Notas de Aula:
Introdução a Complexidade de Espaço

Leandro M. Zatesko, doutorando*
leandro.zatesko@uffs.edu.br

16 de novembro de 2016

Sumário

1 Introdução **1**
1.1 O Problema da Mochila 1
1.2 Buscas em Grafos 3

2 Definições preliminares **4**
2.1 Máquinas de Turing com múltiplas fitas 4
2.2 Espaço de uma Máquina de Turing e grafo das configurações 5
2.3 Classes de Complexidade de Espaço 5

3 Principais fatos conhecidos **7**
3.1 O Método da Alcançabilidade 7
3.2 O Teorema de Savitch 8
3.3 Completude para $\mathcal{NL}$, $\mathcal{P}$, $\mathcal{PSPACE}$ e o Teorema de Immerman-Szelepcsényi 9

4 Exercícios **10**

1 Introdução

Nestas notas, discorreremos brevemente sobre alguns dos principais tópicos referentes a Complexidade de Espaço e sobre suas principais relações com Complexidade de Tempo. Boa parte do conteúdo abordado pode ser encontrada facilmente na literatura sobre Complexidade Computacional, como em [Pap93]. Tentamos adequar as definições e notações para se manter uma certa uniformidade com as demais notações e definições usadas pelo professor na presente componente curricular. Ainda, ressaltamos que a omissão das demonstrações de modo algum lhes subtrai a importância. Antes, o objetivo do presente documento deve ser entendido como introduzir e motivar o estudante ao assunto e lhe fornecer um roteiro inicial de estudo.

1.1 O Problema da Mochila

Consideremos o clássico *Problema da Mochila*:

Problema 1.1 (KNAPSACK).

*Aula ministrada na componente curricular de Complexidade Computacional (CI739), do Programa de Pós-Graduação em Informática da UFPR, sob a orientação do Prof. Dr. Renato Carmo

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.

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

O Algoritmo 1 é um algoritmo com *backtracking* que temos para KNAPSACK imediatamente da definição do problema. A complexidade de tempo deste algoritmo é evidentemente $O(2^n)$. No entanto, embora este algoritmo não seja eficiente em tempo, é-o em espaço.

BACKTRACKING_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** BACKTRACKING_KNAPSACK$(n-1, C, v_1, \ldots, v_{n-1}, w_1, \ldots, w_{n-1})$;
3 **devolva**

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

Algoritmo 1: Uma solução para KNAPSACK usando *backtracking*

Como discorreremos melhor na Seção 2 (p. 4), quando tratamos de complexidade de espaço, não consideramos o espaço da entrada nem o da saída. No caso específico do Algoritmo 1, podemos considerar que cada chamada recursiva que é aberta necessita apenas dos inteiros n e C e do endereço (um ponteiro) a partir do qual os dados $(v_1, \ldots, v_n, w_1, \ldots, w_n)$ podem ser lidos, de acordo com o modelo computacional utilizado. Formalmente, conforme definiremos melhor também na Seção 2 (p. 4), consideraremos para todos os efeitos que um ponteiro será um inteiro i que designa uma posição na fita da entrada de uma Máquina de Turing, mas também podemos pensar em ponteiros como posições numa memória de acesso randômico de um programa RAM, por exemplo, ou no conceito análogo específico do modelo computacional em questão. Note-se que em qualquer desses modelos faz sentido assumir que o espaço necessário para armazenar um ponteiro é logarítmico no tamanho da entrada (Figura 1), sendo que o que muda de modelo para modelo é o tempo para acessar uma certa posição da entrada a partir de um ponteiro: em Máquinas de Turing, por exemplo, podemos ter de percorrer toda a entrada para acessarmos a posição desejada.

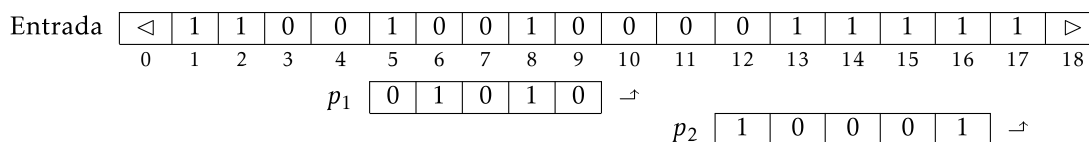

Figura 1: Uma entrada de tamanho $n = 17$ e dois ponteiros p_1 e p_2, que não precisam de mais de $\lfloor \lg 20 \rfloor + 1 = 5$ *bits* cada, para posições na entrada.

Voltando ao Algoritmo 1, cada chamada recursiva consome apenas espaço $O(\log n + \log C)$, para armazenar n, C e o ponteiro para a posição na entrada a partir da qual podem ser lidos os valores $v_1, \ldots, v_n, w_1, \ldots, w_n$. Como haverá não mais que $n+1$ chamadas recursivas abertas ao mesmo tempo, temos que a complexidade de espaço do Algoritmo 1 é $O(n(\log n + \log C))$. Note-se, uma vez que $n(\log n + \log C) = O((n + \log C)^2)$, que esta complexidade é quadrática no tamanho da entrada, o qual podemos considerar $O(n + \log C)$.

O principal motivo da ineficiência do Algoritmo 1 deve-se ao fato de que muitas chamadas recursivas podem ser abertas para uma mesma subinstância. Utilizando Programação Dinâmica, podemos memorizar

os estados já computados para evitar recomputação e assim derivar uma solução mais eficiente em tempo, conforme apresentamos no Algoritmo 2.

```
DP_KNAPSACK(n, C, v_1, ..., v_n, w_1, ..., w_n):
1  para i de 0 até n, faça:
2     para c de 1 até C, faça:
3        se i = 0, S[i][c] ← 0;
4        senão, se w_i > c, S[i][c] ← S[i − 1][c];
5        senão, S[i][c] ← max{S[i − 1][c], v_i + S[i − 1][c − w_i]}.
6  devolva S[n][C].
```

Algoritmo 2: Uma solução para KNAPSACK usando Programação Dinâmica

Embora o Algoritmo 2 seja mais eficiente em tempo que o Algoritmo 1 (p. 2) — sua complexidade de tempo é $O(nC)$, pseudopolinomial no tamanho da entrada —, é mais custoso no que diz respeito ao espaço, pois, além dos espaços necessários para as variáveis i e c (cada um logarítmico no tamanho da entrada), faz-se necessário armazenar os valores dos estados $\mathsf{S}[i][c]$, totalizando uma complexidade de espaço também $O(nC)$. Uma otimização possível decorre de perceber que cada linha da matriz S depende apenas da linha anterior, resultando num algoritmo de complexidade de espaço $O(C)$, mas ainda pseudopolinomial.

Este aparente dilema entre complexidade de tempo e de espaço, tão comum em algoritmos que utilizam Programação Dinâmica e em outras técnicas de *design* de algoritmos, é um dos objetos de estudo da Complexidade Computacional, para o qual, não muito diferentemente do que ocorre com outros tópicos, ainda há muitas questões em aberto.

1.2 Buscas em Grafos

Uma vez que grafos modelam naturalmente as configurações de uma Máquina de Turing Não-determinística e as relações de alcançabilidade entre elas (cf. Seção 3.1, p. 7), buscas em grafos ocupam um lugar muito importante em Complexidade Computacional. O Algoritmo 3 apresenta o pseudocódigo de uma busca genérica para visitar todos os vértices de um grafo dirigido G alcançáveis a partir de um vértice s.

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

Algoritmo 3: Busca genérica em grafos

Se o grafo G é representado computacionalmente por uma lista de adjacências, e se assumimos que as operações de inserir um vértice em X, remover um vértice de X, visitar um vértice e verificar se um vértice está visitado podem ser todas executadas em tempo constante, então, é fácil ver que a complexidade de tempo do Algoritmo 3 é $O(|V(G)| + |E(G)|)$, i.e. linear no tamanho da entrada. Embora *linear* seja o melhor que podemos esperar no que diz respeito ao tempo de qualquer algoritmo que necessite ler toda a entrada, uma complexidade de espaço linear não é muito desejável, pois, na prática, representa que o algoritmo precisa manter alocado num dado momento um volume de memória linearmente proporcional ao tamanho da entrada, ao invés de poder trabalhar unicamente com ponteiros para informações na entrada e outras variáveis de espaço logarítmico.

Por exemplo, consideremos o grafo de todas as configurações possíveis de um Jogo de Damas, o qual possui um tamanho de ordem de grandeza 10^{22}. Fazer uma busca nesse grafo pode ser inviável num

computador convencional que faz cerca de 10^9 operações por segundo, pois $10^{22}/10^9 = 10^{13}$ segundos equivalem a mais de 300 mil anos. Todavia, como a complexidade de tempo é linear, um avanço tecnológico que aumente a ordem de grandeza do número de operações por segundo reduz na mesma proporção o tempo necessário a se esperar. Para o supercomputador mais rápido do mundo, o chinês TaihuLight, que realiza quase 10^{17} operações por segundo, é razoável admitir que uma busca no grafo do Jogo de Damas demoraria um tempo da ordem de 10^5 segundos (pouco mais de 27 horas). Mas não apenas o tempo do nosso algoritmo é linear, como o espaço também o é. A saber, a ordem de grandeza da capacidade de armazenamento total de todos os dispositivos de memória, HDs, mídias removíveis etc. no mundo é estimada em 10^{21}, o que significa que o mundo precisaria ter 10 vezes a capacidade de armazenamento que possui para podermos pensar em executar um algoritmo linear em espaço para um grafo tão grande.

Estas questões concernentes ao espaço de buscas em grafos e a se problemas que usualmente são resolvidos com buscas em grafos podem ser resolvidos em espaço logarítmico são também bastante estudadas pela Complexidade Computacional, tendo, porém, levantado muitas perguntas que ainda carecem de respostas.

2 Definições preliminares

2.1 Máquinas de Turing com múltiplas fitas

Quando tratamos de complexidade de espaço formalmente, em Máquinas de Turing, costumamos trabalhar com múltiplas fitas, a fim de podermos separar o espaço da entrada e o espaço da saída do espaço de fato consumido pela máquina.

Definição 2.1. Uma *Máquina de Turing Determinística com múltiplas fitas* (MTD-k) é uma 5-tupla $M = (k, Q, \Sigma, s, f, \delta)$ em que:

$\underline{k}$ é um inteiro positivo que representa o *número de fitas* de M;

$\underline{Q}$ é um conjunto finito que representa o *conjunto de estados* de M;

$\underline{\Sigma}$ é o *alfabeto* de M, um conjunto finito com pelo menos três elementos, dois dos quais devem ser obrigatoriamente os símbolos especiais $\lhd$ (marcador de início) e $\rhd$ (marcador de fim);

$\underline{s \in Q}$ é o *estado inicial* de M;

$\underline{f \in Q}$ é o *estado final* de M;

$\underline{\delta}$ é a *função de transição* de M, uma função parcial de $Q \times \Sigma^k$ em $Q \times (\Sigma \times \{-1, 0, 1\})^k$ que satisfaz as seguintes propriedades:

(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 $q \neq f$.

Definição 2.2. Uma *Máquina de Turing Não-determinística com múltiplas fitas* (MTND-k) é uma 5-tupla $M = (k, Q, \Sigma, s, f, \Delta)$ em que k, Q, Σ, s e f são como numa MTD-k e:

$\underline{\Delta}$ é a *relação de transição* de M, uma relação binária de $Q \times \Sigma^k$ em $Q \times (\Sigma \times \{-1, 0, 1\})^k$ que satisfaz as seguintes propriedades:

(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 $i \in \{1, \ldots, k\}$, $\sigma'_i = \lhd$ e somente se $\sigma_i = \lhd$ e $d_i = 1$;

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

Note-se que as definições acima convencionam que o conteúdo da primeira fita de uma Máquina de Turing nunca é alterado. Os demais conceitos tradicionais relacionadas a Máquinas de Turing, como *configuração*, *configuração inicial*, *configuração final*, *conteúdo da (i-ésima) fita numa configuração*, *traçado*, *traçado finito*, *traçado infinito*, *aceitação*, *rejeição* etc. podem ser formalizados de modo análogo ao realizado pelo professor em sala de aula (cf. Exercício 1, p. 10), mas convencionando que:

- 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*.

Assim, quando descrevemos o funcionamento de Máquinas de Turing através de pseudocódigos algorítmicos, podemos entender as *variáveis internas* do pseudocódigo (aquelas que não são nem de entrada nem de saída) como cada uma alocada numa fita, desde que as variáveis de entrada sejam somente de leitura e as usadas para a saída sejam somente de escrita.

2.2 Espaço de uma Máquina de Turing e grafo das configurações

Sob as convenções da Seção 2.1 (p. 4), podemos definir nosso conceito de *espaço* de uma Máquina de Turing:

Definição 2.3. Sendo M uma Máquina de Turing com múltiplas fitas, Determinística ou Não-determinística, o *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 — note-se que o traçado é único se M é determinística.

Ainda, podemos definir a *complexidade de espaço* de M em função de um tamanho de entrada n, a qual costumamos denotar por $S_M(n)$, de modo análogo a como fazemos com complexidade de tempo.

Quando é impossível construir um traçado infinito partindo da configuração inicial de uma MTND-k M para qualquer entrada válida x, dizemos que M é *munida de garantia de parada*. Observe-se que, neste caso, o número total de configurações que é possível atingir através de M partindo da configuração inicial para uma dada entrada x é sempre finito. Com isso, podemos construir um conceito muito importante.

Definição 2.4. Sendo M uma MTND-k munida de garantia de parada, x uma entrada válida para M e C_0 a configuração inicial de M para x, o *grafo das configurações* de M para x, denotado por $G(M,x)$, é o grafo dirigido definido por:

(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_1C_2 existe em $G(M,x)$ se e somente se $C_1 \xrightarrow[M]{} C_2$.

2.3 Classes de Complexidade de Espaço

Assim como podemos fazer com complexidade de tempo, definimos $\mathcal{SPACE}(f(n))$ como a classe de todos os problemas computacionais de decisão que podem ser decididos por uma Máquina de Turing Determinística

em complexidade de espaço $O(f(n))$, e definimos $\mathcal{NSPACE}(f(n))$ analogamente para Máquinas de Turing Não-determinísticas. Em particular,

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

Conforme veremos na Seção 3 (p. 7), $\mathcal{PSPACE} = \mathcal{NPSPACE}$,

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

e, embora se acredite que todas as inclusões acima sejam próprias, sabemos apenas, do Teoremas das Hierarquias de Tempo e de Espaço, que $\mathcal{NL} \neq \mathcal{PSPACE}$, que $\mathcal{P} \neq \mathcal{EXP}$ e que $\mathcal{NP} \neq \mathcal{NEXP}$.

Um exemplo de problema que pertence a $\mathcal{NL}$ é o *Problema da Alcançabilidade em Grafos Dirigidos*:

Problema 2.5 (STCON[1]).

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?

Evidentemente STCON $\in \mathcal{P}$, já que pode ser resolvido deterministicamente em tempo polinomial (porém em espaço também polinomial) através de uma busca. Uma solução não-determinística de espaço logarítmico para STCON apresentamos no Algoritmo 4. No pseudocódigo, a instrução **escolha não-deterministicamente** representa que, para qualquer *escolha* que se faça, existe a transição apropriada na Máquina de Turing que implementa o algoritmo.

```
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.
```

Algoritmo 4: Um algoritmo não-determinístico para STCON

Observe-se que a corretude do Algoritmo 4 decorre da observância de que existe um caminho em G de s a t se e somente se existe um passeio com não mais de $|V(G)| - 1$ arestas de s a t. Assim, se (G, s, t) é uma instância positiva de STCON, *alguma* sequência de *escolhas* define um traçado que conduz a correspondente Máquina de Turing da configuração inicial para a instância a uma configuração final de aceitação, ainda que possa haver outros traçados que terminem em rejeição. Em contrapartida, e assimetricamente, se (G, s, t) é uma instância negativa, *toda* sequência de *escolhas* define um traçado que termina em rejeição, o que completa a demonstração da corretude do algoritmo. Observe-se também que, para não cair eventualmente em espaço linear na entrada, o algoritmo não memoriza os vértices já visitados, sendo suas variáveis internas apenas as variáveis i, u e v, todas de espaço logarítmico no tamanho da entrada.

A saber, não se conhece ainda algoritmo *determinístico* de espaço logarítmico para STCON. Na verdade, como veremos na Seção 3 (p. 7), STCON é um problema $\mathcal{NL}$-completo, i.e. a existência de um algoritmo

[1] STCON é sigla de source-target-connectivity.

determinístico de espaço logarítmico para STCON implica que $\mathcal{L} = \mathcal{NL}$, o que se acredita não ser o caso. Em contrapartida, a variante de STCON para grafos não-dirigidos, conhecida por USTCON, está em $\mathcal{L}$, conforme foi provado em 2008 pelo cientista da Computação israelense Omer Reingold [Rei08].

3 Principais fatos conhecidos

3.1 O Método da Alcançabilidade

Um fato intuitivo sobre qualquer modelo razoável de computação é que um algoritmo nunca pode consumir mais espaço do que tempo, o que implica que, para toda função de complexidade $f : \mathbb{N} \to \mathbb{N}$, $\mathcal{TIME}(f(n)) \subseteq \mathcal{SPACE}(f(n))$ e $\mathcal{NTIME}(f(n)) \subseteq \mathcal{NSPACE}(f(n))$, limitando a complexidade de tempo superiormente pela complexidade de espaço. O Teorema 3.1, por outro lado, estabelece a grosso modo um limitante superior para a complexidade de espaço por uma função exponencial na complexidade de tempo.

A técnica usada para demonstrar o Teorema 3.1 é chamada por alguns autores de *Método da Alcançabilidade*, cunhada em diversos trabalhos na década de 1970. A técnica parte da observação de que, num grafo de configurações de uma MTND-k M para uma entrada x de comprimento n, cada configuração pode ser representada computacionalmente em espaço $O(\log n + s_M(x))$, já que: a primeira fita, como é somente de leitura, pode ser representada em espaço $O(\log n)$ apenas pelo índice da posição em que se encontra o cabeçote; as fitas internas podem ser representadas em espaço $O(s_M(x))$ pela própria definição de $s_M(x)$; a última fita não precisa ser representada computacionalmente, já que não é determinante na computação, por ser somente de escrita.

Teorema 3.1. *Para toda função de complexidade* $f : \mathbb{N} \to \mathbb{N}$ *existe uma constante real* $c > 1$ *tal que*

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

Demonstração. Seja N uma Máquina de Turing Não-determinística munida de garantia de parada que satisfaz $S_N(n) = O(f(n))$, e seja M uma Máquina de Turing Determinística que, para uma dada entrada x, realiza uma busca em $G(N,x)$ por um caminho da configuração inicial de N para x até alguma configuração final de aceitação, devolvendo 1 se encontra ou 0 caso contrário. É imediato que para toda entrada válida x, $N(x) = M(x)$, restando-nos mostrar que $T_M(n) = O(c^{f(n)+\log n})$ para alguma constante real $c > 1$. No Algoritmo 5 esclarecemos o funcionamento de M para que possamos prosseguir com a análise.

$M(x)$:
1 construa a configuração inicial C_0 de N para x;
2 $S \leftarrow \{C_0\}$; $V \leftarrow \{C_0\}$;
3 **enquanto** $S \neq \emptyset$, **faça**:
4 remova uma configuração C do conjunto S;
5 **para toda** configuração C' tal que $C \underset{N}{\longrightarrow} C'$ e tal que $C' \notin V$, **faça**:
6 **se** C' é uma configuração final de aceitação, **devolva** 1;
7 insira C' em V e em S;
8 **devolva** 0.

Algoritmo 5

Assim, podemos assumir que:

- as linhas 1 e 2 podem ser executadas em tempo linear em $|``C_0"| = O(\log n + f(n))$, uma vez que cada configuração de $G(N,x)$ pode ser representada em espaço $O(\log n + f(n))$;
- o laço da linha 3 é iterado para toda configuração $C \in V(G(N,x))$ no máximo uma vez, sendo que em cada iteração:

- o teste do laço pode ser executado em tempo constante;
- a linha 4 pode ser executada em tempo linear em $|``C''| = O(\log n + f(n))$;
- o laço da linha 5 é iterado para toda aresta $CC' \in E(G(N,x))$ no máximo uma vez, sendo que em cada iteração:
  - a configuração C' pode ser encontrada mediante uma mera consulta à tabela da relação de transição Δ de N, o que pode ser feito em tempo linear em $|``C'''| = O(\log n + f(n))$, e o teste $C' \notin V$ pode ser executado em tempo $O(|V(G(N,x))|)$;
  - as linhas 6 e 7 podem ser executadas em tempo $O(\log n + f(n))$;

- a linha 8 pode ser executada em tempo constante.

Logo, temos que

$$t_M(x) = O\left(\sum_{C \in V(G(N,x))} \sum_{CC' \in E(G(N,x))} (\log n + f(n))\right),$$

o que implica que $t_M(x) = O(|E(G(N,x))|(\log n + f(n)))$. Ora, podemos mostrar que $|E(G(N,x))|$ e $|V(G(N,x))|$ são assintoticamente equivalentes e que $|V(G(N,x))| = O(nc^{f(n)})$ para uma constante real $c > 1$ (cf. Exercício 2, p. 10). Como para alguma constante real $r > 1$

$$nc^{f(n)}(\log n + f(n))) \leqslant c^{r \log_c n + f(n)} \leqslant (c^r)^{\log_c n + f(n)},$$

temos que $T_M(n) = O((c')^{f(n)+\log n})$ para $c' = c^r > 1$. ♦

Corolário 3.2. $\mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{NPSPACE} \subseteq \mathcal{EXP} \subseteq \mathcal{NEXP}$.

3.2 O Teorema de Savitch

Embora não conheçamos um algoritmo determinístico de espaço logarítmico para STCON, conseguimos construir um algoritmo determinístico de espaço polilogarítmico, como o Algoritmo 6.

```
LIMITED_PATH'(G, s, t, k):
1  se s = t ou (st ∈ E(G) e k = 1), devolva 1;
2  se k ⩽ 1, devolva 0;
3  para todo u ∈ V(G), faça:
4     se LIMITED_PATH'(G, s, u, ⌊k/2⌋) e LIMITED_PATH'(G, u, t, ⌈k/2⌉), então:
5        devolva 1;
6  devolva 0.
LIMITED_PATH(G, s, t):
1  devolva LIMITED_PATH(G, s, t, |V(G)| − 1).
```

Algoritmo 6: Um algoritmo determinístico de espaço $O((\log|V(G)|)^2)$ para STCON

Este algoritmo deve-se ao matemático Walter Savitch e, quando aplicado a grafos de configurações de Máquinas de Turing Não-determinísticas, nos traz o seguinte resultado, conforme observado pelo próprio Savitch:

Teorema 3.3 (Teorema de Savitch [Sav70]). *Para toda função de complexidade* $f : \mathbb{N} \to \mathbb{N}$, $\mathcal{NSPACE}(f(n)) \subseteq \mathcal{SPACE}((f(n))^2)$.

Corolário 3.4. $\mathcal{PSPACE} = \mathcal{NPSPACE}$.

3.3 Completude para $\mathcal{NL}$, $\mathcal{P}$, $\mathcal{PSPACE}$ e o Teorema de Immerman-Szelepcsényi

Assim como construímos o conceito de completude para a classe $\mathcal{NP}$ sob reduções de tempo polinomial, podemos construir também o conceito de completude para a classe $\mathcal{NL}$, desde que sob reduções de espaço logarítmico. Tal como 3SAT é $\mathcal{NP}$-completo, 2SAT é $\mathcal{NL}$-completo. Para mostrarmos a $\mathcal{NL}$-completude de 2SAT, precisamos primeiro mostrar, usando grafos de configurações, que STCON é $\mathcal{NL}$-completo, e que existe uma redução de espaço logarítmico de STCON a $\overline{\text{2SAT}}$, o que não é difícil (cf. Exercício 7, p. 10), sendo $\overline{\text{2SAT}}$ o complementar de 2SAT. Mas isso só nos prova que 2SAT é co$\mathcal{NL}$-completo, não que é $\mathcal{NL}$-completo. No entanto, a demonstração se completa graças ao seguinte resultado surpreendente, provado de modo independente pelos matemáticos Neil Immerman e Róbert Szelepcsényi (cf. Exercício 8, p. 10):

Teorema 3.5 (Teorema de Immerman-Szelepcsényi [Imm88, Sze87])**.** *Para toda função de complexidade* $f : \mathbb{N} \to \mathbb{N}$, $\mathcal{NSPACE}(f(n)) = \text{co}\mathcal{NSPACE}(f(n))$.

Corolário 3.6. $\mathcal{NL} = \text{co}\mathcal{NL}$.

Também podemos, sob reduções de espaço logarítmico, definir o conceito de $\mathcal{P}$-completude. Não se sabe se os problemas $\mathcal{NL}$-completos são também $\mathcal{P}$-completos, assim como também não se sabe se os problemas $\mathcal{P}$-completos estão em $\mathcal{NL}$. Note que se algum problema $\mathcal{NL}$-completo for também $\mathcal{P}$-completo, ou algum problema $\mathcal{P}$-completo for um problema de $\mathcal{NL}$, teremos $\mathcal{NL} = \mathcal{P}$. Destarte, em certo sentido, podemos pensar em problemas $\mathcal{P}$-completos como problemas de $\mathcal{P}$ para os quais se acredita não haver algoritmo de espaço logarítmico, nem mesmo não-determinístico. Outra importante subclasse de $\mathcal{P}$ e superclasse de $\mathcal{NL}$ é a classe $\mathcal{NC}$, a classe dos problemas que podem ser decididos em tempo polilogarítmico paralelizando-se a computação num número polinomial de processadores. Como também se acredita que $\mathcal{NC} \neq \mathcal{P}$, podemos pensar em problemas $\mathcal{P}$-completos também como problemas de $\mathcal{P}$ com uma estrutura inerentemente sequencial, difíceis de serem paralelizados. Exemplos de problemas $\mathcal{P}$-completos são o *Problema do Valor Booleano* (EVALUATION) e o *Problema da Satisfatibilidade de Cláusulas de Horn* (HornSAT) (cf. Exercício 9, p. 10).

Problema 3.7 (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?

Problema 3.8 (HornSAT)**.**

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

Pergunta: F é satisfatível?

No que diz respeito à completude de problemas para a classe $\mathcal{PSPACE}$, podemos ser mais generosos e permitir reduções de tempo polinomial. Um exemplos clássico de problema $\mathcal{PSPACE}$-completo é o *Problema da Fórmula Booleana Quantificada* (QBF ou QSAT) (cf. Exercício 11, p. 10):

Problema 3.9 (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?

Perceba-se que a restrição de QSAT em que $q(x) = \exists$ para todo $x \in V$ é exatamente o problema SAT.

4 Exercícios

Exercício 1. Formalize, para Máquinas de Turing Determinísticas e Não-determinísticas com múltiplas fitas, os conceitos de *configuração*, *configuração inicial*, *configuração final*, *conteúdo da (i-ésima) fita numa configuração*, *traçado*, *traçado finito*, *traçado infinito*, *aceitação*, *rejeição* e todos os demais que julgar necessários, de acordo com as convenções estabelecidas na Seção 2.1 (p. 4).

Exercício 2. Mostre que para toda Máquina de Turing Não-determinística M munida de garantia de parada e toda entrada válida x para M, o número de arestas no grafo $G(M,x)$ é linear no número de vértices, o qual por sua vez é assintoticamente limitado superiormente por $|x|c^{t_M(x)}$, sendo $c > 1$ uma constante real.

Exercício 3. Mostre o Corolário 3.2 (p. 8).

Exercício 4. Mostre que o Algoritmo 6 (p. 8) de fato está correto e de fato possui complexidade de espaço $O((\log|V(G)|)^2)$.

Exercício 5. Mostre o Teorema de Savitch (Teorema 3.3, p. 8).

Exercício 6. Pesquise sobre o Teorema da Hierarquia de Espaço (e sobre o da Hierarquia de Tempo também) e mostre que se $\mathcal{SPACE}((\log n)^2) \subseteq \mathcal{P}$, então $\mathcal{L} \neq \mathcal{P}$.

Exercício 7. Mostre que todo problema em $\mathcal{NL}$ pode ser reduzido em espaço logarítmico para STCON. Mostre também uma redução de espaço logarítmico de STCON para $\overline{\text{2SAT}}$.

Exercício 8. Pesquise sobre a demonstração do Teorema de Immerman-Szelepcsényi (Teorema 3.5, p. 9).

Exercício 9. Mostre que EVALUATION e HornSAT são de fato $\mathcal{P}$-completos.

Exercício 10. Embora seja imediato compor reduções de tempo polinomial, não é tão imediato assim compor reduções de espaço logarítmico, ferramenta crucial para o desenvolvimento dos conceitos de $\mathcal{NL}$-completude e $\mathcal{P}$-completude. Mostre que se existe uma redução de espaço logarítmico de um problema P_1 a um problema P_2, e se existe redução de espaço logarítmico de um problema P_2 a um problema P_3, então existe redução de espaço logarítmico de P_1 a P_3.

Exercício 11. Pesquise sobre a demonstração de que QSAT é de fato $\mathcal{PSPACE}$-completo.

Referências

[Imm88] N. Immerman: *Nondeterministic space is closed under complementation*. SIAM J. Comput., 17:935–938, 1988.

[Pap93] C. H. Papadimitriou: *Computational Complexity*. Addison-Wesley, 1ª edição, 1993.

[Rei08] O. Reingold: *Undirected connectivity in log-space*. J. ACM, 55(4)(17), 2008.

[Sav70] W. J. Savitch: *Relationships between nondeterministic and deterministic tape complexities*. J. Comput. System Sci., 4(2):177–192, 1970.

[Sze87] R. Szelepcsényi: *The method of forcing for nondeterministic automata*. Bulletin of the EATCS, 33:96–100, 1987.