Programação Dinâmica
LIS, LCS e Knapsack

Laboratório de Programação Competitiva - 2020

Pedro Henrique Paiola

Programação Dinâmica

- Introduzido por Richard Bellman da década de 50, em um projeto militar na RAND Corporation
- O termo foi utilizado para encobrir o propósito do projeto, pois o Secretário de Defesa da época abominava pesquisa matemática

“A década de 1950 não foi boa para a pesquisa em matemática. Tivemos um cavalheiro muito interessante em Washington chamado Wilson. Ele foi secretário de Defesa, e realmente tinha um **medo patológico e ódio da palavra ‘pesquisa’**. Não estou usando o termo levemente; eu estou usando-o precisamente. Seu rosto ficava vermelho, e ele ficava violento se as pessoas usassem o termo ‘pesquisa’ em sua presença. **Você pode imaginar como ele se sentia então, sobre o termo 'matemática'**.” (Richard Bellman)

Programação Dinâmica

- Aplicado a problemas com estrutura recursiva.
- Divisão e conquista.
- Ideia: armazenar a solução de subproblemas para a resolução de subproblemas futuros.
- A ideia é simples, o desafio é aplicar isso em diferentes problemas.

Programação Dinâmica

- Propriedades necessárias do problema:
  - Sub-estrutura ótima:
    - A solução ótima do problema é composta pela solução ótima de partes menores e mais simples do problema.
  - Sobreposição de problemas:
    - As partes menores são sobrepostas, portanto, elas podem ser armazenadas para evitar recálculo.

Programação Dinâmica

- Estratégia básica:
  - Definir os subproblemas
  - Escrever a recorrência que relaciona os subproblemas
  - Reconhecer e solucionar os casos bases

Programação Dinâmica

- Dicas do Thiago Alexandre de como entender programação dinâmica:
  - Decorar algoritmos não adianta, entenda a lógica e as diferentes técnicas.
  - Estudar, entender e treinar problemas recursivos
  - Estudar, entender e treinar problemas clássicos de PD
  - Resolva problemas e compare com outras soluções
  - O que outras soluções têm de melhor ou pior?

PD x Outros paradigmas

- Algoritmo Guloso
    - Melhor solução local
- Backtracking
    - Busca exaustiva
    - Problemas não se repetem
    - Complexidade fatorial/exponencial
- Programação Dinâmica
    - Melhor solução global/solução ótima
    - Busca exaustiva “inteligente”
    - Evita recalcular problemas que já ocorreram
    - Complexidade polinomial

Maior subsequência crescente (LIS)

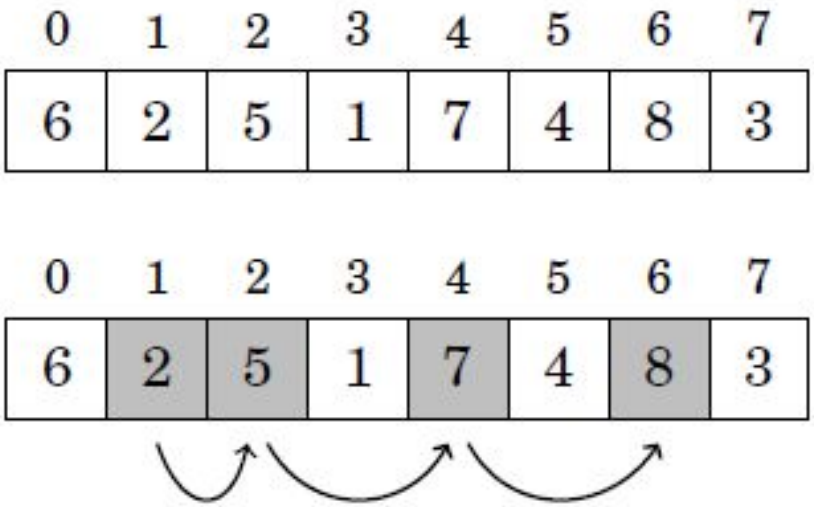
- Subsequência: uma subsequência de uma sequência de elementos X é uma sequência X' com zero ou mais elementos de X removidos.
    - É uma sequência de elementos de X não necessariamente contíguos.

Exemplo:

X = {**A**, B, **C**, B, **D**, **C**, B}
X'= {A, C, D, C}

Maior subsequência crescente (LIS)

- Maior subsequência crescente: dado uma sequência de números, determinar a maior subsequência de valores crescentes.



Maior subsequência crescente (LIS)

- É um problema de PD?
  - Dado um vetor de n elementos, podemos determinar a subsequência máxima do vetor v[0...n-1] a partir das subsequências máximas dos vetores v[0...n-2], v[0...n-3] ...
  - Intuitivamente, isso não é difícil perceber, mas como fazer essa relação e de modo eficiente? Calma! Um passo de cada vez

Maior subsequência crescente (LIS)

- Definição dos estados
  - No passo anterior, concluímos que podemos determinar a subsequência máxima do vetor v[0...n-1] a partir das subsequências máximas dos vetores v[0...n-2], v[0...n-3] ...
  - A partir disso, parece interessante definir o estado do nosso problema como o índice em que acaba nosso vetor
  - Subsequência máxima que **TERMINA** na posição i: `lis(i)`
  - Subsequência máxima do vetor inteiro: `max(lis(i)), 0 <= i < n`

Maior subsequência crescente (LIS)

- Relação entre os estados
  - Agora temos que definir/encontrar uma relação de recorrência.
  - Problema base: lis(0), nesse caso estamos considerando apenas o primeiro elemento do vetor, obviamente a maior subsequência crescente possível é 1 (considerando o único elemento possível)
    - **lis(0) = 1**

Maior subsequência crescente (LIS)

- Relação entre os estados
  - E o passo da recursão?
  - para lis(i) queremos encontrar a subsequência máxima considerando até a posição i.
  - Para isso, vamos considerar as posições j / j < i

Maior subsequência crescente (LIS)

- Relação entre os estados
  - Se a[j] > a[i], não vamos considerar a lis(j), pois o elemento a[i] não pode ser inserida nela
  - Se a[j] <= a[i], então a[i] pode ser inserido na lis(j), gerando uma uma subsequência de tamanho lis(j)+1

```
lis(0) = 1
lis(i) = max(1, 1 + lis(j)), para 0 <= j < i e a[j] <= a[i]
```

Maior subsequência crescente (LIS)

- Esta solução do problema tem complexidade $O(n^2)$.
- Por força bruta, teriamos complexidade exponencial (testando todas as possíveis subsequências)
- Existem outras possíveis soluções, utilizando Programação Dinâmica e Busca Binária ou alguma estrutura de dados que trabalhe com *range queries*. Estas soluções atingem complexidade O(n.logn)
- Para mais detalhes: https://cp-algorithms-brasil.com/Diversos/ss.html

Maior subsequência crescente (LIS)

- Implementação (Top-down):

```
memo[] = {1, -1, -1, -1, ...}
lisMax = 0;    //resposta final
int lis(int i){
    if (memo[i] != -1)
        return memo[i];
    memo[i] = 1;
    for(int j = 0; j < i; j++)
        if (a[j] <= a[i])
            memo[i] = max(memo[i], lis(j) + 1);
    lisMax = max(lisMax, memo[i]);
    return memo[i];
}
```

Maior subsequência crescente (LIS)

- Implementação (Bottom-up):

```
int lis(int n){
    int memo[n], lisMax = 0;
    for(int i = 0; i < n; i++){
        memo[i] = 1;
        for(int j = 0; j < i; j++){
            if (a[j] <= a[i])
                memo[i] = max(memo[i], memo[j] + 1);
        }
        lisMax = max(lisMax, memo[i]);
    }
    return lisMax;
}
```

Maior subsequência comum (LCS)

- Problema: dadas as sequências X[0..m-1] e Y[0..n-1], encontrar uma sequência Z tal que Z é subsequência de X e de Y e tem comprimento máximo.
- Exemplo:

  X = {A, B, C, B, D, A, B}

  Y = {B, D, C, A, B, A}

  LCS(X,Y) = {B, C, B, A}

Maior subsequência comum (LCS)

- **Força bruta**: testar todas as subsequências se X para ver se ela também é uma subsequência de Y.
- Há 2^m subsequências de X para serem verificadas
- Cada subsequência gasta tempo O(n) para ser verificada.
- Complexidade total: $O(n.2^m)$

Maior subsequência comum (LCS)

- Como dito anteriormente, uma subsequência de X é uma sequência X' com zero ou mais elementos de X removidos.
- Pensando nisso, nosso objetivo pode ser visto como minimizar o número de elementos removidos de duas sequências para que elas se tornem iguais (ou, de forma equivalente, maximizar o número de elementos inseridos).

Maior subsequência comum (LCS)

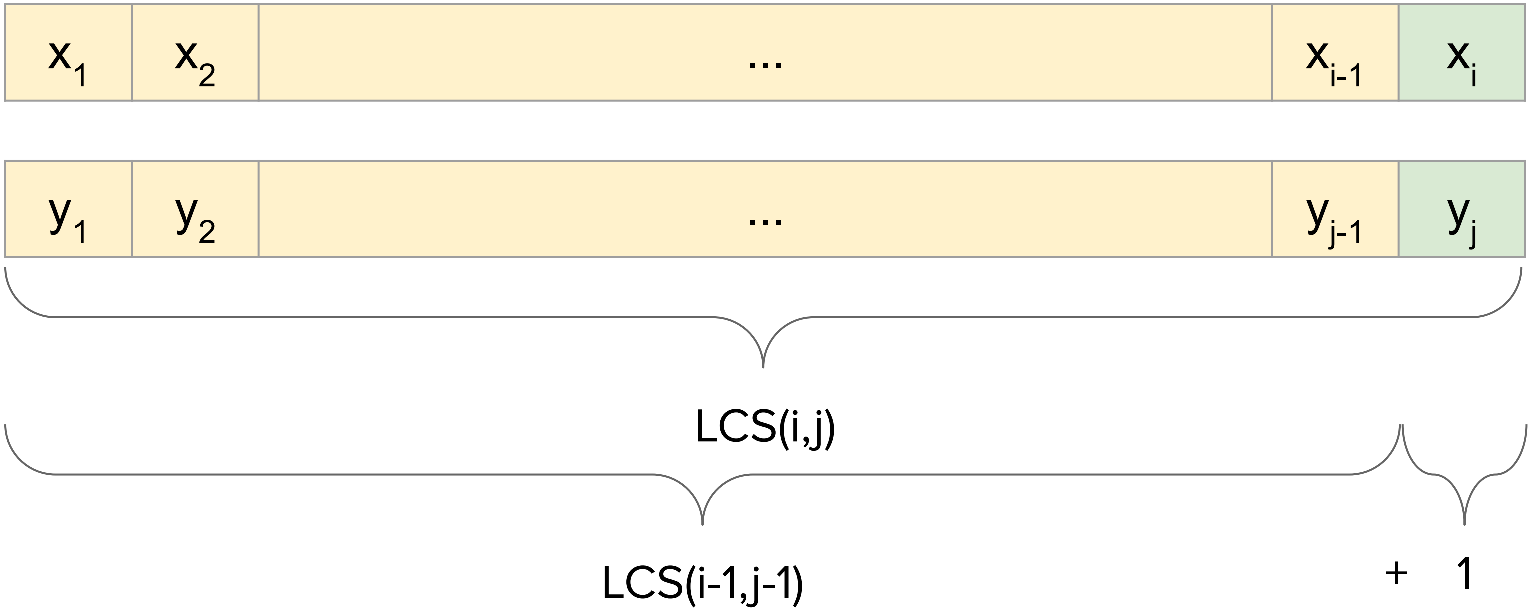
- **Teorema:** Seja Z[1..k] uma LCS de X[1..m] e Y[1..n]
  a. Se $x_m = y_n$ então $z_k = y_n = x_m$ e Z[1..k-1] é uma LCS de X[1..m-1] e Y[1..n-1]
  b. Se $x_m \neq y_n$ então $z_k \neq x_m$, sendo assim Z[1..k] é uma LCS de X[1..m-1] e Y[1..n]
  c. Se $x_m \neq y_n$ então $z_k \neq y_n$, sendo assim Z[1..k] é uma LCS de X[1..m] e Y[1..n-1]
- Esse teorema mostra que este problema atende a propriedade da Subestrutura Ótima.

Maior subsequência comum (LCS)

$$LCS(i,j) = \begin{cases} 0 & \text{se } i = 0 \text{ ou } j = 0 \\ LCS(i-1, j-1) + 1 & \text{se } i,j > 0 \text{ e } x_i = y_j \\ max(LCS(i, j-1), LCS(i-1, j)) & \text{se } i,j > 0 \text{ e } x_i \neq y_j \end{cases}$$

Maior subsequência comum (LCS)

- Se $x_i = y_j$



Maior subsequência comum (LCS)

- Se x_i != y_j

x_1	x_2	...	x_{i-1}	x_i

y_1	y_2	...	y_{j-1}	y_j

- Opção 1: retirar x_i => LIS(i-1, j)

x_1	x_2	...	x_{i-1}

y_1	y_2	...	y_{j-1}	y_j

Maior subsequência comum (LCS)

- Se x_i != y_j

x_1	x_2	...	x_{i-1}	x_i

y_1	y_2	...	y_{j-1}	y_j

- Opção 2: retirar y_j => LIS(i, j-1)

x_1	x_2	...	x_{i-1}	x_i

y_1	y_2	...	y_{j-1}

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B						
A						
C						
B						
A						
D						max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B						
A						
C						
B						
A						
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B						
A						
C						
B						
A				max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B						
A						
C						
B						
A			+1	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B						
A						
C						
B		+1				
A			+1	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B						
A						
C	max					
B		+1				
A			+1	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B						
A	+1					
C	max					
B		+1				
A			+1	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B						
A	1					
C	max					
B		+1				
A			+1	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B						
A	1					
C	1					
B		+1				
A			+1	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B						
A	1					
C	1					
B		2				
A			+1	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B						
A	1					
C	1					
B		2				
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B						
A	1					
C	1					
B		2		max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B						
A	1					
C	1					
B		**2**	max	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B						
A	1					
C	1		max			
B		2	max	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B						
A	1					
C	**1**	max	max			
B		2	max	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B						
A	**1**	max				
C	1	max	max			
B		2	max	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		+1				
A	1	max				
C	1	max	max			
B		2	max	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1				
A	1	max				
C	1	max	max			
B		2	max	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1				
A	1	1				
C	1	max	max			
B		2	max	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1				
A	1	1				
C	1	1	max			
B		2	max	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1				
A	1	1	+1			
C	1	1	max			
B		2	max	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1				
A	1	1	2			
C	1	1	max			
B		2	max	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1				
A	1	1	2			
C	1	1	2			
B		2	max	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1				
A	1	1	2			
C	1	1	2			
B		2	2	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1				
A	1	1	2			
C	1	1	**2**	max		
B		2	2	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1				
A	1	1	**2**	max		
C	1	1	2	max		
B		2	2	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1		max		
A	1	1	2	max		
C	1	1	2	max		
B		2	2	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		**1**	max	max		
A	1	1	2	max		
C	1	1	2	max		
B		2	2	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	max		
A	1	1	2	max		
C	1	1	2	max		
B		2	2	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1		
A	1	1	2	max		
C	1	1	2	max		
B		2	2	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1		
A	1	1	2	2		
C	1	1	2	max		
B		2	2	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1		
A	1	1	2	2		
C	1	1	2	2		
B		2	2	max		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1		
A	1	1	2	2		
C	1	1	2	2		
B		2	2	2		
A			3	max		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1		
A	1	1	2	2		
C	1	1	2	2		
B		2	2	2		
A			3	3		
D					+1	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1		
A	1	1	2	2		
C	1	1	2	2		
B		2	2	2		
A			3	3		
D					4	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1		
A	1	1	2	2		
C	1	1	2	2		
B		2	2	2		
A			3	3		max
D					4	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1		
A	1	1	2	2		
C	1	1	2	2		
B		2	2	2		
A			3	**3**	max	max
D					4	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1		
A	1	1	2	2		
C	1	1	2	2		
B		2	2	**2**	max	
A			3	3	max	max
D					4	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1		
A	1	1	2	2		
C	1	1	2	**2**	max	
B		2	2	2	max	
A			3	3	max	max
D					4	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1		
A	1	1	2	**2**	max	
C	1	1	2	2	max	
B		2	2	2	max	
A			3	3	max	max
D					4	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	**1**	max	
A	1	1	2	2	max	
C	1	1	2	2	max	
B		2	2	2	max	
A			3	3	max	max
D					4	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1	1	
A	1	1	2	2	max	
C	1	1	2	2	max	
B		2	2	2	max	
A			3	3	max	max
D					4	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1	1	
A	1	1	2	2	2	
C	1	1	2	2	max	
B		2	2	2	max	
A			3	3	max	max
D					4	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1	1	
A	1	1	2	2	2	
C	1	1	2	2	2	
B		2	2	2	max	
A			3	3	max	max
D					4	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1	1	
A	1	1	2	2	2	
C	1	1	2	2	2	
B		2	2	2	2	
A			3	3	max	max
D					4	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1	1	
A	1	1	2	2	2	
C	1	1	2	2	2	
B		2	2	2	2	
A			3	3	3	max
D					4	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1	1	
A	1	1	2	2	2	
C	1	1	2	2	2	
B		2	2	2	**2**	max
A			3	3	3	max
D					4	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1	1	
A	1	1	2	2	**2**	
C	1	1	2	2	2	+1
B		2	2	2	2	max
A			3	3	3	max
D					4	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1	1	
A	1	1	2	2	2	
C	1	1	2	2	2	3
B		2	2	2	2	max
A			3	3	3	max
D					4	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1	1	
A	1	1	2	2	2	
C	1	1	2	2	2	3
B		2	2	2	2	3
A			3	3	3	max
D					4	max

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1	1	
A	1	1	2	2	2	
C	1	1	2	2	2	3
B		2	2	2	2	3
A			3	3	3	3
D					4	max

Maior subsequência comum (LCS)

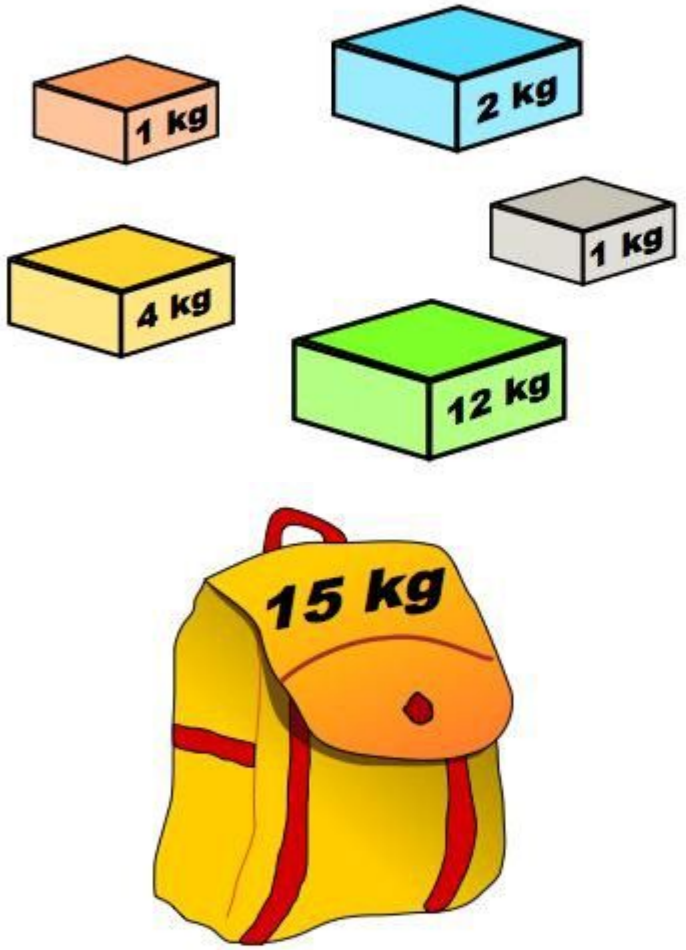
	A	B	A	Z	D	C
B		1	1	1	1	
A	1	1	2	2	2	
C	1	1	2	2	2	3
B		2	2	2	2	3
A			3	3	3	3
D					4	4

Maior subsequência comum (LCS)

	A	B	A	Z	D	C
B		1	1	1	1	
A	1	1	2	2	2	
C	1	1	2	2	2	3
B		2	2	2	2	3
A			3	3	3	3
D					4	4

Problema da Mochila

- Problema:
  - Uma mochila suporta até W quilos
  - Itens devem ser adicionados à mochia
    - Cada item tem um peso w[i] e um valor v[i]
    - w[i] e v[i] são inteiros
- Objetivo:
  - Qual o valor máximo que não ultrapassa o limite da mochila?



Problema da Mochila

- Caso base:
  - Se a capacidade da mochila ou a quantidade de itens for zero, então o valor máximo é zero.
- Passo da recursão
  - Senão, há duas opções: incluir ou não incluir (considerando o problema da mochila binária, onde não há repetições de itens)
- Queremos maximizar o valor total carregado sem ultrapassar a capacidade da mochila.

$$\max \sum_{i=0}^{n} v_i \cdot x_i \qquad \text{sujeito a} \quad \sum_{i=0}^{n} w_i \cdot x_i \leq W \qquad x_i \in \{\,0, 1\,\}$$

Problema da Mochila

$$f(w,n) = \begin{cases} 0, & w = 0 \text{ ou } n = 0 \\ \max(\text{não adicionar, adicionar}), & \text{caso contrário} \end{cases}$$

$$f(w,n) = \begin{cases} 0, & w = 0 \text{ ou } n = 0 \\ \max\{\ f(w,\ n-1),\ value[n-1]\ +\ f(\ w - weight[n-1],\ n-1\ )\}, & \text{caso contrário} \end{cases}$$

Problema da Mochila

- Capacidade da mochila: 12
- v = {100, 55, 50}
- w = {10, 6, 6}

f(12, 3) = max{f(12, 2), 50 + f(6,2)}

	0	1	2	...	6	...	12
0							
1							
2							
3							max

Problema da Mochila

- Capacidade da mochila: 12
- v = {100, 55, 50}
- w = {10, 6, 6}

f(12, 2) = max{f(12, 1), 55 + f(6,1)}

	0	1	2	...	6	...	12
0							
1							
2							max
3							max

Problema da Mochila

- Capacidade da mochila: 12
- v = {100, 55, 50}
- w = {10, 6, 6}

f(12, 1) = max{f(12, 0), 100 + f(2,0)}

	0	1	2	...	6	...	12
0							
1							max
2							max
3							max

Problema da Mochila

- Capacidade da mochila: 12
- v = {100, 55, 50}
- w = {10, 6, 6}

f(12, 1) = max{0, 100 + 0}

	0	1	2	...	6	...	12
0			0				0
1							100
2							max
3							max

Problema da Mochila

- Capacidade da mochila: 12
- v = {100, 55, 50}
- w = {10, 6, 6}

f(6, 1) = f(6,0), não pode pegar o item 1 pois w[0] = 10 > 6

	0	1	2	...	6	...	12
0			0				0
1					f(6,0)		100
2							max
3							max

Problema da Mochila

- Capacidade da mochila: 12
- v = {100, 55, 50}
- w = {10, 6, 6}

f(6, 1) = f(6,0), não pode pegar o item 0 pois w[0] = 10 > 6

	0	1	2	...	6	...	12
0			0		0		0
1					0		100
2							max
3							max

Problema da Mochila

- Capacidade da mochila: 12
- v = {100, 55, 50}
- w = {10, 6, 6}

f(12, 2) = max{100, 55 + 0}

	0	1	2	...	6	...	12
0			0		0		0
1					0		100
2							100
3							max

Problema da Mochila

- Capacidade da mochila: 12
- v = {100, 55, 50}
- w = {10, 6, 6}

$f(6, 2) = \max\{f(6,1), 55 + f(0,1)\}$

	0	1	2	...	6	...	12
0			0		0		0
1					**0**		100
2					max		100
3							max

Problema da Mochila

- Capacidade da mochila: 12
- v = {100, 55, 50}
- w = {10, 6, 6}

f(6, 2) = max{0, 55 + 0}

	0	1	2	...	6	...	12
0			0		0		0
1	0				0		100
2					55		100
3							max

Problema da Mochila

- Capacidade da mochila: 12
- v = {100, 55, 50}
- w = {10, 6, 6}

f(12, 3) = max{f(12, 2), 50 + f(6,2)}

	0	1	2	...	6	...	12
0			0		0		0
1	0				0		100
2					55		100
3							max

Problema da Mochila

- Capacidade da mochila: 12
- v = {100, 55, 50}
- w = {10, 6, 6}

f(12, 3) = max{100, 50 + 55}

	0	1	2	...	6	...	12
0			0		0		0
1	0				0		100
2					55		100
3							105

Problema da Mochila

- Capacidade da mochila: 12
- v = {100, 55, 50}
- w = {10, 6, 6}

f(12, 3) = max{100, 50 + 55}

	0	**1**	**2**	**...**	**6**	**...**	**12**
0			0		0		0
1	0				0		100
2					55		100
3							105

Problema da Mochila - Top Down

```
int knapsack(int w, int n){
    if(memo[w][n] != -1)
        return memo[w][n];
    if(w == 0 || n == 0)
        return memo[w][n] = 0;
    if(weight[n-1] > w)
        return memo[w][n] = knapsack(w, n-1);
    return memo[w][n] = max(knapsack(w, n-1), value[n-1] +
                            knapsack(w - weight[n-1], n-1));
}
```

Problema da Mochila - Bottom Up

```
for(int i=0; i<=n; i++)
    dp[i][0] = 0;
for(int j=0; j<=w; j++)
    dp[0][j] = 0;
for(int i=1; i<=n; i++)
    for(int j=1; j<=w; j++){
        if(weight[i-1] > j)
            dp[i][j] = dp[i-1][j];
        else
            dp[i][j] = max(dp[i-1][j], dp[i-1][j-weight[i-1]]
                                       + value[i-1]);
    }
```

Mochila: otimizando espaço

- Em nossa solução, estamos utilizando uma matriz dp[MAX_W, MAX_N].
- Dependendo do problema, isso pode ocasionar estouro de memória!
- Existem algumas formas de otimizar nossa solução para não precisarmos de uma matriz tão grande. Veja algumas delas nos seguintes links:

https://www.geeksforgeeks.org/space-optimized-dp-solution-0-1-knapsack-problem
https://codeforces.com/blog/entry/47247?#comment-316200
https://medium.com/@ThatOneKevin/knapsack-problems-part-1-8465fb2d53e9

Mochila Ilimitada (com repetição)

- Uma variação comum do Problema da Mochila.
- Neste caso podemos considerar que temos uma quantidade ilimitada de cada item. Sendo assim, um mesmo item pode ser colocado mais de uma vez dentro da mochila.

Mochila Ilimitada (com repetição)

- A ideia da nossa solução não irá se alterar muito. De certa forma, será até mais simples.
- Para uma certa capacidade **i** da mochila, verificamos todos os itens **j** que podem ser colocados nela (w[j] <= i) e qual resulta em maior valor (v[j] + dp[i-w[j]])

$$f(i) = \begin{cases} 0 & \text{se } i = 0 \\ max\{v[j] + f(i - w[j])\} & \forall j | w[j] \leq i \end{cases}$$

Mochila Ilimitada (com repetição)

```
int knapsack(int n, int w){
    memset(dp, 0, sizeof(dp));
    for(int j=1; j<=w; j++){
        for(int i=1; i<=n; i++){
            if(weight[i-1] <= j)
                dp[j] = max(dp[j], dp[j-weight[i-1]] + v[i-1]);
        }
    }
    return dp[w];
}
```

Referências

Thiago Alexandre Domingues de Souza. Palestra sobre Programação Dinâmica.

Giulia Moura, João Pedro Comini e Pedro H. Paiola. Programação Competitiva I.

https://sites.google.com/site/ldsicufal/disciplinas/programacao-avancada/notas-de-aula---programao-dinmica

https://www.geeksforgeeks.org/longest-common-subsequence-dp-4/

https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_longest_common_subsequence.htm

https://neps.academy/lesson/164

http://www.facom.ufms.br/~marco/analise2007/aula12_4.pdf

https://github.com/icmcgema/gema/blob/master/09-Programacao_Dinamica.ipynb

Referências

https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-bool.html

https://www.geeksforgeeks.org/space-optimized-dp-solution-0-1-knapsack-problem

https://www.geeksforgeeks.org/unbounded-knapsack-repetition-items-allowed/