Complexidade de Espaço

Prof^a Jerusa Marchi

Departamento de Informática e Estatística

Universidade Federal de Santa Catarina

e-mail: jerusa.marchi@ufsc.br

Complexidade Computacional

- Até o momento apenas a complexidade de *tempo* (medida em número de passos) foi considerada
- Pode-se considerar outra classe de problemas que inclui todos os problemas $\mathcal{NP}$ e parece incluir outros mais
  - Complexidade de Espaço

Complexidade de Espaço

- Seja uma M MT determinística que para sobre todas as entradas. A complexidade de espaço de M é a função $f : \mathcal{N} \mapsto \mathcal{N}$ onde $f(n)$ é o número máximo de células de fita que M visita sobre qualquer entrada de comprimento n.
- Se M é uma MT não determinística na qual todos os ramos param sobre todas as entradas, define-se sua complexidade de espaço $f(n)$ como o número máximo de células de fita que M visita sobre qualquer ramo de sua computação para qualquer entrada de comprimento n

Complexidade de Espaço

- Classe $SPACE(f(n))$
  - Classe de problemas (ou linguagens) que são decidíveis por Máquinas de Turing Deteminísticas de espaço $O(f(n))$
- Classe $NSPACE(f(n))$
  - Classe de problemas (ou linguagens) que são decidíveis por Máquinas de Turing Não-Deteminísticas de espaço $O(f(n))$
- Poderia-se distinguir então duas classes a $\mathcal{P}SPACE$ e $\mathcal{NP}SPACE$?

Complexidade de Espaço

- Teorema de Savitch
  - Um dos resultados mais antigos da complexidade de espaço
  - Para qualquer função $f : \mathcal{N} \rightarrow \mathcal{R}^{+}$, onde $f(n) \geq n$, $NSPACE(f(n)) \subseteq SPACE(f^2(n))$.
    - **Ideia da prova:** Precisamos simular uma MTND de espaço $f(n)$ de forma determinística. Uma abordagen ingênua é tentar simular todos os ramos da computação da MTND, um por um. A simulação precisaria guardar qual ramo ela está explorando em um dado momento de modo que seja capaz de passar para o próximo (backtrack). Um ramo que usa espaço $f(n)$ pode rodar por $2^{O(f(n))}$ passos, pois cada passo pode ser uma escolha não determinística. Explorar os ramos sequencialmente demandaria registrar todas as escolhas utilizadas em um ramo específico de modo a ser capaz de encontrar o próximo ramo. O que pode vir a usar espaço $2^{O(f(n))}$.

Teorema de Savitch

- **Ideia da prova (cont.):** Em vez disso, podemos ver o problema como um problema mais geral: recebemos duas configurações da MTND, c_1 e c_2, juntamente com um número t, e devemos testar se a MTND pode ir de c_1 a c_2 dentro de t passos (problema da originabilidade). Tomando c_1 (conf. inicial e c_2 (conf. de aceitação) e t como sendo o número máximo de passos que a MTND pode tomar, podemos determinar se a máquina aceita sua entrada. Para resolver o problema da originabilidade, montamos um algoritmo recursivo que busca por uma configuração intermediária c_m dentro de $t/2$ passos, testando recursivamente (i) c_1 pode chegar a c_m em $t/2$ passos e (ii) c_m pode chegar a c_2 em $t/2$ passos. A reutilização do espaço para cada um dos dois testes permite economia de espaço.

Teorema de Savitch

- **Ideia da prova (cont.):** Contudo, é necessário armazenar a pilha de recursão. Cada nível da recursão utiliza $O(f(n))$ para armazenar uma configuração. A profundidade da recursão é $log\ t$, onde t é o tempo máximo que a máquina não determinística pode usar qualquer ramo. Temos $t = 2^{O(f(n))}$, portanto $log\ t = O(f(n))$. Logo a simulação determinística usa espaço $O(f^2(n))$.

Teorema de Savitch

- **Prova:** Seja N uma MTND que decide uma linguagem A em espaço $f(n)$. Construímos uma MTD M que decide A. A MT M utiliza o procedimento `podeoriginar`, que testa se uma das configurações de N pode originar outra dentro de um número especificado de passos.
  Seja w a cadeia de entrada de N. Para configurações c_1 e c_2 de N sobre w, e um inteiro t, $\texttt{podeoriginar}(c_1, c_2, t)$ dá como saída *aceite* se N pode ir da configuração c_1 para a configuração c_2 em t ou menos passos. Se não, `podeoriginar` dá como saída *rejeita*.

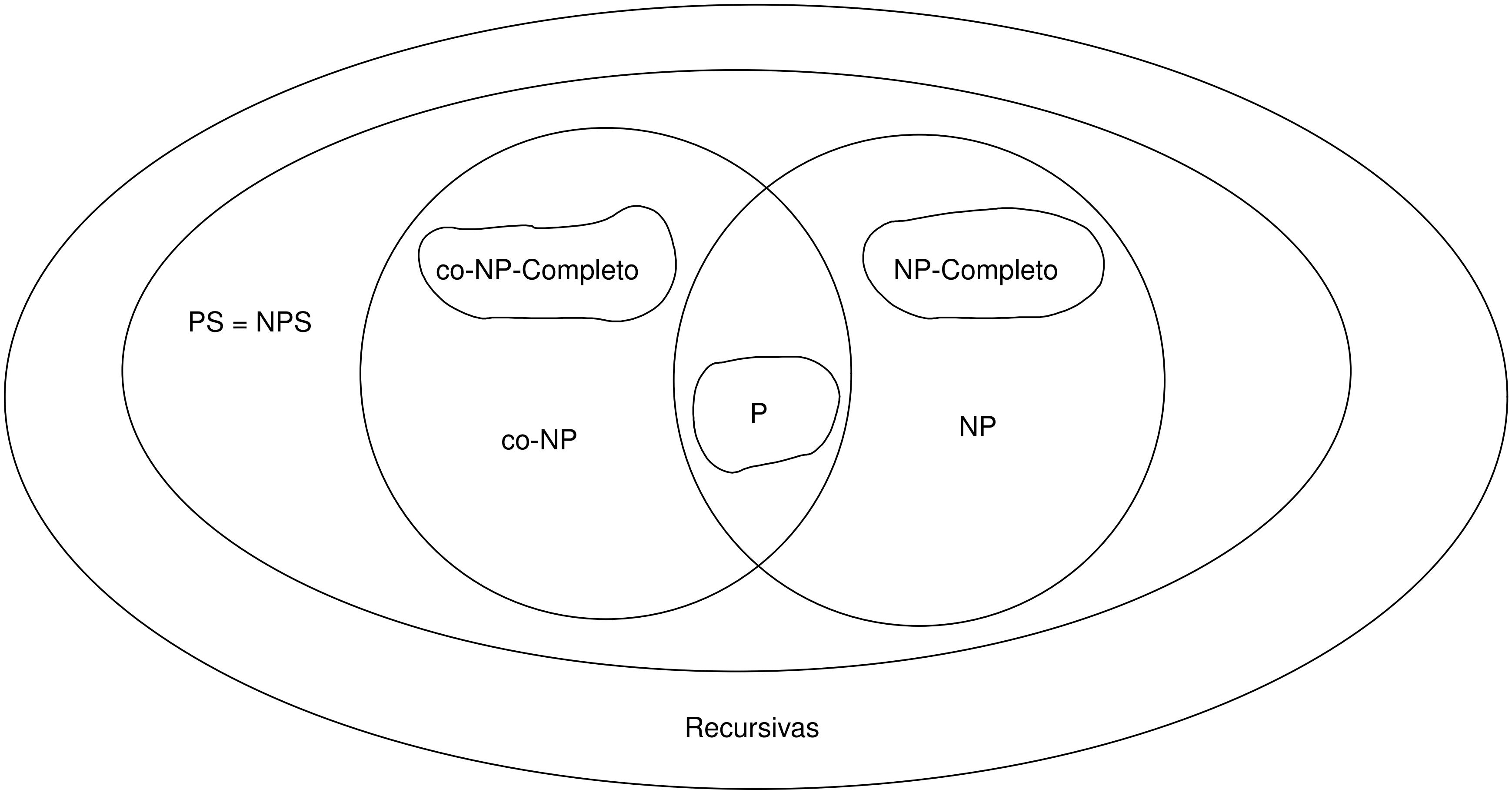
Teorema de Savitch

- **Prova:** Por conveniência assumimos t como uma potência de 2:
  `podeoriginar` = 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 de computação de N. *Aceite* se sim.*Rejeite* se ambos os testes falharem.
  2. Se $t > 1$ então para cada configuração c_m de N sobre w usando espaço $f(n)$:
     (a) Rode `podeoriginar`$(c_1, c_m, t/2)$
     (b) Rode `podeoriginar`$(c_m, c_2, t/2)$
     (c) Se ambos os passos aceitarem, *aceite*
  3. Se não houver aceitação, *rejeite*.

Teorema de Savitch

- **Prova:** M opera da seguinte forma:
  M = sobre a entrada w:
  1. Dê como saída o resultado de `podeoriginar`$(c_1, c_2, 2^{df(n))})$
     (para uma constante d escolhida de forma que N não tenha mais do que $2^{df(n)}$) configurações.

Classes de Complexidade



PSPACE Completude

- Uma Linguagem B é PSPACE-Completa se ela satisfaz duas condições:
  1. B está em PSPACE, e
  2. Toda A em PSPACE é redutível em tempo polinomial a B.
- Se B meramente satisfaz a condição 2, dizemos que ela é PSPACE-Difícil.

PSPACE Completude

- Exemplo: Problema TQBF (true quantified Boolean formula)
  - Generalização do problema SAT, envolvendo quantificadores universais ($\forall$ e $\exists$)
  - $\forall x \exists y (y > x)$
  - $\exists y \forall x (y > x)$
  - Foma normal prenex - todos os quantificadores aparecem no início da fórmula.
  - Quando cada variável aparece dentro do escopo de um quantificador a fórmula é dita completamente quantificada

PSPACE Completude

- Exemplo: Problema TQBF - Verificar se uma fórmula booleana completamente quantificada é satisfeita
  - Inicia-se demonstrando que é possível atribuir valores às variáveis e calcular, recursivamente, a veracidade da fórmula
  - Para mostrar que toda linguagem A se reduz a TQBF em tempo polinomial, supõe-se uma MT limitada por espaço polinomial para A e então apresenta-se uma redução em tempo polinomial que mapeia uma cadeia para uma fórmula booleana quantificada ϕ que codifica uma simulação da MT sobre aquela entrada. A fórmula é verdadeira sse a máquina aceita. (usa a mesma técnica usada no teorema de Savitch).

As Classes L e NL

- Classes de complexidade de espaço sublineares
  - L é a classe de linguagens que são decidíveis em espaço logaritmico em uma MT determinística.
  - NL é a classe de linguangens que são decidíveis em espaço logaritmico em uma MTND.
- Para computar em tempo logaritmico (digamos $lg\ n$), o tempo é insuficiente para ler a entrada inteira.
- Para computar em espaço logaritmico, o tempo é suficiente para ler a entrada, mas o espaço é insuficiente para armazenar a entrada

As Classes L e NL

- Para considerar essa situação é necessário modificar a MT. A MT possui agora 2 fitas:
  - uma de entrada de somente leitura - cuja cabeça detecta os símbolos mas não os modifica
  - uma de leitura e escrita - funciona da forma usual (fita de trabalho)
- Somente as células visitadas na fita de trabalho contam para a complexidade de espaço

As Classes L e NL

- Exemplo: $L = \{0^k 1^k \mid k \geq 0\}$
  - A fita de entrada é de somente leitura. Os $0's$ são varridos e um contador binário incrementa o número lido na fita de trabalho. A representação em binário faz com que o espaço para o contador seja logaritmico. Portanto o algoritmo roda em $O(log\ n)$.