Teoria de Problemas

Jorge Muniz Barreto
UFSC-INE
Curso:Teoria da Computação

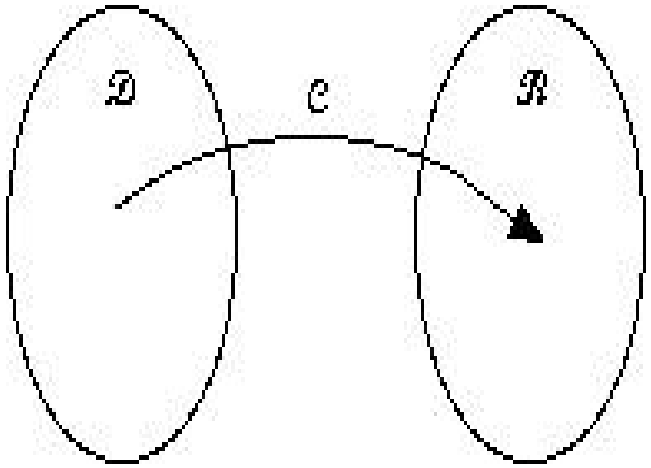
Que é um problema?

- **Enunciado:**

George Polya:

- Quais são os dados (D)?
- Quais são as respostas possíveis (R)?
- Qual a condição (C)?

- **Caracterização**



Definição de Problema

- Um problema é o objeto matemático P=<D,R,c>, consistindo de dois conjuntos não vazios, D os dados e R os resultados possíveis e de uma relação binária c, a condição, que caracteriza uma solução satisfatória, associando a cada elemento do conjunto de dados a Solução única desejada

Exemplo: diagnóstico médico

Um problema de diagnóstico médico P envolve:

- O conjunto de dados disponível d ε D, onde d são os dados (observação da anamnese, sintomas, resultados de laboratório, etc) que pertencem ao conjunto D de dados possíveis.
- R é o conjunto de doenças possíveis.
- A condição que caracteriza uma solução satisfatória consiste em encontrar o par <d,r> onde
- r R é o diagnóstico desejado.

Exemplo: raiz de polinômio

- solução do problema da busca das raízes de um polinômio com coeficientes reais consiste em associar a cada conjunto de coeficientes de um polinômio particular p(x) de grau n, n números complexos c_n de modo a satisfazer a condição de que o valor de p(x) fazendo x = c para todo n seja nulo.

Como Definir uma Função?(1/5)

- **Enumeração exaustiva**

  Neste caso fornece-se todos os conjuntos de pares, dado, resultado. Evidentemente, este modo de definir uma função, só se aplica no caso que o conjunto de dados é finito.

- **Exemplo:** seja uma agenda de telefones. Ela pode ser considerada como a função que associa a cada nome de pessoa seu telefone.

Como Definir uma Função?(2/5)

Declarativamente:

Definir declarativamente um problema é dar propriedades que devem ser satisfeitas pela solução do problema.

Exemplo 1: Dado um número real associa dois números cuja soma de seus quadrados \'e igual ao número real dado. A solução pode ser visualizada como um círculo, centrado na origem de um plano com coordenadas ortonormais (eixos ortogonais e de mesma escala), de raio igual ao número dado.

Como Definir uma Função?(3/5)

- **Declarativamente-Exemplo 2**:

  Seja a função característica do conjunto das equações diofantinas de quarta órdem que tem solução. Ora a partir de 3 sabe-se não haver teorema permitindo saber se o problema tem ou não solução. Logo, o que resta é tentar todos as possibilidades... e como existem infinitos números inteiros não se pode ter certeza, se calculando o problema tem solução ou ainda não foi achada ou não tem solução!

Como Definir uma Função?(4/5)

- **Por um algoritmo:**

  Um programa de computador define a correspondência entre dados e resultados sempre que ele para conseguindo chegar a uma solução. Portanto um programa pode ser considerado como um modo de definir um problema.

- **Exemplo:** Formulário de Imposto de Renda em um País com leis mais complicadas que o nosso...

Como Definir uma Função?(5/5)

- **Por exemplos:**

  Pode-se reconhecer que, neste caso, a solução não é única: todas as funções que sejam iguais dentro da região em que o problema é definido são válidas. Trata-se de fazer uma aproximação.

- Costuma-se empregar redes neurais com aprendizado supervisionado. Usam-se os exemplos para treinar a rede e obtem-se valores estimados da solução para os outros valores usando a propriedade de generalização das redes

Computabilidade

- Intuitivamente uma função é dita computável se é possível calcular seu valor, dado qualquer elemento do seu domínio.

- Será toda função, bem definida, computável?

NEM SEMPRE!!!

Computabilidade

- **Enumeração**

  Sempre computável, basta ler o segundo elemento do par.

- **Por programa:**

  Impossível saber se o programa para: não computável.

- **Exemplos:**

  Obtem-se aproximação da solução. Caso excelente para tratamento por redes neurais.

Definição 1.2 *Resolver um problema definido declarativamente é construir a função T que acha a sua definição algorítmica, e executar o algoritmo achado. Assim trata-se de achar T onde:*

$$T : \Delta \rightarrow \aleph$$

sendo: T: *transformador de forma de definição de problemas*

Δ : *conjunto de definições declarativas de problemas*

$\aleph$: *conjunto de algoritmos*

Computabilidade

Programa constante

Read x;

While x ≠ 10 do

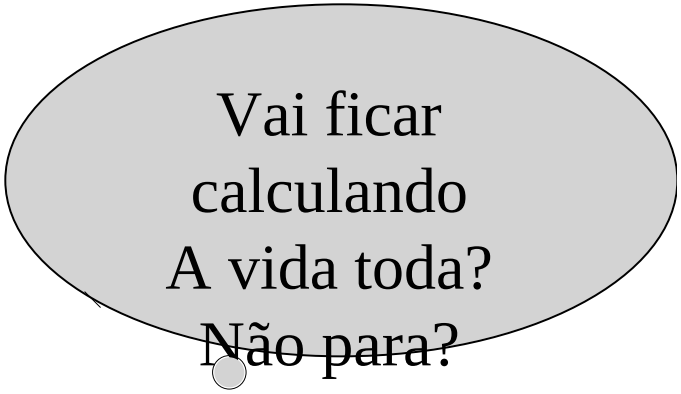
x := x + 1;

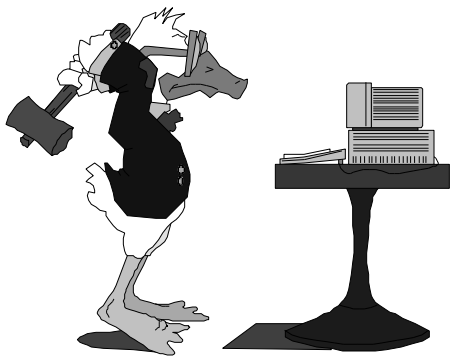
Print x;

End;

****

Ora para x>10 o programa não para!





Computabilidade parcial

- Uma função é parcialmente computável se é possível calcular seu valor para apenas alguns elementos do seu domínio de definicão.
- O exemplo anterior era parcialmente computável.

Decidibilidade

- Decidibilidade é o caso particular de computabilidade quando a função só admite dois valores.
- Quando se fala se um problema é solúvel tem-se um problema de decidibilidade.
- Um problema é parcialmente dicidível se ele é decidível para um subconjunto próprio do seu conjunto de argumentos admissíveis.

Problema da Parada (1/5)

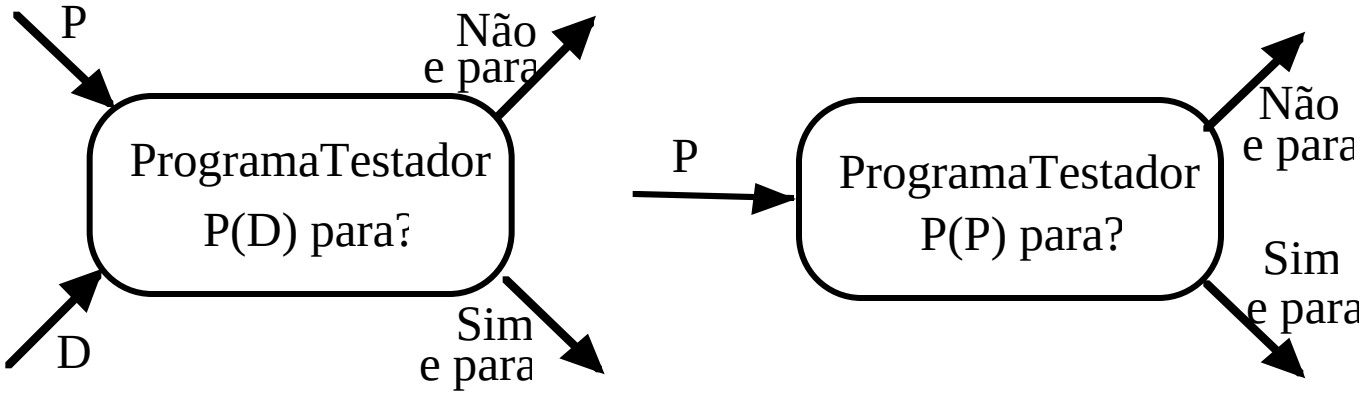
- Trata-se em outras palavras decidir se um programa é um algoritmo, ou seja, um programa que acaba.
- Se o número de dados é finito, o problema consiste em verificar para todos os dados.
- Caso contrário, é impossível provar que ele para para qualquer dado do conjunto de dados possíveis.

Problema da Parada (2/5)

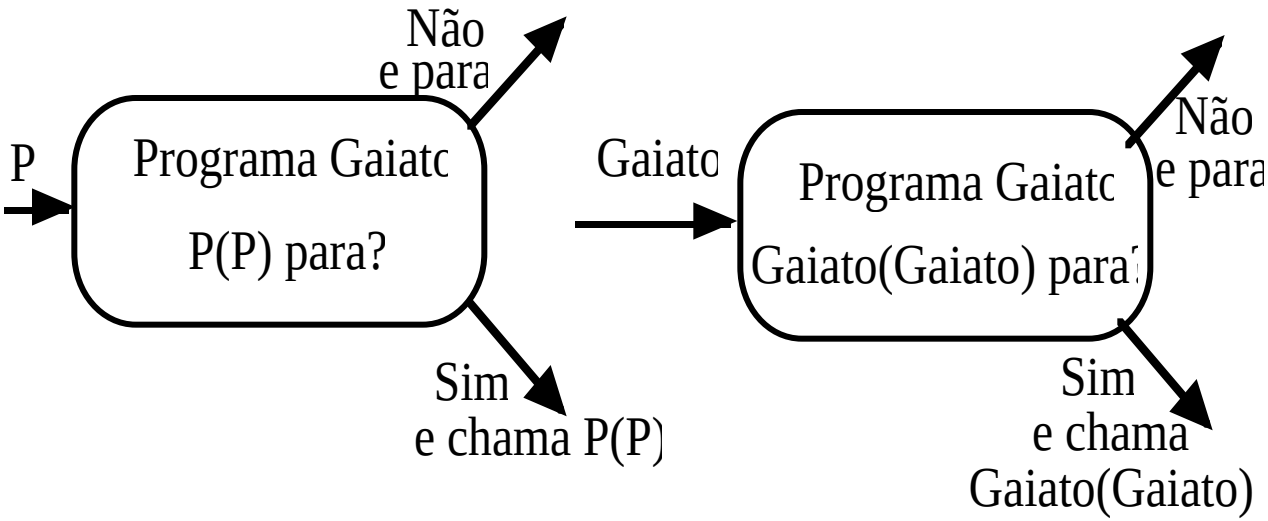
- Admita-se que exista um programa “Testador” que recebe como dados um programa a testar se para “P” e o seu conjunto de dados “D”.
- A saida de “Testador” que sempre para sera “Sim” ou “Não” conforme “P” para ou não para todos os dados de “D”.

Problema da Parada (3/5)

- Nada impede de considerar “P” como dado de “P” e assim tem-se o caso de Testador testando se “P(P)” para. Este é o âmago da demosntração por diagonalizacão.



Problema da Parada (4/5)



Programa Gaiato
leia (P)
se Testador P(P) = "Não" **então** retorne "Não" e pare
senão retorne "Sim" e chame Gaiato

Fim Gaiato

Problema da parada (5/5)

- **Conclusão**:
- Se Gaiato(Gaiato) para, chama Gaiato(Gaiato), logo não para e se a resposta é que não para , então para. Conclui-se portanto que esta impossibilidade foi encontrada por uma das premissas ser falsa. Como a única premissa feita foi a da existência do programa ``Testador", pode-se afirmar não existir tal programa.

Equações Diofantinas (1/2)

- As equações diofantinas tem este nome em homenagem a Diofantus, matemático grego que estudou sistemas de equações algébricas onde os coeficientes são números naturais e se buscam Soluções no conjunto dos naturais.
- No caso de uma equação a uma incógnita, do tipo:

$$ay = bx$$

- a condição para existência de solução é
- que $a \neq 0$ e "b" divisível por "a". Neste caso existe uma infinidade de soluções dadas por:

$$y = \frac{b}{a} x \quad \text{onde} \quad x = 0,1,2,3\ldots$$

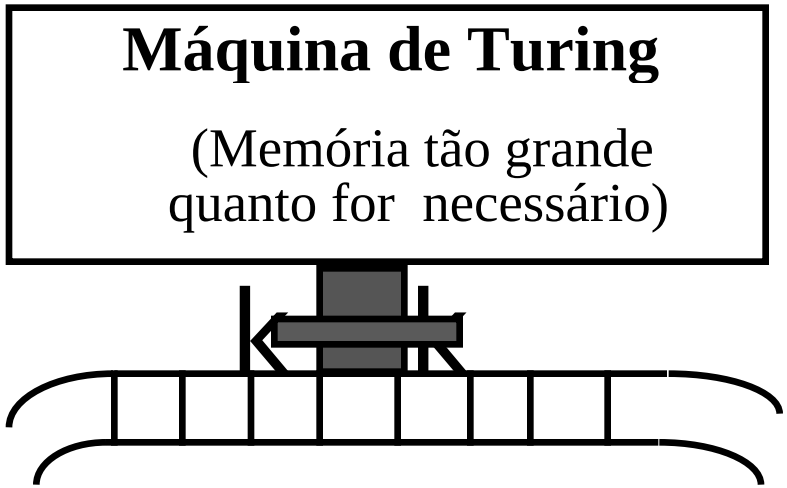
Equacões Diofantinas (2/2)

- Para o caso de equações lineares simultâneas não existe teorema assegurando a existência de solução a partir de 3 incógnitas.

E assim aparece o nosso exemplo!

Máquina de Turing

- **Hipótese de Church-Turing:**
- Toda função computável pode ser calculada pela Máquina de Turing. As não computáveis não podem.
- A **Máquina de Turing** tem uma unidade de controle, uma fita que serve de entrada e saída e uma memória ilimitada.

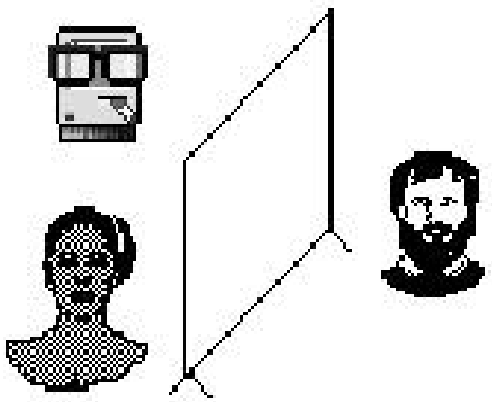


Máquina de Turing

- **Ações:**
- ir para um novo estado (modificar a memória da máquina);
- escrever algo na fita;
- mover a cabeça de leitura para esquerda ou para direita.

- **Em função de:**
- estado em que se encontra a máquina;
- símbolo de entrada;

Teste de Turing

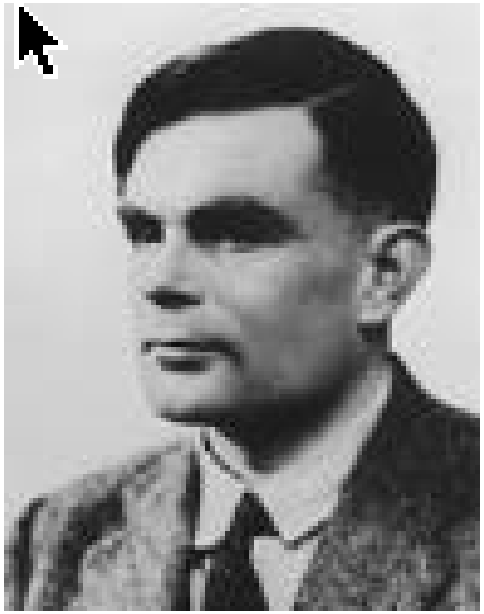


- Foi o idealizador do Teste de Turing de Inteligência.
- Um interrogador faz perguntas a um computador e a um ser humano sem saber quem é quem.

Ganha o interrogador se em um tempo razoável puder dizer com argumentos convincentes quem é quem. Computador e seu par ganham, sendo declarados inteligentes se o interrogador não conseguir.

Alan Mathison Turing (1912-1954)

- Durante a segunda guerra foi convocado pelo governo inglês tendo trabalhado de 1939/45 no Ministerio de Relações Exteriores em trabalho confidencial que até hoje mantém esse status.
- Em homenagem a seus trabalhos em computação, deu seu nome ao mais prestigioso prêmio dado anualmente pela ACM.



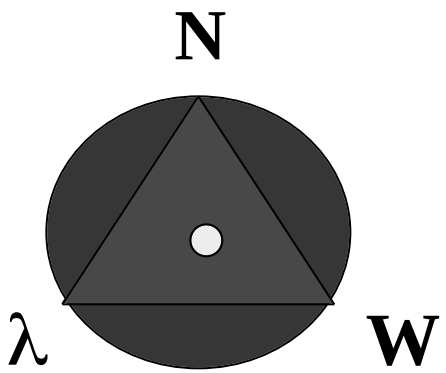
Suicidou-se em 1954, vítima da intolerância

Máquina de Turing

- O grande interesse da Máquina de Turing é oferecer um meio experimental de verificar a computabilidade de um problema. Basta verificar se esta ela pode resolve-lo.

Equivalentes:

- Cálculo λ
- Linguagens While
- Neurocomputadores



Maquina de Turing e Sequencial

- Tanto a Maquina de Turing quanto a Máquina Sequencial tem estado, entrada, saida, etc…
- Qual a difereça entre as duas que faz a Máquina de Turing mais potente?
- A Máquina de Turing é capaz de escrever na sua entrada modificando-a.

Complexidade

- Computabilidade diz respeito a se um problema, modelado como função pode ou não ser resolvido.
- Complexidade diz respeito à quantidade de recursos necessários para resolver um problema.

Complexidade

- Normalmente quando se fala de complexidade se pensa em computadores digitais, ou CBI como sera visto mais adiante.
- Os recursos mais pensados são:
  - **Memória**
  - **Tempo.**

Complexidade

- A complexidade de um problema, com relação a um conjunto bem definido de recursos, é definida como aquela que considera o modo mais parcimonioso de uso de recursos conhecido para a solução do problema.

Complexidade (Importância)

- Se fosse perfeitamente conhecido como determinar a complexidade de um problema no caso das quatro abordagens CA (Computação algoritmica), IAS, IAC, IAE e IAH, seria possível, dado um problema, antes de tentar resolvê-lo, determinar estas complexidades e escolher aquela que fosse a menor, usando-a na resolução do problema.

Complexidade

- Um algoritmo é dito de complexidade *linear* quando a quantidade de recursos para sua execução aumenta proporcionalmente à quantidade de dados envolvida no enunciado do problema.
- polinomial quando a quantidade de recursos para sua execução aumenta mais devagar do que algum polinômio função da quantidade de dados envolvida no enunciado do problema.

Complexidade

- Um algoritmo polinomial é dito de grau N, se existe um polinômio de grau N que cresça com os dados mais rapidamente que cresce a quantidade de recursos necessária à execução do algoritmo e não existe polinômio de grau N-1 com tal característica.

Mostrar que a multiplicaçnao de matrizes, usando a definicão, é dada por:

$$(m \times n \times p \quad m \times n \times (p-1)) \approx 2n^3$$

Complexidade

- Um problema é dito **NP-Completo** se não se conhece algoritmo de órdem polinomial capaz de resolvê-lo.

- Problemas ditos NP-Completos são abordados com solucões aproximadas em IA, usando heurísticas.

NP-Completos

- Problemas NP-Completos são geralmente sinônimo de problemas cuja solucao exata é computável mas se o número de variáveis aumentar se torna de solução impossível.

- Nunca se provou que não existe algoritmo polinomial que resolva os problemas NP-completos.

NP-Completos (Exemplo)

- **Caixeiro viajante:**
- Um caixeiro viajante deve visitar n cidades. Ele conhece o preço do deslocamento entre cada par de cidades. Para economizar, deseja-se conhecer o percurso que lhe permita visitar todas as cidades, voltar a origem e de menor custo.
- Apesar da aparência de pouca utilidade este é o problema a ser resolvido por um caminhão tanque de distribuição de petróleo.

Computabilidade Conexionista

- Chama-se *neurocomputador* um computador em que o funcionamento interno é feito por redes neurais munido de dispositivos de entrada e saída. Até o presente momento tudo foi feito com computadores digitais em mente, os quais tem essencialmente no uso de instruções sua diferença dos neurocomputadores. Por esta razão, e para evitar confusão, eles serão chamados de *Computadores Baseados em Instruções* ou CBI (IBC em inglês).

Computabilidade Conexionista

- Com o advento dos neurocomputadores veio a pergunta:

“Será que algum problema que não podia ser resolvido pela Máquina de Turing, ou por um CBI, pode ser resolvido por um neurocomputador?”

Computabilidade Conexionista

Em um CBI tem-se:

1 - o computador virtual (circuitos e programas),

2 - o ato de fazer o computador apto a resolver um problema específico (carregar o programa na máquina),

3 - resolver o problema (executar o programa).

Em Neurocomputador:

1 - a rede de neurônios com entradas e saídas (simulado ou materialmente implementado),

2 - um meio de fixar os pesos das conexões, muitas vezes usando um algorítmo de aprendizagem (equivalente a carregar o programa),

3 - usar a rede educada e resolver o problema com os dados a serem usados na entrada da rede (identico à rodar o programa).

Computabilidade Conexionista

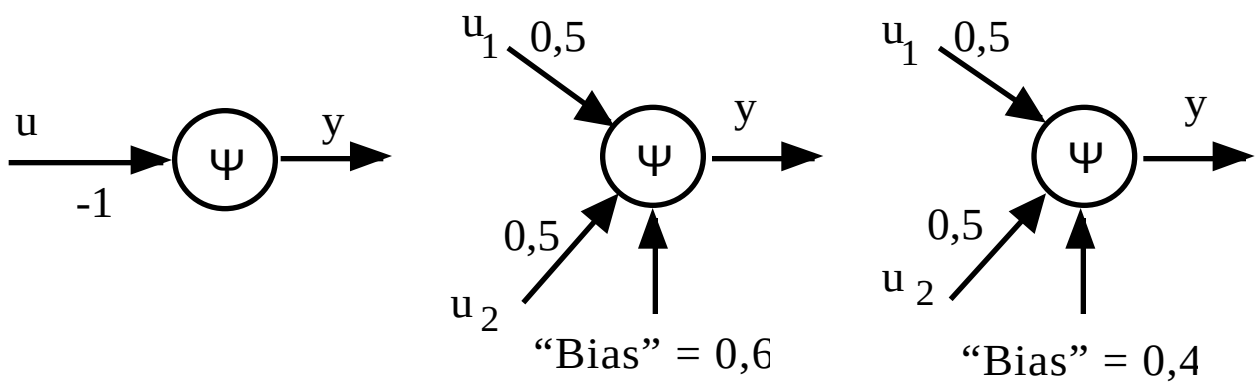
A computabilidade de um problema depende dos pontos 1 e 2. Com efeito, a possibilidade de resolver um problema depende do apoio material que se dispõe e se existe um programa (caso de um CBI) ou se existe um conjunto de pesos de conexões (caso de um neurocomputador) capaz de resolver o problema. Por outro lado a complexidade do problema depende do ponto 3, ou seja rodar o programa ou excitar a rede com os dados a serem usados.

Computabilidade Conexionista

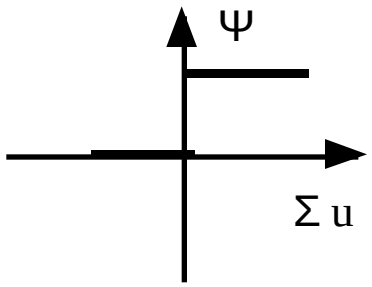
- **Teorema:** Todo problema que pode ser resolvido por um CBI poderá ser resolvido, por uma RNA munida de convenientes dispositivos de entrada e saida.
- **Prova**:
- A prova se baseia em que pode-se construir um CBI com redes neurais. Basta lembrar que com redes neurais é possível construir circuitos lógicos.

Computabilidade Conexionista

- **NÃO** **E** **OU**

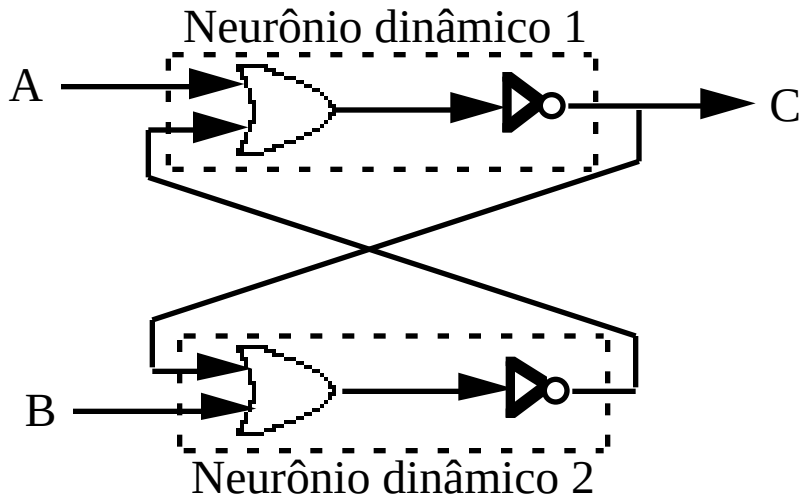


Circuitos da lógica combinacional



Complexidade Conexionista

- **Circuito fundamental da lógica sequencial:**



Computabilidade Conexionista

- Todo problema que pode ser resolvido por uma RNA pode ser aproximado, com a precisão desejada por um CBI.
- **Justificativa**:simular uma RNA em um CBI envolve apenas produto escalar e cálculo de uma função. Pode-se portanto fazer programa para simular qualqueer rede em um CBI.

  Como no CBI existe quantização das variáveis em jogo, a simulação é uma aproximação.

Complexidade Conexionista

- Embora pouco seja conhecido sobre complexidade quando se usa um neurocomputador, sabe-se que em termos de complexidade as coisas são diferentes em termos de CBI e neurocomputadores.
- Para ver que as coisas são diferentes basta considerar um exemplo simples. Seja o caso de um circuito implementando uma RNA direta síncrona com três camadas. Suponha-se ainda que ela foi treinada para associar padrões (por exemplo um sistema especialista de diagnóstico associando sintomas aos correspondentes diagnósticos). Como não há retroações, o tempo para a rede dar uma resposta é sempre o mesmo: três ciclos de relógio! E isso para qualquer número de sintomas e doenças.

Complexidade Conexionista

- Existem muito poucos estudos sobre complexidade de problemas voltada para a solução por neurocomputadores. Note-se que não se trata de falar da complexidade de problemas ligados ao aprendizado das RNA, tais como a complexidade de um algoritmo de aprendizado de redes, que tem sido tratado por vários autores e é reconhecido como problema importante.

Complexidade Conexionista

- A complexidade a que o presente estudo se refere é a quantidade de recursos em termos de RNA necessãrios para poder resolver um determinado problema, eventualmente considerando-se uma certa precisão. Estes recursos incluem o tipo de rede a ser escolhido, a topologia da rede, etc.

Complexidade? Computabilidade?

- Algumas vêzes estes dois conceitos se misturam… Seja o Teorema: “Uma rede direta com duas camadas de entrada e saida não consegue resolver problemas linearmente não separáveis.”
- Como não consegue resolver poderia indicar ser um problema de computabilidade…
- Falso! Pensando assim, guardar 200 telefones seria problema de computabilidade, se considerasse meu celular...

Livro Perceptrons, Minsky, Papert

- O livro citado é uma obra remarcável na teoria da complexidade conexionista, por dar guias de estruturas e problemas que podem ser resolvidos por cada uma.
- Em particular o problema da separabilidade linear é notório.

Complexidade de RNA

- **Teorema 1**:

  Toda RNA constituída apenas de neurônios estáticos, incluindo ciclos (ou seja, com retroação), é equivalente a uma outra rede estática sem ciclos.

- **Teorema 2:**

  Toda rede direta, com topologia por camadas, com neurônios lineares é equivalente a uma rede linear contendo apenas duas camadas de neurônios separadas por uma camada de conexões.

Paradigma de Programação

- Programar um neurocomputador pode ser considerado como um novo paradigma de programação, em que não é necessário nem definir o algoritmo nem definir o problema precisamente. Basta introduzir no computador relações entre conceitos ou usar exemplos de problemas parecidos já resolvidos para serem usados na fase de aprendizado. A RNA, usando sua capacidade de generalização se torna capaz de resolver o problema desejado.

Tipos de Problemas

- Problemas Algorítmicos Numéricos;
- Problemas Algorítmicos Não-Numéricos;
- Problemas Não Algoritmicos;
- Heurísticas.

Características de Problemas

- São conhecidos os passos para achar a solução?
- O problema é decomponível?
- Passos para as soluções podem ser desfeitos?
- O universo é predizível?
- Uma boa solução é relativa ou absoluta?
- O conhecimento disponível é consistente?
- Qual a importância do conhecimento?

São conhecidos os passos para achar a solução?

- o problema é suficientemente bem definido que se pode dizer como vai ser a solução.
- Por exemplo, no problema do jogo do 8, sabe-se exatamente o que é possivel fazer e como fazer. Entretanto seja identificar se uma assinatura em um cheque é ou não falsa. Ora, neste caso quem faz o exame usa técnicas difíceis de serem expressas por regras fixas.

O problema é decomponível?

Agora temos uma brilhante solução!
Só falta descobrir qual o problema!