UDESC

Teoria da Computação

Prof. Diego Buchinger
diego.buchinger@outlook.com
diego.buchinger@udesc.br

Complexidade de Espaço

Introdução

O espaço requerido para computar também é importante

- Podemos classificar problemas de acordo com a sua complexidade de espaço
- A complexidade de espaço compartilha muitas das características da complexidade de tempo
- É usual utilizar a notação assintótica e o modelo de máquinas de Turing para medir o custo de armazenamento de algoritmos

A complexidade de espaço *f(n)* é o número máximo de células de fita que M visita sobre qualquer entrada de comprimento *n*. Dizemos que uma MT determinística M que para sobre todas as entradas roda [executa] em espaço *f(n)*.

Introdução

Para uma MT não-determinística M na qual todos os ramos param sobre todas as entradas, sua complexidade de espaço *f(n)* é o número máximo de células de fita que M visita sobre qualquer ramo de computação, para qualquer entrada de tamanho *n*.

Assim, são determinadas duas classes de complexidade de espaço:

- $SPACE(\,f(n)\,) =$ { L | L é uma linguagem decidida por uma MT determinística de espaço O(f(n)) }
- $NSPACE(\,f(n)\,) =$ { L | L é uma linguagem decidida por uma MT não determinística de espaço O(f(n)) }

Exemplo SPACE

Com relação a tempo, foi mostrado que SAT ∈ NP-Completo. Contudo, com relação à tempo, SAT pode ser resolvido com um algoritmo de espaço linear!

- **Espaço** parece ser “mais poderoso” do que **tempo**, porque o primeiro pode ser reusado, ao passo que o segundo não.

M1 = “Sobre a entrada $\langle \phi \rangle$, onde ϕ é uma fórmula booleana:
1. Para cada atribuição de verdade às variáveis $x_1, \ldots, x_m$ de ϕ:
1.1. Calcula o valor de ϕ naquela atribuição de verdade.
2. Se ϕ alguma vez teve valor 1 (V), *aceite*; se não, *rejeite*.”

$$\text{SAT} \in SPACE(\,O(n)\,)$$

Exemplo NSPACE

Testar se um AFN aceita todas as cadeias

$$TODAS_{AFN} = \{\langle A \rangle \mid A \text{ é } um\ AFN\ e\ L(A) = \Sigma^*\}$$

- Existe um algoritmo de **espaço linear** não-determinístico que decide o complemento dessa linguagem: $\overline{TODAS_{AFN}}$.

(usar não-determinismo para adivinhar uma cadeia que é rejeitada e espaço linear para guardar em quais estados o AFN poderia estar em cada momento)

N= "Sobre a entrada $\langle M \rangle$, onde M é um AFN:

1. Coloque um marcador sobre o estado inicial do AFN.

2. Repita 2^q vezes, onde *q* é o número de estados de *M*:

2.1. Escolha não deterministicamente um símbolo de entrada e modifique as posições dos marcadores sobre os estados de *M* para simular a leitura daquele símbolo.

3. Se um marcador, em algum momento, for colocado sobre um estado de aceitação, *rejeite*; caso contrário, *aceite*."

Exemplo NSPACE

Testar se um AFN aceita todas as cadeias

$$TODAS_{AFN} = \{\langle A \rangle \mid A \text{ é } um\ AFN\ e\ L(A) = \Sigma^*\}$$

- Existe um algoritmo de **espaço linear** não-determinístico que decide o complemento dessa linguagem: $\overline{TODAS_{AFN}}$.

(usar não-determinismo para adivinhar uma cadeia que é rejeitada e espaço linear para guardar em quais estados o AFN poderia estar em cada momento)

N= “Sobre a entrada $\langle M \rangle$, onde M é um AFN:

1. Coloque um m

2. Repita 2^q vez

2.1. Escolha nã

e modifique

de *M* para si

3. Se um marc

um estado de

NOTA: se M aceita quaisquer cadeias, ele deve aceitar uma de comprimento no máximo 2^q; isso porque, em uma cadeia mais longa, a localização dos marcadores apenas se repetiriam.

Teorema de Savitch

Máquinas determinísticas parecem necessitar de um tempo exponencial para simular máquinas não-determinísticas, contudo, necessitam uma quantidade surpreendentemente pequena de espaço para realizar esta simulação!

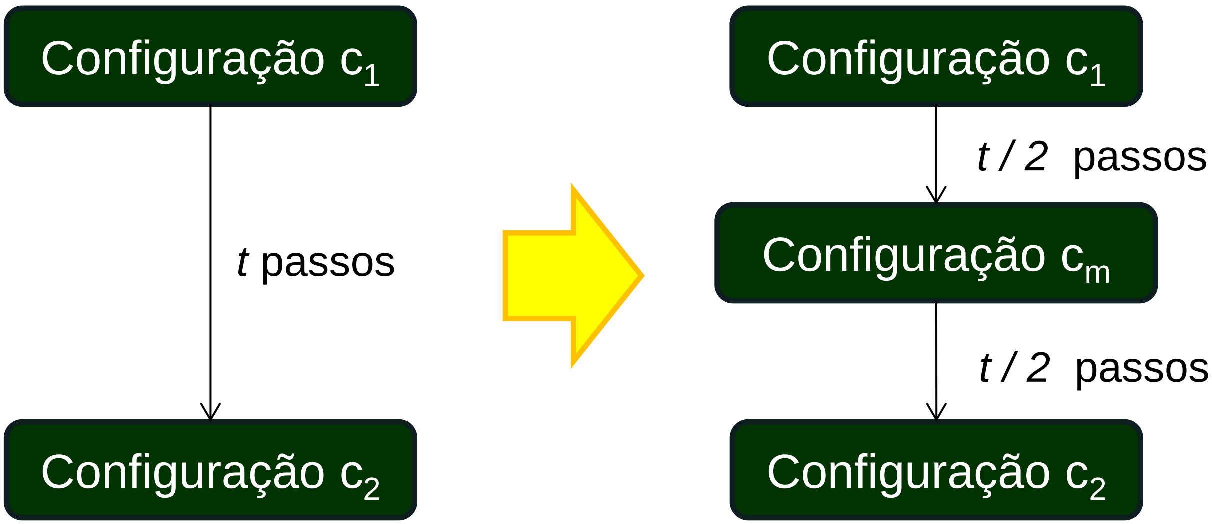
Teorema de Savitch
Para qualquer função $f: N \rightarrow R^{+}$, onde $f(n) \geq n$,
$NSPACE\big(f(n)\big) \subseteq \mathrm{SPACE}(f^{2}(n))$

- **Abordagem 1**: simulação de cada ramo de computação exige espaço exponencial.

- **Abordagem 2**: res. baseada no problema da originabilidade – dadas duas configurações, c_1 e c_2, e um número t, decidir se é possível partir de c_1 e chegar em c_2 com t passos.

Teorema de Savitch

- **Abordagem 2**: problema da originabilidade



Aplicamos um algoritmo recursivo que busca em cada passo uma configuração intermediária c_m. A reutilização do espaço para cada um dos dois testes recursivos permite uma economia significativa de espaço!

Teorema de Savitch

- **Abordagem 2**: problema da originabilidade

ORIGINAR = "Sobre a entrada $c_1, c_2\ e\ t$:

1. Se $t = 1$, então teste diretamente se $c_1 = c_2$ ou se c_1 origina c_2 em um passo conforme as regras de M. Aceite se um dos testes for bem-sucedido; rejeite se ambos falharem.

2. Se $t > 1$, então para cada configuração c_m de M sobre *w* usando espaço *f*(*n*):

2.1. Rode $ORIGINAR(\ c_1,\ c_{m_,}\ \frac{t}{2})$

2.2. Rode $ORIGINAR(\ c_{m_,}\ \ c_2,\ \frac{t}{2})$

2.3. Se ambos os passos 3 e 4 aceitam, então *aceite*.

3. Se não houve aceitação, *rejeite*.

N = "Sobre a entrada w:

1. Dê como saída o resultado $ORIGINAR(\ c_{inicial},\ \ c_{aceita}\ , 2^{df(n)})$.

d é uma constante que garante que M não tenha mais do que $2^{df(n)}$ configurações

Teorema de Savitch

- **Abordagem 2**: problema da originabilidade

- Este algoritmo precisa de espaço para armazenar a pilha de recursão, sendo que cada nível usa *O(f(n))* para armazenar uma configuração.
- A profundidade da recursão é *log t*, onde *t* é o tempo máximo que a máquina não-determinista pode usar. Para uma MT determinística temos $t = 2^{O(f(n))} \rightarrow \log t = O(f(n))$

$$O(f(n)) \times O(f(n)) = O(f^2(n))$$

Classes de Problemas

Por analogia as classes P e NP, temos as classes:

Definição: PSPACE é a classe de linguagens que são decidíveis em espaço polinomial sobre uma MT determinística.

$$PSPACE = \bigcup_k SPACE(n^k)$$

Definição: NPSPACE é a classe de linguagens que são decidíveis em espaço polinomial sobre uma MT não-determinística.

$$NPSPACE = \bigcup_k NSPACE(n^k)$$

Classes de Problemas

Por analogia as classes P e NP, temos as classes:

Definição: PSPACE é a classe de linguagens que são decidíveis em espaço ... a.

Considerando o **teorema de Savitch**, podemos afirmar algo sobre a relação entre **PSPACE** e **NPSPACE**?

$$PSPACE \subseteq NPSPACE$$
$$PSPACE \supseteq NPSPACE$$
$$PSPACE = NPSPACE$$

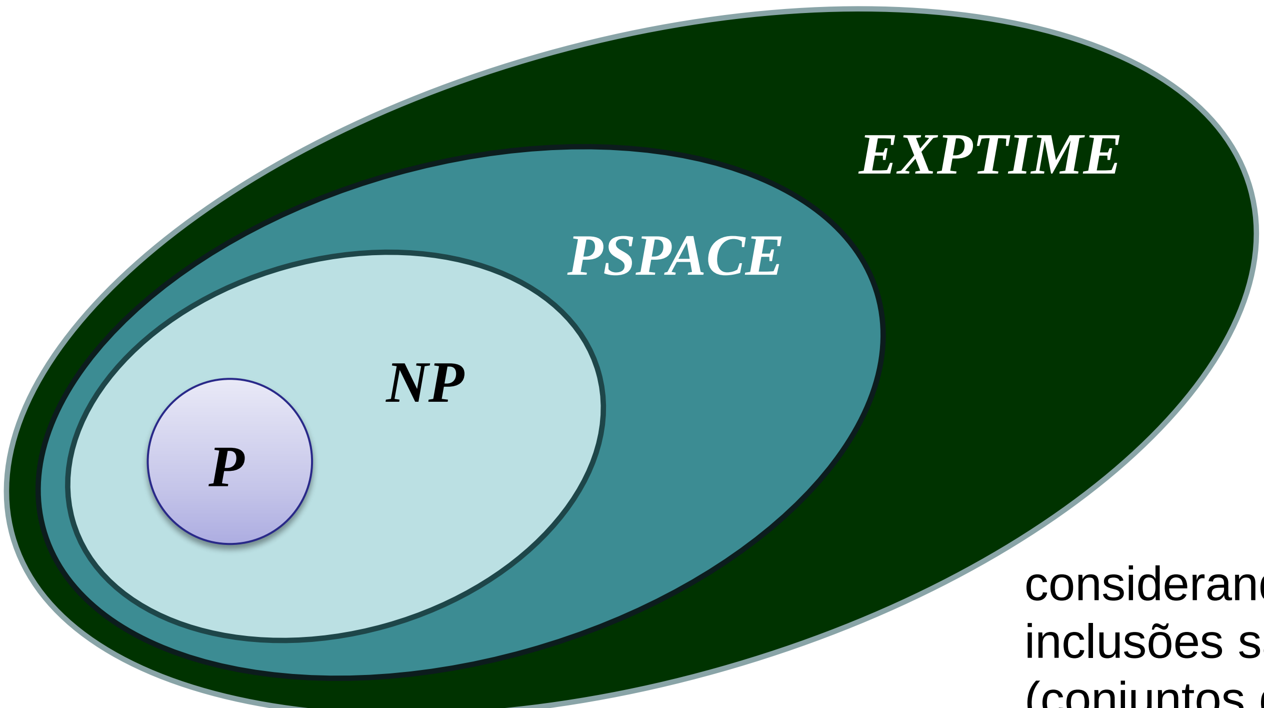
Definição: ... são decidíveis em espaço polinomial sobre uma MT não-determinística.

$$NPSPACE = \bigcup_k NSPACE(n^k)$$

Classes de Problemas

Relacionando classes de problemas em relação a tempo e espaço:

$$P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME$$



considerando que todas as inclusões são próprias (conjuntos diferentes)

PSPACE-Completude

De modo análogo à NP-Completude existe PSPACE-Completude

Uma linguagem B é **PSPACE-completa** se ela:

1. B está em PSPACE;
2. Toda linguagem A em PSPACE é redutível em tempo polinomial a B (A $\leq_p$B).

Se somente a condição (2) é satisfeita, então dizemos que B é **PSPACE-difícil**

OBS: as reduções devem ser feitas em tempo polinomial e não em espaço polinomial. O objetivo da redução é transformar um problema em outro de maneira fácil e rápida (e ao fazer isso, o espaço também só pode crescer de forma polinomial)

PSPACE-Completude

Exemplos de problemas PSPACE-completo:

- TQBF – *True Quantified Boolean Formula*

  Fórmulas Booleanas Completamente Quantificada Verdadeira

  Similar ao SAT mas usando os quantificadores $\forall$ *e* $\exists$

- Estratégias Vencedoras para Jogos:

  Computar se existe uma sequência de passos que sempre vence um jogo (jogo da velha, damas, xadrez)

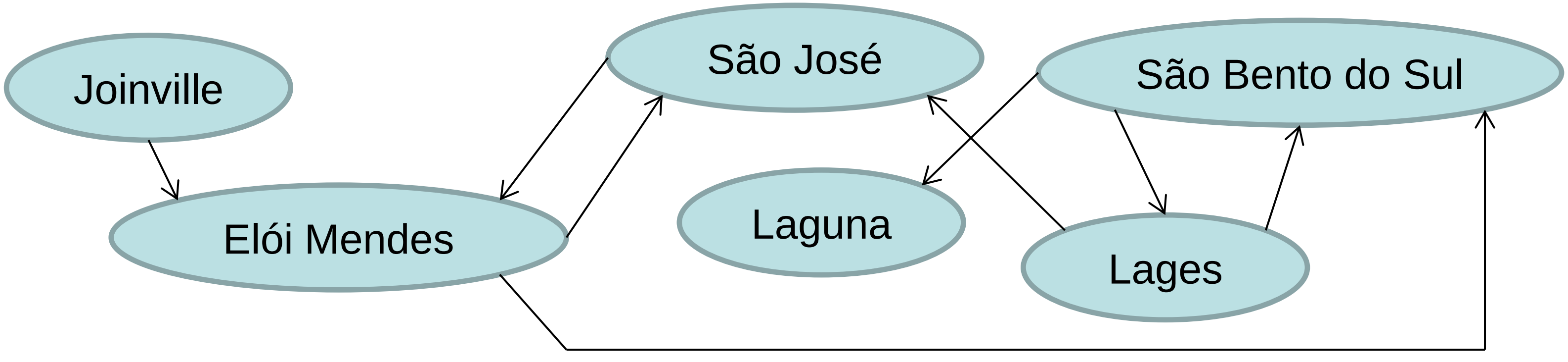
  - *JOGO-da-FÓRMULA:* idem ao TQBF onde um jogador escolhe o valor para uma variável por vez:

$$\exists x_1 \forall x_2 \exists x_3 \left[(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3}) \right]$$

PSPACE-Completude

Exemplos de problemas PSPACE-completo:

- Estratégias Vencedoras para Jogos:
  - *Geografia Generalizada:* um jogador escolhe um nome de cidade por vez, mas o nome da próxima cidade deve iniciar com a última letra da última cidade dita. Um mesmo nome não pode ser repetido. Quem não souber continuar na sua vez, perde.

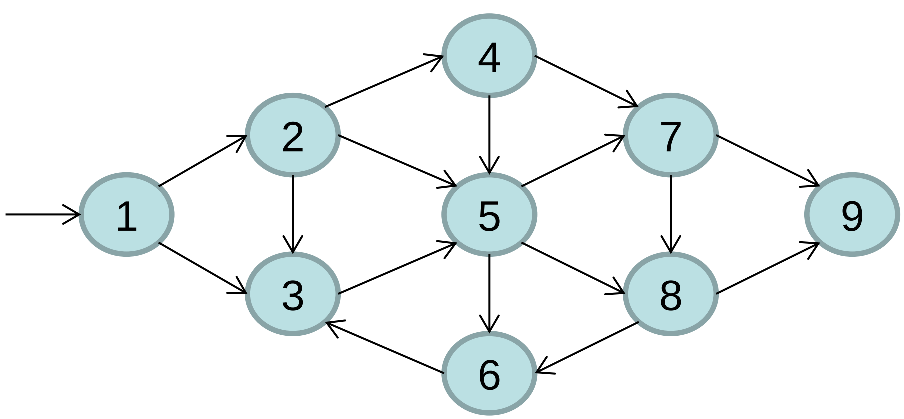


PSPACE-Completude

Exemplos de problemas PSPACE-completo:

- Estratégias Vencedoras para Jogos:
  - *Geografia Generalizada:*

    Existe alguma estratégia vencedora para o Jogador 1, se existem apenas dois jogadores jogando?



Classes L e NL

- Qualquer algoritmo que precisa realizar a leitura da entrada possui uma complexidade de **tempo** no mínimo linear [$\Omega(n)$]
- Com relação a complexidade de **espaço**, esse requisito muitas vezes não é necessário, sendo utilizado uma quantidade *sublinear* de espaço.
- Este tipo de problema exige o uso de MT de duas fitas:

Memória somente leitura (ex: CD-ROM / ROM)

0	1	1	1	0	0	1	0	1	1

Espaço adicional utilizado

0	0

Classes L e NL

Definição: L é a classe de linguagens que são decidíveis em espaço logarítmico em uma MT determinística.

$$L = SPACE(\log n)$$

Definição: NL é a classe de linguagens que são decidíveis em espaço logarítmico em uma MT não-determinística.

$$NL = NSPACE(\log n)$$

Classes L e NL

Exemplo L: $A = \{0^k 1^k \mid k \geq 0\}$

Já vimos diversas soluções para este problema (M_1, M_2, M_3)

[Lembre-se que a fita 1 é somente leitura!]

M_3 = "Sobre a cadeia de entrada *w*:

1. Faça uma varredura na fita e ***rejeite*** se algum 0 for encontrado à direita de algum 1.
2. Faça uma varredura nos 0s sobre a fita 1 até o primeiro 1. Ao mesmo tempo, copie os 0s para a fita 2.
3. Faça uma varredura nos 1s sobre a fita 1 até o final da entrada. Para cada 1 lido sobre a fita 1, corte um 0 sobre a fita 2. Se todos os 0s estiverem cortados antes que todos os 1s sejam lidos, ***rejeite***.
4. Se todos os 0s tiverem sido cortados, ***aceite***. Se restar algum 0, ***rejeite***.

Usa espaço linear - [$O(n)$]

Classes L e NL

Exemplo L: $A = \{0^k 1^k \mid k \geq 0\}$

Já vimos diversas soluções para este problema (M_1, M_2, M_3)

[Lembre-se que a fita 1 é somente leitura!]

M_3 = "Sobre a cadeia de entrada *w*:

1. Faça uma varredura na fita e <u>*rejeite*</u> se algum 0 for encontrado à direita de algum 1.
2. Faça uma varredura nos 0s sobre a fita 1 até o primeiro 1. Ao mesmo tempo, conte (somando), em binário, o número de 0s encontrados.
3. Faça uma varredura nos 1s sobre a fita 1 até o final da entrada. Para cada 1 lido sobre a fita 1, subtraia um na fita 2. Se a fita 2 possuir apenas um zero e for lido um 1 na fita 1, *rejeite*.
4. Se a fita 2 tiver apenas um zero, <u>*aceite*</u>, caso contrário, <u>*rejeite*</u>.

Usa espaço linear - [$O(\log n)$]

Classes L e NL

Exemplo NL: $CAM = \{\langle G, s, t\rangle \mid G \text{ é } um\ grafo\ direcionado\ que\ tem\ um\ caminho\ direcionado\ de\ s\ para\ t\}$

*Obs: queremos saber apenas **se existe** um caminho

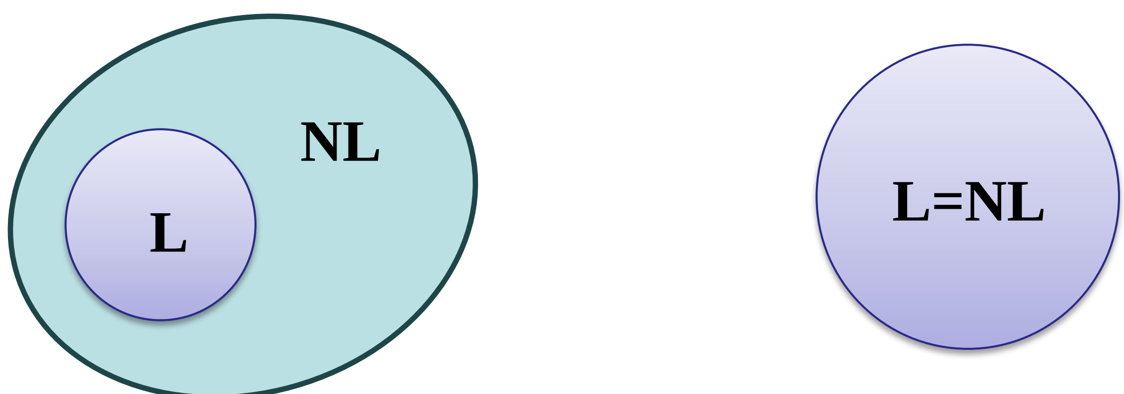
- CAM está em P
- Não se sabe se CAM está em L
- Mas sabe-se que CAM está em NL
  - A MTN M adivinha não deterministicamente os nós que fazem parte de um caminho de *s* para *t*
  - A máquina registra somente a posição do nó atual a cada passo na fita de trabalho
  - Se M atinge *t*, então *aceite*; caso rode por *m* passos e não chegue em *t*, então *rejeite*;

Classes L e NL

O teorema de Savitch afirma que podemos converter MTs não-determinísticas para MTs determinísticas aumentando a complexidade de espaço de forma quadrática, desde que $f(n) \geq n$

Isso também é verdade se $f(n) \geq \log n$ sendo que precisamos armazenar no máximo $\log\left(n2^{Of(n)}\right)$

L vs. NL



Similar à questão P vs. NP

NL-Completude

O problema $CAM \in L$?

Não sabemos provar que sim nem que não.

Novamente buscamos agrupar os problemas mais difíceis em NL e nos focar apenas nestes para demonstrar alguma relação.

- Os problemas mais difíceis em NL são denominados de problemas NL-Completos
- Todo problema em NL pode ser reduzido aos problemas NL-Completos com *espaço log* (não mais em tempo polinomial!)

$A \leq_L B \quad \equiv$ A é redutível em espaço log à B

NL-Completude

Definição: uma linguagem B é NL-Completa se:

1. $B \in NL$
2. Toda A em NL é redutível em espaço log à B.

Se qualquer linguagem NL-Completa está em L, então L = NL

- CAM é NL-Completa

UDESC