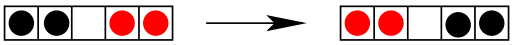
Análise e Técnicas de Algoritmos
Período 2003.1

Backtracking e Branch-and-Bound

Backtracking

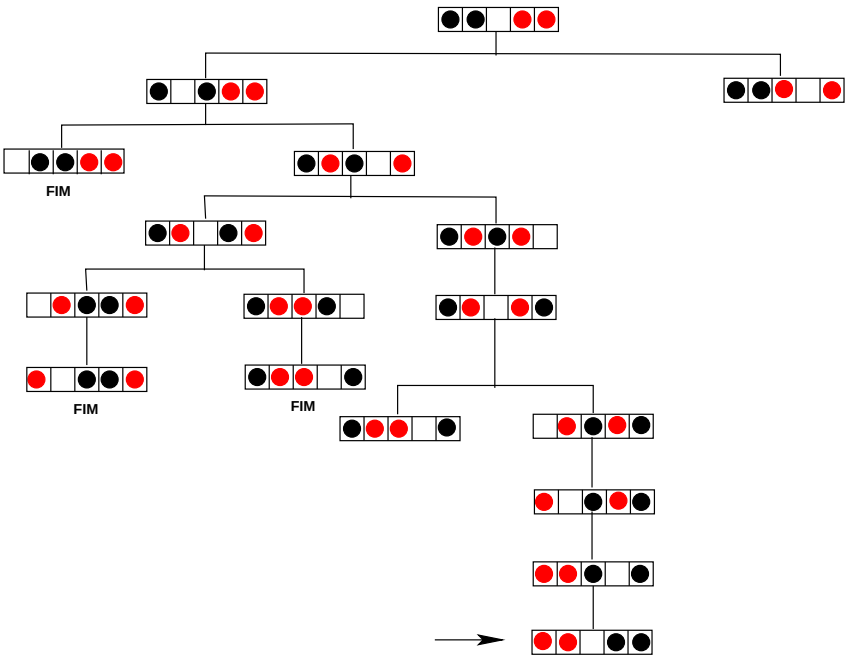
Considere os seguintes problemas:
Problema 1: Um jogo com n bolas vermelhas e o mesmo número de bolas pretas. O tabuleiro do jogo consiste de uma única linha com $2n + 1$ espaços onde as bolas podem ser colocadas. Inicialmente, as bolas vermelhas estão posicionadas na extremidade direita do tabuleiro. As bolas pretas estão na outra extremidade do tabuleiro. Existe um espaço livre, separando as bolas vermelhas e pretas. O objetivo é reverter as posições das bolas, como mostra a figura abaixo.



Para tanto, as bolas vermelhas só podem se movimentar para a esquerda e as pretas para a direita. Não é possível retroceder as bolas. Em cada movimento uma bola pode:

- Mover um espaço, se o espaço estiver vazio.
- Pular sobre exatamente uma bola de cor diferente, se o espaço logo após a bola estiver vazio.

Como atingir o objetivo?



Qual foi a técnica que você utilizou para resolver este problema?

Problema 2: Suponha que um aluno de ATAL necessita planejar suas atividades *pós universidade* para cada dia da semana. As opções são ditadas de acordo com os seguintes objetivos mínimos:

- 20 horas de estudo de ATAL.
- 5 horas de diversão.
- 5 horas de atividades físicas.

Vamos assumir ainda que:

1. Segunda, terça, quarta e quinta são dias reservados ao estudo. É possível escolher estudar 0, 2 ou 4 horas nestes dias.
2. Sexta é dia de diversão. As opções disponíveis são: assistir TV (2 horas), ir ao Iguatemi (5 horas) ou sair para beber (5 horas), ou não fazer nada (0 horas).
3. Sábado é dia de diversão e também de atividades físicas. As opções de diversão são as mesmas da sexta. Para as atividades físicas é possível caminhar (1 hora), jogar futebol (2 horas), musculação (3 horas) ou não fazer nada (0 horas).
4. Domingo é dia de estudar e de atividades físicas. As opções de atividades físicas são as mesmas do sábadao. Os horários para estudo são os mesmos da segunda a quinta.

Como resolver este problema?
Os problemas descritos são de dois tipos:

1. Aqueles que é necessário encontrar todas as soluções possíveis.
2. Achar uma solução ótima sujeita a restrições explícitas.

A solução para os dois tipos é bastante semelhante:

- fazer uma lista de todos os candidatos a respota possíveis.
- Examinar todas as respostas ou algumas delas.
- Retornar a solução.

Na teoria, esta abordagem funciona sempre que a lista de candidatos a solução do problema é finita. Desta forma, é possível examinar todos os candidatos. Na prática, porém, essa abordagem não é muito útil porque o número de candidatos é grande.

Backtracking é uma estratégia para sistematicamente examinar a lista de possíveis soluções. A idéia de backtracking é eliminar a explícita verificação de uma boa parte dos possíveis candidatos. Para tanto, o problema deve respeitar restrições que maximizam/minimizam alguma função de otimização. Os seguintes passos são respeitados:

1. Definir um espaço solução para o problema. este espaço solução deve incluir pelo menos uma solução ótima para o problema.

2. Organizar o espaço solução de forma que seja facilmente pesquisado. A organização típica é uma árvore.

3. Proceder a busca em profundidade.

Backtracking é, portanto, uma estratégia que se aplica em problemas cuja solução pode ser definida a partir de uma seqüência de n decisões, que podem ser modeladas por uma árvore que representa todas as possíveis seqüências de decisão. De fato, se existir mais de uma disponível para cada uma das n decisões, A busca exaustiva da árvore é exponencial. A eficiência desta estratégia depende da possibilidade de limitar a busca, ou seja, podar a árvore, eliminando as sub-árvores que não levam a nenhuma solução.

Em resumo:

- Backtracking é um refinamento da abordagem *força bruta.*
- Sistematicamente busca por uma solução para o problema, considerando todas as opções disponíveis.
- Em geral, avaliamos algumas restrições para minimizar ou maximizar a função de otimização associada.
- O conjunto de soluções possíveis (espaço de solução) é obtido através de soluções parciais que são incrementadas até se chegar a soluções do problema.

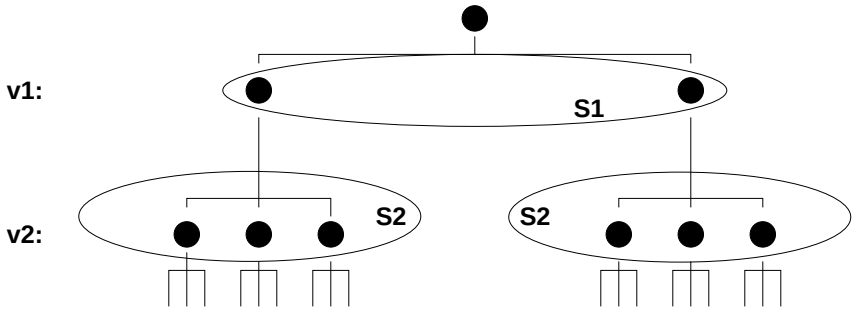
A Idéia Geral: Usando o Espaço de Soluções

As soluções são representadas por n-tuplas ou vetores de solução $\langle v_1, v_2, \cdots, v_n \rangle$. Cada v_i é escolhido a partir de um conjunto finito de opções S_i.

Um algoritmo de backtracking inicia com um vetor vazio. Em cada etapa, o vetor é extendido com um novo valor formando um novo vetor que pode representar uma solução parcial do problema. Na avaliação de um vetor $\langle v_1, \cdots, v_i \rangle$, se for constatado que ele não pode representar uma solução parcial, o algoritmo faz o backtrack, eliminando o último valor do vetor, e continua tentando extender o vetor com outros valores alternativos.

Um aspecto bastante importante nos problemas que são tratados por soluções baseadas em backtracking, é a definição de restrições. Existem dois tipos de restrições: restrições explícitas e restrições implícitas.

Restrições explícitas correspondem às regras que restringem cada v_i em tomar valores de um determinado conjunto. Isto está relacionado com a representação do problema e as escolhas possíveis.



Restrições implícitas determinam como os v_i se relacionam entre si.

Exemplo 1: O Problema das 8 Rainhas

- **Definição:** Colocar 8 rainhas em um tabuleiro de xadrez de forma que nenhuma rainha ataque uma outra.
- **Solução:** Uma 8-tupla $\langle v_1, v_2, \cdots, v_8 \rangle$, onde v_i é a coluna da rainha i.
- **Restrições Explícitas:** $S_i = \{1, 2, \cdots, 8\}$, $1 \leq i \leq n$.
- **Restrições Implícitas:**
  1. Nenhum v_i pode ser igual ao outro.
  2. Duas rainhas não podem estar na mesma diagonal.

Observando a decsrição do problema é possível efetuar as seguintes observações:

- A solução força bruta deve considerar um espaço solução de tamanho $C_8^{64} = 4.426.165.368$.
- Com a restrição explícita, é possível reduzir o espaço de solução para 8^8.
- Com as restrições implícitas, o tamanho do espaço de solução cai para 8!.

Exemplo 2: Soma de Subconjuntos

- **Definição:** Dados n números positivos $(W_i, 1 \leq i \leq n)$ e um valor positivo M, achar todos os subconjuntos dos W_i cuja soma é M.
- **Solução:** Uma k-tupla com os índices dos números a incluir.
- **Restrições Explícitas:** $v_i = \{j | j \text{ é inteiro}, 1 \leq j \leq n\}$.
- **Restrições Implícitas:**
  1. $v_i \neq v_j, i \neq j$.
  2. $\Sigma = M$.
  3. $v_i < v_{i+1}, 1 \leq i < n$.

Geração da Árvore

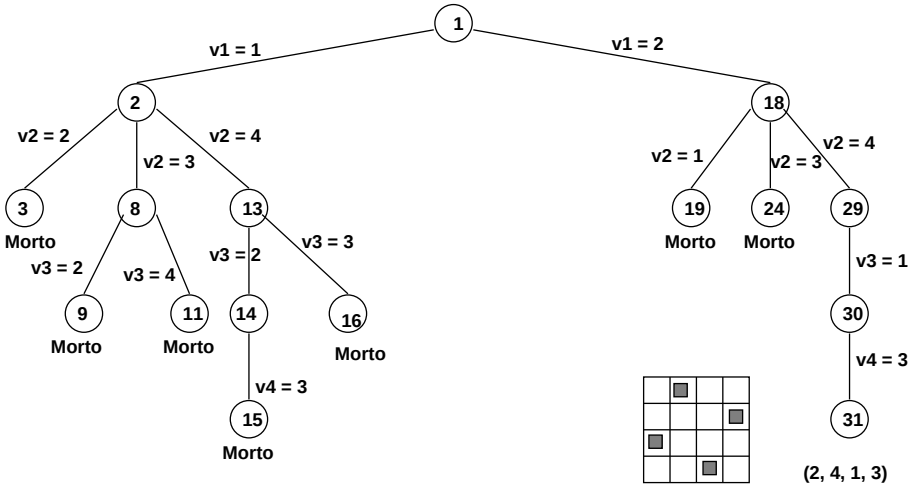
Se S_i é o domínio de v_i, então $S_1 \times \cdots \times S_m$ é o espaço de soluções do problema. O critério de validação (determinado pelas restrições implícitas) determina que parte deste conjunto solução necessita ser considerado.

A busca pelo espaço de soluções pode ser representado por um caminhamento em profundidade de uma árvore.

Para criar a árvore que representa o espaço de solução fazemos:

1. Começar da raiz e gerar outros nós.
2. Um nó que foi gerado, mas que não foi totalmente explorado é dito *nó vivo*.
3. O nó cujos filhos estão sendo gerados é dito *nó de expansão (nó-E)*.
4. As funções de poda são uasadas para detonar os nós vivos antes da geração de todos os seus filhos.
5. Um *nó morto* é aquele que foi podado ou que todos os filhos já foram gerados.

Exemplo: 4 Rainhas



Algoritmo Genérico

Um esquema genérico de algoritmos de backtracking leva em consideração dois tipos de testes:

1. Teste que verifica se o vetor de tamanho k pode levar a uma solução do problema. Em caso positivo, o vetor representa uma solução parcial de tamanho k.

2. teste que verifica se o vetor v já é solução.

O algoritmo genérico pode ser descrito assim:

backtrack($v[1..k]$)

```
▷v é uma solução parcial de tamanho k
if v é uma solução then
  escreva v
else
  for cada k + 1 vetor w em que w[1..k] = v[1..k] do
    backtrack(w[1..k + 1])
```

O Problema da Mochila

O problema da mochila já é nosso velho conhecido. Vamos supor a seguinte instância de problema:

- 4 tipos de objetos: $\langle o_1, o_2, o_3, o_4 \rangle$.
- Existem vários objetos do mesmo tipo.
- Pesos dos objetos: 2, 3, 4 e 5 unidades, respectivamente.
- Valor dos objeos: 3, 5, 6 e 10, respectivamente.
- Capacidade da mochila: 8.

A solução:

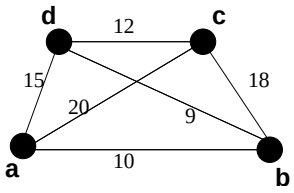
backpack(i, r)

```
b ← 0
for k ← i até n do
  if w[k] ≤ r then
    b ← max(b, v[k] + backpack(k, r − w[k]))
return b
```

O Caixeiro-Viajante

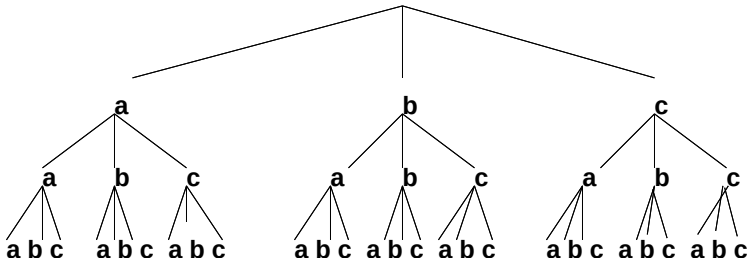
Este problema assume um conjunto de n cidades e um caixeiro-viajante que necessita visitar cada uma das cidades. Por questão de economia, ele deve visitar uma única vez cada cidade e por fim retornar a cidade de origem. Portanto, o problema consiste em determinar uma turnê de custo mínimo.

No exemplo abaixo, a rota (a, b, d, c) é a de custo mínimo (51).

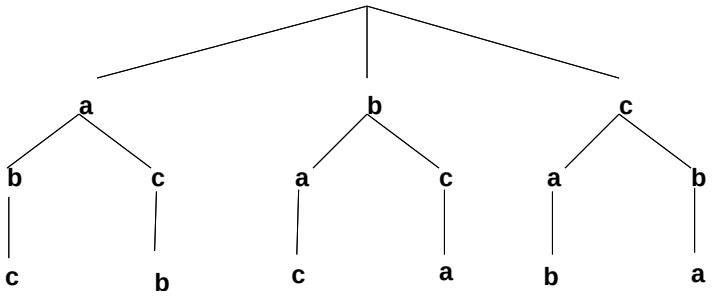


O problema do caixeiro viajante é NP-hard (veremos isso mais na frente). isso implica dizer que não existe uma solução em tempo polinomial para ele. A solução por backtracking define um vetor de cidades $(v_1, \cdots, v_n)$ que representa a melhor rota.

Como critério de poda podemos utilizar o número de cidade na rota obtida. esse número não pode ser maior do que $|V|$. Neste caso, o algoritmo deve buscar $|V|^{|V|}$ possibilidades.



Se o critério de poda verificar a repetição de cidades, o número de possibilidades é reduzido para $|V|!$.



Considerações Adicionais

- Forma eficiente de implementar busca exaustiva.
- Linguagens da área de programação em lógica geralmente trazem algum mecanismo que dá suporte ao baxktracking. Exemplo: Prolog, OPS5, etc.

- Programas em backtracking são, por natureza, combinatórios.
- Requer muita memória, já que a quantidade de variáveis locais transportada em cada chamada recursiva é diretamente proporcional ao tamanho do problema.

Branch-and-Bound

Branch-and-bound (BB) é uma estratégia bastante similar a backtracking. Também utiliza uma estrutura de árvore para explorar todas as possíveis soluções. A principal diferença entre as duas estratégias está na forma como a árvore é explorada. Como vimos, backtracking utiliza uma busca em profundidade. Na estratégia BB, utiliza busca em aplitude. Através de computação auxiliar, BB decide em cada etapa qual dos nós deve ser o próximo a ser explorado. Uma lista de prioridade mantém os nós que foram gerados mas que ainda permanecem inexplorados.