Descrição do problema e de sua complexidade

- O problema binário da mochila (BKP):

  Entrada: um conjunto de n itens. Para cada item $i \in N = \{1, \ldots, n\}$ estão associados dois valores inteiros positivos w_i e c_i representando seu peso e seu custo, respectivamente. Dois valores inteiros positivos: W (capacidade da mochila) e C (custo alvo).
  Pergunta: Existe um subconjunto I de N tal que $\sum_{i \in I} w_i \leq W$ e $\sum_{i \in I} c_i \geq C$?

- Hipóteses:
  - $w_i \leq W$ para todo $i \in N$;
  - $\sum_{i \in N} w_i > W$.
- Complexidade:
  - BKP está em $\mathcal{NP}$.
  - Cook (1971): SAT é $\mathcal{NP}$-completo.
  - Karp (1972): $\texttt{SAT} \propto \texttt{CLIQUE} \propto \texttt{3DM} \propto \texttt{PART} \propto \texttt{BKP}$.

Tratamento de Problemas $\mathcal{NP}$-difíceis

- *Versão de otimização*:
  - encontrar $I \subseteq N$ satisfazendo $\sum_{i \in I} w_i \leq W$ que maximiza $\sum_{i \in I} c_i$.
  - A versão de otimização está em $\mathcal{NP}$-difícil: imediato.

Algoritmo exato de Programação Dinâmica

- Componentes básicas de um algoritmo de Programação Dinâmica:
  - Uma solução ótima contém soluções ótimas de subproblemas semelhantes ao problema original;
  - O valor ótimo está relacionado aos valores ótimos destes subproblemas por meio de uma fórmula de recorrência;
  - Os valores subótimos são tabelados de modo a impedir recálculos;
- Valores tabelados para BKP
  - $A[i, d]$: peso do subconjunto *mais leve* dos i primeiros itens com valor exatamente d;
  - $A[i, d] = \infty$ se não existir tal conjunto;
  - **idéia**: dadas duas soluções de mesmo custo, dá-se preferência àquela que tem menos peso;

Algoritmo exato de Programação Dinâmica

- Fórmula de recorrência:

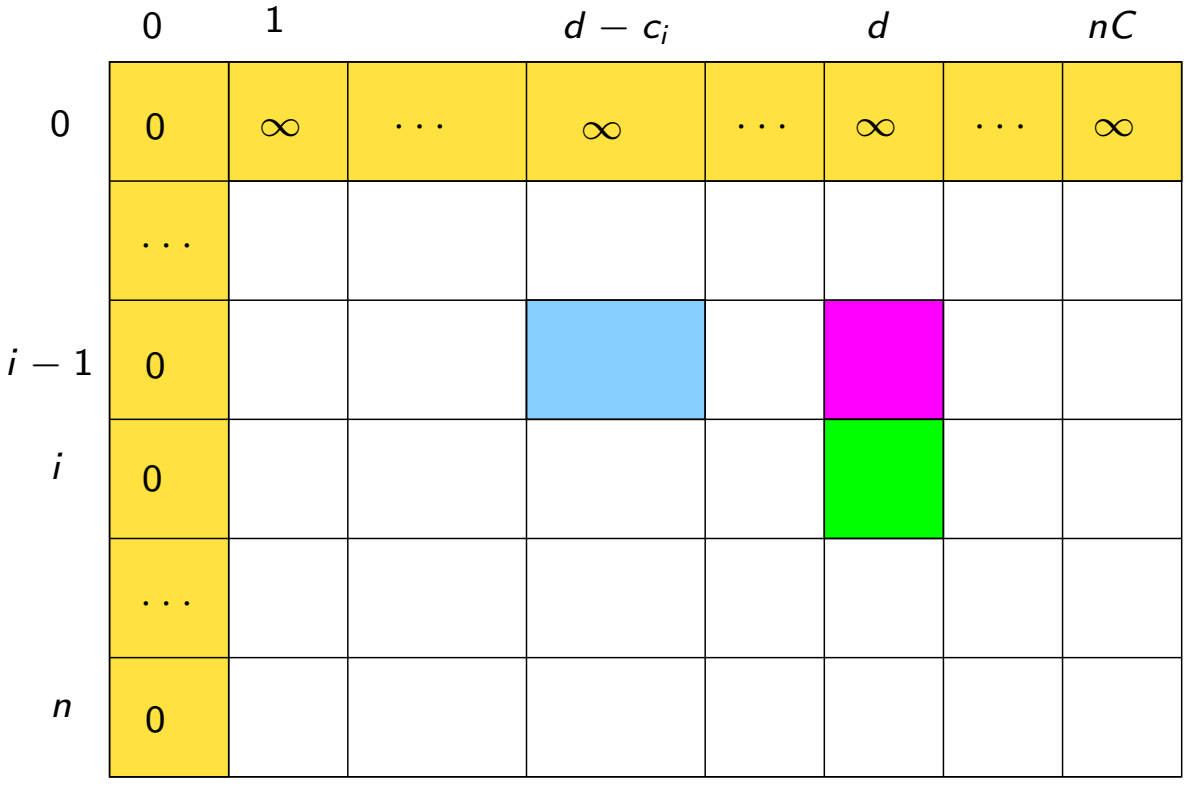
$$A[i,d] = \begin{cases} \min\{A[i-1,d], A[i-1,d-c_i] + w_i\}, & \text{se } c_i < d, \\ A[i-1,d], & \text{caso contrário.} \end{cases}$$

- Dimensão da matriz A: $(n+1) \times (nC+1)$.
- Inicialização de A:
  - Definição: $C = \max_{i \in N}\{c_i\}$;
  - *Primeira coluna*: $A[i,0] = 0$, para todo $i \in N \cup \{0\}$;
  - *Primeira linha*: $A[0,d] = \infty$, para todo $d \in \{1,\ldots,nC\}$;
- Valor ótimo: $C^* = \max\{d : A[n,d] \leq W\}$

(maior índice de uma coluna cujo valor da célula na última linha não excede W)

Algoritmo exato de Programação Dinâmica

- Preenchimento da matriz A:



$$A[i,d] = \begin{cases} \min\{A[i-1,d], A[i-1,d-c_i] + w_i\}, c_i < d, \\ A[i-1,d], \text{ caso contrário.} \end{cases}$$

Algoritmo exato de Programação Dinâmica

- Preenchimento da matriz A:

	0	1		$d - c_i$		d		nC
0	0	∞	$\cdots$	∞	$\cdots$	∞	$\cdots$	∞
	$\cdots$							
$i-1$	0							
i	0							
	$\cdots$							
n	0							

Complexidade: $O(n^2 C)$ *(pseudopolinomial !)*

Algoritmos aproximados

- Princípios básicos de um algoritmo aproximado:
  - Se A é um algoritmo de aproximação (relativa) para um problema de maximização então, para toda instância de entrada E, tem-se que

$$\frac{z^A(E)}{z^*(E)} \geq (1-\epsilon),$$

    onde,
    - z^A é o valor da solução obtida por A para E;
    - z^* é o valor ótimo para E;
    - ϵ é uma constante no intervalo $[0, 1)$.
  - O valor $(1-\epsilon)$ é o fator de aproximação de A.

Um algoritmo aproximado para BKP

- Seja ϵ um valor fixo no intervalo $[0, 1)$;
- Seja `ProgDin-BKP` uma rotina que implementa o algoritmo exato de Programação Dinâmica;
- Algoritmo:

`Mochila-APX`(n, W, c, w, ϵ);
1. $C \leftarrow \max_{i \in N}\{c_i\}$;
2. $K \leftarrow \frac{\epsilon C}{n}$;
3. **para** todo $i \in N$, **faça** $c'_i = \lfloor \frac{c_i}{K} \rfloor$;
4. **retorne** a solução de `ProgDin-BKP`(n, W, c', w);

fim.

- **Teorema**: `Mochila-APX` é uma $(1-\epsilon)$ aproximação para `BKP`.

  Prova: E instância qualquer de `BKP`; I solução de E obtida por `Mochila-APX`; I^* uma solução ótima de E e z^* o valor ótimo.

$$c'_i = \lfloor \tfrac{c_i}{K} \rfloor \Longrightarrow \begin{cases} c_i \geq K c'_i & (\star) \\ c'_i \geq (c_i/K) - 1 \text{ ou } K c'_i \geq c_i - K & (\dagger) \end{cases}$$

$$\begin{aligned} z^A(E) = \textstyle\sum_{i \in I} c_i &\geq \textstyle\sum_{i \in I} K c'_i \ (\star) \\ &\geq \textstyle\sum_{i \in I^*} K c'_i \ (I \text{ ótima para custos } c') \\ &\geq \textstyle\sum_{i \in I^*} (c_i - K) = \sum_{i \in I^*} c_i - |I^*| K \ (\dagger) \\ &\geq \textstyle\sum_{i \in I^*} c_i - nK = \sum_{i \in I^*} c_i - \epsilon C \\ &\geq z^* - \epsilon z^* = (1-\epsilon) z^*. \qquad \square \end{aligned}$$

Complexidade do Algoritmo

`Mochila-APX`(n, W, c, w, ϵ);
1. $C \leftarrow \max_{i \in N}\{c_i\}$;
2. $K \leftarrow \frac{\epsilon C}{n}$;
3. **para** todo $i \in N$, **faça** $c'_i = \lfloor \frac{c_i}{K} \rfloor$;
4. **retorne** a solução de `ProgDin-BKP`(n, W, c', w).

- Dominada pela execução de `ProgDin-BKP`: $O(n^2 C') = O(n^2 \frac{C}{K}) = O(n^3 \frac{1}{\epsilon})$.
- `Mochila-APX` é um esquema de aproximação completamente polinomial (`FPTAS`).