Complexidade de Espaço

Leandro M. Zatesko
leandro.zatesko@ufpr.br

7 de dezembro de 2016

- Introdução
- Definições preliminares
- Principais fatos conhecidos
- Exercícios

- Introdução
  - O Problema da Mochila
  - Buscas em grafos
- Definições preliminares
- Principais fatos conhecidos

KNAPSACK

Instância: Um inteiro positivo n, o qual representa o número de itens disponíveis; um inteiro não-negativo C, o qual representa a capacidade da mochila em unidades de peso; n inteiros não-negativos $v_1, \ldots, v_n$, os quais representam os valores de cada item; n inteiros não-negativos $w_1, \ldots, w_n$, os quais representam os pesos de cada item.

Solução: O maior valor total que é possível levar na mochila sem exceder sua capacidade, i.e.

$$\max_{\substack{S \subseteq \{1,\ldots,n\} \\ \sum_{i \in S} w_i \leqslant C}} \sum_{i \in S} v_i.$$

KNAPSACK

Instância: Um inteiro positivo n, o qual representa o número de itens disponíveis; um inteiro não-negativo C, o qual representa a capacidade da mochila em unidades de peso; n inteiros não-negativos $v_1, \ldots, v_n$, os quais representam os valores de cada item; n inteiros não-negativos $w_1, \ldots, w_n$, os quais representam os pesos de cada item.

Solução: O maior valor total que é possível levar na mochila sem exceder sua capacidade, i.e.

$$\max_{\substack{S \subseteq \{1,\ldots,n\} \\ \sum_{i \in S} w_i \leqslant C}} \sum_{i \in S} v_i.$$

Bcktrck_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

se $n = 0$, devolva 0;
se $w_n > C$, devolva

$$\text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1});$$

devolva

$$\max\begin{cases} \text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}), \\ v_n + \text{Bcktrck\_Knapsack}(n-1, C - w_n, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}). \end{cases}$$

Complexidade de tempo? $O(2^n)$
Complexidade de espaço? $O(n(\log n + \log C))$

Bcktrck_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

se $n = 0$, devolva 0;
se $w_n > C$, devolva

$$\text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1});$$

devolva

$$\max\begin{cases} \text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}), \\ v_n + \text{Bcktrck\_Knapsack}(n-1, C - w_n, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}). \end{cases}$$

Complexidade de tempo? $O(2^n)$

Complexidade de espaço? $O(n(\log n + \log C))$

Bcktrck_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

se $n = 0$, devolva 0;
se $w_n > C$, devolva

$$\text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1});$$

devolva

$$\max\begin{cases} \text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}), \\ v_n + \text{Bcktrck\_Knapsack}(n-1, C - w_n, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}). \end{cases}$$

Complexidade de tempo? $O(2^n)$

Complexidade de espaço? $O(n(\log n + \log C))$

Bcktrck_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

se $n = 0$, devolva 0;
se $w_n > C$, devolva

$$\textsc{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1});$$

devolva

$$\max\begin{cases} \textsc{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}), \\ v_n + \textsc{Bcktrck\_Knapsack}(n-1, C - w_n, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}). \end{cases}$$

Complexidade de tempo? $O(2^n)$
Complexidade de espaço? $O(n(\log n + \log C))$

BCKTRCK_KNAPSACK$(n, C, v_1, \dots, v_n, w_1, \dots, w_n)$:

1 se $n = 0$, devolva 0;

2 se $w_n > C$, devolva

$$\text{BCKTRCK\_KNAPSACK}(n-1, C, v_1, \dots, v_{n-1}, w_1, \dots, w_{n-1});$$

3 devolva

$$\max \begin{cases} \text{BCKTRCK\_KNAPSACK}(n-1, C, v_1, \dots, v_{n-1}, w_1, \dots, w_{n-1}), \\ v_n + \text{BCKTRCK\_KNAPSACK}(n-1, C - w_n, v_1, \dots, v_{n-1}, w_1, \dots, w_{n-1}). \end{cases}$$

Complexidade de tempo? $O(2^n)$

Complexidade de espaço? $O(n(\log n + \log C))$

Bcktrck_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

se $n = 0$, devolva 0;
se $w_n > C$, devolva

$$\text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1});$$

devolva

$$\max\begin{cases}\text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}),\\ v_n + \text{Bcktrck\_Knapsack}(n-1, C - w_n, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}).\end{cases}$$

Complexidade de tempo? $O(2^n)$
Complexidade de espaço? $O(n(\log n + \log C))$

Bcktrck_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

se $n = 0$, devolva 0;
se $w_n > C$, devolva

$$\text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1});$$

devolva

$$\max \begin{cases} \text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}), \\ v_n + \text{Bcktrck\_Knapsack}(n-1, C - w_n, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}). \end{cases}$$

Complexidade de tempo? $O(2^n)$
Complexidade de espaço? $O(n(\log n + \log C))$

Bcktrck_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

se $n = 0$, devolva 0;
se $w_n > C$, devolva

$$\text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1});$$

devolva

$$\max \begin{cases} \text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}), \\ v_n + \text{Bcktrck\_Knapsack}(n-1, C - w_n, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}). \end{cases}$$

Complexidade de tempo? $O(2^n)$

Complexidade de espaço? $O(n(\log n + \log C))$

Bcktrck_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

1 **se** $n = 0$, **devolva** 0;

2 **se** $w_n > C$, **devolva**

$$\text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1});$$

3 **devolva**

$$\max \begin{cases} \text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}), \\ v_n + \text{Bcktrck\_Knapsack}(n-1, C - w_n, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}). \end{cases}$$

Complexidade de tempo? $O(2^n)$

Complexidade de espaço? $O(n(\log n + \log C))$

Bcktrck_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

1 se $n = 0$, devolva 0;

2 se $w_n > C$, devolva

$$\text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1});$$

3 devolva

$$\max \begin{cases} \text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}), \\ v_n + \text{Bcktrck\_Knapsack}(n-1, C - w_n, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}). \end{cases}$$

Complexidade de tempo? $O(2^n)$

Complexidade de espaço? $O(n(\log n + \log C))$

Bcktrck_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

1 **se** $n = 0$, **devolva** 0;

2 **se** $w_n > C$, **devolva**

$$\text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1});$$

3 **devolva**

$$\max \begin{cases} \text{Bcktrck\_Knapsack}(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}), \\ v_n + \text{Bcktrck\_Knapsack}(n-1, C - w_n, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1}). \end{cases}$$

Complexidade de tempo? $O(2^n)$

Complexidade de espaço? $O(n(\log n + \log C))$

DP_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

para i **de** 0 **até** n, **faça**:
  para c **de** 1 **até** C, **faça**:
    se $i = 0$, $S[i][c] \leftarrow 0$;
    senão, **se** $w_i > c$, $S[i][c] \leftarrow S[i-1][c]$;
    senão, $S[i][c] \leftarrow \max\{S[i-1][c], v_i + S[i-1][c - w_i]\}$.
devolva $S[n][C]$.

Complexidade de tempo? $O(nC)$
Complexidade de espaço? $O(C)$

DP_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

para i **de** 0 **até** n, **faça**:
  para c **de** 1 **até** C, **faça**:
    se $i = 0$, $\mathrm{S}[i][c] \leftarrow 0$;
    senão, **se** $w_i > c$, $\mathrm{S}[i][c] \leftarrow \mathrm{S}[i-1][c]$;
    senão, $\mathrm{S}[i][c] \leftarrow \max\{\mathrm{S}[i-1][c], v_i + \mathrm{S}[i-1][c-w_i]\}$.
devolva $\mathrm{S}[n][C]$.

Complexidade de tempo? $O(nC)$
Complexidade de espaço? $O(C)$

DP_KNAPSACK$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

para i **de** 0 **até** n, **faça**:
    para c **de** 1 **até** C, **faça**:
        se $i = 0$, $S[i][c] \leftarrow 0$;
        senão, **se** $w_i > c$, $S[i][c] \leftarrow S[i-1][c]$;
        senão, $S[i][c] \leftarrow \max\{S[i-1][c], v_i + S[i-1][c-w_i]\}$.
devolva $S[n][C]$.

Complexidade de tempo? $O(nC)$

Complexidade de espaço? $O(C)$

DP_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

para i **de** 0 **até** n, **faça**:
    para c **de** 1 **até** C, **faça**:
        se $i = 0$, $S[i][c] \leftarrow 0$;
        senão, se $w_i > c$, $S[i][c] \leftarrow S[i-1][c]$;
        senão, $S[i][c] \leftarrow \max\{S[i-1][c], v_i + S[i-1][c-w_i]\}$.
devolva $S[n][C]$.

Complexidade de tempo? $O(nC)$
Complexidade de espaço? $O(C)$

DP_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

```
para i de 0 até n, faça:
    para c de 1 até C, faça:
        se i = 0, S[i][c] ← 0;
        senão, se w_i > c, S[i][c] ← S[i−1][c];
        senão, S[i][c] ← max{S[i−1][c], v_i + S[i−1][c−w_i]}.
devolva S[n][C].
```

Complexidade de tempo? $O(nC)$

Complexidade de espaço? $O(C)$

DP_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

```
para i de 0 até n, faça:
   para c de 1 até C, faça:
      se i = 0, S[i][c] ← 0;
      senão, se w_i > c, S[i][c] ← S[i − 1][c];
      senão, S[i][c] ← max{S[i − 1][c], v_i + S[i − 1][c − w_i]}.
devolva S[n][C].
```

Complexidade de tempo? $O(nC)$

Complexidade de espaço? $O(C)$

DP_KNAPSACK$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

```
para i de 0 até n, faça:
    para c de 1 até C, faça:
        se i = 0, S[i][c] ← 0;
        senão, se w_i > c, S[i][c] ← S[i − 1][c];
        senão, S[i][c] ← max{S[i − 1][c], v_i + S[i − 1][c − w_i]}.
devolva S[n][C].
```

Complexidade de tempo? $O(nC)$
Complexidade de espaço? $O(C)$

DP_Knapsack($n, C, v_1, \ldots, v_n, w_1, \ldots, w_n$):

```
para i de 0 até n, faça:
    para c de 1 até C, faça:
        se i = 0, S[i][c] ← 0;
        senão, se w_i > c, S[i][c] ← S[i − 1][c];
        senão, S[i][c] ← max{S[i − 1][c], v_i + S[i − 1][c − w_i]}.
devolva S[n][C].
```

Complexidade de tempo? $O(nC)$
Complexidade de espaço? $O(C)$

DP_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

```
para i de 0 até n, faça:
    para c de 1 até C, faça:
        se i = 0, S[i][c] ← 0;
        senão, se w_i > c, S[i][c] ← S[i − 1][c];
        senão, S[i][c] ← max{S[i − 1][c], v_i + S[i − 1][c − w_i]}.
devolva S[n][C].
```

Complexidade de tempo? $O(nC)$
Complexidade de espaço? $O(C)$

DP_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

para i **de** 0 **até** n, **faça**:
    para c **de** 1 **até** C, **faça**:
        se $i = 0$, $\mathrm{S}[i][c] \leftarrow 0$;
        senão, **se** $w_i > c$, $\mathrm{S}[i][c] \leftarrow \mathrm{S}[i-1][c]$;
        senão, $\mathrm{S}[i][c] \leftarrow \max\{\mathrm{S}[i-1][c], v_i + \mathrm{S}[i-1][c - w_i]\}$.
devolva $\mathrm{S}[n][C]$.

Complexidade de tempo? $O(nC)$
Complexidade de espaço? $O(C)$

DP_Knapsack$(n, C, v_1, \ldots, v_n, w_1, \ldots, w_n)$:

para i **de** 0 **até** n, **faça**:
    para c **de** 1 **até** C, **faça**:
        se $i = 0$, $\mathrm{S}[i][c] \leftarrow 0$;
        senão, **se** $w_i > c$, $\mathrm{S}[i][c] \leftarrow \mathrm{S}[i-1][c]$;
        senão, $\mathrm{S}[i][c] \leftarrow \max\{\mathrm{S}[i-1][c], v_i + \mathrm{S}[i-1][c-w_i]\}$.
devolva $\mathrm{S}[n][C]$.

Complexidade de tempo? $O(nC)$
Complexidade de espaço? $O(C)$

- Introdução
  - O Problema da Mochila
  - Buscas em grafos
- Definições preliminares
- Principais fatos conhecidos

Busca_Genérica(G, s):

```
visite s e inicialize uma estrutura de dados X com s;
enquanto X não estiver vazia, faça:
    remova um vértice u de X;
    para toda aresta uv tal que v ainda não foi visitado, faça:
        visite v e insira v em X.
```

Complexidade de tempo? $O(|V(G)| + |E(G)|)$
Complexidade de espaço? $O(|V(G)|)$

Busca_Genérica(G, s):

visite s e inicialize uma estrutura de dados X com s;
enquanto X não estiver vazia, **faça**:
  remova um vértice u de X;
  para toda aresta uv tal que v ainda não foi visitado, **faça**:
    visite v e insira v em X.

Complexidade de tempo? $O(|V(G)| + |E(G)|)$

Complexidade de espaço? $O(|V(G)|)$

Busca_Genérica(G, s):

```
visite s e inicialize uma estrutura de dados X com s;
enquanto X não estiver vazia, faça:
    remova um vértice u de X;
    para toda aresta uv tal que v ainda não foi visitado, faça:
        visite v e insira v em X.
```

Complexidade de tempo? $O(|V(G)| + |E(G)|)$

Complexidade de espaço? $O(|V(G)|)$

Busca_Genérica(G, s):

```
visite s e inicialize uma estrutura de dados X com s;
enquanto X não estiver vazia, faça:
    remova um vértice u de X;
    para toda aresta uv tal que v ainda não foi visitado, faça:
        visite v e insira v em X.
```

Complexidade de tempo? $O(|V(G)| + |E(G)|)$
Complexidade de espaço? $O(|V(G)|)$

Busca_Genérica(G, s):

visite s e inicialize uma estrutura de dados X com s;
enquanto X não estiver vazia, **faça**:
    remova um vértice u de X;
    para toda aresta uv tal que v ainda não foi visitado, **faça**:
        visite v e insira v em X.

Complexidade de tempo? $O(|V(G)| + |E(G)|)$
Complexidade de espaço? $O(|V(G)|)$

Busca_Genérica(G, s):

visite s e inicialize uma estrutura de dados X com s;
enquanto X não estiver vazia, faça:
    remova um vértice u de X;
    para toda aresta uv tal que v ainda não foi visitado, faça:
        visite v e insira v em X.

Complexidade de tempo? $O(|V(G)| + |E(G)|)$
Complexidade de espaço? $O(|V(G)|)$

Busca_Genérica(G, s):

```
visite s e inicialize uma estrutura de dados X com s;
enquanto X não estiver vazia, faça:
    remova um vértice u de X;
    para toda aresta uv tal que v ainda não foi visitado, faça:
        visite v e insira v em X.
```

Complexidade de tempo? $O(|V(G)| + |E(G)|)$
Complexidade de espaço? $O(|V(G)|)$

Busca_Genérica(G, s):

visite s e inicialize uma estrutura de dados X com s;
enquanto X não estiver vazia, **faça**:
  remova um vértice u de X;
  para toda aresta uv tal que v ainda não foi visitado, **faça**:
    visite v e insira v em X.

Complexidade de tempo? $O(|V(G)| + |E(G)|)$
Complexidade de espaço? $O(|V(G)|)$

Busca_Genérica(G, s):

```
visite s e inicialize uma estrutura de dados X com s;
enquanto X não estiver vazia, faça:
    remova um vértice u de X;
    para toda aresta uv tal que v ainda não foi visitado, faça:
        visite v e insira v em X.
```

Complexidade de tempo? $O(|V(G)| + |E(G)|)$
Complexidade de espaço? $O(|V(G)|)$

Busca_Genérica(G, s):

```
visite s e inicialize uma estrutura de dados X com s;
enquanto X não estiver vazia, faça:
    remova um vértice u de X;
    para toda aresta uv tal que v ainda não foi visitado, faça:
        visite v e insira v em X.
```

Complexidade de tempo? $O(|V(G)| + |E(G)|)$
Complexidade de espaço? $O(|V(G)|)$

Busca_Genérica(G, s):

visite s e inicialize uma estrutura de dados X com s;
enquanto X não estiver vazia, faça:
    remova um vértice u de X;
    para toda aresta uv tal que v ainda não foi visitado, faça:
        visite v e insira v em X.

Complexidade de tempo? $O(|V(G)| + |E(G)|)$

Complexidade de espaço? $O(|V(G)|)$

Jogo de Damas	10^{22}
PC	$10^{22}/10^{9} = 10^{13}$ s (300 mil anos)
TaihuLight	$10^{22}/10^{17} = 10^{5}$ s (27 horas)
Armazenamento total global	10^{21}

Jogo de Damas	10^{22}
PC	$10^{22}/10^{9} = 10^{13}$ s (300 mil anos)
TaihuLight	$10^{22}/10^{17} = 10^{5}$ s (27 horas)
Armazenamento total global	10^{21}

Jogo de Damas	10^{22}
PC	$10^{22}/10^{9} = 10^{13}$ s (300 mil anos)
TaihuLight	$10^{22}/10^{17} = 10^{5}$ s (27 horas)
Armazenamento total global	10^{21}

Jogo de Damas	10^{22}
PC	$10^{22}/10^9 = 10^{13}$ s (300 mil anos)
TaihuLight	$10^{22}/10^{17} = 10^5$ s (27 horas)
Armazenamento total global	10^{21}

- Introdução
- Definições preliminares
  - Máquinas de Turing com múltiplas fitas
  - Espaço de uma Máquina de Turing e grafo das configurações
  - Classes de Complexidade de Espaço
- Principais fatos conhecidos

Máquina de Turing Determinística com múltiplas fitas

$$M = (k, Q, \Sigma, s, f, \delta)$$

$$\delta : Q \times \Sigma^k \to Q \times (\Sigma \times \{-1, 0, 1\})^k$$

(i) sendo $(q, \sigma_1, \ldots, \sigma_k) \in Q \times \Sigma^k$, $(q', (\sigma_1', d_1), \ldots, (\sigma_k', d_k)) = \delta(q, \sigma_1, \ldots, \sigma_k)$, $\sigma_1' = \sigma_1$ e, para todo $i \in \{1, \ldots, k\}$, $\sigma_i' = \lhd$ e somente se $\sigma_i = \lhd$ e $d_i = 1$;

(ii) para todo $(q, \sigma_1, \ldots, \sigma_k) \in Q \times \Sigma^k$, $\delta \downarrow (q, \sigma_1, \ldots, \sigma_k)$ se e somente se $q \neq f$.

Máquina de Turing Determinística com múltiplas fitas

$$M = (k, Q, \Sigma, s, f, \delta)$$

$$\delta : Q \times \Sigma^k \to Q \times (\Sigma \times \{-1, 0, 1\})^k$$

(i) sendo $(q, \sigma_1, \ldots, \sigma_k) \in Q \times \Sigma^k$, $(q', (\sigma_1', d_1), \ldots, (\sigma_k', d_k)) = \delta(q, \sigma_1, \ldots, \sigma_k)$, $\sigma_1' = \sigma_1$ e, para todo $i \in \{1, \ldots, k\}$, $\sigma_i' = \triangleleft$ e somente se $\sigma_i = \triangleleft$ e $d_i = 1$;

(ii) para todo $(q, \sigma_1, \ldots, \sigma_k) \in Q \times \Sigma^k$, $\delta \downarrow (q, \sigma_1, \ldots, \sigma_k)$ se e somente se $q \neq f$.

Máquina de Turing Determinística com múltiplas fitas

$$M = (k, Q, \Sigma, s, f, \delta)$$
$$\delta : Q \times \Sigma^k \to Q \times (\Sigma \times \{-1, 0, 1\})^k$$

(i) sendo $(q, \sigma_1, \ldots, \sigma_k) \in Q \times \Sigma^k$, $(q', (\sigma'_1, d_1), \ldots, (\sigma'_k, d_k)) = \delta(q, \sigma_1, \ldots, \sigma_k)$, $\sigma'_1 = \sigma_1$ e, para todo $i \in \{1, \ldots, k\}$, $\sigma'_i = \lhd$ e somente se $\sigma_i = \lhd$ e $d_i = 1$;

(ii) para todo $(q, \sigma_1, \ldots, \sigma_k) \in Q \times \Sigma^k$, $\delta \downarrow (q, \sigma_1, \ldots, \sigma_k)$ se e somente se $q \neq f$.

Máquina de Turing Determinística com múltiplas fitas

$$M = (k, Q, \Sigma, s, f, \delta)$$
$$\delta : Q \times \Sigma^k \to Q \times (\Sigma \times \{-1, 0, 1\})^k$$

(i) sendo $(q, \sigma_1, \ldots, \sigma_k) \in Q \times \Sigma^k$, $(q', (\sigma_1', d_1), \ldots, (\sigma_k', d_k)) = \delta(q, \sigma_1, \ldots, \sigma_k)$, $\sigma_1' = \sigma_1$ e, para todo $i \in \{1, \ldots, k\}$, $\sigma_i' = \triangleleft$ e somente se $\sigma_i = \triangleleft$ e $d_i = 1$;

(ii) para todo $(q, \sigma_1, \ldots, \sigma_k) \in Q \times \Sigma^k$, $\delta \downarrow (q, \sigma_1, \ldots, \sigma_k)$ se e somente se $q \neq f$.

Máquina de Turing Não-determinística com múltiplas fitas

$$M = (k, Q, \Sigma, s, f, \Delta)$$

$$\Delta \subseteq (Q \times \Sigma^k) \times (Q \times (\Sigma \times \{-1, 0, 1\})^k)$$

(i) sendo $(q, \sigma_1, \ldots, \sigma_k) \in Q \times \Sigma^k$ e $(q', (\sigma'_1, d_1), \ldots, (\sigma'_k, d_k)) \in Q \times (\Sigma \times \{-1, 0, 1\})^k$ tais que $(q, \sigma_1, \ldots, \sigma_k) \underset{\Delta}{\longrightarrow} (q', (\sigma'_1, d_1), \ldots, (\sigma'_k, d_k))$, $\sigma'_1 = \sigma_1$ e, para todo $i \in \{1, \ldots, k\}$, $\sigma'_i = \triangleleft$ e somente se $\sigma_i = \triangleleft$ e $d_i = 1$;

(ii) para todo $(q, \sigma_1, \ldots, \sigma_k) \in Q \times \Sigma^k$, existe $(q', (\sigma'_1, d_1), \ldots, (\sigma'_k, d_k)) \in Q \times (\Sigma \times \{-1, 0, 1\})^k$ satisfazendo $(q, \sigma_1, \ldots, \sigma_k) \underset{\Delta}{\longrightarrow} (q', (\sigma'_1, d_1), \ldots, (\sigma'_k, d_k))$ se e somente se $q \neq f$.

Máquina de Turing Não-determinística com múltiplas fitas

$$M = (k, Q, \Sigma, s, f, \Delta)$$
$$\Delta \subseteq (Q \times \Sigma^k) \times (Q \times (\Sigma \times \{-1, 0, 1\})^k)$$

(i) sendo $(q, \sigma_1, \ldots, \sigma_k) \in Q \times \Sigma^k$ e $(q', (\sigma_1', d_1), \ldots, (\sigma_k', d_k)) \in Q \times (\Sigma \times \{-1, 0, 1\})^k$ tais que $(q, \sigma_1, \ldots, \sigma_k) \underset{\Delta}{\longrightarrow} (q', (\sigma_1', d_1), \ldots, (\sigma_k', d_k))$, $\sigma_1' = \sigma_1$ e, para todo $i \in \{1, \ldots, k\}$, $\sigma_i' = \triangleleft$ e somente se $\sigma_i = \triangleleft$ e $d_i = 1$;

(ii) para todo $(q, \sigma_1, \ldots, \sigma_k) \in Q \times \Sigma^k$, existe $(q', (\sigma_1', d_1), \ldots, (\sigma_k', d_k)) \in Q \times (\Sigma \times \{-1, 0, 1\})^k$ satisfazendo $(q, \sigma_1, \ldots, \sigma_k) \underset{\Delta}{\longrightarrow} (q', (\sigma_1', d_1), \ldots, (\sigma_k', d_k))$ se e somente se $q \neq f$.

Máquina de Turing Não-determinística com múltiplas fitas

$$M = (k, Q, \Sigma, s, f, \Delta)$$
$$\Delta \subseteq (Q \times \Sigma^k) \times (Q \times (\Sigma \times \{-1, 0, 1\})^k)$$

(i) sendo $(q, \sigma_1, \ldots, \sigma_k) \in Q \times \Sigma^k$ e $(q', (\sigma_1', d_1), \ldots, (\sigma_k', d_k)) \in Q \times (\Sigma \times \{-1, 0, 1\})^k$ tais que $(q, \sigma_1, \ldots, \sigma_k) \underset{\Delta}{\longrightarrow} (q', (\sigma_1', d_1), \ldots, (\sigma_k', d_k))$, $\sigma_1' = \sigma_1$ e, para todo $i \in \{1, \ldots, k\}$, $\sigma_i' = \triangleleft$ e somente se $\sigma_i = \triangleleft$ e $d_i = 1$;

(ii) para todo $(q, \sigma_1, \ldots, \sigma_k) \in Q \times \Sigma^k$, existe $(q', (\sigma_1', d_1), \ldots, (\sigma_k', d_k)) \in Q \times (\Sigma \times \{-1, 0, 1\})^k$ satisfazendo $(q, \sigma_1, \ldots, \sigma_k) \underset{\Delta}{\longrightarrow} (q', (\sigma_1', d_1), \ldots, (\sigma_k', d_k))$ se e somente se $q \neq f$.

Máquina de Turing Não-determinística com múltiplas fitas

$$M = (k, Q, \Sigma, s, f, \Delta)$$
$$\Delta \subseteq (Q \times \Sigma^k) \times (Q \times (\Sigma \times \{-1, 0, 1\})^k)$$

(i) sendo $(q, \sigma_1, \ldots, \sigma_k) \in Q \times \Sigma^k$ e $(q', (\sigma_1', d_1), \ldots, (\sigma_k', d_k)) \in Q \times (\Sigma \times \{-1, 0, 1\})^k$ tais que $(q, \sigma_1, \ldots, \sigma_k) \underset{\Delta}{\longrightarrow} (q', (\sigma_1', d_1), \ldots, (\sigma_k', d_k))$, $\sigma_1' = \sigma_1$ e, para todo $i \in \{1, \ldots, k\}$, $\sigma_i' = \triangleleft$ e somente se $\sigma_i = \triangleleft$ e $d_i = 1$;

(ii) para todo $(q, \sigma_1, \ldots, \sigma_k) \in Q \times \Sigma^k$, existe $(q', (\sigma_1', d_1), \ldots, (\sigma_k', d_k)) \in Q \times (\Sigma \times \{-1, 0, 1\})^k$ satisfazendo $(q, \sigma_1, \ldots, \sigma_k) \underset{\Delta}{\longrightarrow} (q', (\sigma_1', d_1), \ldots, (\sigma_k', d_k))$ se e somente se $q \neq f$.

- o conteúdo da primeira fita de uma Máquina de Turing nunca é alterado;
- a entrada de uma Máquina de Turing é fornecida sempre na primeira fita, uma fita somente de leitura;
- a saída de uma Máquina de Turing é obtida sempre da última fita, uma fita somente de escrita;
- toda fita que não é nem a primeira nem a última é chamada de *fita interna*.

- o conteúdo da primeira fita de uma Máquina de Turing nunca é alterado;
- a entrada de uma Máquina de Turing é fornecida sempre na primeira fita, uma fita somente de leitura;
- a saída de uma Máquina de Turing é obtida sempre da última fita, uma fita somente de escrita;
- toda fita que não é nem a primeira nem a última é chamada de *fita interna*.

- o conteúdo da primeira fita de uma Máquina de Turing nunca é alterado;
- a entrada de uma Máquina de Turing é fornecida sempre na primeira fita, uma fita somente de leitura;
- a saída de uma Máquina de Turing é obtida sempre da última fita, uma fita somente de escrita;
- toda fita que não é nem a primeira nem a última é chamada de *fita interna*.

- o conteúdo da primeira fita de uma Máquina de Turing nunca é alterado;
- a entrada de uma Máquina de Turing é fornecida sempre na primeira fita, uma fita somente de leitura;
- a saída de uma Máquina de Turing é obtida sempre da última fita, uma fita somente de escrita;
- toda fita que não é nem a primeira nem a última é chamada de *fita interna*.

- Introdução
- Definições preliminares
  - Máquinas de Turing com múltiplas fitas
  - Espaço de uma Máquina de Turing e grafo das configurações
  - Classes de Complexidade de Espaço
- Principais fatos conhecidos

Espaço...

...de uma configuração C de M ($s_M(C)$) é o comprimento da concatenação dos conteúdos das fitas internas;

...de um traçado Z em M ($s_M(Z)$) é o máximo dentre os espaços das configurações do traçado;

...de M para uma entrada x ($s_M(x)$) é o máximo dentre os espaços de todos os traçados de M que conduzem sua configuração inicial para x a uma configuração final.

Complexidade de espaço $S_M(n)$

Espaço...

...de uma configuração C de M ($s_M(C)$) é o comprimento da concatenação dos conteúdos das fitas internas;

...de um traçado Z em M ($s_M(Z)$) é o máximo dentre os espaços das configurações do traçado;

...de M para uma entrada x ($s_M(x)$) é o máximo dentre os espaços de todos os traçados de M que conduzem sua configuração inicial para x a uma configuração final.

Complexidade de espaço $S_M(n)$

Grafo das configuração $G(M, x)$

(i) o conjunto de vértices de $G(M, x)$ é o conjunto de todas as configurações C de M tais que $C_0 \xrightarrow[M]{n} C$ para algum $n \in \mathbb{N}$;

(ii) sendo C_1 e C_2 vértices de $G(M, x)$, a aresta dirigida $C_1 C_2$ existe em $G(M, x)$ se e somente se $C_1 \xrightarrow[M]{} C_2$.

- Introdução
- Definições preliminares
  - Máquinas de Turing com múltiplas fitas
  - Espaço de uma Máquina de Turing e grafo das configurações
  - Classes de Complexidade de Espaço
- Principais fatos conhecidos

$$\mathcal{L} = \mathcal{SPACE}(\log n),$$
$$\mathcal{NL} = \mathcal{NSPACE}(\log n),$$
$$\mathcal{PSPACE} = \bigcup_{k \geqslant 0} \mathcal{SPACE}(n^k) \quad \text{e}$$
$$\mathcal{NPSPACE} = \bigcup_{k \geqslant 0} \mathcal{NSPACE}(n^k).$$

$$\mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PSPACE} = \mathcal{NPSPACE} \subseteq \mathcal{EXP} \subseteq \mathcal{NEXP}$$

$$\mathcal{L} = \mathcal{SPACE}(\log n),$$
$$\mathcal{NL} = \mathcal{NSPACE}(\log n),$$
$$\mathcal{PSPACE} = \bigcup_{k \geqslant 0} \mathcal{SPACE}(n^k) \quad \text{e}$$
$$\mathcal{NPSPACE} = \bigcup_{k \geqslant 0} \mathcal{NSPACE}(n^k).$$

$$\mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PSPACE} = \mathcal{NPSPACE} \subseteq \mathcal{EXP} \subseteq \mathcal{NEXP}$$

STCON

Instância: Um grafo dirigido G, uma origem $s \in V(G)$ e um destino $t \in V(G)$.

Pergunta: Existe caminho dirigido em G de s a t?

STCON $\in \mathcal{P}$

STCON

Instância: Um grafo dirigido G, uma origem $s \in V(G)$ e um destino $t \in V(G)$.

Pergunta: Existe caminho dirigido em G de s a t?

$$\text{STCON} \in \mathcal{P}$$

NON_DETERMINISTIC_WALK(G, s, t):

```
u ← s;
para i de 1 até |V(G)| − 1, faça:
    escolha não-deterministicamente uma aresta uv ∈ E(G);
    se v = t, devolva 1;
    u ← v;
devolva 0.
```

STCON $\in \mathcal{NL}$

USTCON $\in \mathcal{L}$

NON_DETERMINISTIC_WALK(G, s, t):

```
u ← s;
para i de 1 até |V(G)| − 1, faça:
    escolha não-deterministicamente uma aresta uv ∈ E(G);
    se v = t, devolva 1;
    u ← v;
devolva 0.
```

$$\text{STCON} \in \mathcal{NL}$$

$$\text{USTCON} \in \mathcal{L}$$

NON_DETERMINISTIC_WALK(G, s, t):

```
1  u ← s;
2  para i de 1 até |V(G)| − 1, faça:
3     escolha não-deterministicamente uma aresta uv ∈ E(G);
4     se v = t, devolva 1;
5     u ← v;
6  devolva 0.
```

$$\text{STCON} \in \mathcal{NL}$$
$$\text{USTCON} \in \mathcal{L}$$

- Introdução
- Definições preliminares
- Principais fatos conhecidos
  - O Método da Alcançabilidade
  - O Teorema de Savitch
  - Completude para $\mathcal{NL}, \mathcal{P}, \mathcal{PSPACE}$ e o Teorema de Immerman–Szelepcsényi

Teorema

Para toda função de complexidade $f\colon \mathbb{N} \to \mathbb{N}$ existe uma constante real $c > 1$ tal que

$$\mathcal{NSPACE}(f(n)) \subseteq \mathcal{TIME}(c^{f(n)+\log n}).$$

$M(x)$:

construa a configuração inicial C_0 de N para x;
$S \leftarrow \{C_0\}$; $V \leftarrow \{C_0\}$;
enquanto $S \neq \emptyset$, **faça**:
    remova uma configuração C do conjunto S;
    para toda configuração C' tal que $C \underset{N}{\longrightarrow} C'$ e tal que $C' \notin V$, **faça**:
        se C' é uma configuração final de aceitação, **devolva** 1;
        insira C' em V e em S;
devolva 0.

Corolário

$\mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{NPSPACE} \subseteq \mathcal{EXP} \subseteq \mathcal{NEXP}$

- Introdução
- Definições preliminares
- Principais fatos conhecidos
    - O Método da Alcançabilidade
    - O Teorema de Savitch
    - Completude para $\mathcal{NL}, \mathcal{P}, \mathcal{PSPACE}$ e o Teorema de Immerman–Szelepcsényi

LIMITED_PATH′(G, s, t, k):

```
se s = t ou (st ∈ E(G) e k = 1), devolva 1;
se k ⩽ 1, devolva 0;
para todo u ∈ V(G), faça:
    se LIMITED_PATH′(G, s, u, ⌊k/2⌋) e LIMITED_PATH′(G, u, t, ⌈k/2⌉), então:
        devolva 1;
devolva 0.
```

LIMITED_PATH(G, s, t):

```
devolva LIMITED_PATH(G, s, t, |V(G)| − 1).
```

Complexidade de espaço $O((\log|V(G)|)^2)$

Limited_Path′(G, s, t, k):

se $s = t$ ou ($st \in E(G)$ e $k = 1$), devolva 1;
se $k \leqslant 1$, devolva 0;
para todo $u \in V(G)$, faça:
    se Limited_Path′$(G, s, u, \lfloor k/2 \rfloor)$ e Limited_Path′$(G, u, t, \lceil k/2 \rceil)$, então:
        devolva 1;
devolva 0.

Limited_Path(G, s, t):

devolva Limited_Path$(G, s, t, |V(G)| - 1)$.

Complexidade de espaço $O((\log|V(G)|)^2)$

Teorema de Savitch

Para toda função de complexidade $f\colon \mathbb{N} \to \mathbb{N}$,
$\mathcal{NSPACE}(f(n)) \subseteq \mathcal{SPACE}((f(n))^2)$.

Corolário

$\mathcal{PSPACE} = \mathcal{NPSPACE}$

Teorema de Savitch

Para toda função de complexidade $f\colon \mathbb{N} \to \mathbb{N}$,
$\mathcal{NSPACE}(f(n)) \subseteq \mathcal{SPACE}((f(n))^2)$.

Corolário

$\mathcal{PSPACE} = \mathcal{NPSPACE}$

- Introdução
- Definições preliminares
- Principais fatos conhecidos
  - O Método da Alcançabilidade
  - O Teorema de Savitch
  - Completude para $\mathcal{NL}$, $\mathcal{P}$, $\mathcal{PSPACE}$ e o Teorema de Immerman–Szelepcsényi

- Existe redução logarítmica de qualquer problema em $\mathcal{NL}$ para STCON (STCON é $\mathcal{NL}$-completo)
- Existe redução logarítmica de STCON a $\overline{\text{2SAT}}$ ($\overline{\text{2SAT}}$ é $\mathcal{NL}$-completo, i.e. 2SAT é co$\mathcal{NL}$-completo)

- Existe redução logarítmica de qualquer problema em $\mathcal{NL}$ para STCON (STCON é $\mathcal{NL}$-completo)
- Existe redução logarítmica de STCON a $\overline{\text{2SAT}}$ ($\overline{\text{2SAT}}$ é $\mathcal{NL}$-completo, i.e. 2SAT é co$\mathcal{NL}$-completo)

Teorema de Immerman–Szelepcsényi

Para toda função de complexidade $f\colon \mathbb{N} \to \mathbb{N}$, $\mathcal{NSPACE}(f(n)) = \text{co}\mathcal{NSPACE}(f(n))$.

Corolário

$\mathcal{NL} = \text{co}\mathcal{NL}$

Corolário

2SAT é $\mathcal{NL}$-completo

Teorema de Immerman–Szelepcsényi

Para toda função de complexidade $f\colon \mathbb{N} \to \mathbb{N}$, $\mathcal{NSPACE}(f(n)) = \mathrm{co}\mathcal{NSPACE}(f(n))$.

Corolário

$\mathcal{NL} = \mathrm{co}\mathcal{NL}$

Corolário

2SAT é $\mathcal{NL}$-completo

Teorema de Immerman–Szelepcsényi

Para toda função de complexidade $f\colon \mathbb{N} \to \mathbb{N}$, $\mathcal{NSPACE}(f(n)) = \mathrm{co}\mathcal{NSPACE}(f(n))$.

Corolário

$\mathcal{NL} = \mathrm{co}\mathcal{NL}$

Corolário

2SAT é $\mathcal{NL}$-completo

EVALUATION

Instância: Um conjunto de variáveis finito V, uma fórmula booleana F sobre V na forma normal conjuntiva e uma valoração T para V.

Pergunta: A valoração T para V satisfaz F?

HornSAT

Instância: Um conjunto de variáveis finito V e uma fórmula booleana F sobre V na forma normal conjuntiva com no máximo um literal positivo por cláusula.

Pergunta: F é satisfatível?

EVALUATION e HornSAT são $\mathcal{P}$-completos!

EVALUATION

Instância: Um conjunto de variáveis finito V, uma fórmula booleana F sobre V na forma normal conjuntiva e uma valoração T para V.

Pergunta: A valoração T para V satisfaz F?

HornSAT

Instância: Um conjunto de variáveis finito V e uma fórmula booleana F sobre V na forma normal conjuntiva com no máximo um literal positivo por cláusula.

Pergunta: F é satisfatível?

EVALUATION e HornSAT são $\mathcal{P}$-completos!

EVALUATION

Instância: Um conjunto de variáveis finito V, uma fórmula booleana F sobre V na forma normal conjuntiva e uma valoração T para V.

Pergunta: A valoração T para V satisfaz F?

HornSAT

Instância: Um conjunto de variáveis finito V e uma fórmula booleana F sobre V na forma normal conjuntiva com no máximo um literal positivo por cláusula.

Pergunta: F é satisfatível?

EVALUATION e HornSAT são $\mathcal{P}$-completos!

QBF ou QSAT

Instância: Um conjunto de variáveis finito V, uma fórmula booleana F sobre V na forma normal conjuntiva e uma *quantificação* das variáveis de V, i.e. uma função q de V em $\{\forall, \exists\}$.

Pergunta: A fórmula booleana F quantificada por q é verdadeira?

QSAT é $\mathcal{PSPACE}$-completo!

We all know how we cope with something big and complex; divide and rule [...] The town is made up from neighbourhoods, which are structured by streets, which contain buildings, which are made from walls and floors, that are built from bricks, etc. eventually down to the elementary particles. And we have all our specialists along the line, from the town planner, via the architect to the solid state physicist and further. Because, in a sense, the whole is "bigger" than its parts, the depth of a hierarchical decomposition is some sort of logarithm of the ratio of the "sizes" of the whole and the ultimate smallest parts. From a bit to a few hundred megabytes, from a microsecond to a half an hour of computing confronts us with completely baffling ratio of 10^9! The programmer is in the unique position that his is the only discipline and profession in which such a gigantic ratio, which totally baffles our imagination, has to be bridged by a single technology. He has to be able to think in terms of conceptual hierarchies that are much deeper than a single mind ever needed to face before. [...] By evoking the need for deep conceptual hierarchies, the automatic computer confronts us with a radically new intellectual challenge that has no precedent in our history.

(Edsger W. Dijkstra)