Backtracking

Notas de aula da disciplina IME 04-10823
ALGORITMOS E ESTRUTURAS DE DADOS II

Paulo Eustáquio Duarte Pinto
(pauloedp arroba ime.uerj.br)

fevereiro/2013

Backtracking

É uma técnica de solução de problemas (construção de algoritmos) que examina o espaço de soluções de forma exaustiva, mas abandonando famílias de caminhos tão logo alguma inviabilidade seja determinada (exaustão inteligente). É particularmente aplicável a problemas NP-Completos.

Quando um caminho é inviável, VOLTAR para caminhos promissores!!!

Backtracking

BANQUETE

Quer-se distribuir n convidados em volta de uma mesa, tal que inimigos não sentem lado a lado.

Exemplo: n = 8.

Lula x Obama
Luana Piovani x Dado Dolabela
Pelé x Maradona
...

P: Como fazer?

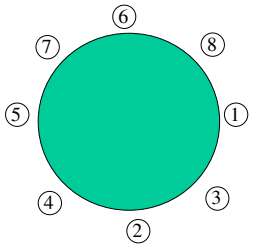
R: Tentar de todas as maneiras...
mas de forma inteligente...

Backtracking

BANQUETE

Este é um problema NP-Completo. A solução é tentar de todas as maneiras possíveis (todas as permutações circulares de 1 a n).

1. Lula
2. L.Piovani
3. Obama
4. D. Dolabela
5. Pelé
6. Maradona
7. ...
8. ...



Permutações circulares:

1 2 3 4 5 6 7 8
1 2 3 4 5 6 8 7
.....
1 3 2 4 5 7 6 8
.....
1 8 7 6 5 4 3 2

Backtracking

Geração de Permutações

Listar todas as n! permutações de um conjunto

```
Perm:
        Para i de 1 a n:
                Se (i ∉ P) Então
                        P ← P + i;
                        Se (|P| = n) Então
                                Imprime
                        Senão
                                Perm;
                        P ← P - i;
        Fp;
Fim;

Externamente:
        P ← Ø;   Perm;
```

Backtracking

Geração de Permutações - Versão 2

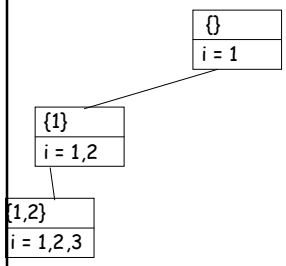
Listar todas as n! permutações de um conjunto

```
Perm:
        Para i de 1 a n:
                Se (not S[i]) Então
                        np ← np + 1;   P[np] ← i;   S[i] ← V;
                        Se (np = n) Então
                                Imprime
                        Senão
                                Perm;
                        np ← np - 1;   S[i] ← F;
        Fp;
Fim;

Externamente:
        S[*] ← F;   np ← 0;   Perm;
```

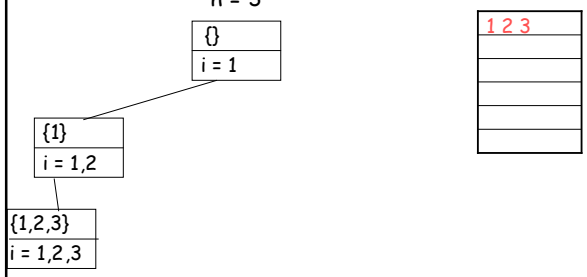
Backtracking - Geração de Permutações

n = 3



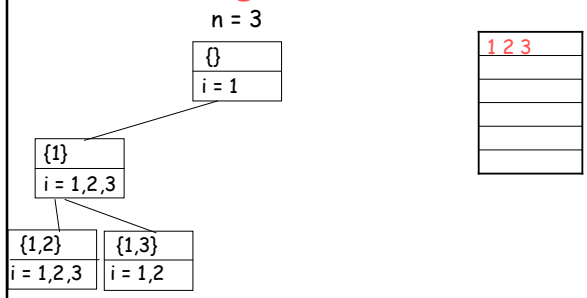
Backtracking - Geração de Permutações

n = 3



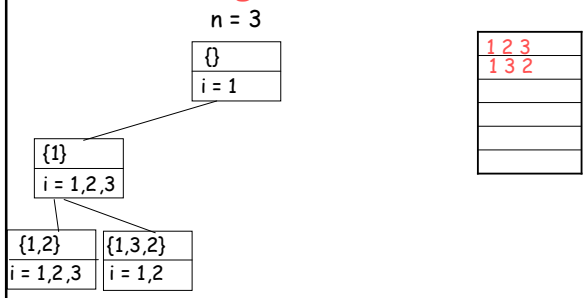
Backtracking - Geração de Permutações

n = 3



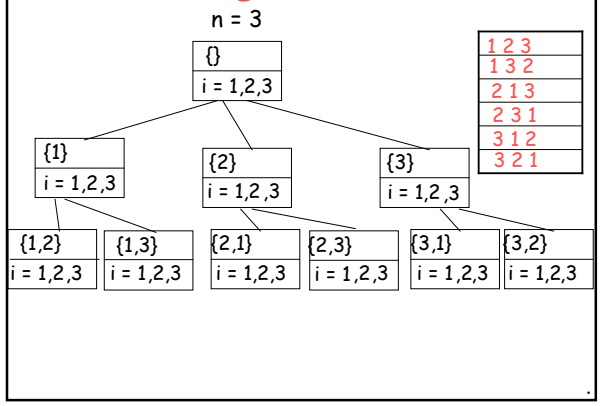
Backtracking - Geração de Permutações

n = 3



Backtracking - Geração de Permutações

n = 3



Backtracking

Geração de Permutações - Torres Pacíficas

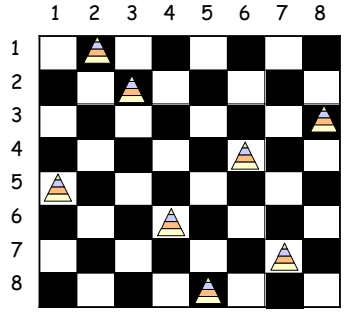
Problema: colocar 8 torres num tabuleiro 8 x 8 tal que as torres não se ataquem.

Solução: cada permutação dos números 1 a 8 é uma solução distinta!
(em cada linha a torre deve ser colocada na coluna indicada pelo i-ésimo elemento da permutação)

Backtracking

Geração de Permutações - Torres Pacíficas

Solução relativa à permutação 23861475.



Backtracking

Geração de Arranjos - A(n, q)

Basta modificar uma instrução do algoritmo para geração de Permutações!!!

```
Arranjo:
        Para i de 1 a n:
                Se (not S[i]) Então
                        np ← np + 1;   P[np] ← i;   S[i] ← V;
                        Se (np = q) Então
                                Imprime
                        Senão
                                Arranjo;
                        np ← np - 1;   S[i] ← F;
        Fp;
Fim;

Externamente:
        S[*] ← F;   np ← 0;   Arranjo;
```

Backtracking

Geração de Permutações circulares

Basta fixar o último elemento da permutação e gerar as permutações para n-1 elementos!!!

```
PermCircular:
        Para i de 1 a n-1:
                Se (not S[i]) Então
                        np ← np + 1;   P[np] ← i;   S[i] ← V;
                        Se (np = (n-1)) Então
                                Imprime
                        Senão
                                PermCircular;
                        np ← np - 1;   S[i] ← F;
        Fp;
Fim;

Externamente:
        S[*] ← F;   np ← 0;   P[n] ← n;   PermCircular;
```

Backtracking

Geração de Combinações - C(n, q)

Gera-se arranjos ordenados (um novo elemento só entra no arranjo se > último). Usa-se sentinela na posição 0. Não é mais necessário o conjunto S.

```
Comb:
        Para i de 1 a n:
                Se (i > P[np]) Então
                        np ← np + 1;   P[np] ← i;
                        Se (np = q) Então
                                Imprime
                        Senão
                                Comb;
                        np ← np - 1;
        Fp;
Fim;

Externamente:
        np ← 0;   P[0] ← 0;   Comb;
```

Backtracking

Geração de Combinações - C(n, q) - Versão 2

Usa-se um parâmetro para indicar os elementos que podem entrar na configuração.

```
Comb (t):
        Para i de t a n:
                np ← np + 1;   P[np] ← i;
                Se (np = q) Então
                        Imprime
                Senão
                        Comb (i+1);
                np ← np - 1;
        Fp;
Fim;

Externamente:
        np ← 0;   Comb(1);
```

Backtracking

Esquema Geral para Backtracking

Quase sempre o objetivo da técnica é exibir uma ou mais configurações de determinados conjuntos. O esquema a seguir aplica-se nesses casos.

```
Config:
    Para cada elemento e_i do conjunto:
      Se e_i pode ser acrescentado à configuração Então
          Acrescentar e_i à configuração.
          Se a configuração está completa Então
              Imprime
          Senão
              Config;
          Retirar e_i da configuração.
    Fp;
Fim;

Externamente:   Esvaziar configuração;   Config;
```

Backtracking

Damas Pacíficas

Problema: colocar 8 Damas num tabuleiro 8 x 8 tal que as Damas não se ataquem.

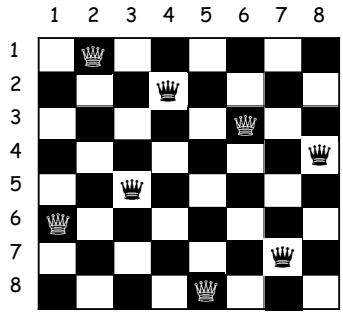
Solução: usar o esquema geral.
sempre se consegue colocar n Damas em um tabuleiro n x n, para n > 3.

Obs: Existem 92 soluções distintas para um tabuleiro 8x8.

Backtracking

Damas Pacíficas

Exemplo de uma solução para n = 8.



Backtracking

Damas Pacíficas

```
Damas_Pacíficas:
        Para c de 1 a 8:
            Se Possivel(l + 1, c) Então
                l ← l + 1;    P[l] ← c;
                Se (l = 8) Então
                        Imprimir P
                Senão
                        Damas_Pacificas;
                l ← l - 1;
        Fp;
Fim;

Externamente:
        l ← 0;    Damas_Pacificas;
```

Backtracking

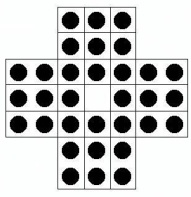
Damas Pacíficas

Exercício: simular as 3 primeiras soluções para um tabuleiro de 5x5 ou 6x6.

Backtracking

Resta 1 (Chinese Checkers)

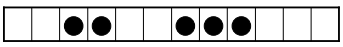
O objetivo é "comer" os pinos, deixando só um pino no tabuleiro



Solução por Backtracking:

Tentar movimentar os pinos de todas as maneiras.

(Entretanto só funciona bem para versões reduzidas...).



Backtracking

Eight (Versão reduzida do 15)

O objetivo é deixar os números ordenados de 1 a 8, por linha.

1	3	
8	2	5
4	7	6

Solução por Backtracking:

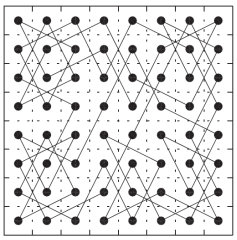
Movimentar o "buraco" de todas as maneiras, mas...

guardar as configurações já consideradas...

Backtracking

Problema do Percurso do Cavalo

O objetivo é fazer o cavalo "passar" por todas as casas do tabuleiro, sem repetições.



Solução por Backtracking:

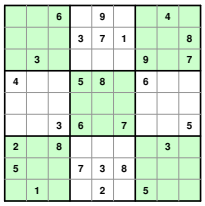
Movimentar o cavalo de todas as maneiras, mas...

funciona melhor se ordenar convenientemente as possibilidades...

Backtracking

Sudoku

O objetivo é completar o preenchimento da matriz tal que, em cada linha, coluna e mini-grid haja uma permutação dos números 1 a 9.



Solução por Backtracking:

Tentar, em cada posição vaga, os números 1 a 9 que não conflitem com os já preenchidos.

Backtracking

Geração de Subconjuntos

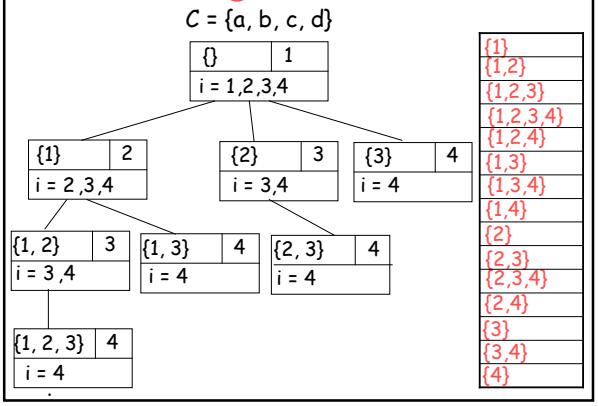
Número de subconjuntos de $C = \{c_1 \ldots c_n\} = 2^n$

```
GeraSub (t):
        Para i de t a n:
                ns ← ns + 1;    S[ns] ← i;
                Imprime;
                Se (i < n) Então
                        GeraSub (i + 1);
                ns ← ns - 1;
        Fp;
Fim;
```

Externamente:

```
ns ← 0;     GeraSub (1);
```

Backtracking - Geração de Subconjuntos



Backtracking

Geração de Subconjuntos

Subconjuntos gerados para: { a, b, c, d }

{ a },
{ a, b },
{ a, b, c },
{ a, b, c, d },
{ a, b, d },
{ a, c },
{ a, c, d },
{ a, d },
{ b },
{ b, c },
{ b, c, d },
{ b, d },
{ c },
{ c, d },
{ d }

Backtracking

Geração de Subconjuntos

Exercício: modificar o algoritmo de geração de subconjuntos para que a geração seja por ordem de tamanho dos subconjuntos.

Backtracking - Soma de Subconjuntos

Problema: Dado um conjunto de números, obter os subconjuntos tal que a soma dos seus elementos seja igual a um valor dado, s.

Exemplo: C = {10, 2, 8, 4, 7, 19, 9}, s = 10

Soluções: 10, 2 + 8

O problema Partição é um caso particular deste.

Backtracking - Soma de Subconjuntos

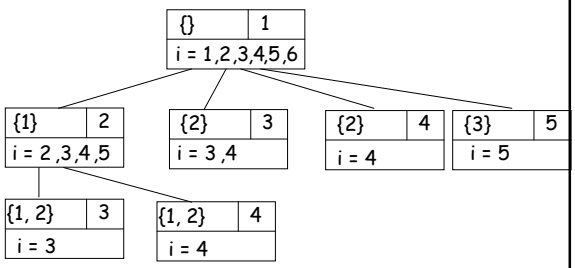
```
SomaSub (t):
        i ← t;
        Enquanto (i ≤ n) e ((soma + C[i]) ≤ s):
                ns ← ns + 1;  S[ns] ← i;    soma ← soma + C[i];
                Se (soma = s)  Então
                        Imprime
                Senão
                        SomaSub (i+1);
                ns ← ns - 1;   soma ← soma - C[i];
                i ← i + 1;
        Fe;
Fim;
```

Externamente:

```
        ns ← 0;   soma ← 0;   Ordena C;   SomaSub (1);
```

Backtracking - Soma de Subconjuntos

C = {1, 2, 2, 3, 4, 5, 9}, s = 4



1, 3
2, 2
4
.

Backtracking - Partição Aproximada

Problema: Particionar um conjunto de números, em 2 outros, tal que a diferença entre suas somas seja mínima.

Exemplo: C = {10, 2, 8, 4, 7, 19, 9}

Solução: {2, 8, 9, 10} e {4, 7, 19}, c/ diferença de 1

O problema Partição é um caso particular deste.

Backtracking - Partição Aproximada

```
Part_aprox (t):
   i ← t;
   Enquanto (i ≤ n) e ((soma +C[i]) < msmp):
        ns ← ns + 1;    S[ns] ← i;    soma ← soma + C[i];
        Se (abs(tot - 2*soma) < dif)  Então
                Guarda;    dif ← abs(tot - 2*soma);
                msmp ← max(soma, tot-soma);
        Part_aprox (i + 1);
        ns ← ns - 1;    soma ← soma - C[i];
        i ← i + 1;
   Fe;
Fim;
```

Externamente:

```
   Ordena C;   soma ← C[1];  ns ← 1;  S[1] ← 1;  Guarda;
   tot ← ΣC[i];  dif ← tot-2*C[1];  msmp ← max(C[1],tot-C[1]);
   Part_aproximada(2);
   Imprime solução guardada;
```

Backtracking - Partição aproximada

C = {7, 13, 17, 31, 38}, tot = 106, msmp = 99, dif = 92

Part/candidato	soma	|tot- 2*soma|	msmp	dif	Melhor partição
{7}	7	92	99	92	{7}
{7} 13	20	66	86	66	{7, 13}
{7, 13} 17	37	32	69	32	{7, 13, 17}
{7, 13,17} 31	68	30	68	30	{7, 13,17, 31}
{7,13,17,31} 38	106				
{7,13,17} 38	75				
{7, 13} 31	51	4	55	4	{7, 13, 31}
{7, 13,31} 38	89				
{7, 13} 38	58				
{7} 17	24	58			
{7, 17} 31	55				
{7} 31	38	30			
{7, 31} 38	76				
{7} 38	45	16			

Backtracking

Partição Aproximada

Exercício: mostrar a Partição Aproximada do conjunto {2, 5, 19, 20, 34},

Exercício: mostrar a Partição Aproximada do conjunto {2, 15, 16, 30}

Backtracking - Jogos

Árvore de jogo

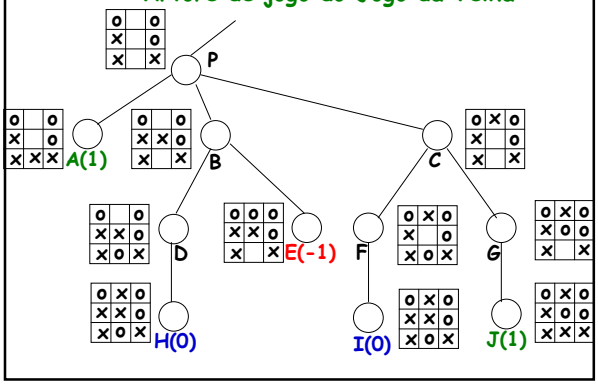
Em jogos de turno pode-se construir a árvore de jogo, para se avaliar a melhor jogada a fazer a cada momento.

A árvore de jogo é construída com Backtracking, para se verificar todas as possibilidades.

Depois a árvore é avaliada com a técnica MaxMin.

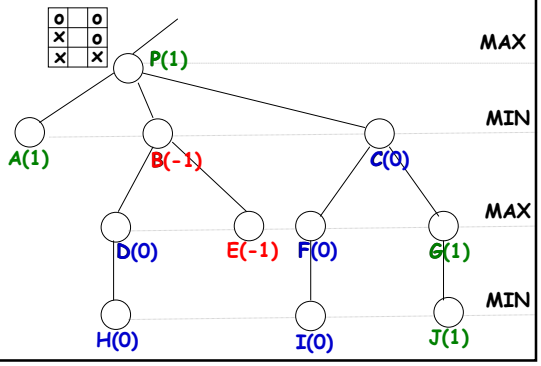
Backtracking - Jogos

Árvore de jogo do Jogo da Velha



Backtracking - Jogos

Avaliação da Árvore de jogo - MAXMIN



Backtracking - Jogos

Avaliação da Árvore de jogo - MAXMIN

```
Avalia_arv (F,  Modo);
      Se (F é folha)  Então
         Retornar (Calcula_valor(F));
      Senão
         Se (Modo = 'MAX')  Então  Valor ← -∞
         Senão                     Valor ← ∞;
         Para cada filho w de F:
             Se (Modo = 'MAX') Então
                 Valor ← max (Valor,  Avalia_arv(w, 'MIN'));
             Senão
                 Valor ← min (Valor,  Avalia_arv(w,  'MAX'));
         Fp;
      Retornar Valor;
Fim;
```

Backtracking - Jogos

EXERCÍCIO

DESENHAR e AVALIAR a árvore de jogo para o Jogo dos Palitos, entre 2 competidores:

São dados 5 palitos e cada um deve tirar, alternadamente, de 1 a 3 palitos. Ganha o que tirar o último palito.

Backtracking - Jogos

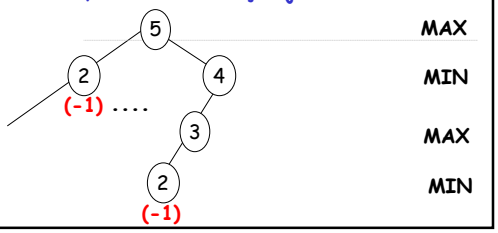
Criação/avaliação da Árvore de jogo - MAXMIN

```
Criaavalia_arv (Modo);
        Se (configuração é final)  Então
           Retornar (Calcula_valor);
        Senão
           Se (Modo = 'MAX')  Então  Valor ← -∞
           Senão                     Valor ← ∞;
           Para cada possível modificação na configuração:
                 Modifica a configuração;
                 Se (Modo = 'MAX') Então
                      Valor ← max (Valor,  Criaavalia_arv('MIN'));
                 Senão
                      Valor ← min (Valor,  Criaavalia_arv('MAX'));
                 Restaura configuração;
           Fp;
        Retornar Valor;
Fim;
```

Backtracking - Jogos

Memorização

Na criação/avaliação simultânea da árvore de jogo pode-se (deve-se) usar memorização, simplificando a construção da árvore. No exemplo abaixo para o jogo com 5 palitos, ao se cria ro segundo nó de valor 2 em nível de mínimo, usa-se a avaliação já feita.



Backtracking - Jogos

Poda de Simetria

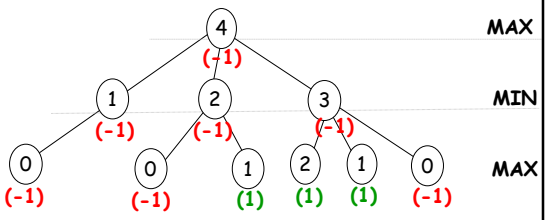
Nos jogos de turnos, onde, dada uma configuração, as jogadas são as mesmas para qualquer um dos dois jogadores que vai jogar, pode-se aplicar a poda por simetria, derivada do princípio de que, para essas situações: T(x, 'MAX') = -T(x, 'MIN'), onde x é uma dada configuração e T a avaliação de determinado nó.

Considere, por exemplo, a árvore de jogo para o jogo de palitos com 4 palitos inicialmente.

A árvore de jogo da página seguinte foi criada usando-se esse princípio.

Backtracking - Jogos

Jogo dos palitos para n = 4. Criação e avaliação usando poda de simetria



Backtracking - Jogos

Poda de Valor limite

Na criação/avaliação simultâneas da árvore de jogo onde há um valor limite para avaliação, pára-se de construir a árvore quando esse valor limite é obtido.

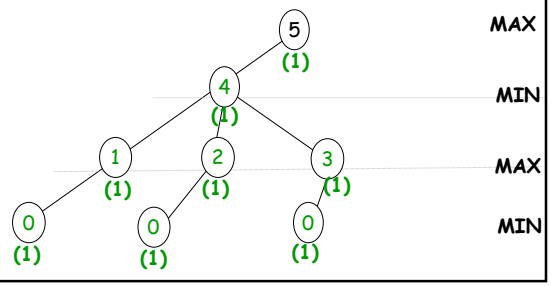
Ordenação de configurações

As diversas podas podem tirar partido do fato de que a ordem das configurações do Backtracking pode ser arbitrada para o que for conveniente para cada problema.

Na próxima página é mostrada a árvore de jogo para 5 palitos usando as podas de valor limite e simetria, escolhendo-se a ordem da retirada de palitos conveniente.

Backtracking - Jogos

Jogo dos palitos para n = 5. Criação e avaliação usando podas de valor limite e simetria, com ordenação conveniente da retirada de palitos



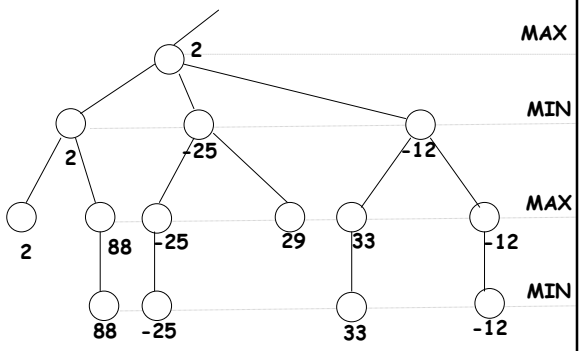
Backtracking - Jogos

Poda Heurística

Em muitas situações (jogo de xadrez, damas, gamão, por exemplo), não é possível criar toda a árvore de jogo. Então cria-se um certo número de níveis da árvore e faz-se a avaliação heurística das folhas e avalia-se a árvore com MAXMIN.

Backtracking - Jogos

Árvore de jogo – Poda Heurística



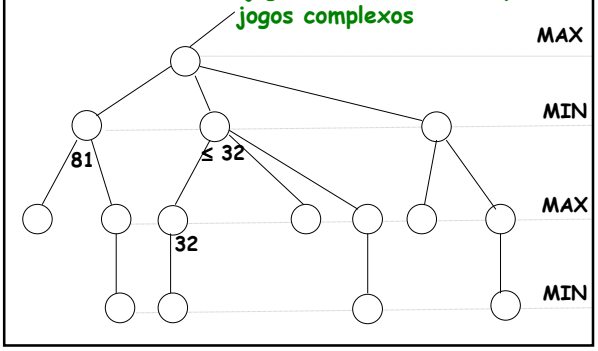
Backtracking - Jogos

Poda Alfa-Beta

A poda alfa-beta permite simplificar a criação da árvore. É uma extensão da poda de valor limite e é usada em geral, para jogos complexos tais como xadrez, damas, gamão.
Na chamada para criação/avaliação são passados dois parâmetros (alfa/beta). No nível de máximo, o parâmetro beta indica o valor mínimo atingido por algum filho do nível anterior. Se algum filho do nível atual atingir ou superar esse valor, pode-se parar a avaliação do nó atual.

Backtracking - Jogos

Árvore de jogo – Poda Alfa-Beta p/ jogos complexos



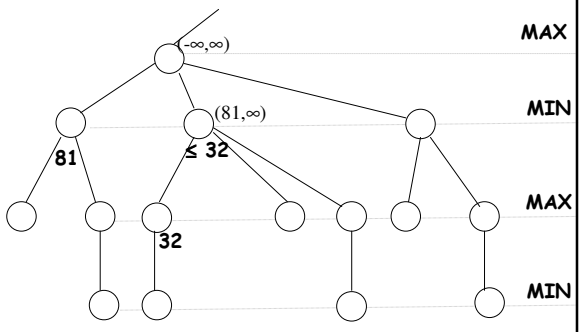
Backtracking - Jogos

Avaliação da Árvore de jogo - Alfabeta

```
Avalia_arv (F,  Modo, α, β);
      Se (F é folha)  Então
         Retornar (Calcula_valor(F));
      Senão
          Para cada filho w de F:
             Se (Modo = 'MAX') Então
                   v ← max (α,  Avalia_arv(w, 'MIN', α, β));
                   Se (v ≥ β ) Então Retornar β;
                   α  ←  v;
             Senão
                   v ← min (β,  Avalia_arv(w, 'MAX', α, β));
                   Se (v ≤ α ) Então Retornar α;
                   β ←  v;
          Fp;
      Retornar v;
Fim;
      Externamente: Avalia_Arv(T, 'MAX', -∞, ∞ )
```

Backtracking - Jogos

Árvore de jogo – Poda Alfa-Beta



Backtracking - Jogos

Jogo de xadrez

Técnicas usadas:

- -Poda heurística
- -Memorização
- -Poda alfa-beta
- -Aprodundamento desigual
- -Busca aquiescente
- -Livro de aberturas
- -Bancos de Dados de finais

ICE: http://www.nxn.kit.net/ICE/

Thinking machine:
http://www.turbulence.org/spotlight/thinking/chess.html

Backtracking

FIM