Problemas Algoritmicos

O que pode ser computado?

Chegamos a um importante ponto do curso. Vamos agora estudar uma das questões mais fundamentais em Ciência da Computação:

Qual seria o limite do poder computacional de uma máquina?

Fato importante: É possível dar exemplos precisos de problemas que estão além da capacidade de computação.

Problemas de Decisão

Um ***problema de decisão*** é uma questão com um conjunto finito de parâmetros e que, para valores específicos desses parâmetros, tem resposta sim ou não.

EX: - A soma de 3 e 5 é igual a 8?

- A soma de x e y é igual a z?

O primeiro problema acima não tem parâmetros (e, portanto, tem um única ***instância***)

O segundo problema tem 3 parâmetros e tem infinitas ***instâncias*** , cada uma correspondente a uma tripla de valores para *x, y* e *z.*

- Um PD pode ser visto como um conjunto de questões, uma para cada possível combinação de valores dos parâmetros do problema.
- Cada uma dessas questões tem resposta sim ou não, e é chamada uma instância do problema.

Problemas de Decisão

Do ponto de vista da Ciência da Computação, qualquer problema cujo conjunto de instâncias é finito não tem muito interesse, pois pode sempre ser resolvido por meio de um algoritmo que simplesmente obtém a resposta para cada instância do problema diretamente de uma tabela que armazena a solução para cada uma das instâncias.

Problemas de Decisão como Linguagens

Para estudar o que significa um problema de decisão ser solúvel computacionalmente, precisamos de um método para padronizar tais problemas e torná-los fáceis de serem entendidos por um computador. A solução é representar cada instância como um string:

Ex: Uma instância para o problema “z é a soma de x e y”, seria o string “2+2=4?”

Q: Podemos agora definir teoricamente o que significa um problema ser solúvel computacionalmente. Como você definiria isso?

Problemas de Decisão como Linguagens

A: Um problema de decisão **P** é solúvel se existe um programa de computador, cujas possíveis entradas são instâncias desse problema (codificadas como strings), e que sempre pára, produzindo a resposta correta para cada instância. Além disso, a codificação das instâncias deve ser, ela própria, computável.

Em teoria de linguagens: Seja Σ um alfabeto no qual podem ser codificadas as instâncias de **P**. Seja *E* o conjunto das instâncias (codificadas) e *L* o conjunto de instâncias cuja resposta é SIM. Então **P** é dito ***decidível*** (ou solúvel computacionalmente) se *L* é decidível.

Problemas de Decisão
Problemas Básicos

Antes de atacar alguns problemas de decisão fundamentais em Computação, vamos considerar se existem algoritmos para…

- …determinar se um dado string pertence a uma dada linguagem, para linguagens de uma certa classe **C,** como linguagens aceitas por DFA's, NFA's, PDA's etc. ***Problema*** $A_{\mathbf{C}}$
- …determinar se uma dada linguagem, de uma certa classe **C**, é vazia. ***Problema*** $E_{\mathbf{C}}$
- …determinar se duas dadas linguagens, de uma certa classe **C,** são iguais. ***Problema*** $EQ_{\mathbf{C}}$

Cada um desses problemas é, por sua vez, codificado como uma linguagem.

Problemas de Decisão
Exemplo de Codificação

Vejamos como A_{DFA} pode ser codificado.

A_{DFA} é o problema de decidir

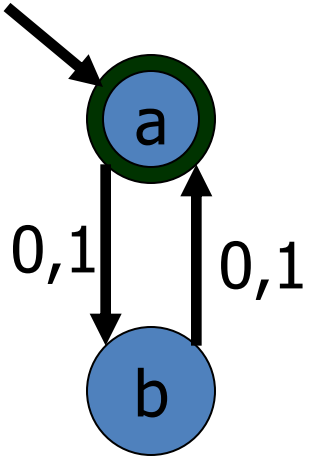
Dado: Um DFA *M* e um string *x*.

Decidir: *M* aceita *x* ?

A linguagem *L* de interesse é a linguagem que consiste de codificações de pares (*M, x*) tais que *M* aceita *x*. A codificação é denotada como "<*M, x* >". Vejamos a seguir como essa codificação pode ser feita.

Problemas de Decisão
Exemplo de Codificação

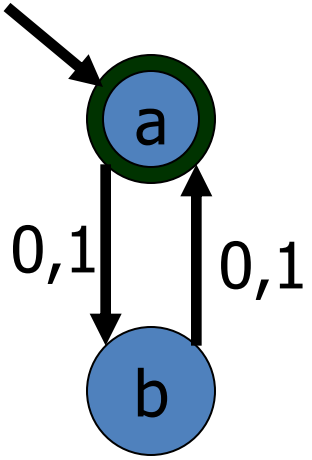
Considere o DFA:



Q: Como ele pode ser representado como um string?

Problemas de Decisão
Exemplo de Codificação

R: Esse DFA seria representado como a 5-tupla:

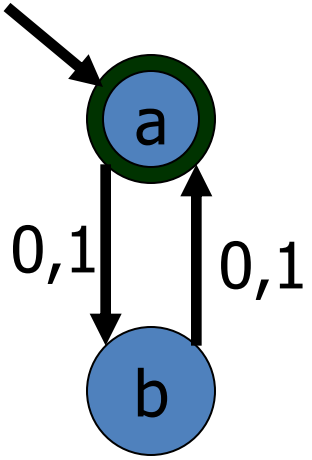


<*M* > =

({a,b},{0,1},{(a,0,b),(a,1,b),(b,0,a),(b,1,a)},a,{a})

Problemas de Decisão
Exemplo de Codificação

R: Esse DFA seria representado como a 5-tupla:



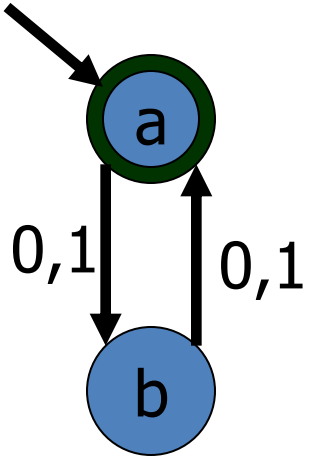
<*M* > =

({a,b},{0,1},{(a,0,b),(a,1,b),(b,0,a),(b,1,a)},a,{a})

Q

Problemas de Decisão
Exemplo de Codificação

R: Esse DFA seria representado como a 5-tupla:



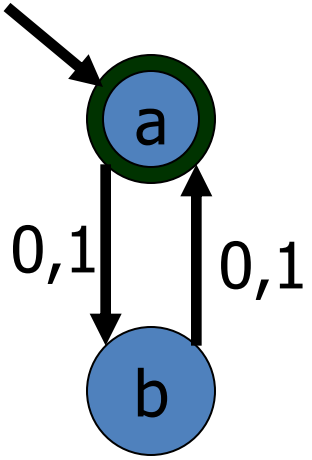
<*M* > =

({a,b},{0,1},{(a,0,b),(a,1,b),(b,0,a),(b,1,a)},a,{a})

Σ

Problemas de Decisão
Exemplo de Codificação

R: Esse DFA seria representado como a 5-tupla:



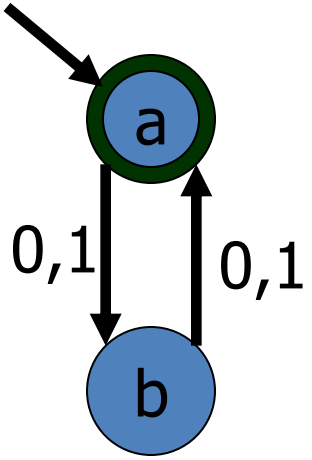
<*M* > =

({a,b},{0,1},{(a,0,b),(a,1,b),(b,0,a),(b,1,a)},a,{a})

δ

Problemas de Decisão
Exemplo de Codificação

R: Esse DFA seria representado como a 5-tupla:



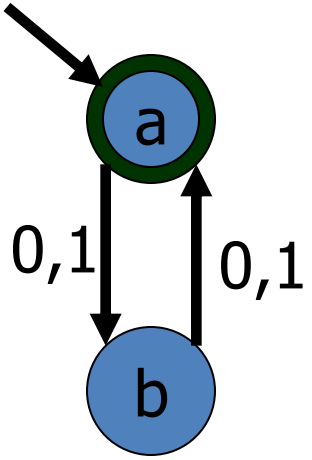
$<M>$ =

({a,b},{0,1},{(a,0,b),(a,1,b),(b,0,a),(b,1,a)},a,{a})

q_0

Problemas de Decisão
Exemplo de Codificação

R: Esse DFA seria representado como a 5-tupla:



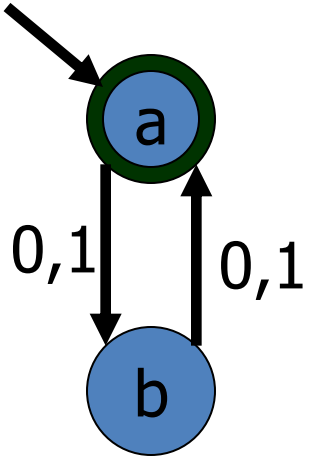
<*M* > =

({a,b},{0,1},{(a,0,b),(a,1,b),(b,0,a),(b,1,a)},a,{a})

F

Problemas de Decisão
Exemplo de Codificação

R: Esse DFA seria representado como a 5-tupla:



<*M* > =

({a,b},{0,1},{(a,0,b),(a,1,b),(b,0,a),(b,1,a)},a,{a})

A codificação <*M,w* > seria obtida adicionando "," e o string de entrada *w*.

Problemas de Decisão
O que de fato fazemos!

Agora que mostramos a idéia de como é possível a definição de problemas de decisão como linguagens (codificando instâncias do problema como strings), vamos, na prática, ignorar essa questão da codificação, lembrando-nos apenas que ela deve poder ser feita computacioalmente.

Mais simplesmente, vamos trabalhar diretamente sobre a estrutura de dados mais natural para a representação do problema (no caso de FA's, um grafo).

A_{DFA}

Q: Como você resolveria A_{DFA} ?

A_{DFA}

R: Simplesmente siga o caminho rotulado pelo string de entrada, a partir do estado inicial. Aceite se ele termina em um estado em *F*. Mais precisamente:

DFAaccept(DFA *M,* String $x_1\, x_2\, x_3 \ldots\, x_n$)

State $q = q_0$ //q_0 definido pela 5-tuple *M*

for(i = 1 to n)

$q = \delta(q,x_i)$ // δ definida pela 5-tuple *M*

if($q \in F$) // *F* definido pela 5-tuple *M*

return ACEPT

else

return REJECT

A_{NFA}

Q: E o problema A_{NFA}?

A_{NFA}

A: Apenas converta o NFA para um DFA e use a solução dada para o problema A_{DFA}.

NFAaccept(NFA *N,* String $x_1 x_2 x_3 \ldots x_n$)

DFA *M* = convertaDeterminista(*N*)

return DFAaccept(*M*, $x_1 x_2 x_3 \ldots x_n$)

A_{NFA}

NFAaccept2(NFA *N,* String $x_1\, x_2\, x_3 \ldots\, x_n$)

StateSet $S = \{q_0\}$ // S como um bit string

for(i = 1 to n)

$S = \delta(S, x_i)$ // é NFA, δ retorna um conj.

if(S e F não são disjuntos)

return ACCEPT

else

return REJECT

E_{DFA}

Vamos focar agora no problema E_{DFA}. Podemos usar o lema do bombeamento para resolver esse problema: suponha que conhecemos o no. de bombeamento *p* do DFA. Então, se não encontrarmos nenhum string com comprimento < *p* que seja aceito pelo DFA, podemos afirmar que o *DFA* não aceita nenhum string.

Q: Porque isso é verdade?

E_{DFA}

R: Suponha que a linguagem seja não vazia. Como verificamos que todos os strings de comprimento < p são rejeitados, o menor string aceito s tem comprimento $\geq p$. Então s é bombeável, podendo ser bombeado de maneira que se obtenha um string mais curto s', o que contradiz a minimalidade de s !

Isso nos dá o seguinte algoritmo:

E_{DFA}

```
DFAempty( DFA M )
   integer p = |Q |
   for (all strings x over Σ with length < p)
        if( DFAaccepts(M,x) == ACCEPT )
             return NONEMPTY
   //only got here if nothing was accepted
   return EMPTY
```

Q: Tempo de execução? Melhorias?

E_{DFA}

R:

Tempo de execução: $O(|\Sigma|^{|Q|})$ –exponencial.

Melhoria: Basta verificar se algum estado de aceitação pode ser atingido a partir do estado inicial, fazendo uma busca em largura (BFS) ou uma busca em profundidade (DFS), do seguinte modo:

E_{DFA}

DFAempty2(DFA M)

State $q = q_0$

Stack S // Inicializa uma pilha LIFO

Set $V = \emptyset$ // Conjunto de estados visitados

S.push(q_0)

while($S \neq \emptyset$)

$q = S$.pop()

if ($q \in F$) return NONEMPTY

$V = V \cup \{q\}$

for (States q' satisfying $q \rightarrow q'$) // um arco

if ($q' \notin V$) S.push(q')

return EMPTY // Apenas chega aqui se F inatingível

Q: O algoritmo usa BFS ou DFS?

E_{DFA}

R: Esse algoritmo usa DFS, já que usa uma pilha.
Se usasse uma fila, seria um algoritmo BFS.

A_{CFG}

A_{CFG} também é decidível. Existe uma solução bastante horrível, usando Forma Normal de Chomsky. Mais adiante veremos um algoritmo bem melhor, de tempo polinomial. (se não houvesse tal algoritmo, compiladores seriam muito ineficientes!)

A forma normal de Chomsky nos dá o seguinte:

LEMA: Seja uma gramática G em CNF. Seja x um string em $L(G)$, e seja n o comprimento de x. Então x é gerado por uma derivação de comprimento $2n$-1, se $n > 0$.

A_{CFG}

LEMA: Seja uma gramática G em CNF. Seja x um string em $L(G)$, e seja n o comprimento de x. Então x é gerado por uma derivação de comprimento $2n-1$, se $n > 0$.

Prova. As primeiras $n-1$ produções são da forma $A \rightarrow BC$ e nos dão o comprimento correto. As últimas n produções são da forma $A \rightarrow a$ e derivam x.

Isso leva ao seguinte algoritmo:

A_{CFG}

CFGaccept(CFG G, String $x=x_1\,x_2\,x_3 \ldots\, x_n$)

CFG G' = ChomskyNormalForm(G)

for (all derivations from start variable of G' of length $\leq 2n+1$)

if (derivation resulted in x)

return ACCEPT

return REJECT

Q1: Porque isso funciona se $x = \varepsilon$?

Q2: Qual a ordem complexidade?

A_{CFG}

A1: Funciona para $x = \varepsilon$ em razão da cláusula "$\leq 2n+1$".

A2: Exponencial. Se considerarmos o tempo da conversão para CNF, esse algoritmo é tremendamente ineficiente.

Exercícios

1. EQ_{DFA}
2. E_{CFG}
3. $CARD_{\text{DFA}}$