Análise e Técnicas de Algoritmos

Jorge Figueiredo

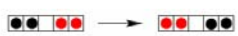
Backtracking and Branch-and-Bound

Agenda

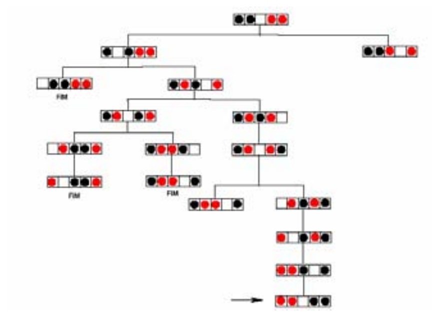
- Conceitos Básicos
- O Problema das Rainhas
- *Template* Genérico
- Mochila Binária

Jogo da Troca de Bolas

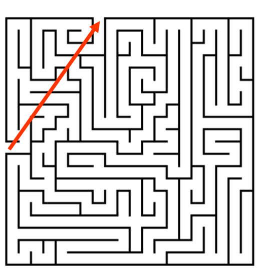
- n bolas vermelhas e n bolas pretas
- Tabuleiro (uma linha) com 2n + 1 posições
- Bolas com a mesma cor em extremidades diferentes, e um espaço vazio separando o conjunto de bolas diferentes.
- Movimentos possíveis:
  - Bola vermelha para a esquerda e preta para a direita
  - Mover um espaço se o espaço está vazio
  - Pular sobre exatamente uma bola de cor diferente, se o espaço logo após a bola estiver vazio



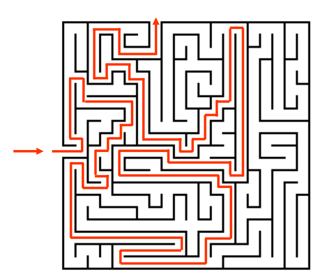
Jogo da Troca de Bolas



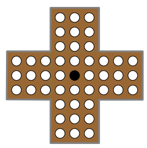
Problema do Labirinto



Problema do Labirinto



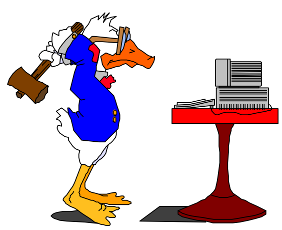
Jogo do “Resta Um”



O Que Estes Problemas Têm em Comum?

- Tomar uma série de decisões entre várias opções.
- Cada decisão leva a um novo conjunto de decisões.
- Alguma(s) seqüência(s) de decisões pode(m) conduzir a solução do problema.
- Encontrar solução:
  1. Fazer uma lista com todos os candidatos possíveis.
  2. Examinar todas as respostas ou alguma delas.
  3. Retornar a solução.
- Problemas de otimização

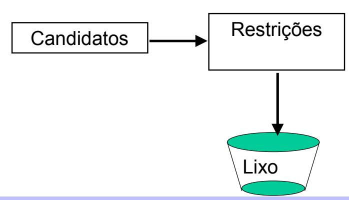
Como Resolver?



Força bruta: na prática esta abordagem não é muito eficiente porque a lista de candidatos é grande.

Backtracking

- Estratégia para sistematicamente a lista de possíveis soluções, eliminando (explicitamente) a verificação de uma boa parte dos possíveis candidatos.
- Pode ser considerado como uma variação de DFS.
- Usa uma *árvore implícita*.



Idéia Geral: Usando o Espaço de Solução

- Soluções representadas por n-tuplas ou vetores de solução:
  - $<v_1, v_2, \ldots, v_n>$
  - Cada v_i é escolhido a partir de um conjunto finito de opções S_i.
- Inicia com um vetor vazio.
- Em cada etapa o vetor é extendido com um novo valor.
- O novo vetor é então avaliado. Se não for solução parcial, o último valor do vetor é eliminado e outro candidato é considerado.

Restrições

- **Restrições Explícitas**: correspondem às regras que restringem cada v_i em tomar valores de um determinado conjunto. Está relacionado com a representação do problema e as escolhas possíveis.
- **Restrições Implícitas**: determinam como os v_i's se relacionam entre si.

O Problema das 8 Rainhas

- Colocar 8 rainhas em um tabuleiro de xadrez de modo que nenhuma rainha ataque uma outra.
- **Solução**: uma 8-tupla $<v_1, v_2, \ldots, v_8>$ em que v_i indica a coluna da rainha i.
- **Restrições Explícitas**: $S_i = \{1, 2, 3, \ldots, 8\}, 1 \le i \le n$.
- **Restrições implícitas**:
  - Nenhum v_i pode ser igual ao outro.
  - Duas rainhas não podem estar na mesma diagonal.
- Tamanho do espaço solução:
  - Força Bruta: 4.426.165.368
  - Com R.E.: 8^8.
  - Com R.I.: 8!.

Soma de Subconjuntos

- Sejam n números positivos (W_i, $1 \le i \le n$) e um valor positivo M, achar todos os subconjuntos de W_i cuja soma é M.
- **Solução**: uma k-tupla com os índices dos números escolhidos.

- **Restrições Explícitas**: $v_i = \{j \mid j \text{ é inteiro}, 1 \le j \le n\}$.
- **Restrições Implícitas**:
  - $v_i \ne v_j$, $i \ne j$.
  - $\Sigma = M$.
  - $v_i < v_{i+1}$, $1 \le i < n$.

Análise e Técnicas de Algoritmos – 2005.1 © Jorge Figueiredo, DSC/UFCG

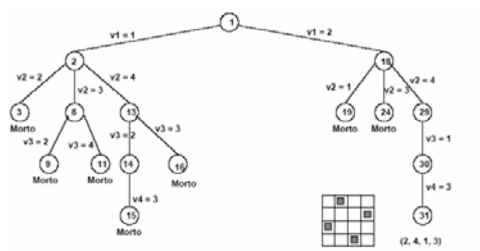
Geração da Árvore

Para criar a árvore que representa o espaço solução fazemos:

1. Começar da raiz e gerar outros nós.
2. Um nó que foi gerado e que não foi totalmente explorado é dito nó vivo.
3. Um nó cujos filhos estão sendo gerados é dito em expansão.
4. Usar função de poda para *detonar* a geração de alguns filhos, se for o caso.
5. Um nó morto é aquele que foi podado ou todos os filhos já foram gerados.

Análise e Técnicas de Algoritmos – 2005.1 © Jorge Figueiredo, DSC/UFCG

Exemplo: Problema das 4 rainhas



Análise e Técnicas de Algoritmos – 2005.1 © Jorge Figueiredo, DSC/UFCG

Algoritmo Genérico

```
Backtrack(v[1..k])
► v é um vetor promissor de tamanho k
    if v é solução then
        escreva v
    for cada vetor promissor w de tamanho k+1
        em que w[1..k] = v[1..k] do
            Backtrack(w[1..k+1])
```

Análise e Técnicas de Algoritmos – 2005.1 © Jorge Figueiredo, DSC/UFCG

Mochila Binária

- Considerar n tipos de objetos (um número adequado de cada objeto)
- Cada objeto i tem: valor (v_i) e peso (w_i)
- Mochila com capacidade W
- Para concretizar:
  - W = 8
  - o_1: (w=2, v=3)
  - o_2: (3, 5)
  - o_3: (4, 6)
  - o_4: (5, 10)

Análise e Técnicas de Algoritmos – 2005.1 © Jorge Figueiredo, DSC/UFCG

Mochila Binária: Algoritmo

```
Backpack(i, r)
► maior lucro usando objetos de tipos i até n e que não exceda r
b ← 0
for k ← i to n do
      if w[k] ≤ r then
            b ← max(b, v[k]+ Backpack(k, r – w[k]))
return b
```


Mochila Binária (Variação)

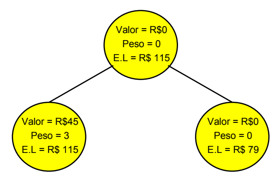
- Um objeto de cada tipo.
- Usar uma função extra que limita poda:
  - Usar a estratégia gulosa para mochila fracionária para computar um limite superior de lucro.
  - Ordenar os objetos por valor/peso.
- Duas possibilidades de backtracking:
  - Limite de peso.
  - Se não existe possibilidade da melhor estimativa de lucro ser maior do que o melhor lucro já encontrado.

Mochila Binária (Variação)

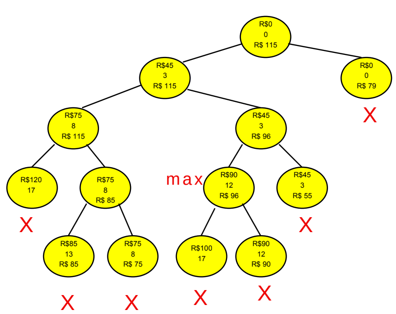
- Considere W= 16 e os seguintes 4 objetos:

i	v_i	w_i	v_i/w_i
1	R$45	3	R$15
2	R$30	5	R$ 6
3	R$45	9	R$ 5
4	R$10	5	R$ 2

Mochila Binária (Variação)



Mochila Binária (Variação)



Branch-and-Bound

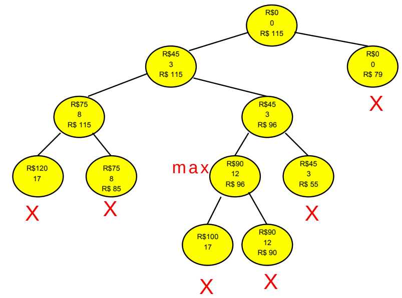
- Variação de backtracking.
- Necessidade de função de poda.
- Em alguns problemas, poda pode ser realizada mais cedo se usarmos BFS em vez de DFS.
- BnB = Backtracking + Função de poda – DFS + BFS + PQ

Mochila Binária (Variação)

- Considere W= 16 e os seguintes 4 objetos:

i	v_i	w_i	v_i/w_i
1	R$45	3	R$15
2	R$30	5	R$ 6
3	R$45	9	R$ 5
4	R$10	5	R$ 2

Mochila Binária (BnB)



Problema de Atribuição de Tarefas

- n agentes para n tarefas.
- Cada agente deve executar exatamente uma única tarefa.
- Se ao agente i é atribuída a tarefa j, um custo $C_{i,j}$ é identificado.
- Problema é atribuir tarefas aos agentes para minimizar o custo total de executar as n tarefas.

	1	2	3	4
A	11	12	18	40
B	14	15	13	22
C	11	17	19	23
D	17	14	20	28

Problema de Atribuição de Tarefas

- Usar BnB.
- Identificar função de poda.
- Guiar BFS (Best-first Search)

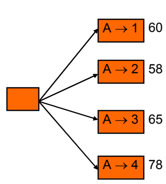
	1	2	3	4	
					87
A	11	12	18	40	
B	14	15	13	22	58
C	11	17	19	23	
D	17	14	20	28	
					73

[58..73]

Problema de Atribuição de Tarefas

	1	2	3	4
A	11	12	18	40
B	14	15	13	22
C	11	17	19	23
D	17	14	20	28

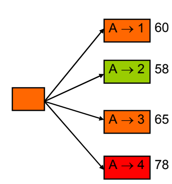
[58..73]



Problema de Atribuição de Tarefas

	1	2	3	4
A	11	12	18	40
B	14	15	13	22
C	11	17	19	23
D	17	14	20	28

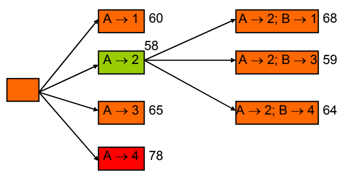
[58..73]



Problema de Atribuição de Tarefas

	1	2	3	4
A	11	12	18	40
B	14	15	13	22
C	11	17	19	23
D	17	14	20	28

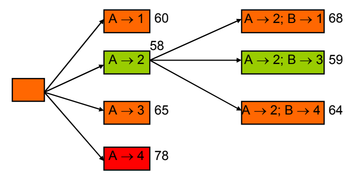
[58..73]



Problema de Atribuição de Tarefas

	1	2	3	4
A	11	12	18	40
B	14	15	13	22
C	11	17	19	23
D	17	14	20	28

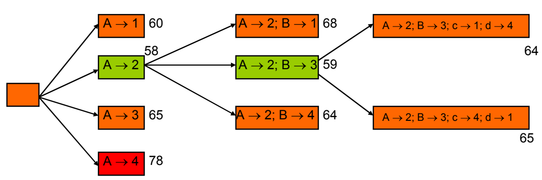
[58..73]



Problema de Atribuição de Tarefas

	1	2	3	4
A	11	12	18	40
B	14	15	13	22
C	11	17	19	23
D	17	14	20	28

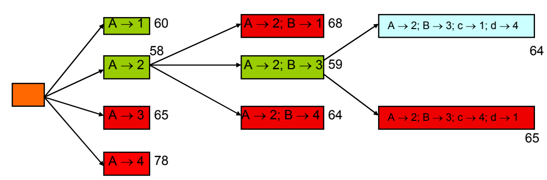
[58..73]



Problema de Atribuição de Tarefas

	1	2	3	4
A	11	12	18	40
B	14	15	13	22
C	11	17	19	23
D	17	14	20	28

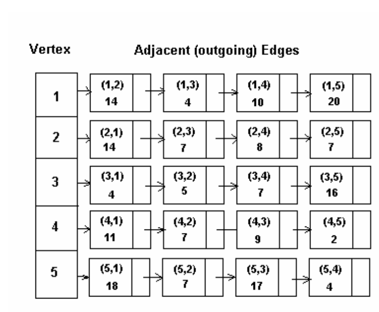
[58..64]



Eficiência de Backtracking e BnB

- Fatores que afetam a eficiência:
  - tempo para gerar o próximo v_k;
  - Cardinalidade de S_k satisfazendo as restrições;
  - tempo de execução da função de poda;
  - Cardinalidade de S_k satisfazendo a função de poda.
- Uma boa função de poda reduz substancialmente o número de nodos considerados.
- Existe um *tradeoff*: uma boa função de poda versus o tempo de avaliá-la.
- Para estimar o número de nodos gerados, podemos usar *Análise Monte-Carlo* (simulação estatística).

Exercício: O Problema do Caixeiro Viajante



Exercício: O Problema do Caixeiro Viajante