Capítulo 3

A Tese de Church–Turing

Até agora no nosso desenvolvimento da teoria da computação apresentamos vários modelos de dispositivos de computação. Autômatos finitos são bons modelos para dispositivos que têm uma quantidade pequena de memória. Autômatos a pilha são bons modelos para dispositivos que têm uma quantidade de memória ilimitada que é utilizável apenas da forma "o último que entra é o primeiro que sai" tal qual uma pilha. Mostramos que algumas tarefas muito simples estão além das capacidades desses modelos. Portanto eles são demasiado restritos para servir como modelos de computadores de propósito geral.

3.1 Máquinas de Turing

Voltamo-nos agora para um modelo muito mais poderoso, primeiramente proposto por Alan Turing em 1936, chamado de ***máquinas de Turing***. Semelhante a um autômato finito mas com uma mémoria ilimitada e irrestrita, uma máquina de Turing é um modelo muito mais preciso de um computador de propósito geral. Uma máquina de Turing pode fazer tudo o que um computador real pode fazer. Entretanto, mesmo uma máquina de Turing não pode resolver certos problemas. Num sentido muito real, esses problemas estão além dos limites teóricos da computação.

O modelo da máquina de Turing usa uma fita infinita como sua memória ilimitada. Ela tem uma cabeça que pode ler e escrever símbolos e mover sobre a fita. Inicialmente a fita contém somente a cadeia de entrada e está em branco em todo o restante. Se a máquina precisa de armazenar informação, ela pode escrever essa informação sobre a fita. Para ler a informação que ela escreveu, a máquina pode mover sua cabeça de volta sobre a posição onde a informação foi escrita. A máquina continua a computar até que ela decide produzir uma saída. As saídas *aceita* e *rejeita* são obtidas pela entrada em estados de aceitação e de rejeição. Se ela não entrar em estado de aceitação ou de rejeição, ela continuará para sempre, nunca parando.

Figura 3.1: Esquema de uma máquina de Turing

A seguinte lista resume as diferenças entre autômatos finitos e máquinas de Turing.

1. Uma máquina de Turing pode tanto escrever sobre a fita quanto ler dela.
2. A cabeça de leitura-escrita pode mover tanto para a esquerda quanto para a direita.
3. A fita é infinita.
4. Os estados especiais para rejeitar e aceitar têm efeito imediato.

Vamos considerar uma máquina de Turing M_1 para testar pertinência na linguagem $B = \{w\#w \mid w \in \{\mathtt{0},\mathtt{1}\}^*\}$. Isto é, desejamos projetar M_1 para aceitar se sua entrada é um membro de B. Para entender M_1 melhor, ponha-se no seu lugar imaginando que você está sobre uma entrada de 1Km de comprimento consistindo de milhões de caracteres. Seu objetivo é determinar se a entrada é um membro de B, ou seja, se a entrada compreende duas cadeias idênticas separadas por um símbolo #. A entrada é demasiado longa para você memorizá-la toda, mas lhe é permitido mover de-frente-para-trás e de-trás-para-frente sobre a entrada e deixar marcas sobre ela. É claro que a estratégia óbvia é ziguezaguear para as posições correspondentes nos dois lados do # e determinar se eles casam. Use marcas para manter o registro de quais posições se correspondem.

Projetamos M_1 para funcionar da mesma maneira. Ela realiza múltiplas varridas sobre a cadeia de entrada com a cabeça de leitura-escrita. A cada passagem ela emparelha um dos caracteres em cada lado do símbolo #. Para manter registro de quais símbolos já foram verificados, M_1 deixa uma marca sobre cada símbolo à medida que ele é examinado. Se ela marca todos os símbolos, isso significa que tudo emparelhou de forma bem sucedida, e M_1 vai para um estado de aceitação. Se ela descobre uma sobra, ela entra em um estado de rejeição. Em resumo, o algoritmo de M_1 é o seguinte.

M_1 = "Sobre a cadeia de entrada w:

1. Faça um varredura na entrada para assegurar que ela contém uma única ocorrência do símbolo #. Se não, *rejeite*.
2. Faça um zigue-zague na fita para fazer corresponder posições nos dois lados do símbolo # para verificar se essas posições contêm o mesmo símbolo. Se elas não contêm, *rejeite*. Marque os símbolos à medida que eles são verificados para manter registro de quais símbolos têm correspondência.
3. Quando todos os símbolos à esquerda do # foram marcados, verifique se existe algum símbolo remanescente à direita do #. Se quaisquer símbolos restam, *rejeite*; caso contrário, *aceite*.

A Figura 3.2 contém várias fotografias instantâneas da fita de M_1 enquanto ela está computando nos estágios 2 e 3 quando iniciada sobre a entrada `011000#011000`.

Figura 3.2: Fotografias instantâneas da máquina de Turing M_1 computando sobre a entrada `011000#011000`

Essa descrição da máquina de Turing M_1 esboça a maneira como ela funciona mas não dá todos os detalhes. Podemos descrever máquinas de Turing em todos os detalhes dando descrições formais análogas àquelas introduzidas para autômatos finitos e autômatos a pilha. A descrição formal especifica cada uma das partes da definição formal do modelo da máquina de Turing a ser apresentada logo mais. Na realidade quase nunca damos descrições formais de máquinas de Turing porque elas tendem a ser muito grandes.

Definição formal de uma máquina de Turing

O coração da definição de uma máquina de Turing é a função de transição δ porque ela diz como a máquina vai de um passo ao próximo. Para uma máquina de Turing, δ toma a forma $Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{E}, \mathrm{D}\}$. Ou seja, quando a máquina está num certo estado q e a cabeça está sobre uma posição da fita contendo um símbolo a, e se $\delta(q, a) = (r, b, \mathrm{E})$, a máquina escreve o símbolo b substituindo o a, e vai para o estado r. O terceiro componente é E ou D e indica se a cabeça move para a esquerda ou para a direita após a escrita. Nesse caso, o E indica um movimento para a esquerda.

Definição 3.1 . . . . . .
Uma ***máquina de Turing*** é uma 7-upla $(Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{aceita}}, q_{\mathrm{rejeita}})$, onde Q, Σ, Γ são todos conjuntos finitos e

1. Q é o conjunto de estados.
2. Σ é o alfabeto de entrada que não contém o símbolo especial ***branco*** $\sqcup$,
3. Γ é o alfabeto da fita, onde $\sqcup \in \Gamma$ e $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{E}, \mathrm{D}\}$ é a função de transição,
5. q_0 é o estado inicial,
6. $q_{\mathrm{aceita}} \in Q$ é o estado de aceitação,
7. $q_{\mathrm{rejeita}} \in Q$ é o estado de rejeição.

Uma máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{aceita}}, q_{\mathrm{rejeita}})$ computa da seguinte forma. Inicialmente M recebe sua entrada $w = w_1 w_2 \ldots w_n \in \Sigma^*$ nas células mais à esquerda da fita, e o restante da fita é branco (i.e., preenchido com símbolos em branco). A cabeça começa na célula mais à esquerda da fita. Note que Σ não contém o símbolo branco, portanto o símbolo branco aparecendo na fita marca o final da entrada. Uma vez que M começa, a computação prossegue conforme as regras descritas pela função de transição. Se M em algum momento tenta mover sua cabeça para à esquerda do final à esquerda da fita, a cabeça permanece no mesmo lugar para aquele movimento, muito embora a função de transição indique E. A computação continua até que ela entre nos estados de aceitação ou rejeição nos quais ela pára. Se nenhum desses casos ocorre, M continua para sempre.

À medida que uma máquina de Turing computa, mudanças ocorrem no estado atual, no conteúdo da fita, e na localização da cabeça. Uma combinação dessas três informações é chamada uma ***configuração*** da máquina de Turing. Configurações frequentemente são representadas de uma maneira especial. Para um estado q e duas cadeias u e v sobre o alfabeto da fita Γ escrevemos $u\,q\,v$ para a configuração onde o

estado atual é q, o conteúdo da fita é uv, e a localização atual da cabeça é o primeiro símbolo de v. A fita contém apenas brancos a partir do último símbolo de v. Por exemplo, $\texttt{1011}q_7\texttt{01111}$ representa a configuração quando a fita é `101101111`, o estado atual é q_7, e a cabeça está atualmente sobre o segundo `0`. A Figura 3.3 mostra uma máquina de Turing com aquela configuração.

Figura 3.3: Uma máquina de Turing com configuração $\texttt{1011}q_7\texttt{01111}$

Aqui formalizamos nosso entendimento intuitivo da maneira pela qual uma máquina de Turing computa. Vamos dizer que uma configuração C_1 ***produz*** uma configuração C_2 se a máquina de Turing pode legalmente ir de C_1 para C_2 em um único passo. Definimos essa noção formalmente da seguinte maneira.

Suponha que tenhamos a, b, e c em Γ, assim como u e v em Γ^* e estados q_i e q_j. Nesse caso uaq_ibv e uq_jacv são duas configurações. Vamos dizer que

$$uaq_ibv \quad \text{produz} \quad uq_jacv$$

se na função de transição $\delta(q_i, b) = (q_j, c, \mathrm{E})$. Isso lida com o caso em que a máquina de Turing move para a esquerda. Para um movimento à direita, vamos dizer que

$$uaq_ibv \quad \text{produz} \quad uacq_jv$$

se $\delta(q_i, b) = (q_j, c, \mathrm{D})$.

Casos especiais ocorrem quando a cabeça está em uma das extremidades da configuração. Para a extremidade esquerda, a configuração $q_i\, bv$ produz $q_j\, cv$ se a transição é de movimento para a esquerda (porque cuidamos para que a máquina não passe da extremidade à esquerda da fita), e ela produz $c\, q_j\, v$ para a transição de movimento à direita. Para a extremidade direita, a configuração $ua\, q_i$ é equivalente a $ua\, q_i\sqcup$ porque assumimos que brancos vêm logo depois da parte da fita representada na configuração. Por conseguinte podemos lidar com esse caso tal qual anteriormente, com a cabeça não mais na extremidade direita.

A ***configuração inicial*** de M sobre a entrada w é a configuração $q_0\, w$, que indica que a máquina está no estado inicial q_0 com sua cabeça na posição mais à esquerda sobre a fita. Em uma ***configuração de aceitação*** o estado da configuração é q_{aceita}. Em uma ***configuração de rejeição*** o estado da configuração é q_{rejeita}. Configurações de aceitação e de rejeição são ***configurações de parada*** e como tal não produzem quaisquer configurações. Uma máquina de Turing M ***aceita*** a entrada w se uma seqüência de configurações $C_1, C_2, \ldots, C_k$ existe onde

1. C_1 é a configuração inicial de M sobre a entrada w,
2. cada C_i produz C_{i+1}, e
3. C_k é uma configuração de aceitação.

A coleção de cadeias que M aceita é ***a linguagem de*** M, denotada $L(M)$.

Definição 3.2 .....
Chame uma linguagem ***Turing-reconhecível*** se alguma máquina de Turing a reconhece.[1]

Quando iniciamos uma MT sobre uma entrada, três resultados são possíveis. A máquina pode *aceitar*, *rejeitar*, ou *entrar em loop*. Por *entrar em loop* queremos dizer que a máquina simplesmente não pára. Ela não está necessariamente repetindo os mesmos passos para sempre como a conotação de entrar em loop pode sugerir. Entrar em loop pode acarretar comportamento simples ou complexo que nunca leva a um estado de parada.

Uma máquina de Turing M pode falhar em aceitar uma cadeia entrando no estado q_{rejeita} e rejeitando, ou entrando em loop. Às vezes distinguir uma máquina que está em loop de uma que está meramente levando um tempo longo é difícil. Por essa razão preferimos máquinas de Turing que parem sobre todas as entradas; tais máquinas nunca entram em loop. Essas máquinas são chamadas ***decisores*** porque elas sempre tomam uma decisão de aceitar ou de rejeitar. Um decisor que reconhece uma linguagem também é dito ***decidir*** aquela linguagem.

Definição 3.3 .....
Chame uma linguagem ***Turing-decidível*** ou simplesmente ***decidível*** se alguma máquina de Turing a decide.[2]

Toda linguagem decidível é Turing-reconhecível mas certas linguagens Turing-reconhecíveis não são decidíveis. Agora damos alguns exemplos de linguagens decidíveis. Apresentamos exemplos de linguagens que são Turing-reconhecíveis mas não decidíveis após desenvolvermos uma técnica para provar indecidibilidade no Caítulo 4.

Exemplos de máquinas de Turing .....

Como fizemos para autômatos finitos e autômatos a pilha, podemos dar uma descrição formal de uma máquina de Turing particular especificando cada uma das suas sete partes. Entretanto, indo para aquele nível de detalhe para máquinas de Turing pode ser incômodo para todas exceto as máquinas minúsculas. Assim, não desperdiçaremos muito tempo dando tais descrições. No máximo daremos apenas descrições de mais alto nível porque elas são suficientemente precisas para nossos propósitos e são muito mais fáceis de entender. Entretanto, é importante lembrar que toda descrição de mais alto nível é na verdade simplesmente uma abreviação da sua contrapartida formal. Com paciência e cuidado poderíamos descrever qualquer das máquinas de Turing neste livro em detalhe formal completo.

Para ajudar você a fazer a conexão entre as descrições formais e as descrições de mais alto nível, damos diagramas de estado nos próximos dois exemplos. Você pode pulá-los se você já se sente confortável com essa conexão.

Exemplo 3.4 .....
Aqui descrevemos uma MT M_2 que reconhece a linguagem consistindo de todas as cadeias de 0's cujo comprimento é uma potência de 2. Ela decide a linguagem $A = \{0^{2^n} \mid n \geq 0\}$.

M_2 = "Sobre a cadeia de entrada w:

[1] Ela é chamada uma linguagem ***recursivamente enumerável*** em alguns outros livros-texto.
[2] Ela é chamada uma linguagem ***recursiva*** em alguns outros livros-texto.

1. Faça uma varredura da esquerda para a direita na fita, apagando um `0` alternadamente. (O primeiro não, o segundo sim, etc.)

2. Se no estágio 1 a fita continha um único `0`, *aceite*.

3. Se no estágio 1 a fita continha mais de um único `0` e o número de `0`'s era ímpar, *rejeite*.

4. Retorne a cabeça para a extremidade esquerda da fita.

5. Vá para o estágio 1."

Cada iteração do estágio 1 corta o número de `0`'s pela metade. À medida que a máquina faz a varredura na fita no estágio 1, ela mantém registro do fato de que o número de `0`'s é par ou ímpar. Se aquele número é ímpar e maior que 1, o número original de `0`'s na entrada não poderia ter sido uma potência de 2. Por conseguinte a máquina rejeita nesta instância. Entretanto, se o número de `0`'s vistos é 1, o número original tem que ter sido uma potência de 2. Portanto nesse caso a máquina aceita.

Agora damos a descrição formal de $M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{aceita}}, q_{\text{rejeita}})$.

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_{\text{aceita}}, q_{\text{rejeita}}\}$,
- $\Sigma = \{\texttt{0}\}$, e
- $\Gamma = \{\texttt{0}, \texttt{x}, \sqcup\}$.
- Descrevemos δ com um diagrama de estados (veja Figura 3.4).
- Os estados inicial, de aceitação, e de rejeição são q_1, q_{aceita}, e q_{rejeita}.

No diagrama de estados da Figura 3.4 o rótulo $\texttt{0} \to \sqcup, \text{D}$ aparece na transição de q_1 para q_2. Significa que, quando no estado q_1 com a cabeça lendo `0`, a máquina vai para o estado q_2, escreve $\sqcup$, e move a cabeça para a direita. Em outras palavras, $\delta(q_1, \texttt{0}) = (q_2, \sqcup, \text{D})$. Para maior clareza usamos a abreviação $\texttt{0} \to \text{D}$ na transição de q_3 para q_4, para indicar que a máquina move para a direita quando lê `0` no estado q_3 mas não altera a fita, portanto $\delta(q_3, \texttt{0}) = (q_4, \texttt{0}, \text{D})$.

Essa máquina começa escrevendo um símbolo em branco sobre o `0` mais à esquerda na fita de modo que ela possa encontrar a extremidade esquerda da fita no estágio 4. Enquanto que normalmente usaríamos um símbolo mais sugestivo tal como # para o delimitador de extremidade esquerda, usamos um branco aqui para manter o alfabeto da fita, e portanto o diagrama de estados, pequeno. O Exemplo 3.6 dá um outro método de encontrar a extremidade esquerda da fita.

Damos uma amostra de execução dessa máquina sobre a entrada `0000`. A configuração inicial é $q_1\texttt{0000}$. A seqüência de configurações que a máquina entra aparece após a Figura 3.4. Leia as colunas de cima para baixo e da esquerda para a direita.

Figura 3.4: Diagrama de estados para a máquina de Turing M_2

Uma amostra de execução de M_2 sobre a entrada 0000:

$q_1$0000	⊔q_5x0x⊔	⊔x$q_5$xx⊔
⊔$q_2$000	q_5 ⊔ x0x⊔	⊔$q_5$xxx⊔
⊔x$q_3$00	⊔q_2x0x⊔	q_5 ⊔ xxx⊔
⊔x0$q_4$0	⊔xq_2x0⊔	⊔$q_2$xxx⊔
⊔x0xq_3⊔	⊔xxq_3x⊔	⊔x$q_2$xx⊔
⊔x0q_5x⊔	⊔xxxq_3⊔	⊔xxq_2x⊔
⊔x$q_5$0x⊔	⊔xxq_5x⊔	⊔xxxq_2⊔
		⊔xxx ⊔ q_{aceita}

Exemplo 3.5 . . . . . .
A seguir vai uma descrição formal de $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{aceita}}, q_{\text{rejeita}})$, a máquina de Turing que informalmente descrevemos na página 127 para decidir a linguagem $B = \{w\#w \mid w \in \{0,1\}^*\}$.

1. $Q = \{q_1, \ldots, q_{14}, q_{\text{aceita}}, q_{\text{rejeita}}\}$,
2. $\Sigma = \{0, 1, \#\}$, e $\Gamma = \{0, 1, \#, x, \sqcup\}$.
3. Descrevemos δ com um diagrama de estados (veja Figura 3.5).
4. Os estados inicial, de aceitação, e de rejeição são q_1, q_{aceita}, q_{rejeita}.

Na Figura 3.5 que ilustra o diagrama de estados da MT M_1, você vai encontrar o rótulo $0, 1 \rightarrow$ D sobre a transição indo de q_3 para si próprio. Esse rótulo significa que a máquina permanece em q_3 e move para a direita quando ela lê um 0 ou um 1 no estado q_3. Ela não muda o símbolo sobre a fita.

Figura 3.5: Diagrama de estados para a máquina de Turing M_1

Como no Exemplo 3.4, a máquina começa escrevendo um símbolo em branco para delimitar a fronteira à esquerda da fita. Dessa vez ela pode escrever um 0 ou um 1 por cima quando fizer isso, e memorizar o símbolo escrito por cima usando um controle finito.

O estágio 1 é implementado pelos estados q_1 até q_7, e os estágios 2 e 3 pelos estados remanescentes. Para simplificar a figura, não mostramos o estado de rejeição ou as transições indo para o estado de rejeição. Essas transições ocorrem implicitamente sempre que um estado não dispõe de uma transição saindo dele para um símbolo específico. Por conseguinte, devido ao fato de que no estado q_5 nenhuma seta de saída com um # está presente, se um # ocorre sob a cabeça de leitura-escrita quando a máquina está no estado q_5, ela vai para o estado q_{rejeita}.

Exemplo 3.6 . . . . . .
Aqui, uma máquina de Turing M_3 está fazendo aritmética elementar. Ela decide a linguagem $C = \{a^i b^j c^k \mid i \times j = k \text{ e } i, j, k \geq 1\}$.

M = "Sobre a cadeia de entrada w:

1. Faça uma varredura na entrada da esquerda para a direita para se assegurar que ela é um membro de $\mathtt{a}^*\mathtt{b}^*\mathtt{c}^*$ e *rejeite* se ela não for.

2. Retorne a cabeça para a extremidade esquerda da fita.

3. Marque um a e faça uma varredura à esquerda até que um b ocorra. Viaje entre os b's e os c's, apagando um de cada até que todos os b's tenham se acabado.

4. Restaure os b's marcados e repita o estágio 3 se existe um outro a para marcar. Se todos os a's estão marcados, verifique se todos os c's também estão marcados. Se sim, *aceite*; caso contrário, *rejeite*."

Vamos examinar os quatro estágios de M_3 mais de perto: No estágio 1 a máquina opera como um autômato finito. Nenhuma escrita é necessária quando a máquina move da esquerda para a direita, mantendo registro, por meio do uso de seus estados, se a entrada está na forma apropriada.

O estágio 2 parece igualmente simples mas contém uma sutileza. Como pode a máquina de Turing encontrar a extremidade esquerda da fita de entrada? Encontrar a extremidade direita da fita é fácil porque ela está terminada com um símbolo branco. Mas a extremidade esquerda não tem terminador inicialmente. Uma técnica que permite a máquina encontrar a extremidade esquerda da fita é marcar o símbolo mais à esquerda de alguma forma quando a máquina inicia com sua cabeça sobre aquele símbolo. Então a máquina pode fazer uma varredura para a esquerda até que ela encontra a marca quando ela deseja reinicializar sua cabeça para a extremidade esquerda. O Exemplo 3.4 ilustrou essa técnica, usando um símbolo branco para a marcar o símbolo à esquerda na fita.

Um método mais elaborado de encontrar a extremidade esquerda da fita aproveita-se da forma pela qual definimos o modelo da máquina de Turing. Lembre-se que, se a máquina tenta mover sua cabeça além da extremidade esquerda da fita, ela permanece no mesmo lugar. Podemos usar essa característica para montar um detector de extremidade esquerda. Para detectar se a cabeça está posicionada na extremidade esquerda a máquina pode escrever um símbolo especial sobre a posição atual, enquanto grava no controle o símbolo que ela substituiu. Então ela pode tentar mover a cabeça para a esquerda. Se ela ainda está sobre o símbolo especial, o movimento para a esquerda não foi bem sucedido, e por conseguinte a cabeça tem que ter estado sobre a extremidade esquerda. Se, ao contrário, ela está sobre um símbolo diferente, alguns símbolos permaneceram à esquerda daquela posição sobre a fita. Antes de ir adiante, a máquina tem que garantir que restaura o símbolo modificado sobre a fita para o símbolo original.

Os estágios 3 e 4 têm implementações simples usando vários estados cada.

Exemplo 3.7

Aqui, uma máquina de Turing M_4 está resolvendo o que é chamado do *problema da diferenciação de elemento*. A ela é dada uma lista de cadeias sobre $\{\mathtt{0},\mathtt{1}\}$ separadas por #'s e sua tarefa é aceitar se todas as cadeias são diferentes. A linguagem é

$$E = \{\#x_1\#x_2\#\cdots\#x_l \mid \text{ cada } x_i \in \{\mathtt{0},\mathtt{1}\}^* \text{ e } x_i \neq x_j \text{ para cada } i \neq j\}.$$

A máquina M_4 funciona comparando x_1 com x_2 até x_l, e então comparando x_2 com x_3 até x_l, e assim por diante. Uma descrição informal da MT M_4 que decide essa linguagem segue.

M_4 = "Sobre a entrada w:

1. Ponha uma marca sobre o símbolo mais à esquerda. Se aquele símbolo era um branco, *aceite*. Se aquele símbolo era um #, continue com o próximo estágio. Caso contrário, *rejeite*.

2. Faça uma varredura para a direita até o próximo # e ponha uma segunda marca sobre ele. Se nenhum # for encontrado antes de um símbolo branco, somente x_1 estava presente, portanto *aceite*.

3. Fazendo zigue-zague, compare as duas cadeias à direita dos #'s marcados. Se eles são iguais, *rejeite*.

4. Mova a marca mais à direita das duas para o próximo # à direita. Se nenhum símbolo # for encontrado antes de um símbolo branco, mova a marca mais à esquerda do próximo # para a sua direita e a marca mais à direita para o # após aquele. Dessa vez, se nenhum # está disponível para a marca mais à direita, todas as cadeias foram comparadas, portanto *aceite*.

5. Vá para o estágio 3."

Essa máquina ilustra a técnica de marcar símbolos da fita. No estágio 2, a máquina põe uma marca sobre um símbolo, # nesse caso. Na implementação real, a máquina tem dois símbolos diferentes, # e $\dot{\#}$, no seu alfabeto de fita. Dizer que uma máquina põe uma marca sobre um # significa que a máquina escreve o símbolo $\dot{\#}$ naquela posição. Remover a marca significa que a máquina marca sobre vários símbolos sobre a fita. Para fazer isso simplesmente incluimos versões de todas esses símbolos da fita com pontos no alfabeto da fita.

Podemos concluir dos exemplos precedentes que as linguagens descritas A, B, C, E são decidíveis. Todas as linguagens decidíveis são Turing-reconhecíveis, portanto essas linguagens são também Turing-reconhecíveis. Demonstrar que uma linguagem é Turing-reconhecível nas não decidível é mais difícil, o que fazemos no Capítulo 4.

3.2 Variantes de máquinas de Turing . . . . . .

Definições alternativas de máquinas de Turing abundam, incluindo versões com múltiplas fitas ou com não-determinismo. Elas são chamadas ***variantes*** do modelo da máquina de Turing. O modelo original e suas variantes razoáveis todos têm o mesmo poder—eles reconhecem a mesma classe de linguagens. Nesta seção descrevemos algumas dessas variantes e as provas de equivalência em poder. Chamamos ***robustez*** essa invariância a certas mudanças na definição. Tanto autômatos finitos como autômatos a pilha são modelos um tanto robustos, mas máquinas de Turing têm um grau impressionante de robustez.

Para ilustrar a robustez do modelo de máquina de Turing vamos variar o tipo de função de transição permitida. Em nossa definição, a função de transição força a cabeça a mover para a esquerda ou para a direita após cada passo; a cabeça não pode simplesmente permanecer parada. Suponha que tivéssemos permitido à máquina de Turing a capacidade de permanecer parada. A função de transição teria então a forma $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{E}, \mathrm{D}, \mathrm{P}\}$. Essa característica poderia permitir às máquinas de Turing reconhecer linguagens adicionais, e por conseguinte adicionar poder ao modelo? É claro que não, porque podemos converter qualquer MT com a característica

adicional de "permanecer parada" para uma que não tem essa característica adicional. Fazemos isso substituindo cada transição com "permanecer parada" por duas transições, uma que move para a direita e a segunda que volta para a esquerda.

Esse pequeno exemplo contém a chave para mostrar a equivalência de variantes da máquina de Turing. Para mostrar que dois modelos são equivalentes precisamos simplesmente de mostrar que podemos simular um por meio do outro.

Máquinas de Turing multi-fitas

Uma **máquina de Turing multi-fitas** é como uma máquina de Turing comum com várias fitas. Cada fita tem sua própria cabeça para ler e escrever. Inicialmente a entrada aparece sobre a fita 1, e as outras começam com branco. A função de transição é modificada para permitir ler, escrever, e mover as cabeças em todas as fitas simultaneamente. Formalmente, ela é

$$\delta : Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{\mathrm{E}, \mathrm{D}\}^k$$

onde k é o número de fitas. A expressão

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, \mathrm{E}, \mathrm{D}, \ldots, \mathrm{E})$$

quer dizer que, se a máquina está no estado q_i e a cabeça 1 até a cabeça k estão lendo os símbolos a_1 até a_k, a máquina vai para o estado q_j, escreve os símbolos b_1 até b_k, e move cada cabeça para a esquerda ou a direita conforme especificado.

Máquinas de Turing multi-fitas parecem ser mais poderosas que máquinas de Turing comuns, mas podemos mostrar que elas são equivalentes em poder. Lembre-se que duas máquinas são equivalentes se elas reconhecem a mesma linguagem.

Teorema 3.8 .....
Toda máquina de Turing multi-fitas tem uma máquina de Turing com uma única fita equivalente.

Prova. Mostramos como converter uma MT multi-fitas M em uma MT S com uma única fita equivalente. A idéia chave é mostrar como simular M com S.

Digamos que M tem k fitas. Então S simula o efeito de k fitas armazenando a informação que elas trazem numa única fita. Ela usa o novo símbolo # como um delimitador para separar o conteúdo das diferentes fitas. Além do conteúdo dessas fitas, S tem que manter registro das posições das cabeças. Ela faz isso escrevendo um símbolo de fita com um ponto acima dele para marcar o lugar onde a cabeça sobre aquela fita estaria. Pense como se fossem fitas e cabeças "virtuais". Tal qual anteriormente, os símbolos com um ponto em cima são simplesmente novos símbolos que foram adicionados ao alfabeto de fita. A Figura 3.6 ilustra como uma fita pode ser usada para representar três fitas.

Figura 3.6: Representando três fitas com uma

S = "Sobre a entrada $w = w_1 \cdots w_n$:

1. Primeiro S põe sua fita no formato que representa todas as k fitas de M. A fita formatada contém
$$\#\dot{w}_1 w_2 \cdots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \cdots \#$$

2. Para simular um único movimento, S faz uma varredura na sua fita a partir do primeiro #, que marca a extremidade esquerda, até o $(k+1)$-ésimo #, que marca a extremidade direita, de modo a determinar os símbolos sob as cabeças virtuais. Então S faz uma segunda varredura para atualizar as fitas conforme a maneira pela qual a função de transição de M determina.

3. Se em qualquer ponto S move uma das cabeças virtuais para a direita sobre um #, essa ação significa que M moveu a cabeça correspondente sobre a parte em branco ainda não lida da fita. Portanto S escreve um símbolo branco sobre essa célula da fita e desloca o conteúdo da fita, a partir dessa célula até o # mais à direita, uma unidade para a direita. E aí continua a simulação como antes."

Corolário 3.9 . . . . . .
Uma linguagem é Turing-reconhecível se e somente se alguma máquina de Turing multi-fitas a reconhece.

Prova. Uma linguagem Turing-reconhecível é reconhecida por uma máquina de Turing comum (com uma única fita), o que é um caso especial de uma máquina de Turing multi-fitas. Isso prova uma direção deste corolário. A outra direção segue do Teorema 3.8.
. . . . . .

Máquinas de Turing não-determinísticas

Uma máquina de Turing não-determinística é definida da maneira esperada. Em qualquer ponto em uma computação a máquina pode proceder conforme várias possibilidades. A função de transição para uma máquina de Turing não-determinística tem a forma
$$\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\mathrm{E}, \mathrm{D}\}).$$

A computação de uma máquina de Turing não-determinística é uma árvore cujos ramos correspondem a possibilidades diferentes para a máquina. Se algum ramo da computação leva ao estado de aceitação, a máquina aceita sua entrada. Se você sente a necessidade de revisar não-determinismo, volte para a Seção 1.2 na página 47. Agora mostramos que não-determinismo não afeta o poder do modelo de máquina de Turing.

Teorema 3.10 . . . . . .
Toda máquina de Turing não-determinística tem uma máquina de Turing determinística equivalente.

. . . . . .
Idéia da Prova. Mostramos que podemos simular qualquer MT não-determinística N com uma MT determinística D. A idéia por trás da simulação é fazer com que D tente todos os possíveis ramos da computação não-determinística de N. Se D chega a encontrar o estado de aceitação sobre um desses ramos, D aceita. Caso contrário, a simulação de D não termina.

Vemos a computação de N sobre uma entrada w como uma árvore. Cada ramo da árvore representa um dos ramos do não-determinismo. Cada nó da árvore é uma configuração de N. A raiz da árvore é a configuração inicial. A MT D faz uma busca nessa árvore por uma configuração de aceitação. Conduzir essa busca cuidadosamente é crucial para que D não deixe de visitar a árvore inteira. Uma idéia tentadora, embora ruim, é fazer com que D explore a árvore usando busca em profundidade. A estratégia da busca em profundidade vai até o fundo num ramo antes de voltar e explorar outros ramos. Se D tivesse que explorar a árvore dessa maneira, D poderia descer para sempre num ramo infinito e perder uma configuração de aceitação em algum outro ramo. Daí projetamos D para explorar a árvore usando busca em largura ao invés de busca em profundidade. Essa estratégia explora todos os ramos até a mesma profundidade antes de continuar a explorar qualquer ramo na próxima profundidade. Esse método garante que D visitará todo nó na árvore até que ela encontre uma configuração de aceitação.

Prova. A MT D simuladora tem três fitas. Pelo Teorema 3.8 esse arranjo é equivalente a ter uma única fita. A máquina D usa suas três fitas de uma maneira particular, conforme ilustrado na Figura 3.7. A fita 1 sempre contém a cadeia de entrada e nunca é alterada. A fita 2 mantém uma cópia da fita de N sobre algum ramo de sua computação não-determinística. A fita 3 mantém registro da posição de D na árvore de computação não-determinística de N.

Figura 3.7: MT determinística D simulando a MT não-determinística N

Vamos primeiro considerar a representação de dados na fita 3. Todo nó na árvore pode ter no máximo b filhos, onde b é o tamanho do maior conjunto de escolhas possíveis dado pela função de transição de N. A todo nó na árvore atribuimos um endereço que é uma cadeia sobre o alfabeto $\Sigma_b = \{1, 2, \ldots, b\}$. Atribuimos o endereço 231 ao nó a que chegamos iniciando na raiz, indo para seu 2º filho, indo para o 3º daquele nó, e finalmente indo para o 1º filho daquele nó. Cada símbolo na cadeia diz qual escolha a fazer a seguir quando se simula um passo num ramo da computação não-determinística de N. Às vezes um símbolo pode não corresponder a nenhuma escolha se muito poucas escolhas estão disponíveis para uma configuração. Nesse caso o endereço é inválido e não corresponde a nenhum nó. A fita 3 contém uma cadeia sobre Σ_b. Ela representa o ramo da computação de N a partir da raiz para o nó endereçado por aquela cadeia, a menos que o endereço seja inválido. A cadeia vazia é o endereço da raiz da árvore. Agora estamos prontos para descrever D.

1. Inicialmente a fita 1 contém a cadeia w, e as fitas 2 e 3 estão vazias.
2. Copie a fita 1 para a fita 2.
3. Use a fita 2 para simular N com entrada w sobre um ramo de sua computação não-determinística. Antes de cada passo de N consulte o próximo símbolo sobre a fita 3 para determinar qual escolha fazer entre aquelas permitidas pela função de transição de N. Se nenhum símbolo mais permanece sobre a fita 3 ou se esse escolha não-determinística for inválida, aborte esse ramo indo para o estágio 4. Também vá para o estágio 4 se uma configuração de rejeição for encontrada. Se uma configuração de aceitação for encontrada, *aceite* a entrada.

4. Substitua a cadeia sobre a fita 3 pela cadeia lexicograficamente seguinte. Simule o próximo ramo da computação de N indo para o estágio 2.

Corolário 3.11 . . . . . .
Uma linguagem é Turing-reconhecível se e somente se alguma máquina de Turing não-determinística a reconhece.

Prova. Qualquer MT determinística é automaticamente uma MT não-determinística e portanto uma direção desse teorema segue imediatamente. A outra direção segue do Teorema 3.10.
. . . . . .

Podemos modificar a prova do Teorema 3.10 de modo que se N sempre pára em todos os ramos de sua computação, D sempre vai parar. Chamamos uma máquina de Turing não-determinística um ***decisor*** se todos os ramos param sobre todas as entradas. O Exercício 3.3 pede a você para modificar a prova dessa maneira para obter o seguinte corolário do Teorema 3.10.

Corolário 3.12 . . . . . .
Uma linguagem é decidível se e somente se alguma máquina de Turing não-determinística a decide.

Enumeradores

Como mencionamos em uma nota de pé-de-página anterior, algumas pessoas usam o termo linguagem ***recursivamente enumerável*** para uma linguagem Turing-reconhecível. Esse termo tem origem num tipo de variante de máquina de Turing chamado um enumerador. Frouxamente definido, um enumerador é uma máquina de Turing com uma impressora em anexo. A máquina de Turing pode usar essa impressora como um dispositivo de saída para imprimir cadeias. Toda vez que a máquina de Turing quer adicionar uma cadeia à lista, ela envia a cadeia para a impressora. O Exercício 3.4 pede que você dê uma definição formal de um enumerador. A Figura 3.8 ilustra um esquema desse modelo.

Figura 3.8: Esquema de um enumerador

Um enumerador começa com uma fita de entrada em branco. Se o enumerador não pára, ele pode imprimir uma lista infinita de cadeias. A linguagem enumerada por E é a coleção de todas as cadeias que ela eventualmente imprime. Além do mais, E pode gerar as cadeias da linguagem em qualquer ordem, possivelmente com repetições. Agora estamos prontos para desenvolver a conexão entre enumeradores e linguagens Turing-reconhecíveis.

Teorema 3.13 . . . . . .
Uma linguagem é Turing-reconhecível se e somente se algum enumerador a enumera.

Prova. Primeiro mostramos que se temos um enumerador E que enumera uma linguagem A, uma MT M reconhece A. A MT M funciona da seguinte forma.

M = "Sobre a entrada w:

1. Rode E. Toda vez que E dá como saída uma cadeia, compare-a com w.
2. Se w vem a aparecer na saída de E, *aceite*."

Claramente M aceita aquelas cadeias que aparecem na lista de E.

Agora fazemos a outra direção. Se a MT M reconhece uma linguagem A, podemos construir o seguinte enumerador E para A. Digamos que $s_1, s_2, \ldots$ é uma lista de todas as possíveis cadeias em Σ^*.

E = "Ignore a entrada.

1. Repita o seguinte para $i = 1, 2, 3, \ldots$
2. Rode M por i passos sobre cada entrada $s_1, s_2, \ldots, s_i$.
3. Se quaisquer computações aceitam, imprima o s_j correspondente."

Se M aceita uma cadeia específica s, eventualmente ela aparecerá na lista gerada por E. Na verdade, ela aparecerá na lista uma quantidade infinita de vezes porque M roda do início sobre cada cadeia para cada repetição do passo 1. Esse procedimento dá o efeito de se rodar M em paralelo sobre todas as possíveis cadeias de entrada.

. . . . . .

Equivalência com outros modelos

Até aqui apresentamos diversas variantes do modelo da máquina de Turing e demonstramos que elas são equivalentes em poder. Muitos outros modelos são muito parecidos com máquinas de Turing, enquanto outros são um tanto diferentes. Todos compartilham a característica essencial de máquinas de Turing, a saber, acesso irrestrito a memória ilimitada, distingüindo-os de modelos mais fracos tais como autômatos finitos e autômatos a pilha. Notavelmente, *todos* os modelos com essa característica acabam sendo equivalentes em poder, desde que eles satisfaçam certos requisitos razoáveis.[3]

Para entender esse fenômeno considere a situação análoga para linguagens de programação. Muitas, tais como Pascal e LISP, parecem um tanto diferentes umas das outras em estilo e estrutura. Algum algoritmo pode ser programado em uma delas e não nas outras? Claro que não—podemos compilar LISP para Pascal e Pascal para LISP, o que significa que as duas linguagens descrevem *exatamente* a mesma classe de algoritmos. Assim o fazem todas as outras linguagens de programação razoáveis. A tão difundida equivalência de modelos computacionais se verifica precisamente pela mesma razão. Quaisquer modelos computacionais que satisfaçam certos requisitos razoáveis podem simular um ao outro e portanto são equivalentes em poder.

Esse fenômeno da equivalência tem um corolário filosófico importante. Muito embora existam muitos modelos computacionais diferentes, a classe de algoritmos que eles descrevem é única. Enquanto que cada modelo computacional individual tem um certo grau de arbitrariedade na sua definição, a classe subjacente de algoritmos que ele descreve é natural porque ela é a mesma classe que outros modelos descrevem. Esse fenômeno também tem tido profundas implicações para a matemática, como mostramos na próxima seção.

[3] Por exemplo, um requisito é a capacidade de realizar somente uma quantidade finita de trabalho em um único passo.

3.3 A definição de algoritmo .....

Informalmente falando, um *algoritmo* é uma coleção de instruções simples para levar adiante alguma tarefa. Comum na vida do dia-a-dia, algoritmos às vezes são chamados *procedimentos* ou *receitas*. Algoritmos também desempenham um papel importante em matemática. A antiga literatura matemática contém descrições de algoritmos para uma variedade de tarefas, tais como encontrar números primos e máximos divisores comuns. Na matemática contemporânea algoritmos abundam.

Muito embora algoritmos tenham tido uma longa história em matemática, a noção de algoritmo propriamente dita não estava definida precisamente até o século XX. Antes disso, os matemáticos tinham uma noção intuitiva do que algoritmos eram e se apoiavam naquela noção ao usá-los e descrevê-los. Mas aquela noção intuitiva era insuficiente para adquirir um entendimento mais profundo de algoritmos. A estória a seguir relata como a definição precisa de algoritmo foi crucial para um importante problema matemático.

Os problemas de Hilbert

Em 1900, o matemático David Hilbert proferiu uma hoje-famosa palestra no Congresso Internacional de Matemáticos em Paris. Em sua apresentação, ele identificou vinte-e-três problemas matemáticos e os colocou como um desafio para o século vindouro. O décimo problema na sua lista dizia respeito a algoritmos.

Antes de descrever aquele problema, vamos discutir brevemente polinômios. Um ***polinômio*** é uma soma de termos, onde cada ***termo*** é um produto de certas variáveis e uma constante chamada um ***coeficiente***. Por exemplo

$$6 \cdot x \cdot x \cdot x \cdot y \cdot z \cdot z = 6x^3yz^2$$

é um termo com coeficiente 6, e

$$6x^3yz^2 + 3xy^2 - x^3 - 10$$

é um polinômio com quatro termos sobre as variáveis x, y, e z. Uma ***raiz*** de um polinômio é uma atribuição de valores a suas variáveis tal que o valor do polinômio seja 0. Esse polinômio tem uma raiz em $x = 5$, $y = 3$, e $z = 0$. Essa raiz é uma ***raiz inteira*** porque todas as variáveis são substituídas por valores inteiros. Alguns polinômios têm uma raiz inteira e alguns não.

O décimo problema de Hilbert era para projetar um algoritmo que testa se um polinômio tem uma raiz inteira. Ele não usou o termo *algoritmo* mas sim "um processo conforme o qual ela pode ser determinada por um número finito de operações."[4] É interessante observar que, da forma com que ele fraseou esse problema, Hilbert pedia explicitamente que um algoritmo fosse "projetado." Por conseguinte ele aparentemente assumiu que tal algoritmo tinha que existir—alguém só precisa encontrá-lo.

Como agora sabemos, nenhum algoritmo existe para essa tarefa; ela é algoritmicamente insolúvel. Para os matemáticos daquele período chegar a essa conclusão com seu conceito intuitivo de algoritmo teria sido virtualmente impossível O conceito intuitivo pode ter sido adequado para dar algoritmos para certas tarefas, mas ele era inútil para mostrar que nenhum algoritmo existe para uma tarefa particular. Provar que um algoritmo não existe requer dispor de uma definição clara de algoritmo. Progresso no décimo problema teve que esperar por aquela definição.

[4] Traduzido do original em alemão.

A definição veio nos artigos de 1936 de Alonzo Church e Alan Turing. Church usou um sistema notacional chamado o λ-cálculo para definir algoritmos. Turing o fez com suas "máquinas." Essas duas definições foram demonstradas serem equivalentes. Essa conexão veio a ser chamada a ***tese de Church–Turing***.

A tese de Church–Turing provê a definição de algoritmo necessária para resolver o décimo problema de Hilbert. Em 1970, Yuri Matijasevič, baseando-se em trabalhos de Martin Davis, Hilary Putnam, e Julia Robinson, mostrou que nenhum algoritmo existe para testar se um polinômio tem raízes inteiras. No Capítulo 4 desenvolvemos as técnicas que formam a base para provar que esse e outros problemas são algoritmicamente insolúveis.

Noção intuitiva de algoritmo	é igual a	*Algoritmos de máquinas de Turing*

Figura 3.9: A Tese de Church–Turing

Vamos frasear o décimo problema de Hilbert em nossa terminologia. Fazer isso ajuda a introduzir alguns temas que exploramos nos Capítulos 4 e 5. Seja

$$D = \{p \mid p \text{ é um polinômio com uma raiz inteira}\}.$$

O décimo problema de Hilbert pergunta em essência se o conjunto D é decidível. A resposta é negativa. Por outro lado podemos mostrar que D é Turing-reconhecível. Antes de fazer isso, vamos considerar um problema mais simples. É um análogo ao décimo problema de Hilbert para polinômios que têm apenas uma única variável, tais como $4x^3 - 2x^2 + x - 7$. Seja

$$D_1 = \{p \mid p \text{ é um polinômio sobre } x \text{ com uma raiz inteira}\}.$$

Aqui está uma máquina de Turing M_1 que reconhece D_1:

M = "A entrada é um polinômio p sobre a variável x.

1. Calcule o valor de p com x substituído sucessivamente pelos valores $0, 1, -1, 2, -2, 3, -3, \ldots$ Se em algum ponto o polinômio resulta no valor 0, *aceite*."

Se p tem uma raiz inteira, M_1 eventualmente a encontrará e aceitará. Se p não tem uma raiz inteira, M_1 vai rodar para sempre. Para o caso multivariado, podemos apresentar uma máquina de Turing semelhante M que reconhece D. Aqui, M passar por todas as possíveis atribuições de valor a suas variáveis com valores inteiros.

Ambas M_1 e M são reconhecedores mas não decisores. Podemos converter M_1 para um decisor para D_1 porque podemos calcular limitantes dentro dos quais as raízes de uma única variável polinomial tem que residir e restringir a busca a esses limitantes. No Problema 3.18 lhe será pedido para mostrar que as raízes de tal polinômio têm que estar entre os valores

$$\pm k\frac{c_{\max}}{c_1},$$

onde k é o número de termos no polinômio, $c_{\max}$ é o coeficiente com o maior valor absoluto, e c_1 é o coeficiente do termo de mais alta ordem. se uma raiz não for encontrada dentro desses limitantes, a máquina *rejeita*. O teorema de Matijasevič mostra que calcular tais limitantes para polinômios multivariados é impossível.

Terminologia para descrever máquinas de Turing

Chegamos a um ponto de virada no estudo da teoria da computação. Continuamos a falar de máquinas de Turing, mas nosso foco real de agora em diante é sobre algoritmos. Ou seja, a máquina de Turing serve meramente como um modelo preciso para a definição de algoritmo. Pularemos a extensa teoria de máquinas de Turing propriamente ditas e não gastaremos muito tempo na programação de baixo-nível de máquinas de Turing. Precisamos apenas estar confortável o bastante com máquinas de Turing para acreditar que elas capturam todos os algoritmos.

Com isso em mente, vamos padronizar a maneira que descrevemos algoritmos de máquinas de Turing. Incialmente, perguntamos: Qual é o nível correto de detalhe a dar ao descrever tais algoritmos? Estudantes comumente fazem essa pergunta, especialmente ao preparar soluções a exercícios e problemas. Vamos entreter três possibilidades. A primeira é a *descrição formal* que explicita por completo os estados da máquina de Turing, função de transição, e assim por diante. É o nível mais baixo, mais detalhado, de descrição. A segunda é uma descrição de mais alto nível, chamada a *descrição de implementação*, na qual usamos prosa em português para descrever a maneira pela qual a máquina de Turing movimenta sua cabeça e a forma pela qual ela armazena dados na sua fita. Nesse nível não damos detalhes de estados ou função de transição. A terceira é a *descrição de alto-nível*, na qual usamos prosa em português para descrever um algoritmo, ignorando o modelo de implementação. Nesse nível não precisamos mencionar como a máquina administra sua fita ou sua cabeça.

Neste capítulo demos descrições formais e de nível-de-implementação de vários exemplos de máquinas de Turing. A prática com descrições de mais baixo nível de máquina de Turing ajuda a você entender máquinas de Turing e ganhar confiança em usá-las. Uma vez que você se sente confiante, descrições de alto-nível são suficientes.

Agora estabelecemos um formato e notação para descrever máquinas de Turing. A entrada para uma máquina de Turing é sempre uma cadeia. Se desejamos prover um objeto diferente de uma cadeia como entrada, temos primeiro que representar aquele objeto como uma cadeia. Cadeias podem facilmente representar polinômios, grafos, gramáticas, autômatos, e qualquer combinação daqueles objetos. Uma máquina de Turing pode ser programada para decodificar a representação de modo que ela pode ser interpretada da maneira que pretendemos. Nossa notação para a codificação de um objeto O em sua representação como uma cadeia é $\langle O \rangle$. Se temos vários objetos $O_1, O_2, \ldots, O_k$, denotamos sua codificação em uma única cadeia por $\langle O_1, O_2, \ldots, O_k \rangle$. A codificação propriamente dita pode ser feita de várias maneiras razoáveis. Não importa qual escolhemos, porque uma máquina de Turing pode sempre traduzir uma dessas codificações em uma outra.

Em nosso formato, descrevemos algoritmos de máquinas de Turing com um segmento indentado de texto dentro de aspas. Quebramos o algoritmo em estágios, cada um usualmente envolvendo muitos passos individuais da computação da máquina de Turing. Indicamos a estrutura de bloco do algoritmo com mais indentação. Se a descrição da entrada é simplesmente w, a entrada é tomada como sendo uma cadeia. Se a descrição da entrada é a codificação de um objeto como em $\langle A \rangle$, a máquina de Turing primeiro implicitamente testa se a entrada codifica apropriadamente um objeto para a forma desejada e a rejeita se ela não codifica.

Exemplo 3.14 .....

Seja A a linguagem consisting de todas as cadeias representando grafos não-direcionados que são conexos. Lembre-se que um grafo é ***conexo*** se todo nó pode ser atingido a par-

tir de qualquer outro nó por meio de uma viagem sobre as arestas do grafo. Escrevemos

$$A = \{\langle G \rangle \mid G \text{ é um grafo não-direcionado conexo}\}.$$

A seguir vai uma descrição de alto-nível de uma MT que decide A.

M = "Sobre a entrada $\langle G \rangle$, a codificação de um grafo G:

1. Selecione o primeiro nó de G e marque-o.
2. Repita o estágio seguinte até que nenhum nó novo seja marcado.
3. Para cada nó em G, marque-o se ele está ligado por uma aresta a um nó que já está marcado.
4. Faça uma varredura em todos os nós de G para determinar se eles todos estão marcados. Se eles estão, *aceite*; caso contrário *rejeite*."

Para prática adicional, vamos examinar alguns detalhes de nível-de-implementação da máquina de Turing M. Usualmente não daremos esse nível de detalhe no futuro e nem você precisará fazê-lo, a menos que especificamente solicitado em um exercício. Primeiro, temos que entender como $\langle G \rangle$ codifica o grafo G como uma cadeia. Considere uma codificação que é uma lista dos nós de G seguida por uma lista das arestas de G. Cada nó é um número decimal, e cada aresta é o par de números decimais que representam os nós nas duas extremidades da aresta. A Figura 3.10 ilustra esse grafo e sua codificação.

Figura 3.10: Um grafo G e sua codificação $\langle G \rangle$

Quando M recebe a entrada $\langle G \rangle$, ela primeiro procura determinar se a entrada é uma codificação apropriada de algum grafo. Para fazer isso, M varre a fita para se assegurar de que existem duas listas e que elas estão na forma apropriada. A primeira lista deve ser uma lista de números decimais distintos, e a segunda deve ser uma lista de pares de números decimais. Então M verifica várias coisas. Primeiro, a lista de nós não deve conter repetições, e segundo, todo nó aparecendo na lista de aresta deve também aparecer na lista de nós. Para o primeiro, podemos usar o procedimento dado no Exemplo 3.7 para a MT M_4 que verificar distinção de elementos. Um método semelhante funciona para o segundo item de verificação. Se w passa nessas verificações, ela é a codificação de algum grafo G. Essa verificação completa a checagem da entrada, e M vai para o estágio 1.

Para o estágio 1, M marca o primeiro nó com um ponto sobre o dígito mais à esquerda.

Para o estágio 2, M varre a lista de nós para encontrar um nó n_1 sem-ponto e o destaca marcando-o diferentemente, digamos, sublinhando o primeiro símbolo. Então M varre a lista novamente para encontrar um nó n_2 marcado com um ponto e o sublinha também.

Agora M varre a lista de arestas. Para cada aresta, M testa se os dois nós sublinhados n_1 e n_2 são aqueles aparecendo naquela aresta, podemos usar o procedimento dado no Exemplo 3.7 para a MT M_4 que verificar distinção de elementos. Um método semelhante funciona para o segundo item de verificação. Se w passa nessas verificações, ela

é a codificação de algum grafo G. Essa verificação completa a checagem da entrada, e M vai para o estágio 1.

Para o estágio 1, M marca o primeiro nó com um ponto sobre o dígito mais à esquerda.

Para o estágio 2, M varre a lista de nós para encontrar um nó n_1 sem-ponto e o destaca marcando-o diferentemente, digamos, sublinhando o primeiro símbolo. Então M varre a lista novamente para encontrar um nó n_2 marcado com um ponto e o sublinha também.

Agora M varre a lista de arestas. Para cada aresta, M testa se os dois nós sublinhados n_1 e n_2 são os que aparecem naquela aresta. Se eles o são, M marca com um ponto n_1, remove as marcas de sublinhar, e continua a partir do início do estágio 2. Se eles não o são, M verifica a próxima aresta na lista. Se não existem mais arestas, $\{n_1, n_2\}$ não é uma aresta de G. Então M move o símbolo de sublinhar em n_2 para o próximo nó marcado com um ponto e agora chama esse nó de n_2. Repete os passos neste parágrafo para verificar, como antes, se o novo par $\{n_1, n_2\}$ é uma aresta. Se não existem mais nós marcados com um ponto, n_1 não está ligado a nenhum nó marcado com um ponto. Então M lança os símbolos de sublinhar de tal modo que n_1 é o próximo nó não-marcado com um ponto e n_2 é o primeiro nó marcado com um ponto e repete os passos neste parágrafo. Se não existem mais nós não-marcados com um ponto, M não foi capaz de encontrar nenhum novo nó para marcar com um ponto, portanto ela vai para o estágio 4.

Para o estágio 4, M varre a lista de nós para determinar se todos estão marcados com um ponto. Se eles estão, ela entra no estado de aceitação; caso contrário ela entra no estado de rejeição. Isso completa a descrição da MT M.

. . . . . .

Exercícios

3.1 Este exercício concerne a MT M_2 cuja descrição e diagrama de estados aparecem no Exemplo 3.4. Em cada um dos itens abaixo, dê a seqüência de configurações nas quais M_2 entra quando iniciado sobre a cadeia de entrada indicada:

a. `0`.

b. `00`.

c. `000`.

d. `000000`.

3.2 Este exercício concerne a MT M_1 cuja descrição e diagrama de estados aparecem no Exemplo 3.5. Em cada um dos itens abaixo, dê a seqüência de configurações nas quais M_1 entra quando iniciado sobre a cadeia de entrada indicada:

a. `11`.

b. `1#1`.

c. `1##1`.

d. `10#11`.

e. `10#10`.