Mais programação dinâmica

KT 6.4

Aproveite para olhar todo o Cap 6 do KT,
que é sobre programação dinâmica.

= "recursão–com–tabela"
= transformação inteligente de recursão em iteração

Mochila

Dados dois vetores $x[1\,..\,n]$ e $w[1\,..\,n]$, denotamos por $x \cdot w$ o produto escalar

$$w[1]x[1] + w[2]x[2] + \cdots + w[n]x[n].$$

Suponha dado um número inteiro não-negativo W e vetores positivos $w[1\,..\,n]$ e $v[1\,..\,n]$.

Uma mochila é qualquer vetor $x[1\,..\,n]$ tal que

$$x \cdot w \leq W \quad \text{e} \quad 0 \leq x[i] \leq 1 \text{ para todo } i$$

O valor de uma mochila é o número $x \cdot v$.

Uma mochila é ótima se tem valor máximo.

Problema booleano da mochila

Uma mochila $x[1 \,.\,.\, n]$ tal que $x[i] = 0$ ou $x[i] = 1$ para todo i é dita booleana.

Problema (Knapsack Problem): Dados (w, v, n, W), encontrar uma mochila boolena ótima.

Exemplo: $W = 50$, $n = 4$

	1	2	3	4	
w	40	30	20	10	
v	840	600	400	100	
x	1	0	0	0	valor $= 840$
x	1	0	0	1	valor $= 940$
x	0	1	1	0	valor $= 1000$

Subestrutura ótima

Suponha que $x[1 \,..\, n]$ é mochila boolena ótima para o problema (w, v, n, W).

Subestrutura ótima

Suponha que $x[1 \mathinner{.\,.} n]$ é mochila boolena ótima para o problema (w, v, n, W).

Se $x[n] = 1$

então $x[1 \mathinner{.\,.} n{-}1]$ é mochila boolena ótima para
$(w, v, n-1, W - w[n])$

Subestrutura ótima

Suponha que $x[1\,..\,n]$ é mochila boolena ótima para o problema (w, v, n, W).

Se $x[n] = 1$

então $x[1\,..\,n{-}1]$ é mochila boolena ótima para
$(w, v, n-1, W - w[n])$

senão $x[1\,..\,n{-}1]$ é mochila boolena ótima para
$(w, v, n-1, W)$

Subestrutura ótima

Suponha que $x[1\,..\,n]$ é mochila boolena ótima para o problema (w, v, n, W).

Se $x[n] = 1$

então $x[1\,..\,n{-}1]$ é mochila boolena ótima para
$(w, v, n-1, W - w[n])$

senão $x[1\,..\,n{-}1]$ é mochila boolena ótima para
$(w, v, n-1, W)$

NOTA. Não há nada de especial acerca do índice n.
Uma afirmação semelhante vale para qualquer índice i.

Simplificação

Problema:
encontrar o valor de uma mochila booleana ótima.

Simplificação

Problema:
encontrar o valor de uma mochila booleana ótima.

$t[i, Y] =$ valor de uma mochila booleana ótima para (w, v, i, Y)

Simplificação

Problema:
encontrar o valor de uma mochila booleana ótima.

$t[i, Y] =$ valor de uma mochila booleana ótima para (w, v, i, Y)

$=$ valor da expressão $x \cdot v$ sujeito às restrições

$$x \cdot w \leq Y,$$

onde x é uma mochila booleana ótima

Simplificação

Problema:
encontrar o valor de uma mochila booleana ótima.

$$\begin{aligned} t[i, Y] = & \text{ valor de uma mochila booleana ótima} \\ & \text{ para } (w, v, i, Y) \\ = & \text{ valor da expressão } x \cdot v \text{ sujeito às restrições} \end{aligned}$$

$$x \cdot w \leq Y,$$

onde x é uma mochila booleana ótima

Possíveis valores de Y: $0, 1, 2, \ldots, W$

Recorrência

$t[i, Y] =$ valor da expressão $x \cdot v$ sujeito à restrição

$$x \cdot w \leq Y$$

Recorrência

$t[i, Y]$ = valor da expressão $x \cdot v$ sujeito à restrição

$$x \cdot w \leq Y$$

$t[0, Y] = 0$ para todo Y

Recorrência

$t[i, Y]$ = valor da expressão $x \cdot v$ sujeito à restrição

$$x \cdot w \leq Y$$

$t[0, Y] = 0$ para todo Y

$t[i, 0] = 0$ para todo i

Recorrência

$t[i, Y]$ = valor da expressão $x \cdot v$ sujeito à restrição

$$x \cdot w \leq Y$$

$t[0, Y] = 0$ para todo Y

$t[i, 0] = 0$ para todo i

$t[i, Y] = t[i-1, Y]$ se $w[i] > Y$

Recorrência

$t[i, Y]$ = valor da expressão $x \cdot v$ sujeito à restrição

$$x \cdot w \leq Y$$

$t[0, Y] = 0$ para todo Y

$t[i, 0] = 0$ para todo i

$t[i, Y] = t[i-1, Y]$ se $w[i] > Y$

$t[i, Y] = \max\{t[i-1, Y], t[i-1, Y-w[i]] + v[i]\}$ se $w[i] \leq Y$

Solução recursiva

Devolve o valor de uma mochila ótima para (w, v, n, W).

REC-MOCHILA (w, v, n, W)
se $n = 0$ **ou** $W = 0$
    então devolva 0
se $w[n] > W$
    então devolva REC-MOCHILA $(w, v, n-1, W)$
$a \leftarrow$ REC-MOCHILA $(w, v, n-1, W)$
$b \leftarrow$ REC-MOCHILA $(w, v, n-1, W-w[n]) + v[n]$
devolva $\max\{a, b\}$

Solução recursiva

Devolve o valor de uma mochila ótima para (w, v, n, W).

```
REC-MOCHILA (w, v, n, W)
se n = 0 ou W = 0
    então devolva 0
se w[n] > W
    então devolva REC-MOCHILA (w, v, n−1, W)
a ← REC-MOCHILA (w, v, n−1, W)
b ← REC-MOCHILA (w, v, n−1, W−w[n]) + v[n]
devolva max {a, b}
```

Consumo de tempo no pior caso é $\Omega(2^n)$

Solução recursiva

Devolve o valor de uma mochila ótima para (w, v, n, W).

```
REC-MOCHILA (w, v, n, W)
se n = 0 ou W = 0
    então devolva 0
se w[n] > W
    então devolva REC-MOCHILA (w, v, n−1, W)
a ← REC-MOCHILA (w, v, n−1, W)
b ← REC-MOCHILA (w, v, n−1, W−w[n]) + v[n]
devolva max {a, b}
```

Consumo de tempo no pior caso é $\Omega(2^n)$

Por que demora tanto?
O mesmo subproblema é resolvido muitas vezes.

Programação dinâmica

Cada subproblema, valor de uma mochila ótima para

$$(w, v, i, Y),$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela t?

Programação dinâmica

Cada subproblema, valor de uma mochila ótima para

$$(w, v, i, Y),$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela t?

Olhe a recorrência e pense...

$t[i, Y] = t[i{-}1, Y]$ se $w[i] > Y$

$t[i, Y] = \max\{t[i-1, Y], t[i{-}1, Y{-}w[i]] + v[i]\}$ se $w[i] \leq Y$

Programação dinâmica

	0	1	2	3	4	5	6	7	Y
0	0	0	0	0	0	0	0	0	
1	0								
2	0	$\star$	$\star$	$\star$	$\star$	$\star$			
3	0					??			
4	0								
5	0								
6	0								
7	0								
i									

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

i \ Y	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

	0	1	2	3	4	5	Y
0	0	0	0	0	0	0	
1	0	0					
2	0						
3	0						
4	0						
i							

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

	0	1	2	3	4	5	Y
0	0	0	0	0	0	0	
1	0	0	0				
2	0						
3	0						
4	0						
i							

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

	0	1	2	3	4	5	Y
0	0	0	0	0	0	0	
1	0	0	0	0			
2	0						
3	0						
4	0						
i							

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

	0	1	2	3	4	5	Y
0	0	0	0	0	0	0	
1	0	0	0	0	500		
2	0						
3	0						
4	0						
i							

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0					
3	0					
4	0					

i Y

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

i \ Y	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0				
3	0					
4	0					

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

i \ Y	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0	400			
3	0					
4	0					

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

i \ Y	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0	400	400		
3	0					
4	0					

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

	0	1	2	3	4	5	Y
0	0	0	0	0	0	0	
1	0	0	0	0	500	500	
2	0	0	400	400	500		
3	0						
4	0						
i							

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

	0	1	2	3	4	5	Y
0	0	0	0	0	0	0	
1	0	0	0	0	500	500	
2	0	0	400	400	500	500	
3	0						
4	0						
i							

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

	0	1	2	3	4	5	Y
0	0	0	0	0	0	0	
1	0	0	0	0	500	500	
2	0	0	400	400	500	500	
3	0	300	400	700	700	800	
4	0	300	400	700	750	850	
i							

Algoritmo de programação dinâmica

Devolve o valor de uma mochila booleana ótima para (w, v, n, W).

```
MOCHILA-BOOLEANA (w, v, n, W)
para Y ← 0 até W faça
    t[0, Y] ← 0
    para i ← 1 até n faça
        a ← t[i−1, Y]
        se w[i] > Y
            então b ← 0
            senão b ← t[i−1, Y−w[i]] + v[i]
        t[i, Y] ← max{a, b}
devolva t[n, W]
```

Algoritmo de programação dinâmica

Devolve o valor de uma mochila booleana ótima para (w, v, n, W).

MOCHILA-BOOLEANA (w, v, n, W)

para $Y \leftarrow 0$ **até** W **faça**
    $t[0, Y] \leftarrow 0$
    para $i \leftarrow 1$ **até** n **faça**
        $a \leftarrow t[i-1, Y]$
        se $w[i] > Y$
            então $b \leftarrow 0$
            senão $b \leftarrow t[i-1, Y-w[i]] + v[i]$
        $t[i, Y] \leftarrow \max\{a, b\}$
devolva $t[n, W]$

Consumo de tempo é $\Theta(nW)$.

Conclusão

O consumo de tempo do algoritmo MOCHILA-BOOLEANA é $\Theta(nW)$.

Conclusão

O consumo de tempo do algoritmo MOCHILA-BOOLEANA é $\Theta(nW)$.

NOTA:
O consumo $\Theta(n2^{\lg W})$ é exponencial!

Conclusão

$\boxed{\text{O consumo de tempo do algoritmo MOCHILA-BOOLEANA é } \Theta(nW).}$

NOTA:
O consumo $\Theta(n2^{\lg W})$ é exponencial!

Explicação: o "tamanho" de W é $\lg W$ e não W
(tente multiplicar $w[1], \ldots, w[n]$ e W por 1000)

Conclusão

O consumo de tempo do algoritmo MOCHILA-BOOLEANA é $\Theta(nW)$.

NOTA:
O consumo $\Theta(n2^{\lg W})$ é exponencial!

Explicação: o "tamanho" de W é $\lg W$ e não W
(tente multiplicar $w[1], \ldots, w[n]$ e W por 1000)

Se W é $\Omega(2^n)$ o consumo de tempo é $\Omega(n2^n)$,
mais lento que o algoritmo força bruta!

Obtenção da mochila

```
MOCHILA (w, n, W, t)
Y ← W
para i ← n decrescendo até 1 faça
    se t[i, Y] = t[i−1, Y]
        então x[i] ← 0
        senão x[i] ← 1
              Y ← Y − w[i]
devolva x
```

Obtenção da mochila

MOCHILA (w, n, W, t)

$Y \leftarrow W$
para $i \leftarrow n$ **decrescendo até** 1 **faça**
  se $t[i, Y] = t[i-1, Y]$
    então $x[i] \leftarrow 0$
    senão $x[i] \leftarrow 1$
      $Y \leftarrow Y - w[i]$
devolva x

Consumo de tempo é $\Theta(n)$.

Versão recursiva

MEMOIZED-MOCHILA-BOOLEANA (w, v, n, W)

para $i \leftarrow 0$ **até** n **faça**

    para $Y \leftarrow 0$ **até** W **faça**

        $t[i, Y] \leftarrow \infty$

devolva LOOKUP-MOC (w, v, n, W)

Versão recursiva

```
LOOKUP-MOC (w, v, i, Y)
se t[i, Y] < ∞
    então devolva t[i, Y]
se i = 0 ou Y = 0 então t[i, Y] ← 0
 senão
    se w[i] > Y
        então
            t[i, Y] ← LOOKUP-MOC (w, v, i−1, Y)
        senão
            a ← LOOKUP-MOC (w, v, i−1, Y)
            b ← LOOKUP-MOC (w, v, i−1, Y−w[i]) + v[i]
            t[i, Y] ← max {a, b}
devolva t[i, Y]
```

Mochila com Repetição (Dasgupta et al., Seção 6.4)

Mochila com Repetição (Dasgupta et al., Seção 6.4)

- **Entrada:** Capacidade $W \geq 0$ e vetores de pesos $w[1 \dots n]$ e de valores $v[1 \dots n]$ positivos.
- **Objetivo:** Achar $x[1 \dots n]$ tal que $x \cdot w = \sum_{k=1}^{n} x_k w_k \leq W$, $x \cdot v = \sum_{k=1}^{n} x_k v_k$ seja máximo e $x_k \in \mathbb{N}$ para todo $k = 1, \dots, n$.
- **Observação:** x_k não precisa mais ser 0 ou 1. Pode ser 2 ou mais.

Passo 1: Propriedade da Subestrutura Ótima

- Imagine uma solução ótima x do problema. Seja $k \in \{1, \dots, n\}$ um item que aparece na solução ótima. Seja x' o vetor obtido de x diminuindo em 1 o valor de x'_k. Ou seja, $x'_k = x_k$-1 e $x'_i = x_i$, $\forall i \neq k$.
- Então x' é solução ótima da instância alterando a capacidade da mochila para $Y = W - w_k$. Ou seja, removendo o item k da mochila ótima, o que estiver na mochila será uma solução ótima caso a mochila fosse menor, de capacidade Y.
- Isso porque, caso contrário e houvesse uma solução melhor para Y, poderíamos obter uma solução melhor para W incluindo de volta o item k. Absurdo, pois x é uma solução ótima para a capacidade W.

Passo 2. Eq. Recorrência - Algoritmo recursivo simples

- Seja $t(Y)$ o maior valor para mochila com capacidade Y.
- $t(0) = 0$
- Para $Y > 0$, testar todos os itens

$$t(Y) = \max_{k \in \{1,\ldots,n\}} \left\{ t(Y - w_k) + v_k \right\},$$

onde o máximo é sobre os itens k tais que $w_k \leq Y$.

Mochila-rep-REC(w, v, n, Y)

se $(Y < 0)$ **então retorne** $-\infty$
se $(Y = 0)$ **então retorne** 0
$B \leftarrow -\infty$
para $k \leftarrow 1$ até n:
    $b \leftarrow$ *Mochila-rep-REC*$(w, v, n, Y - w_k) + v_k$
    se $(B < b)$: $B \leftarrow b$
retorne B

Passo 2. Eq. Recorrência - Algoritmo recursivo simples

Sobreposição de Subproblemas

- **Chamada inicial**: *Mochila-rep-REC*(w, v, n, W)
- **Tempo**: Exponencial $\Omega(n^Y)$ (se cada $w_k = 1$).
- **Indução**: $T(Y) \geq n \cdot T(Y-1) + 1$
- **Superposição**: A instância $(Y - w_1 - w_2)$ é chamada por $(Y - w_1)$ e $(Y - w_2)$.

Mochila-rep-REC(w, v, n, Y)

```
se (Y < 0) então retorne −∞
se (Y = 0) então retorne 0
B ← −∞
para k ← 1 até n:
        b ← Mochila-rep-REC(w, v, n, Y − w_k) + v_k
        se (B < b):  B ← b
retorne B
```

Passo 2b. Memoização (Alg. rec. + memória) - Top Down

Mochila-memo(w, v, n, W, B)

$B[0] \leftarrow 0$
para $Y \leftarrow 1$ **até** W:
    $B[Y] \leftarrow -1$
retorne *Mochila-rep-REC-memo*(w, v, n, W, B)

Mochila-rep-REC-memo(w, v, n, Y, B)

se $(B[Y] \geq 0)$ **então retorne** $B[Y]$
se $(Y < 0)$ **então retorne** $-\infty$
se $(Y = 0)$ **então retorne** 0
$B[Y] \leftarrow -\infty$
para $k \leftarrow 1$ até n:
    $b \leftarrow$ *Mochila-rep-REC-memo*$(w, v, n, Y - w_k) + v_k$
    se $(B[Y] < b)$: $B[Y] \leftarrow b$
retorne $B[Y]$

Passo 3. Algoritmo p/ Valor Ótimo (Bottom-up, não rec)

Mochila-rep-PD(w, v, n, W)

```
Criar vetor B[0 ... W]
B[0] ← 0
para Y ← 1 até W:
    B[Y] ← −∞
    para k ← 1 até n:
        se w_k ≤ Y então
            b ← B[Y − w_k] + v_k
            se (B[Y] < b): B[Y] ← b
retorne B[W]
```

Tempo pseudo-polinomial $\Theta(n \cdot W)$

Passo 4. Algoritmo p/ obter Solução Ótima (Bottom-up)

Mochila-rep-PD(w, v, n, W)

```
Criar vetor B[0 ... W]
B[0] ← 0
para Y ← 1 até W:
    B[Y] ← −∞
    para k ← 1 até n:
        se w_k ≤ Y então
            b ← B[Y − w_k] + v_k
            se (B[Y] < b): B[Y] ← b;  R[Y] ← k
retorne B[W] e R
```

Tempo pseudo-polinomial $\Theta(n \cdot W)$

Passo 4b. Alg. p/ escrever Solℓ. Ótima (rec. / não rec.)

Print-Opt(w, W, R)

$Y \leftarrow W$
enquanto $Y > 0$ **faça**:
    $k \leftarrow R[Y]$
    print "Item k"
    $Y \leftarrow Y - w_k$

Tempo $\Theta(W)$

Print-Opt-rec(w, Y, R)

se $Y \leq 0$ **então retorne**
$k \leftarrow R[Y]$
print "Item k"
Print-Opt-rec$(w, Y - w_k, R)$

Chamada principal: *Print-Opt-rec*(w, W, R)

Exercício resolvido (troco possível?)

Exercício: No Brasil, há 5 tipos de moedas: 5, 10, 25, 50 e 100 centavos. Na Algoritmênia, há n tipos de moedas: $1 \leq b_1 < \ldots < b_n \leq 100$ centavos. Faça um algoritmo de tempo $O(nW)$ que receba um valor W em centavos e os tipos de moeda $b_1 < \ldots < b_n$ e diga se é possivel representar W com moedas (se sim, diga como fazer). Por exemplo, no Brasil, podemos representar 40 centavos (25+10+5), mas 41 a 44 não.

Solução:

- É possível repetir moedas: $20 = 10+10$
- Para cada $k \in \{1, \ldots, n\}$: faça $w_k \leftarrow b_k$ e $v_k \leftarrow b_k$
- Executar *Mochila-rep-PD*(w, v, n, W)
- Se retornar W, é possível representar W. Se não, não é possível.
- Se sim, executar *Print-Opt*(w, W, R)
- Explicação: Pesos w_k e valores v_k são iguais p/ cada item no problema da mochila. Logo, a soma dos pesos de qualquer solução é igual a soma dos valores. Como a capacidade W é a maior soma possível de pesos (e é o valor que se quer representar com moedas), então o valor máx. é W se e só se for possível representar W.

Exercício resolvido (troco possível sem repetir?)

Exercício: Faça um algoritmo de tempo $O(nW)$ que receba um valor W em centavos e as moedas $b_1 < \ldots < b_n$ e diga se é possivel representar W sem repetir moedas (se sim, diga como fazer). Por exemplo, no Brasil, podemos representar 40 centavos (25+10+5 ao invés de 10+10+10+10), mas 20 e 45 não (25+10+10 e 25+10+5+5 não podem).

Solução:

- Não é possível repetir moedas. Ideia: Mochila sem repetição.
- Para cada $k \in \{1, \ldots, n\}$: faça $w_k \leftarrow b_k$ e $v_k \leftarrow b_k$
- Executar $\mathit{Mochila\text{-}PD}(w, v, n, W)$
- Se retornar W, é possível representar W. Se não, não é possível.
- Se sim, executar $\mathit{Print\text{-}Opt}(w, n, W, R)$
- Explicação: Pesos w_k e valores v_k são iguais p/ cada item no problema da mochila. Logo, a soma dos pesos de qualquer solução é igual a soma dos valores. Como a capacidade W é a maior soma possível de pesos (e é o valor que se quer representar com moedas), então o valor máx. é W se e só se for possível representar W.

Exercício resolvido: Questão 5 da Lista 2

Lista 2 - Questão 5 Escreva um algoritmo $O(nT)$ que recebe um inteiro positivo T e uma lista com n inteiros positivos $(a_1, a_2, \ldots, a_n)$ e decide se existe algum subconjunto desses inteiros cuja soma é igual a T.

Solução:

- Subconjunto não repete números. **Ideia:** Mochila sem repetição.
- Para cada $k \in \{1, \ldots, n\}$: faça $w_k \leftarrow a_k$ e $v_k \leftarrow a_k$
- Executar *Mochila-PD*(w, v, n, T)
- Se retornar T, é possível somar T (em um subconjunto). Se não, não é possível.
- Se sim, executar *Print-Opt*(w, n, T, R)
- Explicação: Pesos w_k e valores v_k são iguais p/ cada item no problema da mochila. Logo, a soma dos pesos de qualquer solução é igual a soma dos valores. Como a capacidade T é a maior soma possível de pesos (e é o valor que se quer representar com moedas), então o valor máx. é T se e só se for possível representar T.