Técnicas de Projeto (Parte 3)
Projeto e Análise de Algoritmo

Pontifícia Universidade Católica de Minas Gerais

Técnicas de Projeto

- Técnicas de retrocesso

Retrocesso (Backtracking)

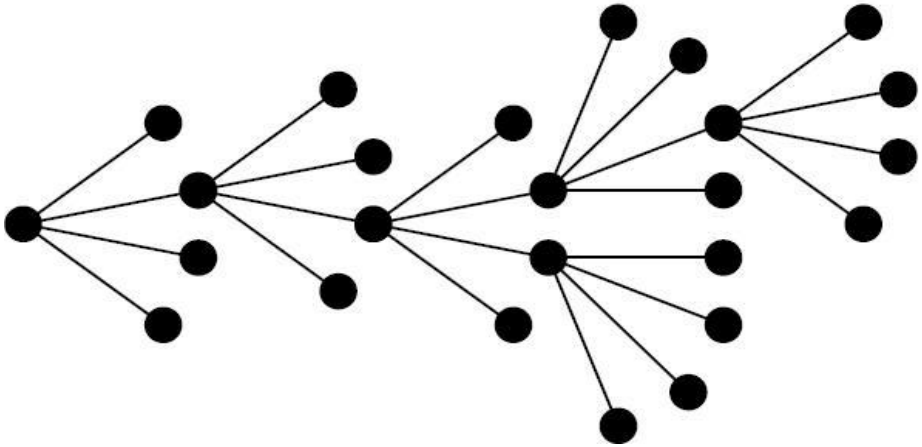
- É um refinamento da Busca Exaustiva ou Força Bruta
  - Não examina explicitamente todas as soluções, pois algumas podem ser eliminadas sem serem examinadas.
- Pode ser aplicado a problemas que possuem a propriedade de soluções candidatas parciais
  - Soluções que são construídas gradativamente, avaliadas a cada passo;
  - Também deve ser possível a realização de testes rápidos para determinar se uma solução parcial pode ser completada até uma solução válida.

Retrocesso (*Backtracking*)

- A técnica de *backtracking* consiste em:
- Construir soluções adicionando um componente de cada vez.
- Avaliar estas soluções candidatas parcialmente construídas:
  - Se uma solução parcialmente construída puder continuar a ser desenvolvida, sem violar as condições do problema, pega-se a primeira opção remanescente.
  - Se não houver opção para o próximo componente, o algoritmo retrocede para trocar o último componente inserido pela próxima opção.

Retrocesso (Backtracking)

- O processo de tentativa gradualmente constrói e percorre uma árvore de sub-tarefas.
- Algoritmos tentativa e erro não seguem uma regra fixa de computação:
  - Passos em direção à solução final são tentados e registrados.
  - Caso esses passos tomados não levem à solução final, eles podem ser retirados e apagados do registro.



Retrocesso (*Backtracking*)

- Algumas aplicações
  - Problemas de Satisfação de Restrições (e.g. Palavras cruzadas);
  - Parsers;
  - Linguagens de Programação Lógica;

Retrocesso (*Backtracking*)

- Enumera um conjunto de soluções parciais candidatas
  - As quais podem ser completadas até uma solução real
  - Cada solução parcial é um vértice em uma árvore
    - Soluções parciais adjacentes diferem entre si por apenas uma extensão.

Retrocesso (*Backtracking*)

- O algoritmo percorre a árvore recursivamente em profundidade
  - A cada vértice, é checado se a solução pode ser completada
  - Caso não possa, toda a subárvore com raiz neste vértice é podada;
  - Caso possa, é checado se já não se trata de uma solução completa;
    - Recursivamente a subárvore também é enumerada.

Retrocesso (*Backtracking*)

- Composição
  - Procedimento de verificação de extensão
    - Verifica se uma solução parcial pode ser estendida
  - Procedimento de verificação de solução completa
    - Verifica se uma solução parcial é completa
  - Procedimento de Extensão
    - Incrementa uma solução parcial
  - Critério de Parada
    - Determina o fim da execução do backtracking

Retrocesso (*Backtracking*)

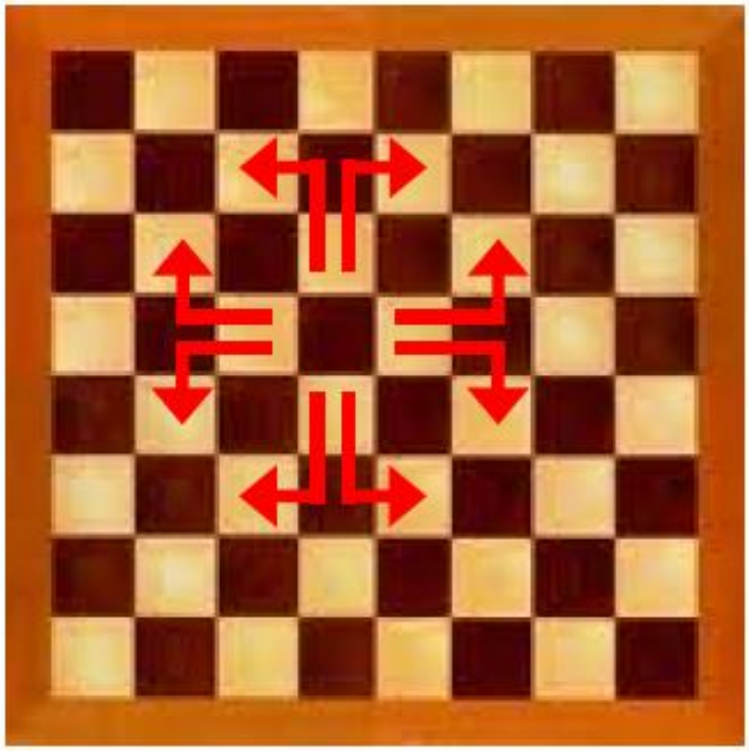
- Os procedimentos de verificação de possibilidade de extensão e solução completa devem ser "leves", pois são utilizados diversas vezes
- Especificamente, o procedimento de verificação de possibilidade de extensão deve ser eficiente, pois quanto mais soluções candidatas houver, mais o *backtracking* se aproxima da busca exaustiva
- O critério de parada deve ser especificado
  - Primeira solução
  - Melhor solução
  - etc.
- Influência no custo computacional.

Retrocesso (*Backtracking*)

- É conveniente construir uma árvore para acompanhar o processo de escolha de opções. As folhas representam nós que não podem levar a uma solução ou soluções completas.
- O espaço de estados representado pela árvore é explorado através de uma busca em profundidade.

Exemplo: O Passeio do Cavalo

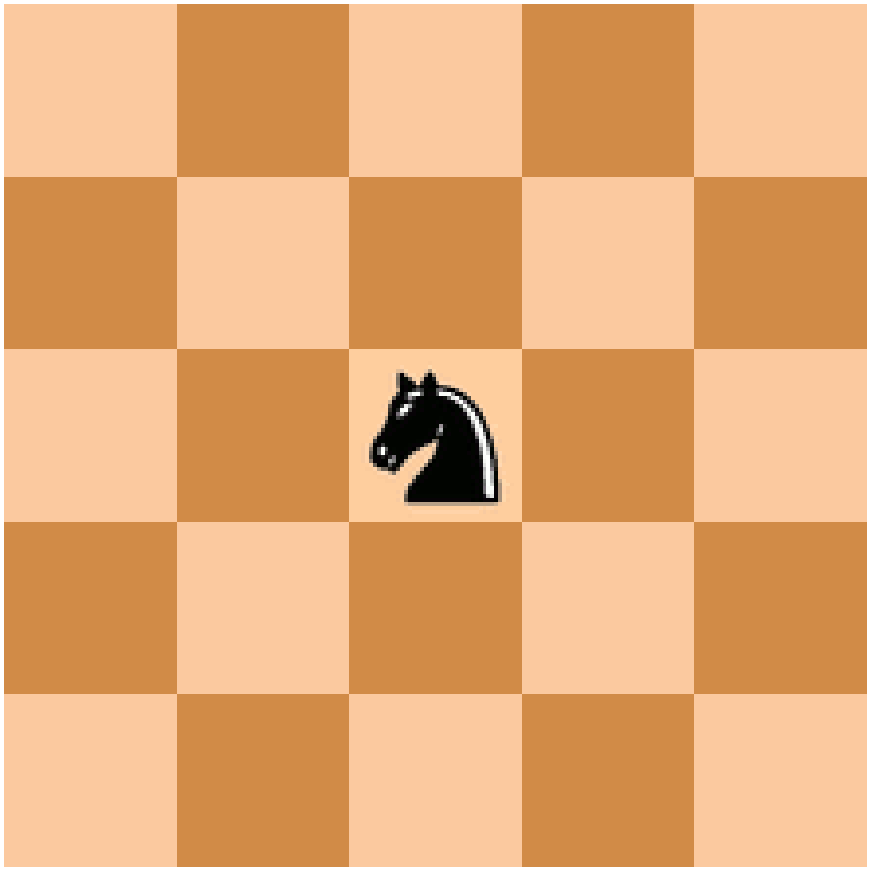
- Problema matemático que envolve os movimentos da peça cavalo do xadrez
  - O cavalo se move em 'L'



Exemplo: O Passeio do Cavalo

- O problema consiste em, a partir de uma posição inicial, construir um circuito em um tabuleiro de xadrez de dimensão ***n*x*n***, de forma que cada casa seja visitada exatamente uma vez
  - No final, retorna-se a posição original;
  - O valor de ***n*** varia
- É uma instância de ciclo Hamiltoniano em grafos, se considerarmos cada casa como um vértice
  - Adjacências somente entre casas acessíveis com o movimento do cavalo.

Exemplo: O Passeio do Cavalo



Exemplo: O Passeio do Cavalo

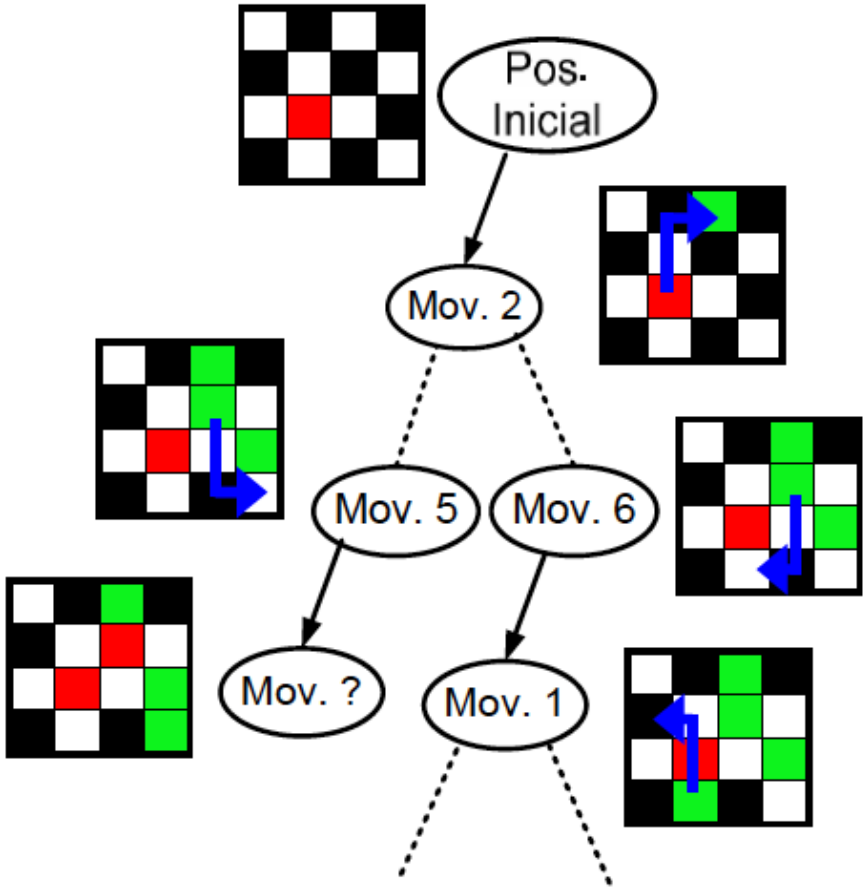
- Procedimento de verificação de extensão
  - Verifica se existe alguma casa não visitada acessível
- Procedimento de verificação de solução completa
  - Verifica se todas as casas foram visitadas.
- Procedimento de Extensão
  - Visita a próxima casa acessível ainda não visitada.
- Critério de Parada
  - Primeira solução completa.

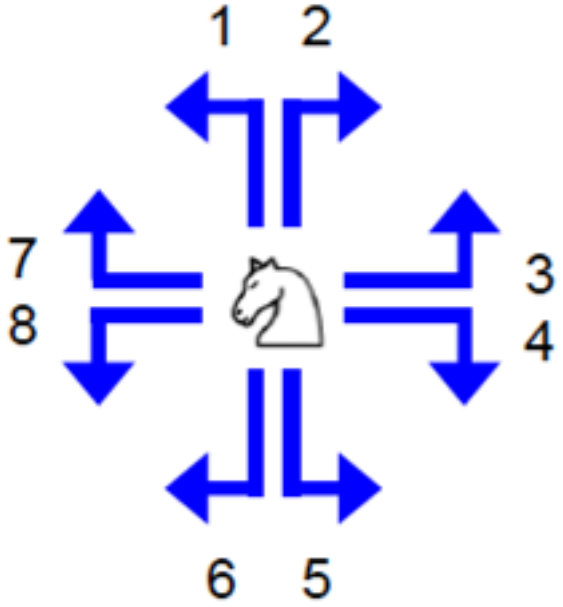
Exemplo: O Passeio do Cavalo

Busca(*Solucao* S)

1. **Enquanto** (!Sucesso **ou** !Solucao_Completa)
2. Seleciona próximo movimento;
    1. **Se** (Existe_Extensao)
        1. Registra o sucesso;
        2. Registra o caminho;
        3. Estende_Solucao(S);
    2. **Senao**
        1. Apaga o registro anterior do caminho;
        2. **Retorna**;

Exemplo: O Passeio do Cavalo





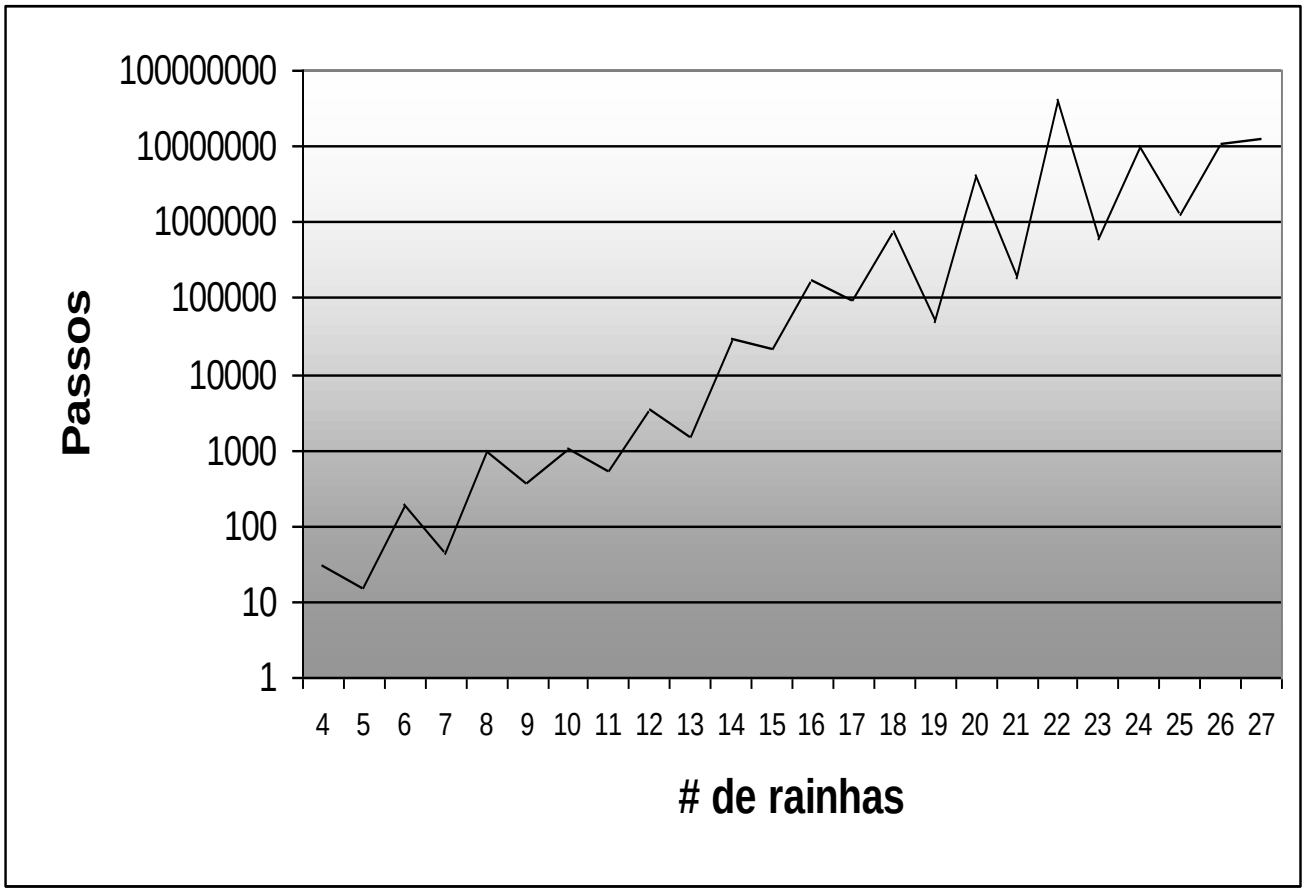
Exemplo: O Passeio do Cavalo

- Em problemas com múltiplas soluções e que apresentam subestrutura ótima é possível realizar podas na árvore de busca
  - Ao encontrarmos uma solução completa, ela vira o valor de referência
  - Continuamos a buscar uma solução melhor
  - Porém, se uma solução parcial possuir valor que ultrapassar o valor de referência, podemos interromper a busca
  - A solução parcial certamente será pior
- **Poda Alfa-Beta**

Exemplo: Problema das *n* rainhas

- Colocar *n* rainhas em um tabuleiro de xadrez *n* x *n* de modo que duas delas não estejam na mesma linha, coluna ou diagonal.
  - Para algumas instâncias de n, o problema não tem solução.
  - O algoritmo posiciona a primeira rainha na casa (1,1).
  - Obviamente, a próxima rainha não poderá ficar na linha 1.
  - Ao ser posicionada em (2,2), verifica-se que está na mesma diagonal da primeira rainha, logo esta solução parcial é inviável.
  - O algoritmo retrocede para escolher a próxima opção (2,3).
  - Eventualmente, o retrocesso pode chegar a raiz.
  - A árvore terá altura *n*.

Exemplo: Problema das *n* rainhas



N	steps
5	15
4	30
7	44
6	196
9	365
11	558
8	981
10	1,067
13	1,463
12	3,315
15	21,624
14	28,380
19	50,710
17	96,579
16	170,748
21	188,133
23	609,996
18	784,510
25	1,265,433
20	4,192,125
24	10,289,900
26	10,737,522
27	12,717,586
22	39,955,071
29	45,966,735
28	87,182,236
31	421,959,504
33	463,557,767
30	1,749,317,724
32	2,887,216,497
34	80,311,184,345

Algorithm

- AlgorithmNQueens(k,n)
- //Using backtracking,this procedure prints all possible
  //placements of n queens on an n*n
  //chessboard so that they are non attacking

```
{
  for(i=1 to n ) do
  {
        if Place(k,i) then
        {
              x[k]=I
              If(k=n) then write (x[1:n]);
              else Nqueens(k+1,n);
        }
  }
}
```

Exemplo: Soma do subconjunto

- Encontrar um subconjunto de um dado conjunto S de n inteiros positivos cuja soma seja igual a um inteiro d.
- Análise:
  - No pior caso, algoritmos de *backtracking* têm comportamento exponencial.
  - O algoritmo, no entanto, acaba podando ramos inviáveis, reduzindo bastante o tempo total.
  - Se apenas uma solução for suficiente, o algoritmo pode parar bem antes do final.
  - O sucesso desta técnica varia bastante de problema a problema e entre instâncias do mesmo problema.
  - É uma técnica geralmente aplicada a problemas combinatoriais, para os quais não existam algoritmos eficientes.
  - A implementação não recursiva requer uma pilha que é O(n) onde *n* é a altura da árvore.

Exemplo: Soma do subconjunto

```
Pilha caminho[]
int somaCaminho =0;
int lista[n];
int d;


int soma(Pilha caminho[]) {...}
vou imprime(Pilha caminho[]) {...}


//Inicializando
caminho = new Pilha();
Lista = new int[] {1,2,7,4,6,3,5,1};
D = 15;
sumaSub(0);
```

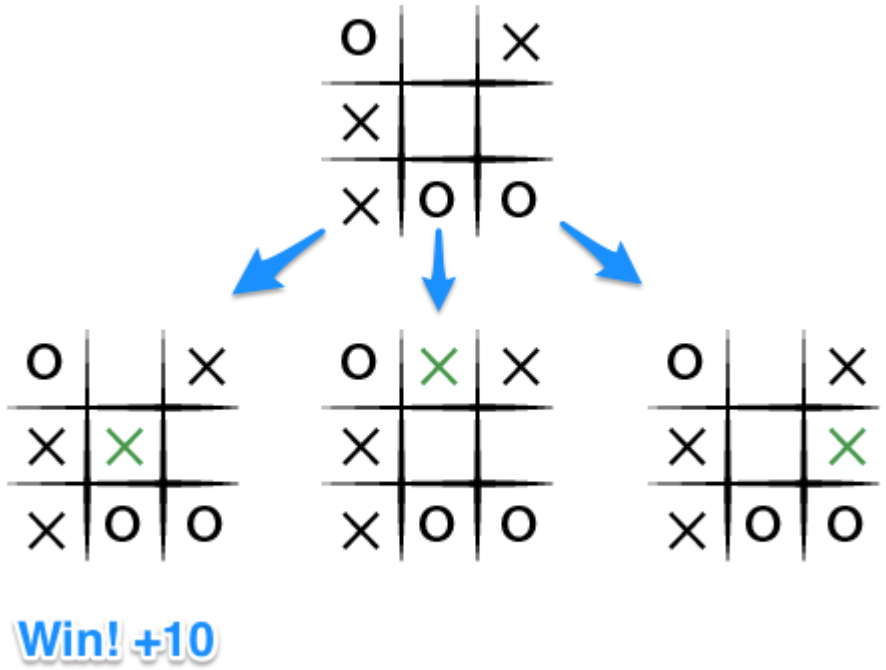
```
void somaSub(int pos) {
    for (int i=pos; i<n;i++){
        caminho.add(lista[i]);
        somaCaminho = soma(caminho);
        if (somaCaminho == d) {
            imprime(caminho);
            PARE;
        } else if (somaCaminho < d)
            somaSub(i+1);
        else
            caminho.removeTopo();
            i++;
    }
}
```

MinMax – Jogo da Velha

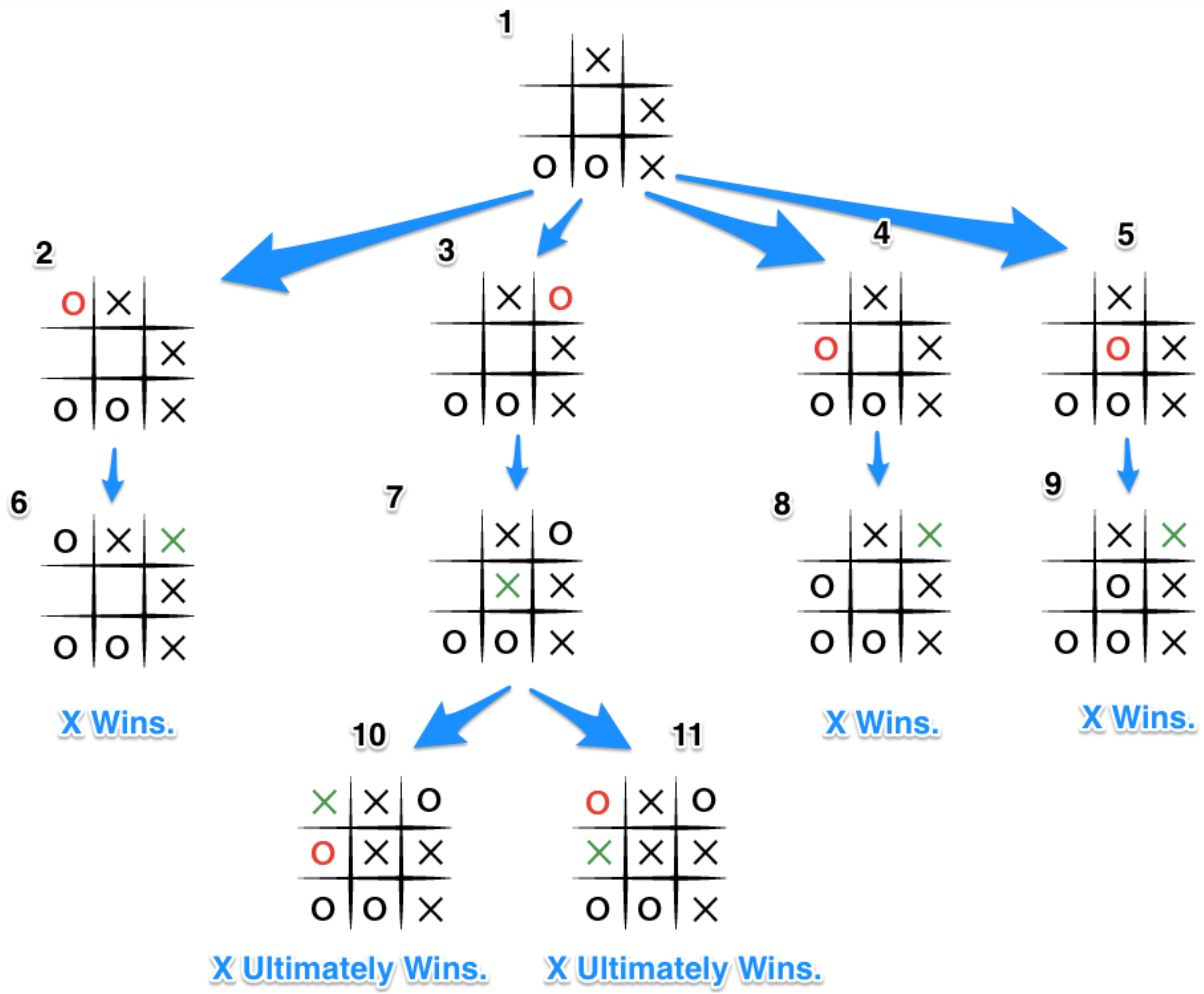
- Jogadores sempre buscam maximizar suas escolhas

- Escolha da IA = +10
- Escolha Adversári0= -10

- Em cada estado a IA escolhe o caminho de maior pontuação



MinMax – Jogo da Velha



MinMax – Jogo da Velha

```
Jogada jogadaIA // variável Global
// Player 1 = IA
// Player 2 = Oponente (teclado)

void Main() {

        while(!fim(jogo)) {

                minMax(jogo)
                jogar(jogo, player1, jogadaIA)
                jogar(jogo, player2, @teclado)
        }
}
```

MinMax – Jogo da Velha

```
int pontuacao(jogo) {

        //Vitoria da IA
        if (vitoria(player1)==true) {
                return 10;
        //Vitoria do Oponente
        } else if (vitoria(player2)==true) {
                return -10;
        }
        else
                return 0;
}
```

MinMax – Jogo da Velha

```
int minMax(jogo) {

        if (fim(jogo))
                return pontuacao(jogo)
        else {

                Pilha<int> pilhaPontuacao[];
                Pilha<Jogada> pilhaMovimentos[];

                foreach(m in possiveisMovimentos(jogo)) {
                        novoJovo = getJogo(m);
                        pilhaPontuacao.push (minMax(novoJogo))
                        pilhaMovimentos.push(m);
                }

                //Calcula o Min ou Max

                if (vezDo(jogador1)) { //IA
                        melhorJogada =  getIDmaxValor(pilhaPontuacao);
                }else {
                        melhorJogada =  getIDminValor(pilhaPontuacao);
                }
                jogadaIA = pilhaMovimentos[melhorJogada];
                return pilhaPontuacao[melhorJogada];

        }
}
```

Retrocesso (*Backtracking*)

- Vantagens
  - Forma simples de resolver problemas complexos
  - Linguagens de programação lógica oferecem suporte
- Desvantagens
  - Pode exigir muita memória para problemas grandes
  - O tamanho da busca em árvore pode crescer rapidamente tornando inviável a solução em tempo razoável
  - Pode ser necessário embutir heurísticas, ao custo da perda de garantia da otimalidade