Tratabilidade

Prof^a Jerusa Marchi

Departamento de Informática e Estatística

Universidade Federal de Santa Catarina

e-mail: jerusa.marchi@ufsc.br

Teoria da Computação

- Não Computável
- Computável
  - Indecidível
  - Decidível
    - Intratável
    - Tratável

      Dentre os problemas decidíveis, quais podem ser computados por Máquinas de Turing que rodam em *tempo polinomial* em relação ao tamanho da entrada

Tratabilidade

- Considerações
  - Os problemas solúveis em tempo polinomial em um computador típico são exatamente os mesmos problemas solúveis em tempo polinomial em uma Máquina de Turing
  - A separação entre problemas que podem ser solucionados em tempo polinomial daqueles que requerem tempo exponencial ou mais é fundamental
    - problemas práticos identificados como requerendo tempo polinomial são quase sempre solúveis em um montante de tempo tolerável
    - Apenas instâncias pequenas dos problemas que requerem tempo exponencial podem ser resolvidas em tempo razoável

Análise de Algoritmos

- Como verificar o desempenho?
  - A posteriori: envolve a execução propriamente dita do algoritmo, medindo-se o tempo de execução.
    - é dependente da arquitetura da máquina, da linguagem de programação, do código gerado pelo compilador
    - as medidas são feitas utilizando benchmarks e posteriormente condensadas através de análise estatística dos dados
  - A priori: feita de forma analítica (Análise de Algoritmos)
    - número de instruções executadas (passos do algoritmo)
    - entrada (há uma relação direta entre entrada e desempenho - algoritmos de ordenação, p.e)

Análise de Algoritmos

- Por que verificar o desempenho?
  - Dado um algoritmo A com complexidade quadrática. Como ele irá se comportar para uma entrada de tamanho 10^9?
  - Dados dois algoritmos A e B, que resolvem o mesmo problema computacional, como podemos decidir qual é o mais eficiente?
  - Dado uma algoritmo A, que resolve um problema Π, A é o melhor algoritmo para solucionar Π?
    - Um algoritmo é **ótimo** quando seu custo de execução é igual ao menor custo possível para solucionar um problema

Análise de Algoritmos

tamanho da entrada	Tempo de processamento	
	Alg. 1	Alg.2
n	1s	1s
$2n$	2s	4s
$3n$	3s	9s
$\ldots$	$\ldots$	$\ldots$
$10n$	10s	100s $(1.6$min$)$
$100n$	100s $(1.6$min$)$	10^4s $(2,8$h$)$
	linear	n^2

Análise de Algoritmos

- Análise de Algoritmos é o estudo teórico do desempenho de programas e do uso de recursos computacionais
  - **Tempo**
  - Espaço
- Tempo de execução de um algoritmo visto como uma função ($T(n)$) do tamanho n da sua entrada
- Além disso, não estamos interessados em analisar detalhadamente o algoritmo, queremos sim uma **estimativa** do esforço de computação quando a entrada tende ao infinito
  - Consideramos tempos constantes
  - Desconsideramos constantes multiplicadoras e termos de mais baixa ordem
- Essa taxa de crescimento do tempo de execução em função do tamanho da entrada constitui a **Complexidade do algoritmo**

Análise de Algoritmos

- A diferença no desempenho de algoritmos é muito mais drástica do que a diferença de processamento de computadores
- Suponha dois computadores
  - C_1: 10 bilhões de instruções/seg (10^{10})
  - C_2: 10 milhões de instruções/seg (10^7)
- Suponha dois algoritmos
  - Algoritmo A_1 - $T_{A_1}(n) = 2n^2$
  - Algoritmo A_2 - $T_{A_2}(n) = 50n \lg n$
- Suponha uma entrada $n = 10$ milhões de números (10^7)

Análise de Algoritmos

- C_1 executando A_1:

$$Tempo = \frac{T_{A_1}(n)}{velocidade} = \frac{2(10^7)^2 \text{instruções}}{10^{10} \text{inst/seg}} = 20.000\text{s} = 5,5\text{h}$$

- C_2 executando A_2:

$$Tempo = \frac{T_{A_2}(n)}{velocidade} = \frac{50 \times 10^7 \text{lg} 10^7 \text{instruções}}{10^7 \text{inst/seg}} = 1162\text{s} = 19,36\text{min}$$

Análise de Algoritmos

- A análise pode ser feita considerando:
  - Melhor Caso
    - Poucos algoritmos funcionam "sempre"no melhor caso
  - Caso Médio
    - depende de uma distribuição de probabilidade (difícil de se obter!)
  - Pior Caso
    - Mais fácil de se obter

Análise de Algoritmos

- Encontrar uma função que descreve o comportamento do algoritmo: Medida de Complexidade
- A complexidade de tempo não representa tempo diretamente, mas o número de vezes que determinadas operações consideradas relevantes são executadas
  - Depende do tamanho da entrada (elementos de um vetor, total de bits utilizados para representar um número, número de vértices e arestas em um grafo,...)

Análise de Algoritmos

- Estamos interessados no comportamento do algoritmo para entradas grandes: comportamento assintótico
  - Notação Assintótica

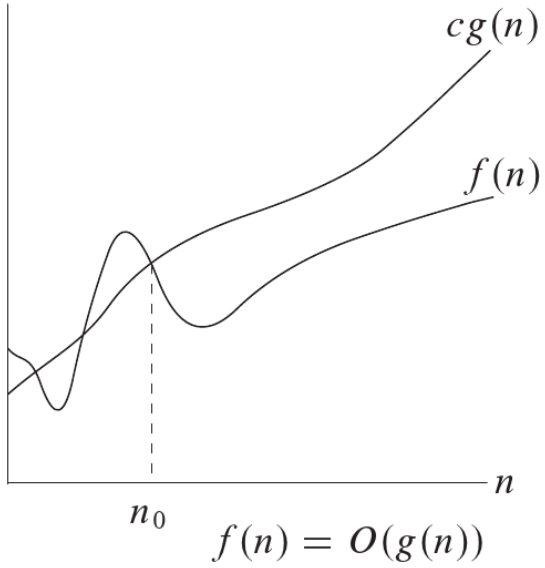
Notação Assintótica

- As notações que utilizaremos para descrever o tempo de execução assintótico de um algoritmo são definidas em termos das funções, cujos domínios são conjuntos dos números naturais $\mathbb{N} = \{0, 1, 2, \cdots\}$
- Contudo
  - algumas vezes são feitos abusos de notação (dominío dos reais, p.ex.)
  - estas notações são aplicadas à funções, portanto algumas abstrações precisam ser feitas

Notação O

- Denota *limite assintótico superior*
- Consiste em determinar uma função que, dada uma constante multiplicativa, domina assintoticamente um conjunto de funções
- Dada uma função $g(n)$, denota-se $O(g(n))$ o conjunto de funções

$$O(g(n)) = \{f(n) : \text{ existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leq f(n) \leq cg(n) \text{ para todo } n \geq n_0\}$$



Notação O

- Apesar de $O(g(n))$ ser um conjunto, tipicamente escrevemos $f(n) = O(g(n))$ ao invés de $f(n) \in O(g(n))$
- Exemplo 1: $f(n) = (n+1)^2$ é $O(n^2)$

n	$f(n)$	$g(n)$	$2g(n)$	$4g(n)$
0	1	0	0	0
1	4	1	2	$\mathbf{4}$
2	9	4	8	$\mathbf{16}$
3	16	9	$\mathbf{18}$	$\mathbf{36}$
4	25	16	$\mathbf{32}$	$\mathbf{64}$

- $f(n) = O(n^2)$ com $c = 2$ e $n_0 = 3$
- $f(n) = O(n^2)$ com $c = 4$ e $n_0 = 1$

Notação O

- Exemplo 2: $f(n) = 3n^3 + 2n^2 + n$ é $O(n^3)$

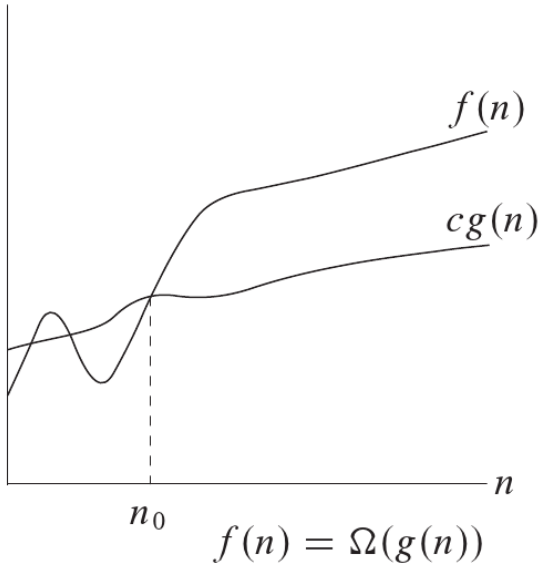
n	$f(n)$	$g(n)$	$4g(n)$	$5g(n)$
0	0	0	0	0
1	6	1	4	5
2	34	8	32	**40**
3	102	27	**108**	**135**
4	228	64	**256**	**320**

- $f(n) = O(n^3)$ com $c = 4$ e $n_0 = 3$
- $f(n) = O(n^2)$ com $c = 5$ e $n_0 = 2$

Notação Ω

- Denota *limite assintótico inferior*
- Dada uma função $g(n)$, denota-se $\Omega(g(n))$ o conjunto de funções

$$\Omega(g(n)) = \{f(n) : \text{ existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leq cg(n) \leq f(n) \text{ para todo } n \geq n_0\}$$



$f(n) = \Omega(g(n))$

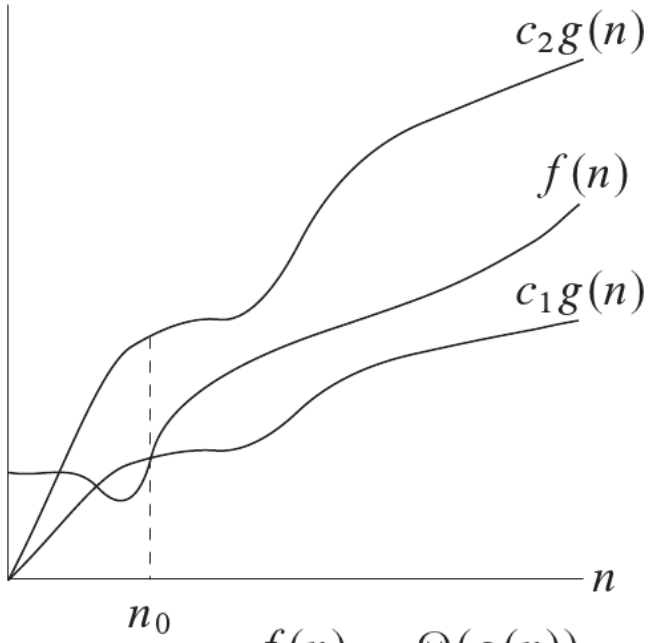
Notação Ω

- Exemplo: $f(n) = 2n^2 + 4n + 5$ é $\Omega(n^2)$
- Basta assumir $c = 2$ e $n_0 = 0$

Notação Θ

- Denota *limite assintótico firme*
- Dada uma função $g(n)$, denota-se $\Theta(g(n))$ o conjunto de funções

$$\Theta(g(n)) = \{f(n) : \text{ existem constantes positivas } c_1, c_2 \text{ e } n_0 \text{ tais que } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ para todo } n \geq n_0\}$$



$f(n) = \Theta(g(n))$

Notação Θ

- Exemplo 1: seja $f(n) = \frac{3n^2}{2} - 2n$, $f(n)$ é $\Theta(n^2)$, pois:

n	$f(n)$	$g(n)$	$2g(n)$
2	2	4	8
4	16	16	32
6	42	36	72
8	80	64	128
10	130	100	200

- $f(n) = \Theta(n^2)$ com $c_1 = 1$, $c_2 = 2$ e $n_0 = 4$

Notação Assintótica

- Observe que para $f(n) = 32n^2 + 17n + 32$
  - $f(n)$ é $O(n^2)$, $O(n^3)$, $\Omega(n^2)$, $\Omega(n)$ e $\Theta(n^2)$
  - $f(n)$ NÃO é $O(n)$, $\Omega(n^3)$, $\Theta(n)$, ou $\Theta(n^3)$

- Teorema: Para quaisquer duas funções $f(n)$ e $g(n)$, tem-se $f(n) = \Theta(g(n))$ se e somente se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$.

Notação Assintótica

- Há ainda outras duas notações para denotar limites que não são *assintoticamente justos*
  - $2n^2 = O(n^2)$ é assintoticamente justo, enquanto que $2n = O(n^2)$ não é
  - $2n = \Omega(n)$ é assintoticamente justo, enquanto que $2n^2 = \Omega(n)$ não é

Notação o

- Define-se $o(g(n))$ como o conjunto

$$o(g(n)) = \{f(n) : \text{ para qualquer constante positiva } c > 0,$$
$$\text{existe uma constante } n_0 > 0 \text{ tal que}$$
$$0 \leq f(n) < cg(n) \text{ para todo } n \geq n_0\}$$

- Exemplo: $2n = o(n^2)$ mas $2n^2 \neq o(n^2)$

Notação ω

- Define-se $\omega(g(n))$ como o conjunto

$$\omega(g(n)) = \{f(n) : \text{ para qualquer constante positiva } c > 0,$$
$$\text{existe uma constante } n_0 > 0 \text{ tal que}$$
$$0 \leq cg(n) < f(n) \text{ para todo } n \geq n_0\}$$

- Exemplo: $\frac{n^2}{2} = \omega(n)$ mas $\frac{n^2}{2} \neq \omega(n^2)$

Propriedades

- Transitividade
  - Se $f = O(g)$ e $g = O(h)$, então $f = O(h)$
  - Se $f = \Omega(g)$ e $g = \Omega(h)$, então $f = \Omega(h)$
  - Se $f = \Theta(g)$ e $g = \Theta(h)$, então $f = \Theta(h)$
- Reflexividade
  - $f = \Theta(f)$
  - $f = O(f)$
  - $f = \Omega(f)$
- Simetria
  - $f = \Theta(g)$ sse $g = \Theta(f)$
- Simetria Transposta
  - $f = O(g)$ sse $g = \Omega(f)$
  - $f = o(g)$ sse $g = \omega(f)$

Comportamento Assintótico

- Complexidade constante - $f(n) = O(1)$
  - O uso destes algoritmos independe do tamanho da entrada e as instruções são executadas um número fixo de vezes
- Complexidade logarítmica - $f(n) = O(log\ n)$
  - Ocorre tipicamente em algoritmos que resolvem um problema transformando-o em problemas menores
  - Ex. $n = 1000, lg\ n = 10$
- Complexidade linear - $f(n) = O(n)$
  - Em geral, um pequeno trabalho é realizado sobre cada um dos elementos de entrada

Comportamento Assintótico

- Complexidade $n\ log\ n$ - $f(n) = O(n\ log\ n)$
  - Complexidade típica de algoritmos que quebram o problema em problemas menores, resolvendo-os e depois juntando as soluções
- Complexidade quadrática - $f(n) = O(n^2)$
  - típica de algoritmos que processa elementos aos pares (2 loops aninhados)
- Complexidade cúbica - $f(n) = O(n^3)$
  - Usados apenas para resolver pequenas instâncias (3 loops aninhados)

Comportamento Assintótico

- Complexidade exponencial - $f(n) = O(2^n)$
  - Busca exaustiva (força bruta). Não úteis na prática
- Complexidade fatorial - $f(n) = O(n!)$
  - Também dito exponencial apesar de crescer mais rapidamente

Complexidade de Tempo

função	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$	$n = 60$
n	.00001s	.00002s	.00003s	.00004s	.00005s	.00006s
n^2	.0001s	.0004s	.0009s	.0016s	.0025s	.0036s
n^3	.001s	.008s	.027s	.064s	.125s	.216s
n^5	.1s	3.2s	24.3s	1.7 min	5.2 min	13.0 min
2^n	.001s	1.0s	17.9 min	12.7 dias	35.7 anos	366 séculos
3^n	.059s	58 min	6.5 anos	3855 séculos	2×10^8 séculos	1.3×10^{13} séculos

Complexidade de Tempo

- A separação entre *algoritmos* **eficientes** *em tempo polinomial* e *algoritmos* **ineficientes** *em tempo exponencial* admite exceções quando as instâncias do problema de interesse são limitadas
  - No quadro, instâncias onde $n \leq 20$ são executadas mais rápido por algoritmos em tempo exponencial (2^n) do que por um algoritmo em tempo polinomial (n^5)
- Porém, apenas alguns poucos algoritmos exponenciais são úteis na prática

Analisando Algoritmos

- Considere a linguagem $L = \{0^n 1^n \mid n \geq 0\}$
- Seja MT M_1 que decide L, M_1 opera da seguinte forma:

$M_1 =$ sobre a entrada w :

1.Faça uma varredura na fita e **rejeite** se algum 0 for encontrado à direita de algum 1

2.Repita

2.1. Faça uma varredura na fita cortanto um único 0 e um único 1

3. Se sobrarem 0's ou 1's rejeite. Caso contrário, aceite.

- Qual é a complexidade de tempo deste algoritmo?

Analisando Algoritmos

$M_1 =$	sobre a entrada w :
$2(n+1)$	1.Faça uma varredura na fita e **rejeite** se algum 0 for encontrado à direita de algum 1
$n/2$	2.Repita
$2(n/2)$	2.1. Faça uma varredura na fita cortanto um único 0 e um único 1
$(n/2)+1$	3. Se sobrarem 0's ou 1's rejeite. Caso contrário, aceite.

- Complexidade: $T(n)n^2/2 + 5n/2 + 3 = O(n^2)$

Analisando Algoritmos

- Considere agora M_2 que opera da seguinte forma:

$M_2 =$	sobre a entrada w :
$2(n+1) = O(n)$	1.Faça uma varredura na fita e **rejeite** se algum 0
	for encontrado à direita de algum 1
$lognO(lg\ n)$	2.Repita enquanto houverem 0's e 1's na fita
$2(n+1) = O(n)$	2.1. Faça uma varredura na fita, verificando se o número to
	de símbolos remanecentes é par ou ímpar. Se ímpar, **rejeit**
$2(n+1) = O(n)$	2.2. Faça uma varredura cortando alternadamente um 0 sim
	outro não. Faça o mesmo com 1's.
$n+1 = O(n)$	2.3. Se nenhum 0 e nenhum 1 permanecerem, **aceite**.
	Caso contrário, **rejeite**

- Complexidade: $O(n) + O(n\ log\ n) = O(n\ log\ n)$

Analisando Algoritmos

- Considere agora a seguinte MT multifitas M_3 que decide $L = \{0^n 1^n \mid n \geq 0\}$

  $M_3 =$ sobre a 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 0's da 1^a fita, copiando-os para a 2^a fita
  3. Faça uma varredura nos 1s até o final da entrada, cortando o 0 correspondente na 2^a fita
  4. Se sobrarem 0's ou 1's rejeite. Caso contrário, aceite.

- Complexidade : $O(n)$

Complexidade de Tempo

- Seja $T(n)$ uma função, onde $T(n) \geq n$. Então, toda MT multifita de tempo $T(n)$ tem uma MT fita única equivalente de tempo $O(T^2(n))$.
- **Prova**: Seja M uma MT de k fitas que roda em tempo $T(n)$. A máquina S opera simulando M copiando inicialmente os conteúdos das k fitas. Para cada passo de M, S faz duas passagens sobre a porção ativa de sua fita (uma para leitura da posição dos cabeçotes e outra para atualização). O comprimento da porção ativa da fita de S determina quanto tempo S leva para varrê-la. Este comprimento é dado pela soma dos comprimentos das porções ativas das k fitas de M. Cada uma tem no máximo $T(n)$ células. Assim, S usa $k * T(n)$ passos para varrer a sua entrada, ou seja $O(T(n))$ passos. Para cada passo de M, S realiza duas varreduras e até k deslocamentos para a direita. Cada uma usa $O(T(n))$; logo o tempo total para S simular um dos passos de M é $O(T(n))$. Como M roda em $T(n)$ passos, S toma $T(n) * O(T(n)) = O(T^2(n))$ passos.

Complexidade de Tempo

- Seja $T(n)$ uma função, onde $T(n) \geq n$. Então, para toda MT não determinística de uma única fita de tempo $T(n)$ exitem uma MT determinística de fita única equivalente de tempo $2^{O(T(n))}$.
- **Prova**: Seja N uma MT não determinística rodando em tempo $T(n)$. Uma máquina não determinística decide quando algum de seus ramos aceita ou todos os seus ramos falham. Sobre uma entrada de tamanho n cada ramo da árvore de computação de N tem um comprimento no máximo $T(n)$. Todo nó na árvore pode ter no máximo b filhos, onde b é o número máximo de escolhas dado pela função de transição de N. Portanto o número máximo de folhas da árvore é $b^{T(n)}$. Para simular N, a máquina determinística multifitas D, examina os caminhos da árvore de N, sempre partindo da raíz. Ou seja, ela examina, no máximo $b^{T(n)}$ caminhos onde cada caminho tem tamanho $T(n)$. Assim, o tempo de execucão de D é de $O(T(n)b^{T(n)}) = 2^{O(T(n))}$.

Problemas de Decisão

- Um problema é dito tratável se é decidível por Máquinas de Turing em **tempo polinomial** em relação ao tamanho da entrada
  - Como vimos, MT decidem linguagens respondendo **sim** ou **não**
  - Vimos também que todo problema computacional pode ser visto como um problema de linguagem
  - É preciso então, dado um problema, modificar a sua formulação para se torne um **Problema de Decisão**

Problema de Decisão

- Um problema de decisão é um problema em que pode-se responder **sim** ou **não**
- Problemas computacionais podem ser modificados para problemas de decisão

Problema de Decisão

- Exemplos:
  - SAT - dado um conjunto de cláusulas $W = \{C_1, ..., C_n\}$ onde C_i é uma disjunção de literais, há alguma atribuição de valores verdade que tornem a fórmula verdadeira?
  - Problema da alcaçabilidade - Dado um grafo orientado $G \subseteq V \times V$, onde $V = \{v_1, ...v_n\}$ é um conjunto finito, e dois vértices v_i e $v_j \in V$, existe um caminho de v_i para v_j?

Problema de Decisão

- Exemplos:
  - Caminho Hamiltoniano - Dado um grafo G, existe algum caminho que passa por cada um dos vértices de G exatamente uma vez?
  - Problema do Caixeiro Viajante (e problemas de otimização em geral) - fornece-se um limite para a função custo
    - Dado um grafo G com $n \geq 2$ vértices e uma matriz $n \times n$ de adjacências representando a distância entre cidades e um inteiro L, há alguma permutação π de $\{1, 2, ..., n\}$ tal que custo(π)$\leq L$?

Classes de Complexidade

- Classe $\mathcal{P}$
  - Uma máquina de Turing $M = (K, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ é dita **polinomialmente limitada** se há um polinômio $p(n)$ tal que, para qualquer entrada x, não há configuração C tal que $(q_0, \triangleright x) \vdash_M^{p(|x|)+1} C$
    - Ou seja, a máquina sempre pára após $p(n)$ passos, onde n é o comprimento da cadeia de entrada

Classe $\mathcal{P}$

- Exemplos
  - Todas as Linguagens Regulares
  - Todas as Linguagens Livres de Contexto
  - 2SAT

Classe $\mathcal{P}$

- Exemplos - Árvore Geradora Mínima
  - Deseja-se conectar um conjunto de n localidades através de uma rede de comunicação. Qual é a forma mais econômica (menor quantidade de cabos) para conectar as n localidades?
    - Modelar as n localidades como um grafo não direcionado $G = (V, E)$, onde os vértices representam as localidades e as arestas assumem um valor $p(u, v)$ que representa o custo (quantidade de cabos) para realizar a conexão
    - Encontrar um subconjunto $T \subseteq E$ que conecta todos os vértices de G e cujo custo total

$$p(T) = \sum_{(u,v) \in T} p(u, v)$$

seja mínimo

Classe $\mathcal{P}$

- Exemplos - Árvore Geradora Mínima
  - A definição do subconjunto de arestas permite a obtenção um grafo $G' = (V, T)$
  - G' será acíclico e T forma uma árvore chamada de *árvore geradora* de G.
  - Algoritmo simples para resolução: Algoritmo de Kruskal
    - Utiliza uma estratégia gulosa
      - Uma solução ótima globalmente pode ser obtida fazendo escolhas localmente ótimas
      - Essa abordagem é utilizada em áreas como Projeto e Análise de Algoritmos e Inteligência Artificial para resolver problemas complexos de forma aproximada

Classe $\mathcal{P}$

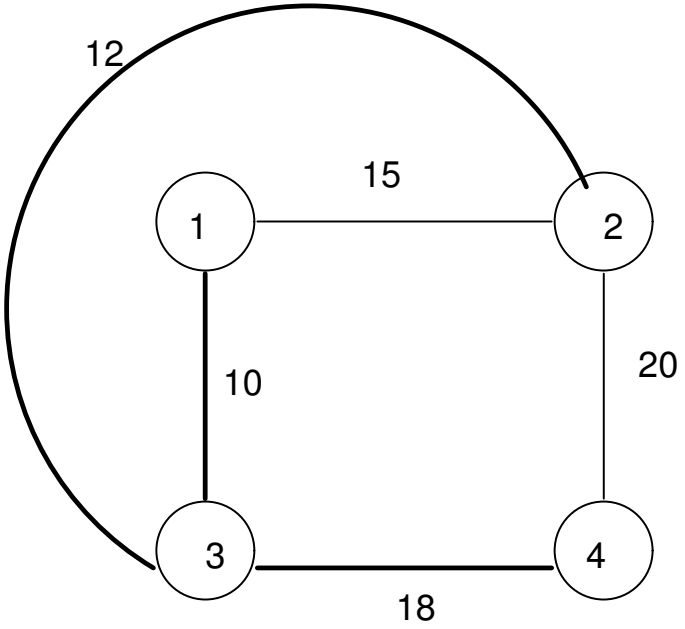
- Exemplos - Árvore Geradora Mínima (Algoritmo de Kruskal)
  - Ideia: Ver cada vértice como uma árvore e ligar árvores com arestas leves, até que haja somente uma árvore (a árvore geradora mínima)
    - Uma aresta é uma **aresta leve** se seu peso for menor do que o de qualquer outra aresta

Classe $\mathcal{P}$

- Exemplos - Árvore Geradora Mínima (Algoritmo de Kruskal)
  - Algoritmo
    - Ordena as arestas por peso
    - Seleciona sempre a aresta com menor peso
    - O conjunto A formado pelas arestas escolhidas é uma floresta
    - A idéia do algoritmo é escolher arestas leves que conectem árvores na floresta
    - Inicialmente, cada vértice v é uma árvore
  - Complexidade do algoritmo $O(E\ lg\ V)$ em uma implementação eficiente. (Heap Binário)

Classe $\mathcal{P}$

- Exemplos - Árvore Geradora Mínima (Algoritmo de Kruskal)



Classe $\mathcal{P}$

- Exemplos - Árvore Geradora Mínima (Algoritmo de Kruskal)
  - Para tornar o problema da AGM um problema de decisão:
    - Dado um grafo G e um limite W, G possui uma árvore geradora de peso W ou menor?
  - Transformando as entradas para computação em MT
    - $\Sigma = \{0, 1, (,), , \}$
    - Atribua valores de 1 a m para os vértices
    - Inicie a codificação com o valor de m em binário e o limite W em binário, separados por vírgula
    - Se há uma aresta que conecta dois nodos v_i e v_j com peso w, coloque (i, j, w) em binário no código

Classe $\mathcal{P}$

- Exemplos - Árvore Geradora Mínima (Algoritmo de Kruskal)
  - Considerando W = 40

$$100, 101000(1, 10, 1111)(1, 11, 1010)(10, 11, 1100)$$
$$(10, 100, 10100)(11, 100, 10010)$$

  - Em uma Máquina de Turing multifitas
    - uma fita para armazenar a entrada
    - uma fita para armazenar os nós e a qual árvore eles pertencem (tabela)
    - uma fita para armazenar a soma dos pesos já computados
    - uma fita para armazenar a última aresta leve encontrada
    - uma fita para armazenar os nós i e j conectados pela aresta leve
    - uma fita para auxiliar na soma

Classe $\mathcal{P}$

- Exemplos - Árvore Geradora Mínima (Algoritmo de Kruskal)
  - Funcionamento da MT multifitas
    - Copia w para a fita 1
    - Constrói a tabela de nós e árvores na fita 2
    - Busca a aresta leve e grava na fita 4, colocando os vértices i e j na fita 5
    - Soma fita 4 com a fita 3, armazenando na fita 6 (compara o valor com o valor de W na fita 1. Se maior, pára e rejeita).
    - Copia fita 6 para fita 3
    - atualiza fita 2 fazendo com que todos os vértices pertencentes a árvore a qual i pertence, pertençam a árvore a qual o vértice j pertence. Se todos pertencerem a uma mesma árvore, pára e aceita.

Classe $\mathcal{P}$

- Exemplos - Árvore Geradora Mínima (Algoritmo de Kruskal)
  - A execução desta codificação em uma máquina de Turing multifitas toma $O(n^2)$
  - Tudo o que uma MT multifitas processa em s passos, uma MT fita única processa em s^2 passos
  - Logo, uma MT fita única processa instâncias de problema da AGM em $O((n^2)^2) = O(n^4)$, ou seja em tempo polinomial.

Classes de Complexidade

- Classe $\mathcal{P}$ - Propriedade
  - A classe $\mathcal{P}$ é fechada com relação à operação de complementação
    - Se uma linguagem L é decidível por uma máquina de Turing M polinomialmente limitada, então seu complemento é decidido pela versão de M que inverte as saídas q_{accept} e q_{reject}
    - Obviamente o limite polinomial não é afetado por essa inversão

Classes de Complexidade

- Uma classe fundamental para o estudo da tratabilidade de problemas é aquela formada por problemas cuja solução pode ser obtida em tempo polinomial por uma MT não determinística
  - Será que tudo o que pode ser computado em tempo polinomial por uma MT não determinística pode ser computado em tempo polinomial por uma MT determinística, talvez com um polinômio de mais alta ordem?
    - Outras extensões de MT comprovadamente podem
- Esta classe é de especial interesse por conter uma infinidade de problemas práticos para os quais, até o momento não se encontrou solução em tempo polinomial

Classes de Complexidade

- Classe $\mathcal{NP}$
  - Uma máquina de Turing não Determinística $M = (K, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ é dita **polinomialmente limitada** se há um polinômio $p(n)$ tal que, para qualquer entrada x, não há configuração C tal que $(q_0, \triangleright x) \vdash_M^{p(|x|)+1} C$
    - Ou seja, nenhuma computação nessa máquina dura mais do que um número polinomial de passos

Classe $\mathcal{NP}$

- Recordando: Uma MT não determinística decide uma linguagem L se:
  - Para cada sentença de entrada que não pertença à linguagem L, todas as computações possíveis da MT devem rejeitar tal entrada
  - Para cada sentença de entrada pertencente a L, exige-se que haja pelo menos uma computação que aceite tal entrada
- a computação não deteminística forma uma árvore, onde:
  - os vértices representam configurações e arestas representam os passos
  - escolhas não deteminísticas representam mais de uma aresta partindo de um vértice
  - a altura da árvore corresponde ao tempo (número de passos)

Classe $\mathcal{NP}$

- Toda MT determinística é um caso particular de uma MT não determinística que não possui múltiplas escolhas em seus movimentos, portanto

$$\mathcal{P} \subseteq \mathcal{NP}$$

- Contudo, aparentemente $\mathcal{NP}$ contém muitos problemas que não estão em $\mathcal{P}$
  - Uma MT não determinística executando em tempo polinomial tem a habilidade de **chutar** um número exponencial de possíveis soluções para um problema e verificar tais soluções em "paralelo"

Classe $\mathcal{NP}$

- O que caracteriza a classe $\mathcal{NP}$ é que uma solução para um problema em $\mathcal{NP}$ pode ser verificada em tempo polinomial ao tamanho da entrada
  - a cadeia que apresenta esta propriedade denomina-se **certificado**
  - somente os problemas $\mathcal{NP}$ possuem certificados

Classe $\mathcal{NP}$

- Definição alternativa (baseada na ideia de certificados):
  - Seja Σ um alfabeto e ";" um símbolo que não pertence a Σ. Seja $L' \subseteq \Sigma^*;\Sigma^*$. Dizemos que L' é **polinomialmente equilibrada** se nela existe um polinômio $p(n)$, tal que, se $x;y \in L'$, então $|y| \leq p(|x|)$
  - Seja $L \subseteq \Sigma^*$ uma linguagem onde ";" $\notin \Sigma$ e $|\Sigma| \geq 2$. Então, $L \in \mathcal{NP}$ se e somente se existir uma linguagem polinomialmente equilibrada $L' \subseteq \Sigma^*;\Sigma^*$, tal que $L' \in \mathcal{P}$ e $L = \{x$: há um $y \in \Sigma^*$, tal que $x;y \in L'\}$

Classe $\mathcal{NP}$

- Exemplo de certificado: Números Compostos
  - Um número é dito composto quando pode ser representado pelo produto de dois números naturais, maiores do que 1. São compostos: $4, 6, 8, 10, 12$
  - Dado um número natural, por exemplo - $4.294.967.297$ - ele é composto?
    - não há um modo claro e eficiente de responder a esta pergunta, contudo, considerando um conjunto C de números compostos, cada número em C tem um certificado
    - O certificado para $4.294.967.297$ é o par $6.700.417$ e 641
    - Para fazer a verificação basta fazer a multiplicação e constatar que $4.294.967.297 \in C$
    - Curiosidade: a fatoração desde número foi descoberta por Leornard Euler em 1732, 92 anos depois de Pierre de Fermat ter conjecturado que não existiria tal fatoração

Classe $\mathcal{NP}$

- Exemplo: Problema da Satisfazibilidade Booleana
  - Para demonstrar que o SAT pertence a $\mathcal{NP}$, deve-se projetar uma MT não determinística M que decide em tempo polinomial quaisquer codificações satisfazíveis de fórmulas booleanas em CNF
  - M opera da seguinte forma:
    - Dada a entrada w verifica se w codifica uma fórmula booleana em CNF, se não rejeita a entrada, nesta verificação, M também conta o número de variáveis, armazenando-as em uma 2^a fita
    - Ao termino desse passo, a 2^a fita de M contém a cadeia $\triangleright I^n$ onde n é o número de variáveis

Classe $\mathcal{NP}$

- Exemplo: Problema da Satisfazibilidade Booleana
  - M entra em uma fase não determinística, substituindo os símbolos I da 2^a fita por valores verdade $\top$ e $\bot$. Para isto, basta adicionar um novo estado q a K em M e incluir novas transições $(q, I, q, \top), (q, I, q, \bot), (q, \top, q, \rightarrow), (q, \bot, q, \rightarrow), (q, \sqcup, q', \sqcup)$ onde q' é o estado a partir do qual a computação prosseguirá
  - Em sua fase final, M opera de modo determinístico, interpretando a cadeia sobre $\{\top, \bot\}^n$ contida da 2^a fita, verificando se cada cláusula da fórmula contém um literal que é $\top$
  - Se todas as cláusulas apresentam um literal $\top$, M aceita a entrada, caso contrário, rejeita.

Classe $\mathcal{NP}$

- Exemplo: Problema da Caixeiro Viajante
  - Assim com o problema SAT, pode-se “testar” em paralelo todas as permutações possíveis, em tempo polinomial
- Diversos outros problemas aparente difíceis também podem ser facilmente solucionados com MT não deteminísticas e portanto pertencem a classe $\mathcal{NP}$

Classes de Complexidade

- Conforme visto, é possível simular uma MT não determinística em uma MT determinística, contudo essa simulação recorre ao exame exaustivo de todas as possíveis computações
- Ou seja, é necessário um número de passos exponencial em n em uma máquina determinística para simular uma computação de n passos de uma máquina não determinística
- A classe $\mathcal{NP}$ não é fechada para o complemento.

Classes de Complexidade

- Classe $\mathcal{EXP}$
  - Uma máquina de Turing $M = (K, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ é dita **exponencialmente limitada** se há um polinômio $p(n)$ tal que, para qualquer entrada x, não há configuração C tal que $(q_0, \triangleright x) \vdash_M^{2^{p(|x|)}+1} C$
    - Ou seja, tal máquina sempre pára após, no máximo, um número exponencial de passos

Classes de Complexidade

$$\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{EXP}$$

- Considerações
  - Determinar se $\mathcal{P} = \mathcal{NP}$ é um problema em aberto
  - Determinar se $\mathcal{NP} = \mathcal{EXP}$ é um problema em aberto
    - O que se sabe, seguramente, é que $\mathcal{P} \subset \mathcal{EXP}$
    - Suspeita-se que ambas as inclusões acima são próprias

Satisfazibilidade Booleana

- Problema da Satisfazibilidade (Satisfação) Booleana
  - Tão importante para o estudo da tratabilidade quanto é o problema da parada para o estudo da decidibilidade

Lógica Booleana

- Descreve proposições simples
  - Está chovendo.
  - O carro é branco.
- Uso de conectivos lógicos para construir proposições compostas
  - A vassoura não está no canto ou está chovendo

Lógica Booleana

- As proposições simples podem ser representadas por meio de variáveis lógicas, como p, q, r,... ou ainda p_1, p_2,...
- A sintaxe da lógica booleana é a seguinte:
  - Símbolos proposicionais: $p_1, p_2, ..., p_n$ (alfabeto infinito, mas contável)
  - Constantes: verdadeiro $(\top)$ e falso $(\bot)$
  - Conectivos lógicos: o conectivo pode ser unário - negação $(\neg)$ ou binários - e $(\wedge)$, ou $(\vee)$ e implicação $(\rightarrow)$
  - Parênteses.

Lógica Booleana

- A sintaxe da lógica proposicional é definida em linguagem formal BNF (Backus-Naur Formalism) como:

$$
\begin{aligned}
\langle \text{Sentença} \rangle &::= \langle \text{Sentença Atômica} \rangle \mid \langle \text{Sentença Complexa} \rangle \\
\langle \text{Sentença Atômica} \rangle &::= \top \mid \bot \\
&\quad \mid p_1 \mid p_2 \mid \ldots p_n \\
\langle \text{Sentença Complexa} \rangle &::= (\langle \text{Sentença} \rangle) \\
&\quad \mid \langle \text{Sentença} \rangle \ \langle \text{Conectivo} \rangle \ \langle \text{Sentença} \rangle \\
&\quad \mid \neg \langle \text{Sentença} \rangle \\
\langle \text{Conectivo} \rangle &::= \vee \mid \wedge \mid \rightarrow
\end{aligned}
$$

Lógica Booleana

- A lógica booleana lida com dois valores: verdadeiro ou falso
- A semântica da lógica booleana é dada pela interpretação da fórmula de acordo com o valor verdade dos símbolos proposicionais e com a aplicação dos conectivos lógicos
  - Tabelas Verdade

p_1	$\neg p_1$	p_2	$p_1 \wedge p_2$	$p_1 \vee p_2$	$p_1 \rightarrow p_2$
F	V	F	F	F	V
F	V	V	F	V	V
V	F	F	F	V	F
V	F	V	V	V	V

Lógica Booleana

- A construção da tabela verdade apresenta o conjunto de **interpretações** possíveis para uma dada fórmula ou proposição complexa
- Se para todas as interpretações, o valor da fórmula é verdadeiro, diz-se que esta fórmula é *teorema* ou uma *tautologia*
- Se há pelo menos uma interpretação que torne a fórmula verdadeira, então a fórmula é dita *válida* ou *satisfazível* (do inglês satisfiable)
- Se não existe uma atribuição de valores verdade que tornem a fórmula verdadeira, então esta fórmula é *insatisfazível* ou uma *contradição*.

Lógica Booleana

- Podemos representar as fórmulas em lógica proposicional em uma forma normalizada utilizando unicamente os operadores $\{\neg, \vee, \wedge\}$
- Formas Normais Canônicas:
  - Simplificar fórmulas complexas
  - Em geral, primeira etapa dos procedimentos de demonstração automática de teoremas

Formas normais canônicas

- Forma normal conjuntiva (ou forma clausal) - CNF
  - conjunção de disjunções

$$(p \vee q) \wedge (\neg q \vee r)$$

  - Formalmente
    - a fórmula está na forma

$$C_1 \wedge C_2 \wedge \ldots C_n$$

    - onde cada cláusula C_i é uma disjunção de literais

$$L_1 \vee L_2 \vee \ldots L_n$$

    - onde cada literal L_i é um símbolo de predicado ou sua negação

Formas normais canônicas

- Forma normal disjuntiva (ou forma clausal dual) - DNF
  - disjunção de conjunções

$$(r \wedge \neg p) \vee (q \wedge r)$$

  - Formalmente
    - a fórmula esta na forma

$$D_1 \vee D_2 \vee \ldots D_n$$

    - onde cada cláusula D_i é uma conjunção de literais

$$L_1 \wedge L_2 \wedge \ldots L_n$$

Formas normais canônicas

- As fórmulas são transformadas utilizando as relações de equivalência

$$A \rightarrow B \equiv \neg A \vee B \text{ (eliminação da implicação)}$$

$$\neg(A \vee B) \equiv \neg A \wedge \neg B \text{ (lei de De Morgan)}$$

$$\neg(A \wedge B) \equiv \neg A \vee \neg B \text{ (lei de De Morgan dual)}$$

$$A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C) \text{ (distributividade)}$$

$$A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C) \text{ (distributividade)}$$

- As representações em formas normais são equivalentes às fórmulas originais

$$W \equiv CNF_W \equiv DNF_W$$

Formas normais canônicas

- Algoritmo Forma Normal Conjuntiva
  - Eliminar todas as ocorrências de $A \rightarrow B$ em *w*, substituindo-as por $\neg A \vee B$.
  - Reduzir o escopo das negações de maneira que só restem negações aplicadas a fórmulas atômicas. Para isto usar as regras:

$$\neg(A \vee B) \Rightarrow (\neg A \wedge \neg B)$$
$$\neg(A \wedge B) \Rightarrow (\neg A \vee \neg B)$$
$$\neg(\neg(A)) \Rightarrow A$$

  - Converter a fórmula para a forma de uma conjunção de disjunções usando a propriedade distributiva do operador $\vee$ sobre o operador $\wedge$:

$$A \vee (B \wedge C) \Rightarrow (A \vee B) \wedge (A \vee C).$$

Forma normal conjuntiva

- Exemplo:
  - “Se chove então eu não saio. Mas, se não chove, eu saio e eu tomo sorvete. Eu saio e tomo sorvete.”

representamos como:	transformando em CNF:
$c \rightarrow \neg s$	$\neg c \vee \neg s$
$\neg c \rightarrow s \wedge t$	$c \vee s$
$s \wedge t$	$c \vee t$
	s
	t

Problema da Satisfazibilidade Booleana

- O problema da satisfazibilidade ou satisfação (SAT) em formas normais conjuntivas consiste em determinar se a fórmula

$$W_{CNF} = C_1 \wedge C_2 \wedge, ..., C_n$$

é satisfazível (do inglês “satisfiable”), ou seja, se há uma combinação de valores para as variáveis tal que a fórmula seja avaliada como *verdadeira*.

Problema da Satisfazibilidade Booleana

- Exemplo 1: Dada a fórmula:

$$W = \{(\neg c \vee \neg s), (c \vee s), (c \vee t), (s), (t)\}$$

W é satisfazível fazendo-se $c = \bot$, $s = \top$, $t = \top$

Problema da Satisfazibilidade Booleana

- Exemplo 2: Dada a fórmula:

$W = \{(p_1 \vee p_2 \vee p_3), (\neg p_1 \vee p_2), (\neg p_2 \vee p_3), (\neg p_3 \vee p_1), (\neg p_1 \vee \neg p_2 \vee \neg p_3)\}$

W é satisfazível?