i,0] = i · α;
4 para j = 1,2,...,n faça
5     D[0,j] = j · α;
6 para i = 1,2,...,m faça
7     para j = 1,2,...,n faça
          D[i,j] = min { D[i-1,j-1] + { 0, se x_i = y_j
                                      { β, caso contrário ,
                       { D[i-1,j] + α,                    ;
                       { D[i,j-1] + α
8
9 devolva D[m,n];
```


Recorrência do Problema de Alinhamento

A partir da subestrutura ótima, obtemos a seguinte recorrência:

$$D(i,j) = \begin{cases} 0, \text{se } i = j = 0, \\ i\alpha, \text{se } j = 0, \\ j\alpha, \text{se } i = 0, \\ \min \begin{cases} D(i-1,j-1) + \big(0 \text{ se } x_i = y_j, \text{ senão } \beta\big), \\ D(i-1,j) + \alpha, \\ D(i,j-1) + \alpha. \end{cases} \end{cases}$$

O valor $D(m,n)$ fornece o custo mínimo de alinhamento entre X e Y.

Exemplo

$$X = \text{ACGT}, \qquad Y = \text{AGA}, \qquad \alpha = 2,\ \beta = 1$$

$D(i,j)$	$j=0$	$j=1$	$j=2$	$j=3$
$i=0$	0	2	4	6
$i=1$	2	0	2	4
$i=2$	4	2	1	3
$i=3$	6	4	2	2
$i=4$	8	6	4	3