

Tratabilidade

Prof^a Jerusa Marchi

Departamento de Informática e Estatística

Universidade Federal de Santa Catarina

e-mail: jerusa@inf.ufsc.br

Teoria da Computação

- Não Computável
- Computável
 - Indecidível
 - Decidível (efetivamente computável)
 - 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

Tratabilidade

- Demonstrar quais problemas podem ser passíveis de soluções em tempo polinomial
 - 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, \dots ou ainda p_1, p_2, \dots
- A sintaxe da lógica booleana é a seguinte:
 - Símbolos proposicionais: p_1, p_2, \dots, p_n (alfabeto infinito, mas contável)
 - Constantes: verdadeiro (\top) e falso (\perp)
 - 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:

$\langle \text{Sentença} \rangle ::= \langle \text{Sentença Atômica} \rangle \mid \langle \text{Sentença Complexa} \rangle$

$\langle \text{Sentença Atômica} \rangle ::= \top \mid \perp$

$\mid p_1 \mid p_2 \mid \dots p_n$

$\langle \text{Sentença Complexa} \rangle ::= (\langle \text{Sentença} \rangle)$

$\mid \langle \text{Sentença} \rangle \langle \text{Conectivo} \rangle \langle \text{Sentença} \rangle$

$\mid \neg \langle \text{Sentença} \rangle$

$\langle \text{Conectivo} \rangle ::= \vee \mid \wedge \mid \rightarrow$

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

$$C_1 \wedge C_2 \wedge \dots C_n$$

- onde cada cláusula C_i é uma disjunção de literais

$$L_1 \vee L_2 \vee \dots L_n$$

- 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

$$D_1 \vee D_2 \vee \dots D_n$$

- onde cada cláusula D_i é uma conjunção de literais

$$L_1 \wedge L_2 \wedge \dots L_n$$

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:

$$c \rightarrow \neg s$$

$$\neg c \rightarrow s \wedge t$$

$$s \wedge t$$

transformando em CNF:

$$\neg c \vee \neg s$$

$$c \vee s$$

$$c \vee t$$

$$s$$

$$t$$

Problema da Satisfazibilidade Booleana

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

$$W_{CNF} = C_1 \wedge C_2 \wedge \dots \wedge C_n$$

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 $c = \perp$, $s = \top$, $t = \top$

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?

Tratabilidade

- 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 $W = \{C_1, \dots, C_n\}$ onde C_i é uma disjunção de literais, há alguma atribuição de valores verdade que tornem a fórmula verdadeira?
- Problema da alcançabilidade - Dado um grafo orientado $G \subseteq V \times V$, onde $V = \{v_1, \dots, v_n\}$ 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 $\{1, 2, \dots, n\}$ tal que $\text{custo}(\pi) \leq L$?

Teoria da Complexidade

- Complexidade de Tempo (número de passos de execução de um algoritmo para resolver um problema)
 - Crescimento Polinomial \times Crescimento Exponencial
- Complexidade de Espaço (quantidade de memória utilizada para resolver um problema)
 - Crescimento Polinomial \times Crescimento Exponencial

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

Complexidade de Tempo

- Comportamento assintótico de funções - Notação O
 - Uma função f é dita de complexidade $O(g(x))$ se ambas funções, para x suficientemente grande, "crescem" da mesma forma
 - $f(x) = 10x^5 + 3x^3$ é $O(x^5)$
 - Formalmente: Sejam duas funções $f(x)$ e $g(x)$

$$O(g(x)) = \left\{ f(x) : \text{existem } c, x_0 \text{ constantes positivas tais que} \right. \\ \left. \forall x : x_0 \leq x : 0 \leq f(x) \leq cg(x) \right\}$$

Complexidade de Tempo

- Comportamento assintótico de funções - Notação O

- Exemplos:

- $f(x) = 10x + 11$ é $O(x)$ - considerando $c = 12$ e $x_0 = 10$

- $f(x) = x^2 + 100$ é $O(x^2)$ - considerando $c = 2$ e $x_0 = 10$

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}

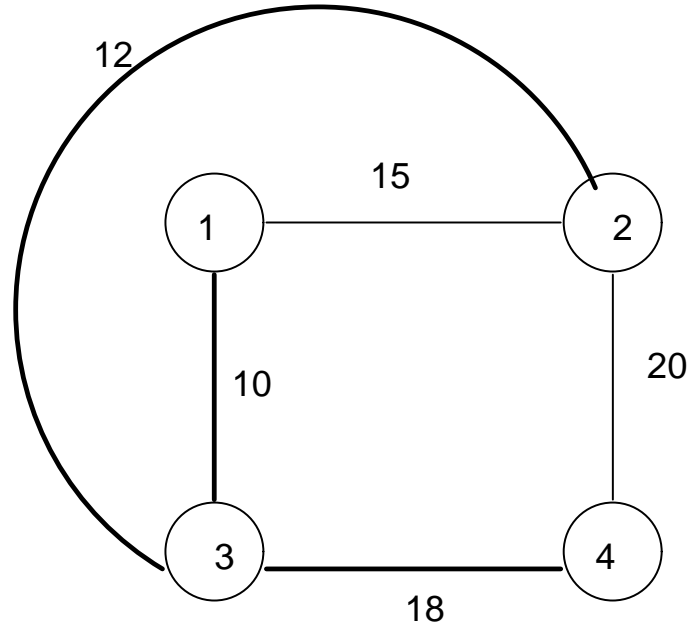
- Exemplos - Árvore Geradora Mínima (Algoritmo de Kruskal)
 - Ideia: partindo de algum vértice de G , escolher a cada passo, a aresta com menor peso para incluir na árvore geradora mínima
 - Algumas definições:
 - Um **corte** $(S, V - S)$ de um grafo não direcionado $G = (V, E)$ é uma partição de V
 - Uma aresta $(u, v) \in E$ **cruza** o corte $(S, V - S)$ se um de seus vértices pertence a S e outro pertence a $V - S$
 - Um corte **respeita** o conjunto A de arestas se nenhuma aresta em A cruza o corte
 - Uma aresta é uma **aresta leve** cruzando o corte se seu peso for menor do que qualquer outra aresta cruzando o corte

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 $O(e(e + m))$ em uma implementação pouco eficiente

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 pertencem, 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 $O(2n^2)$
 - 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 $O((2n^2)^2) = O(4n^4)$, 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



Exemplos

- Problema do Caixeiro Viajante
- Circuito Hamiltoniano
- Problema da Alcançabilidade
- Problema da Satisfazibilidade Booleana

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 determinística forma uma árvore, onde:
 - os vértices representam configurações e arestas representam os passos
 - escolhas não determinísticas 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 $L = \{x : \text{há um } y \in \Sigma^*, \text{ tal que } x; y \in L'\}$

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 desse número foi descoberta por Leonhard 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 2ª fita
 - Ao termino desse passo, a 2ª 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 2ª fita por valores verdade \top e \perp . Para isto, basta adicionar um novo estado q a K em M e incluir novas transições $(q, I, q, \top), (q, I, q, \perp), (q, \top, q, \rightarrow), (q, \perp, q, \rightarrow), (q, \sqcup, q', \sqcup)$ 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 $\{\top, \perp\}^n$ contida da 2ª 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 determinísticas 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

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