Programação Dinâmica

Investigação Operacional

J.M. Valério de Carvalho
vc@dps.uminho.pt

Departamento de Produção e Sistemas
Escola de Engenharia, Universidade do Minho

December 15, 2014

Conteúdo

- Introdução
- Problema do saco de mochila
- Modelo de programação dinâmica
- Princípio da optimalidade
- Princípio da separabilidade
- Lotes de Produção
- Carga computacional

A leitura destes slides não dispensa o estudo dos apontamentos da disciplina: A.J. Guimarães Rodrigues, "Apontamentos de Investigação Operacional, Universidade do Minho"

Aplicação

- Resolução de problemas de optimização que podem ser formalizados como um processo de decisão sequencial.
- Existe um conjunto discreto de instantes de tempo (ou fases do processo ou estágios), em que são tomadas decisões.
- Designação "dinâmica" está associada a "tempo", embora os estágios possam ter um significado diferente.

Processo

- O sistema é caracterizado por estados, identificados por uma grandeza física.
- As decisões apenas são condicionadas pelo estado corrente do sistema, e conduzem o sistema a um novo estado no estágio seguinte.
- A cada decisão está associada uma contribuição de estágio (*e.g.*, lucro, custo).

Política e Valor

- Uma política é uma sequência de decisões válidas que se traduz numa evolução do estado do processo ao longo do tempo.
- Cada política conduz o processo a um dado estado.
- A esse estado podemos associar um valor, que resulta da combinação das contribuições de estágio das decisões que levaram a esse estágio.

Relação de recorrência

- O valor de um dado estado deve poder ser estabelecido apenas em função do valor do estado que resulta da transição e da respectiva contribuição de estágio.
- O valor de um estado estabelecido com base numa equação de recorrência geral do seguinte tipo:

$$f_j(y) \quad = \quad \phi\{f_{j-1}(y'), r_j\},$$

que traduz que o valor $f_j(y)$ do estado y no estágio j é expresso como uma função ϕ de um outro estado num estágio adjacente e de uma contribuição de estágio r_j.

Problema do saco de mochila binário

$$\begin{array}{ll} \max & \sum_{j=1}^{n} p_j x_j \\ suj. & \sum_{j=1}^{n} w_j x_j \leq W \\ & x_j = 0 \text{ ou } 1, \; j = 1, 2, \ldots, n \end{array}$$

- Estágio j : cada item j, $j = 1, 2, \ldots, n$, é colocado sequencialmente.
- Estado $e_j(y)$: espaço y ocupado do saco de mochila até ao estágio j, $y = 0, 1, \ldots, W$
- Decisões: em cada estágio j, em cada estado $e_j(y)$, incluir, ou não, o item.
- As decisões alternativas dependem da quantidade de espaço ainda disponível, $W - y$.
- No caso de w_j ser maior que o espaço disponível, a única decisão possível é não incluir o item no saco.

Problema do saco de mochila binário (cont.)

- Rede acíclica: cada decisão conduz a um estado em que a quantidade de recurso utilizada é maior.
- Contribuição de estágio: a inclusão do item j no saco tem um valor p_j.
- Valor óptimo de um estado $f_j(y)$: máximo valor da função objectivo que é possível obter com itens com índice inferior a j, utilizando y unidades de espaço.
- Óptimo do problema: valor óptimo do estado $e_n(W)$

Problema do saco de mochila binário (cont.)

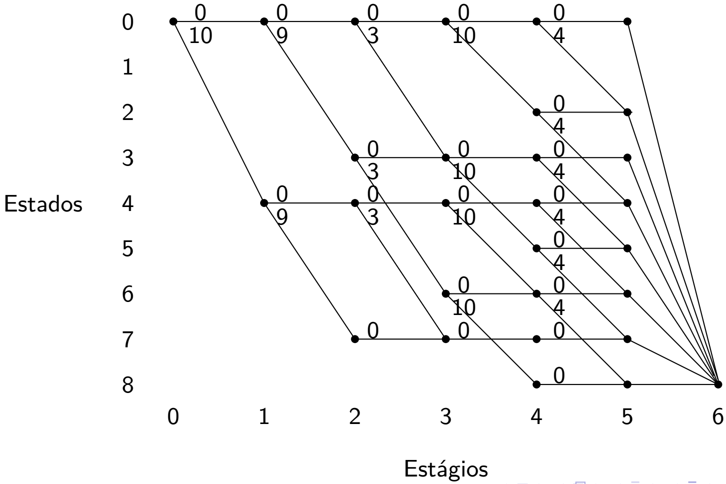
Equação de recorrência: valor de um estado y no estágio j expresso em função do valor de um estado no estágio $j-1$ e de uma contribuição do estágio j:

$$
\begin{aligned}
f_0(0) &= 0 \\
f_j(y) &= \max_{\substack{x_j = 0 \text{ ou } 1: \\ w_j x_j \leq W - y}} \{f_{j-1}(y - w_j x_j) + x_j p_j\},
\end{aligned}
$$

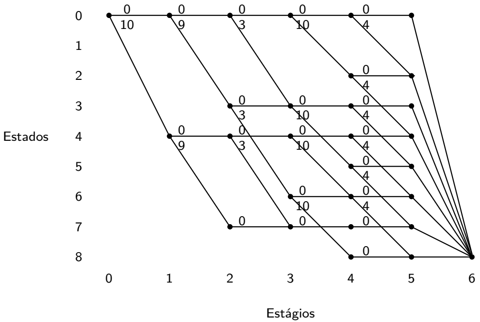
$$0 \leq y \leq W, j = 1, 2, \ldots, n$$

Exemplo

$$\begin{array}{ll} \max & 10x_1 + 9x_2 + 3x_3 + 10x_4 + 4x_5 \\ suj. & 4x_1 + 3x_2 + 3x_3 + 2x_4 + 2x_5 \leq 8 \\ & x_j = 0 \text{ ou } 1, \forall j \end{array}$$

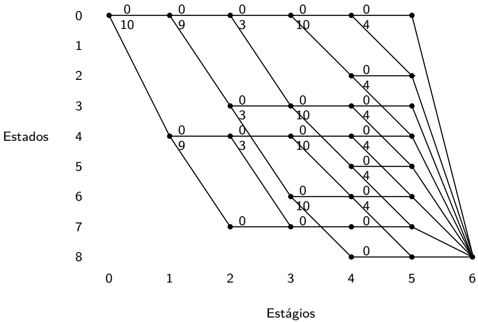


Estágio 1



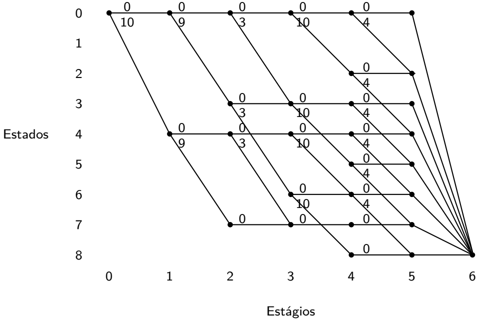
$$f_0(0) = 0$$

$$f_1(0) = f_0(0) + 0 = 0$$
$$f_1(4) = f_0(0) + 10 = 10$$



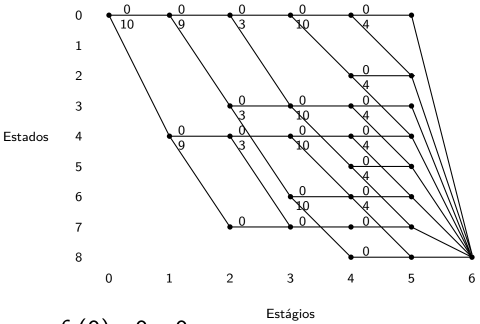
$$
\begin{aligned}
f_2(0) &= f_1(0) + 0 = 0 \\
f_2(3) &= f_1(0) + 9 = 9 \\
f_2(4) &= f_1(4) + 0 = 10 \\
f_2(7) &= f_1(4) + 9 = 19
\end{aligned}
$$

Estágio 3



$$
\begin{aligned}
f_3(0) &= f_2(0) + 0 = 0 \\
f_3(3) &= \max\{f_2(0) + 3, f_2(3) + 0\} = \max\{3, 9\} = 9 \\
f_3(4) &= f_2(4) + 0 = 10 \\
f_3(6) &= f_2(3) + 3 = 12 \\
f_3(7) &= \max\{f_2(4) + 3, f_2(7) + 0\} = \max\{13, 19\} = 19
\end{aligned}
$$

Estágio 4



$$
\begin{aligned}
f_4(0) &= f_3(0) + 0 = 0 \\
f_4(2) &= f_3(0) + 10 = 10 \\
f_4(3) &= f_3(3) + 0 = 9 \\
f_4(4) &= f_3(4) + 0 = 10 \\
f_4(5) &= f_3(3) + 10 = 19 \\
f_4(6) &= \max\{f_3(4) + 10, f_3(6) + 0\} = \max\{20, 12\} = 20 \\
f_4(7) &= f_3(7) + 0 = 19 \\
f_4(8) &= f_3(6) + 10 = 22
\end{aligned}
$$

Estágios 5 e 6

$$
\begin{aligned}
f_5(0) &= f_4(0) + 0 = 0 \\
f_5(2) &= \max\{f_4(0) + 4, f_4(2) + 0\} = \max\{4, 10\} = 10 \\
f_5(3) &= f_4(3) + 0 = 9 \\
f_5(4) &= \max\{f_4(2) + 4, f_4(4) + 0\} = \max\{14, 10\} = 14 \\
f_5(5) &= \max\{f_4(3) + 4, f_4(5) + 0\} = \max\{13, 19\} = 19 \\
f_5(6) &= \max\{f_4(4) + 4, f_4(6) + 0\} = \max\{14, 20\} = 20 \\
f_5(7) &= \max\{f_4(5) + 4, f_4(7) + 0\} = \max\{23, 19\} = 23 \\
f_5(8) &= \max\{f_4(6) + 4, f_4(8) + 0\} = \max\{24, 22\} = 24
\end{aligned}
$$

$$
f_6(8) = \max_i\{f_5(i)\} = 24
$$

Estágio 6 corresponde a considerar o espaço de sobra.
Esta solução corresponde a colocar 3 itens no saco de mochila, dos tipos 1,3 e 4.

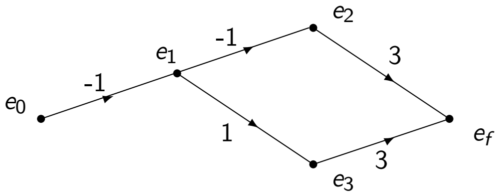
Princípio da Optimalidade

Vamos considerar uma sequência óptima de decisões desde um estado inicial e_0 *até um estado final* e_f, *definida pelo caminho óptimo* $e_0 e_1 e_2 \ldots e_i \ldots e_j \ldots e_f$.

O caminho entre dois quaisquer nodos e_i *e* e_j *do caminho óptimo, deve, por sua vez, constituir a sequência óptima de decisões entre os estados* e_i *e* e_j.

Ilustração do princípio da optimalidade

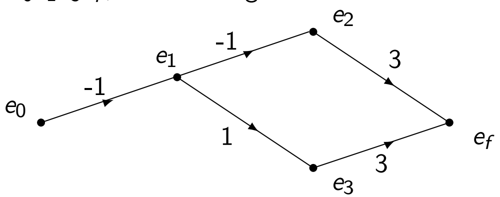
- Problema de minimização da soma das contribuições.
- Determinar o caminho mais curto numa rede de programação dinâmica.
- O caminho óptimo desde o estado inicial e_0 para o estado final e_f é $e_0 e_1 e_2 e_f$, com valor igual a 1.



- a porção deste caminho óptimo entre e_1 e e_f, ou seja, $e_1 e_2 e_f$, é o caminho óptimo entre esses dois estados, com valor igual a 2.

Caso em que o princípio da optimalidade não se verifica

- Problema de minimização do produto das contribuições.
- O caminho óptimo desde o estado e_0 para o estado final e_f é o caminho $e_0 e_1 e_3 e_f$, com valor igual a -3.

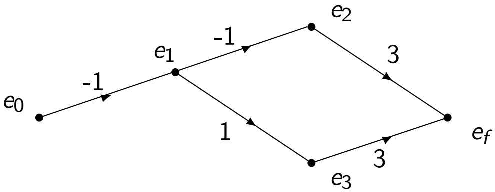


- A porção deste caminho óptimo entre e_1 e e_f, ou seja, $e_1 e_3 e_f$, não é o caminho óptimo entre esses dois estados.
- O caminho óptimo entre e_1 e e_f é o caminho $e_1 e_2 e_f$, com valor igual a -3.
- Neste caso, o princípio da optimalidade não se verifica, e, como tal, a programação dinâmica não pode ser utilizada.

Princípio da Optimalidade [Enunciado de Richard Bellman]

Uma política óptima tem a propriedade de, quaisquer que sejam o estado e a decisão iniciais, as restantes decisões devem constituir uma política óptima com respeito ao estado resultante da primeira transição.

Ilustração da não-aplicação do Princípio da Optimalidade



- A política $e_0 e_1 e_3 e_f$ não é uma política óptima, porque, para o estado e_0 e para a decisão inicial (que conduz ao estado e_1), as restantes decisões $e_3 e_f$ não constituem uma política óptima com respeito ao estado resultante da primeira transição, o estado e_1.
- De facto, a política óptima, com respeito ao estado e_1, é a política que conduz ao estado e_0 com menor custo, ou seja, $e_1 e_2 e_f$.

Princípio da Separabilidade

- O valor de um dado estado $f_j(y)$ do estado y no estágio j deve poder ser *separado*,
- e expresso em função **apenas** do valor de um outro estado num estágio adjacente, $f_{j-1}(y')$, e de uma contribuição de estágio r_j,
- através de uma relação de recorrência:

$$f_j(y) \quad = \quad \phi\{f_{j-1}(y'), r_j\}$$

- A caracterização do estado y' deve ser suficiente para determinar o seu valor, $f_{j-1}(y')$ (e também quais são as decisões alternativas admissíveis em y'),
- não devendo ser necessário usar a memória do que ocorreu em estágios anteriores, desde 1 até $j-1$.

Princípio da Separabilidade: exemplos

- contribuições de estágio aditivas:

$$
\begin{aligned}
f_j(y) &= \sum_{t=1}^{j} r_t \\
f_j(y) &= \sum_{t=1}^{j-1} r_t + r_j \\
f_j(y) &= f_{j-1}(y') + r_j
\end{aligned}
$$

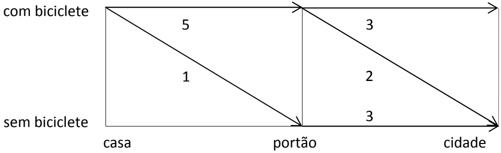
- contribuições de estágio multiplicativas (positivas):

$$
\begin{aligned}
f_j(y) &= \prod_{t=1}^{j} r_t \\
f_j(y) &= \prod_{t=1}^{j-1} r_t \times r_j \\
f_j(y) &= f_{j-1}(y') \times r_j
\end{aligned}
$$

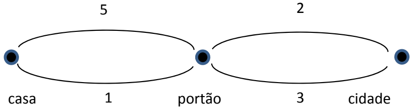
Caso em que o princípio da separabilidade não se verifica

- exemplo: viajar de casa até à cidade.
- alternativas são usar a bicicleta ou ir a pé, com as contribuições (tempo de viagem) indicadas.
- qual dos modelos está correcto?

Modelo 1:



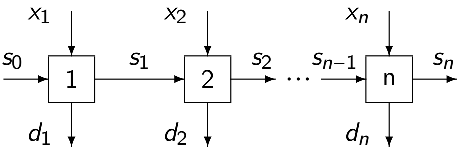
Modelo 2:



- porque é que o princípio da separabilidade não se verifica no Modelo 2?

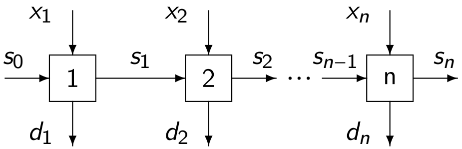
Lotes de Produção *(lotsizing)*

- Determinar a dimensão dos lotes a fabricar em cada período, dentro de um horizonte de planeamento.
- Em cada período, se o número de unidades disponíveis (*i.e.*, as unidades produzidas no período mais as existentes em stock) for superior à procura nesse período, as unidades remanescentes podem ser armazenadas em stock para venda em períodos subsequentes, segundo o seguinte esquema:



- Objectivo: minimização da soma dos custos de produção e dos custos de armazenagem, satisfazendo a procura em cada período.

Modelo dos lotes de produção



Variáveis de decisão:
x_j número de unidades produzidas no período j,
s_j stock existente após o período j.

Dados:
d_j procura existente no período j
c_j custo unitário de produção dos artigos no período j
h_j custo unitário de posse de inventário no período j
x_j^{max} número máximo de unidades produzidas no período j
s_j^{max} nível máximo de stock no período j

Modelo dos lotes de produção (cont.)

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n} c_j x_j + h_j s_j \\
\text{suj.} \quad & x_j + s_{j-1} - s_j = d_j \ , \ \forall j \\
& x_j \geq 0 \ , \ \forall j \\
& x_j \leq x_j^{max} \ , \ \forall j \\
& s_j \leq s_j^{max} \ , \ \forall j
\end{aligned}
$$

- Estágio j : cada período j, $j = 1, 2, \ldots, n$, é decidido sequencialmente o número de unidades a produzir.
- Estado $s_j(y)$: número de unidades em stock y no estágio j, $y = 0, 1, \ldots, s_j^{max}$
- Decisões: em cada estágio j, em cada estado $s_j(y)$, decidir o número de unidades a produzir.
- As decisões alternativas dependem da procura e do número de unidades em stock.

Exemplo

- Horizonte de planeamento (T): 4 períodos
- Procura em cada período de 1, 3, 2 e 2, respectivamente.
- Capacidade máxima de produção, x_j^{max} : 3 unidades em cada período.
- Nível máximo de stock, s_{max} : 2 unidades.
- Custos unitários de armazenagem, h_j : 1 U.M./ artigo x período.

Exemplo (cont.)

- Custos de produção incluem um custo de preparação das máquinas, k_j, e um custo variável proporcional ao número de artigos, p_j ::

$$c_j(x_j) = \begin{cases} k_j + p_j x_j & \text{, se } x_j > 0 \\ 0 & \text{, se } x_j = 0, \end{cases}$$

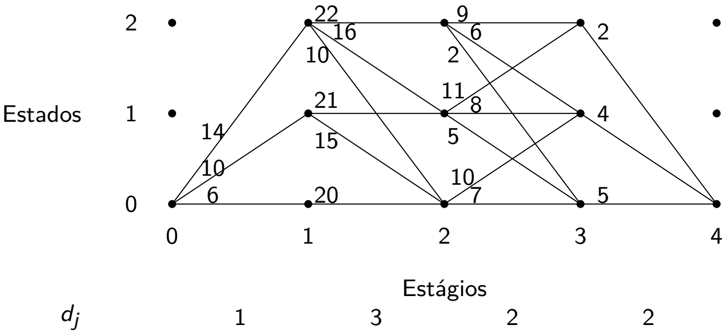
  para qualquer período $j = 1, 2, \ldots, T$.

- O valores dos coeficientes de custo para cada período são dados pelo seguinte Quadro:

j	1	2	3	4
custo variável p_j	4	6	3	2
custo fixo k_j	2	2	1	1

Exemplo (cont.)

j	1	2	3	4
custo variável p_j	4	6	3	2
custo fixo k_j	2	2	1	1



Cálculo das contribuições de estágio

Exemplo

- A contribuição de estágio que conduz do estado $e_1(2)$, ou seja o estado de nível de stock 2 no estágio 1, ao estado $e_2(1)$ tem o valor 16. Essa transição corresponde à produção de 2 unidades.
- No período 2, a procura é de 3 unidades. Para responder à procura, é usada 1 unidade de stock, passando o nível de stock de 2 unidades para 1 unidade, mais as 2 unidades produzidas nesse período.

Cálculo dos custos

- os custos de produção são iguais a 14, e resultam da soma de 2 unidades de custos de preparação ($k_2 = 2$) com 12 unidades de custos variáveis de produção de 2 unidades a um custo unitário de 6 ($p_2 = 6$).
- os custos de inventário são iguais a 2, porque havia, no início do período, duas unidades em stock, sendo o custo unitário de posse de 1 unidade/artigo x período ($h_2 = 1$).

Resolução

$$
\begin{aligned}
f_4(0) &= 0 \\
\\
f_3(2) &= 2 \\
f_3(1) &= 4 \\
f_3(0) &= 5 \\
\\
f_2(2) &= \min\{f_3(2)+9, f_3(1)+6, f_3(0)+2\} = \min\{11,10,7\} = 7 \\
f_2(1) &= \min\{f_3(2)+11, f_3(1)+8, f_3(0)+5\} = \min\{13,12,10\} = 10 \\
f_2(0) &= \min\{f_3(1)+10, f_3(0)+7\} = \min\{14,12\} = 12 \\
\\
f_1(2) &= \min\{f_2(2)+22, f_2(1)+16, f_2(0)+10\} = \min\{29,26,22\} = 22 \\
f_1(1) &= \min\{f_2(1)+21, f_2(0)+15\} = \min\{31,27\} = 27 \\
f_1(0) &= \min\{f_2(0)+20\} = 32 \\
\\
f_0(0) &= \min\{f_1(2)+14, f_1(1)+10, f_1(0)+6\} = \min\{36,37,38\} = 36
\end{aligned}
$$

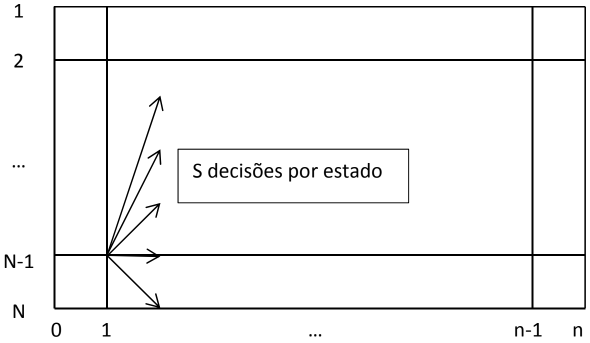
Resolução (cont.)

- A solução óptima tem um custo de 36.
- Analisando as transições efectuadas, a solução óptima corresponde a produzir 3 unidades no período 1, 1 unidade no período 2, 2 unidades no período 3 e 2 unidades no período 4.

Carga computacional

Dado um modelo de programação dinâmica com:

- n estágios
- N estados
- S decisões (acções) em cada estado (nota:$S \leq N$)



podemos calcular o número de operações para determinar a política (caminho) óptima, usando abordagens diferentes ...

Carga computacional da enumeração completa

enumeração de todos os caminhos e selecção do melhor:

- há NS^n caminhos.
- Carga computacional para cada caminho: n operações (*e.g.*, adição dos custos dos n arcos do caminho).
- Após calcular o valor do primeiro caminho,
- para cada um dos restantes $NS^n - 1$ caminhos, há uma comparação do valor do caminho com o melhor até ao instante, para escolher o melhor de todos.

algoritmo exponencial:

- carga total: $nNS^n + NS^n - 1 = (n+1)NS^n - 1$
- que não pode ser expressa como função polinomial de n,
- que aparece em expoente.

Carga computacional da programação dinâmica

aplicação da relação de recorrência, em cada nó da rede:

- S operações, uma para cada alternativa, e
- $S-1$ comparações (para seleccionar a melhor alternativa).
- Repetindo as operações em cada um dos nN nós da rede,

algoritmo polinomial em N e n

- carga total: $nN(2S-1)$,
- que pode ser expressa como função polinomial de n, N e S.

mas vamos ver que é, de facto, *pseudo-polinomial.*

Exemplo

Comparação do esforço computacional para:

- $n = N = S = 12$
- enumeração completa: 1.390.911.669.927.935
- programação dinâmica: 3312

questão:

Programação dinâmica e complexidade computacional

- O problema do saco de mochila pertence à classe de problemas NP-completos.
- Será que a programação dinâmica fornece um algoritmo polinomial?

resposta: não,

- porque a carga computacional da programação dinâmica é exponencial em relação à dimensão dos dados do *input*, porque:

se usarmos uma *representação razoável* para os dados de *input*:

- os $2n+1$ dados de um problema de saco de mochila:
- $2n$ valores associados aos itens, respectivamente, p_j e $w_j, j = 1, \ldots, n$,
- e o valor de W (peso máximo do saco de mochila).
- requerem um comprimento logarítmico, como vamos ver...

e a carga computacional da programação dinâmica $nN(2S-1)$,

- que envolve N (o número de estados, que é igual ao valor W), é exponencial em relação ao comprimento (logarítmico) da representação dos dados de *input*.

Representação *razoável* dos dados do problema

- em computadores, os números inteiros são representados na base binária.
- comprimento da *string* para representar o número x é $\lceil \log_2(x+1) \rceil$.
- exemplo: número 30 na base 10 representa-se por 11110 na base 2.

- Representação numa base B ($B \geq 2$) é da mesma ordem de grandeza:

$$\log_B n = \frac{\log n}{\log B}.$$

- porque $\log B$ é uma constante.

No entanto, se representarmos os dados na base unária:

- Representação na base unária: usa-se apenas um símbolo, *e.g.*, "/".
- exemplo: número 5 na base 10 representa-se por ///// na base 1.

- Se usarmos esta representação *unária* dos dados de *input*, N é polinomial em relação à dimensão dos dados do *input*,
- e a programação dinâmica fornece um algoritmo polinomial!

- Para estabelecer a complexidade de um problema, deve usar-se sempre uma *representação razoável* dos dados de *input*.
- Não há vantagem em dizer-se que se tem um algoritmo polinomial em relação a uma representação *exponencialmente grande* dos dados (ou seja, não é aceitável esconder a dificuldade do problema no seu *input*).

Algoritmos pseudo-polinomiais

- A complexidade do problema do saco de mochila resulta da dimensão dos dados do problema.
- *Se consideramos N como uma constante,* a programação dinâmica fornece um algoritmo polinomial para o problema do saco de mochila;
- daí a designação de pseudo-polinomial.

- Na prática, os valores de N são geralmente razoavelmente pequenos,
- tornando a programação dinâmica atractiva em muitos problemas.

Problemas NP-completos: classificação

- Dentro da classe de problemas NP-completos, diferenciam-se os problemas cuja complexidade é devida à sua própria estrutura e os problemas cuja complexidade resulta da dimensão dos seus dados:

- exemplo: o problema do caixeiro viajante é *fortemente* NP-completo (*strongly NP-complete*).
- Não são conhecidos algoritmos polinomiais para o problema do caixeiro viajante, mesmo que os dados sejam representados na base unária.

- exemplo: o problema do saco de mochila é *fracamente*-NP-completo (*weakly NP-complete*).
- Há algoritmos pseudo-polinomiais para a sua resolução.

Fim