Backtracking

Túlio Toffolo – www.toffolo.com.br
Marco Antônio Carvalho – marco.opt@gmail.com

BCC402 – Aula 10

Algoritmos e Programação Avançada

Backtracking

- Backtracking é um **refinamento** do algoritmo de busca por força bruta (ou enumeração exaustiva), no qual boa parte das soluções podem ser eliminadas sem serem explicitamente examinadas.
- Se aplica em:
  - Problemas cuja solução pode ser definida a partir de uma seqüência de decisões.
  - Problemas que podem ser modelados por uma árvore que representa todas as possíveis seqüências de decisão.

Backtracking

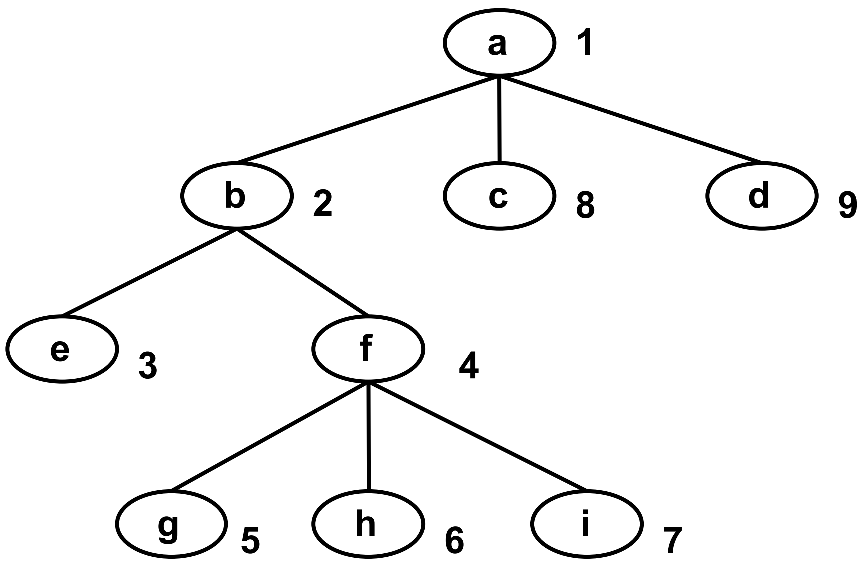
- Se existir mais de uma decisão disponível para cada uma das **n** decisões, a busca exaustiva será exponencial.
- A eficiência da estratégia depende da <u>possibilidade de limitar a busca</u>, ou seja, **<u>podar a árvore</u>** eliminando os ramos que não levam à solução desejada.
- Para tanto é necessário definir um espaço de solução para o problema:
  - Que inclua a **<u>solução ótima</u>**
  - Que possa ser **<u>pesquisada de forma organizada</u>** (tipicamente como uma árvore).

Backtracking

- Técnica em procedimentos de busca que corresponde ao retorno de uma exploração.
- Ex: Busca-em-Profundidade (já visto)
  - Quando chegamos a um nó v pela primeira vez, cada aresta incidente a v é explorada e então o controle volta (backtracks) ao nó a partir do qual v foi alcançado.

Busca em Profundidade

- Ordem de visita dos nós da árvore de busca:



- a, b, e, (b), f, g, (f), h, (f), i, (f), (b), (a), c, (a), d
  - (parênteses indicam caminho em backtracking)

BACKTRACKING

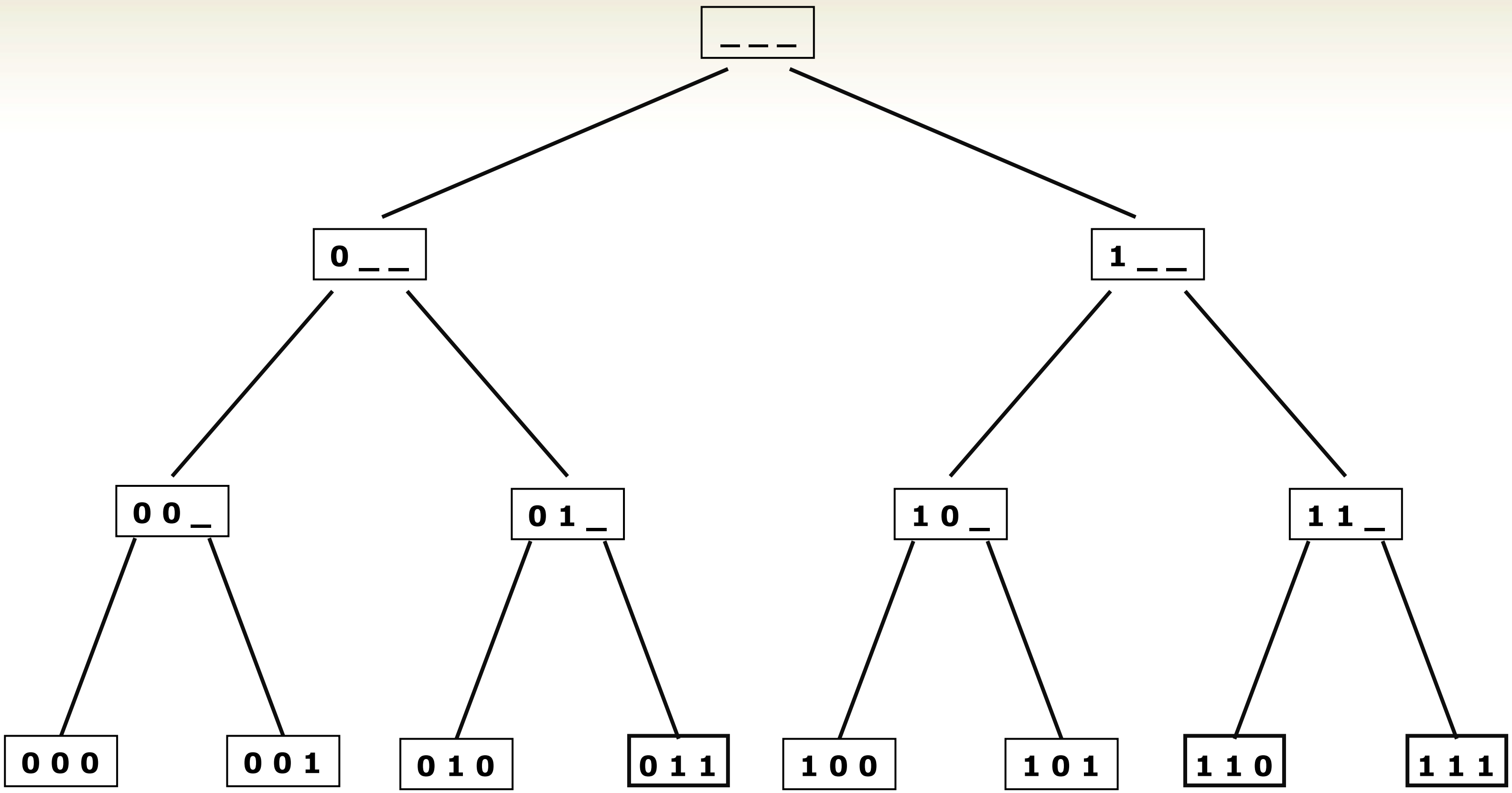
EXEMPLO

Exemplo

- Problema:
  - Encontrar todos os números binários de 3 bits em que a soma de 1's seja maior ou igual a 2.

- A única forma de resolver é checar todas as possíveis combinações?
- Estas possibilidades serão chamadas doravantes de **espaço de busca**.

Exemplo



Como reduzir o <u>espaço de busca</u>?

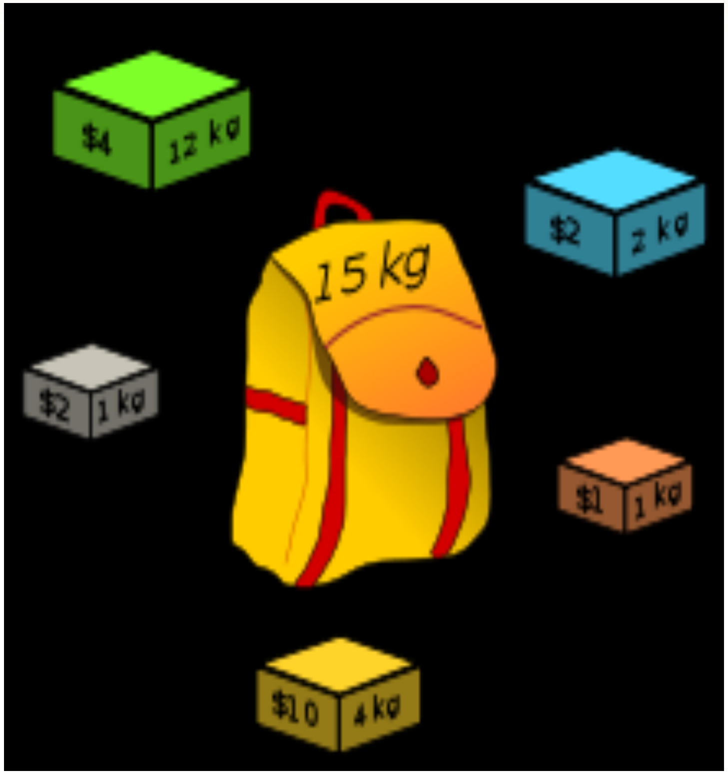
APLICAÇÃO DE BACKTRACKING

PROBLEMA DA MOCHILA

Problema da Mochila

- Deve-se preencher uma mochila com diversos itens com pesos e valores (benefícios) diferentes.

- O objetivo é que se preencher a mochila com o maior valor (benefício) possível, sem ultrapassar a capacidade (peso máximo).



Problema da Mochila

- Entrada:
  - capacidade da mochila, **K**
  - **n** itens com pesos $\mathbf{p_i}$ e valores $\mathbf{v_i}$
- O objetivo é obter um conjunto **S** de itens tais que:
  - A soma dos pesos dos itens em **S** seja menor ou igual a **K**
  - A soma dos valores dos itens em **S** seja a maior possível

Guloso não funciona

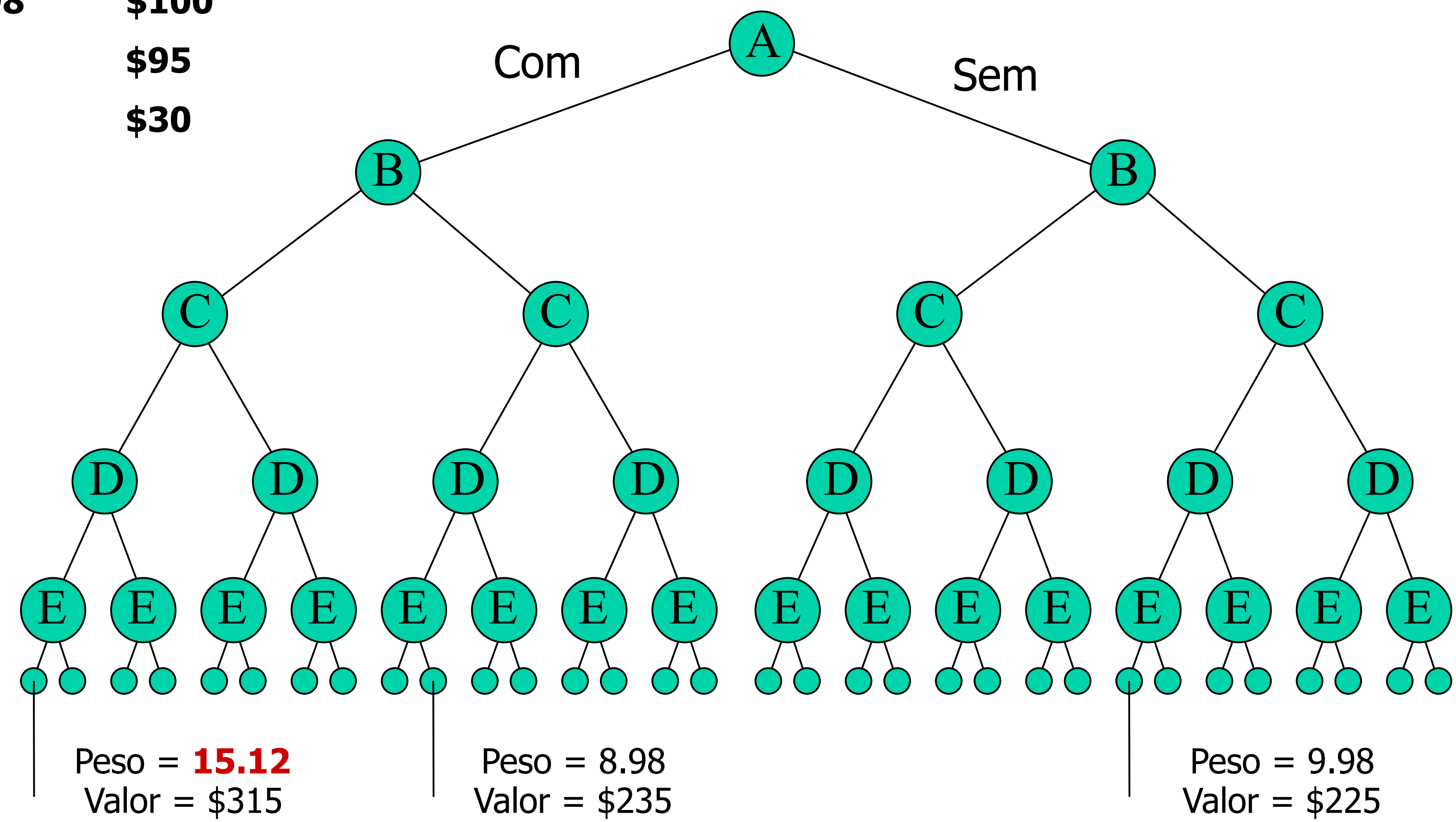
- Qual item escolher primeiro?
  - Maior densidade?
  - Maior valor?
  - Menor peso?

Solução de Força Bruta

- Gerar todas as possiveis combinações
  - Com n itens, existem 2^n soluções
  - Checar se cada solução satisfaz limite peso
  - Salvar a condição que melhor representa a solução
- Pode ser representada como uma árvore

Solução Força Bruta – Mochila de 10kg

A	**2**	**$40**
B	**3,1**	**$50**
C	**1,98**	**$100**
D	**5**	**$95**
E	**3**	**$30**

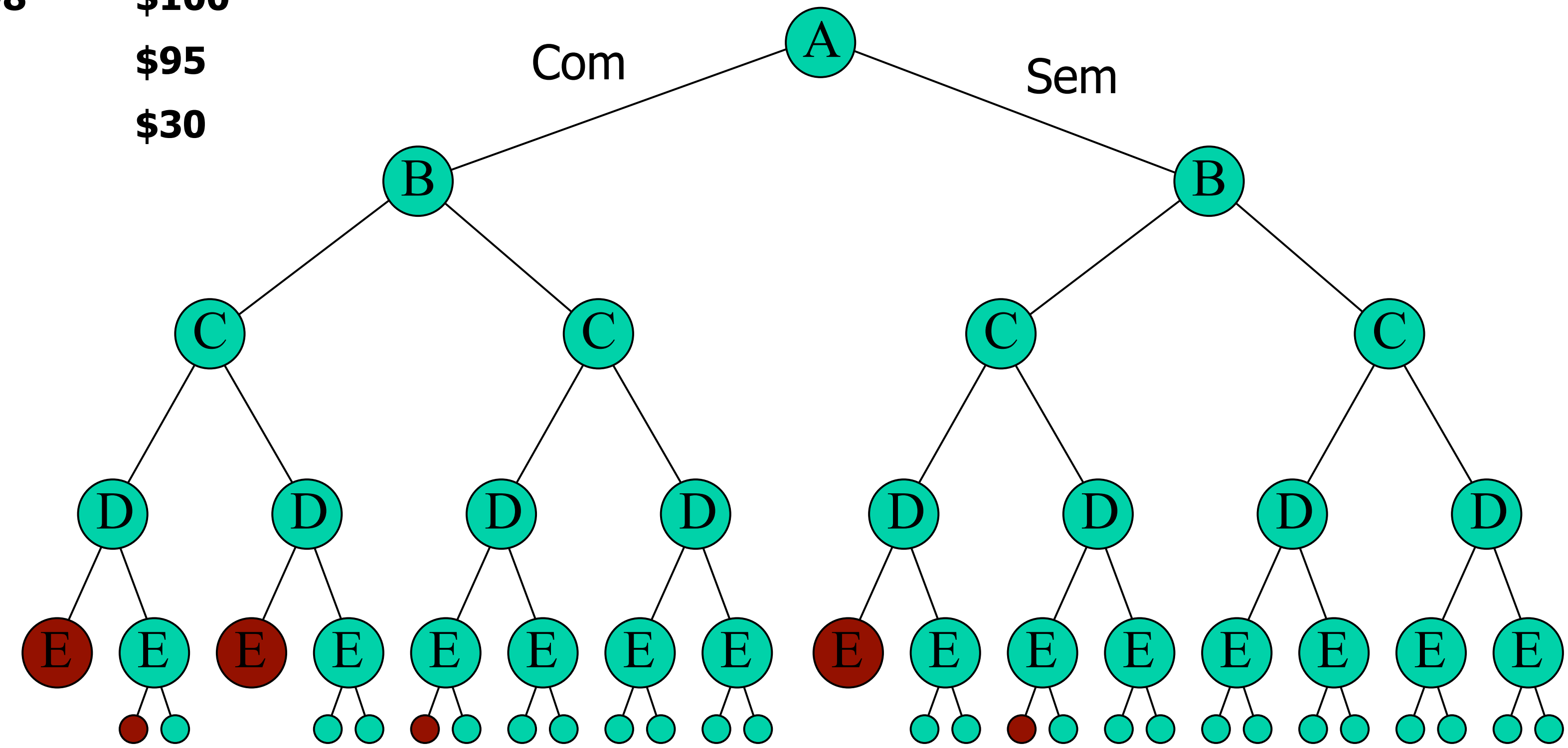


Backtracking

- Se alcançamos um ponto em que a solução não é mais viável, não precisamos continuar exporando a solução.
  - Podemos "voltar atrás" (backtrack) a partir deste ponto.
- No exemplo backtracking se torna bastante útil:
  - Na medida em que o número de itens cresce.
  - Na medida em que a capacidade da mochila diminui.

Backtracking < 10kg

A	**2**	**$40**
B	**3,1**	**$50**
C	**1,98**	**$100**
D	**5**	**$95**
E	**3**	**$30**

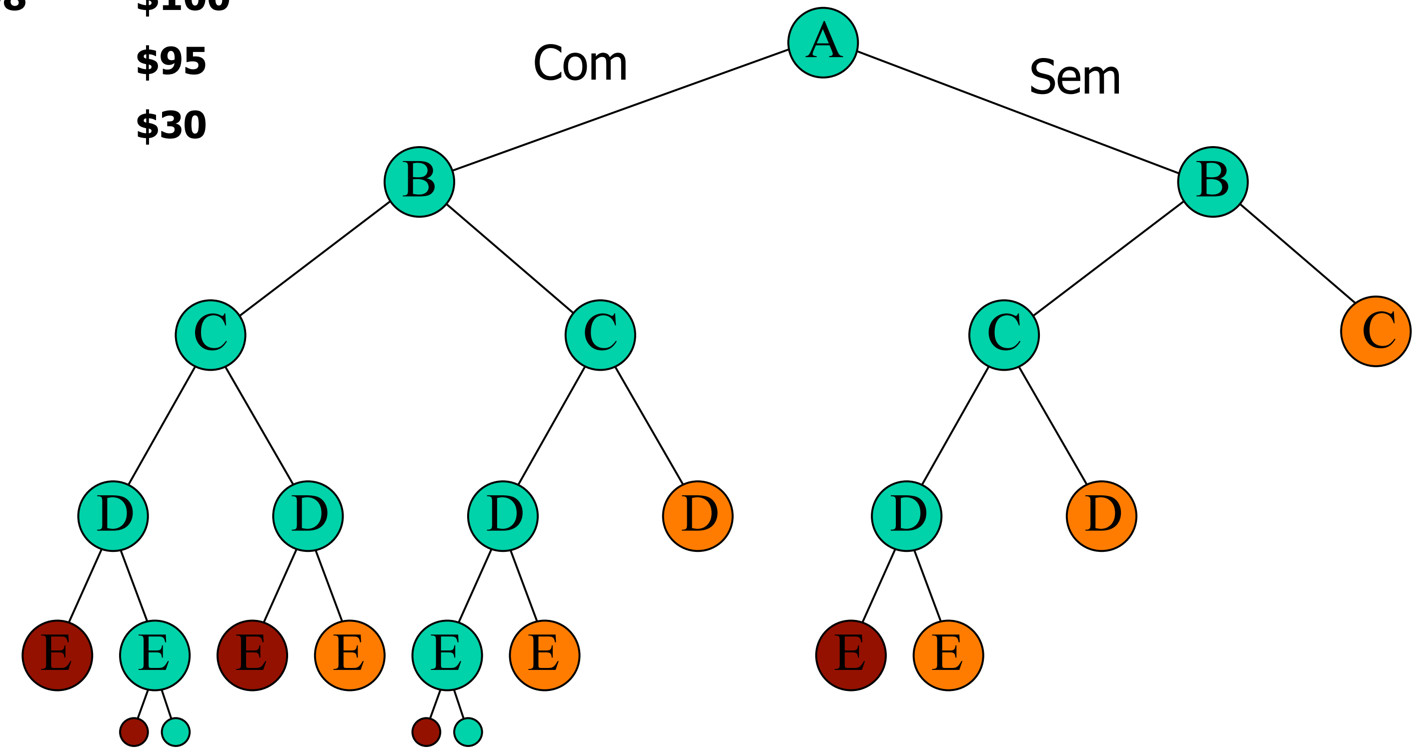


Backtracking

- Pode-se voltar atrás também quando se sabe que a melhor solução da subárvore é pior do que a melhor solução já encontrada.
  - Fundamento utilizado por muitos algoritmos
  - Exemplo típico: **Branch and Bound**

Backtracking com cortes por qualidade

A	**2**	**$40**
B	**3,1**	**$50**
C	**1,98**	**$100**
D	**5**	**$95**
E	**3**	**$30**



Backtracking - Solução genérica

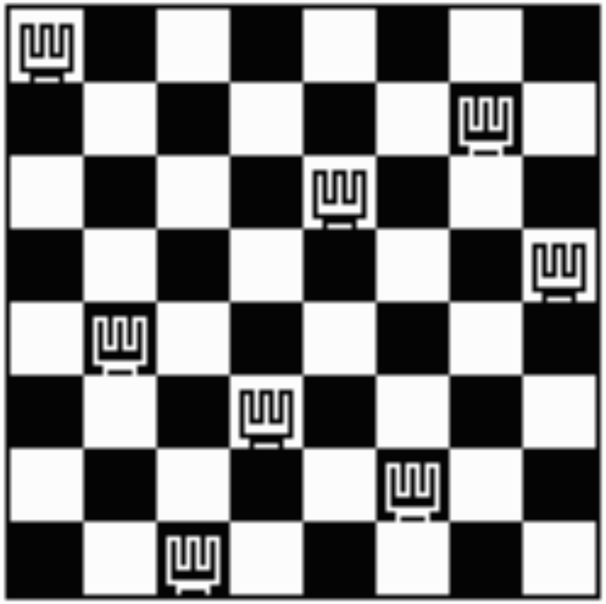
```
def backtrack(v): # v é o nó sendo pesquisado
    if (promissor(v)):
        if (existe_solucao(v)):
            armazena solucao(v)
        else:
            for filho in v:
                backtrack(filho)
}
```

APLICAÇÃO DE BACKTRACKING

PROBLEMA DAS OITO RAINHAS

O Problema das Oito Rainhas

- Colocar oito rainhas num tabuleiro de xadrez, de forma que nenhuma delas seja atacada por outra.
- Em outras palavras: escolher oito posições no tabuleiro de forma que não haja duas delas compartilhando a mesma linha, coluna ou diagonal.



Algoritmo: Problema das Oito Rainhas

- Enquanto não houver oito rainhas no tabuleiro faça:
  - Se na próxima linha existir uma coluna que não está sob ataque de uma rainha já no tabuleiro, coloque uma rainha nesta posição
  - Caso contrário volte à linha anterior (backtrack)
  - Mova a rainha o mínimo necessário para a direita de forma que ela não fique sob ataque

O Problema das Oito Rainhas

Q							

O Problema das Oito Rainhas

Q							
		Q					

O Problema das Oito Rainhas

Q							
		Q					
				Q			

O Problema das Oito Rainhas

Q							
		Q					
				Q			
	Q						

O Problema das Oito Rainhas

Q							
		Q					
				Q			
	Q						
			Q				

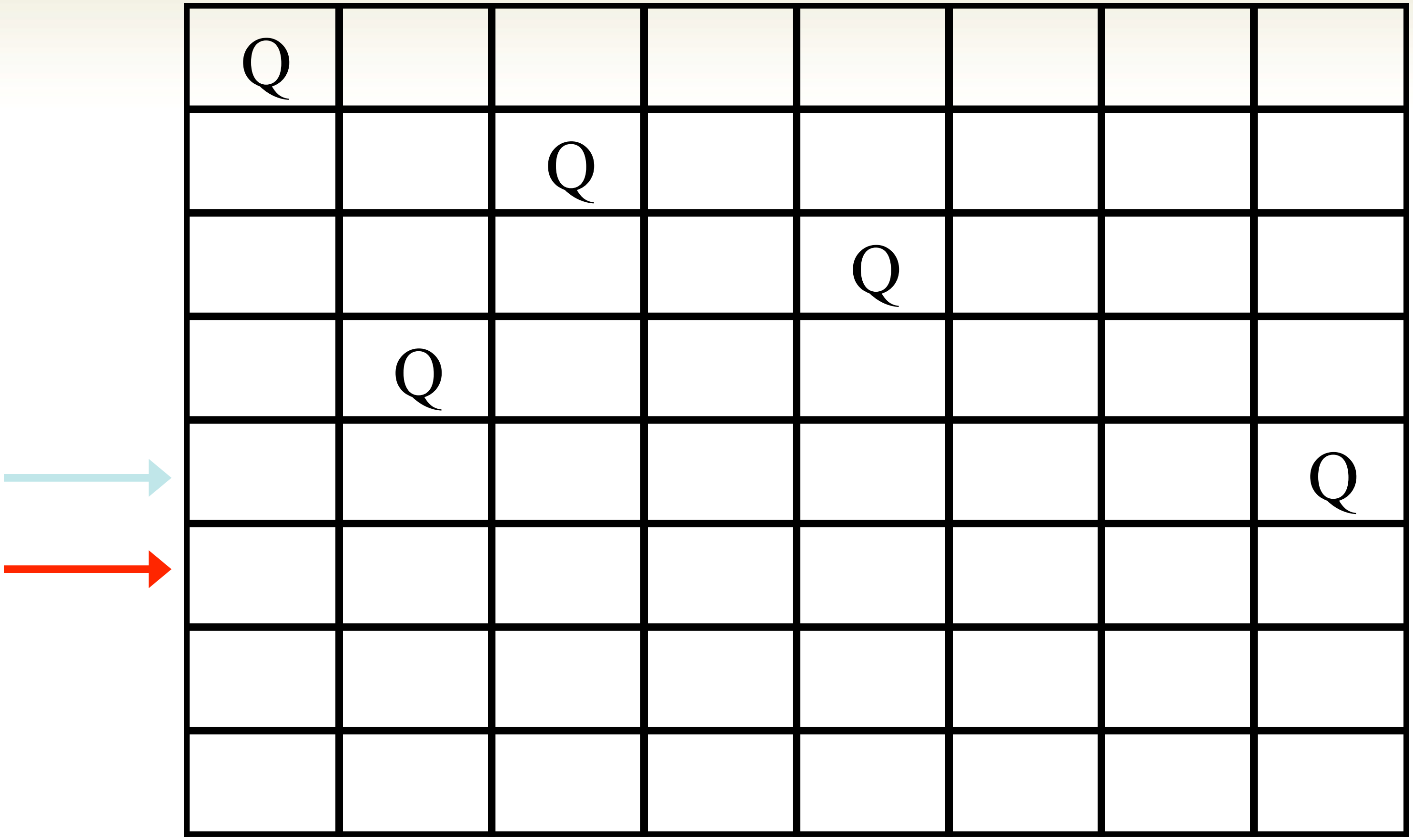
O Problema das Oito Rainhas

Q							
		Q					
				Q			
	Q						
			Q				

O Problema das Oito Rainhas

Q							
		Q					
				Q			
	Q						
							Q

O Problema das Oito Rainhas



O Problema das Oito Rainhas

Q							
		Q					
				Q			
	Q						

O Problema das Oito Rainhas

Q							
		Q					
				Q			
						Q	

O Problema das Oito Rainhas

Q							
		Q					
				Q			
						Q	
	Q						

O Problema das Oito Rainhas

Q							
		Q					
				Q			
						Q	
	Q						
			Q				

O Problema das Oito Rainhas

Q							
		Q					
				Q			
						Q	
	Q						
			Q				
					Q		

O Problema das Oito Rainhas

Q							
		Q					
				Q			
						Q	
	Q						
			Q				
					Q		

O Problema das Oito Rainhas

Q							
		Q					
				Q			
						Q	
	Q						
			Q				

O Problema das Oito Rainhas

Q							
		Q					
				Q			
						Q	
	Q						
			Q				

O Problema das Oito Rainhas

Q							
		Q					
				Q			
						Q	
	Q						

O Problema das Oito Rainhas

Q							
		Q					
				Q			
						Q	
	Q						

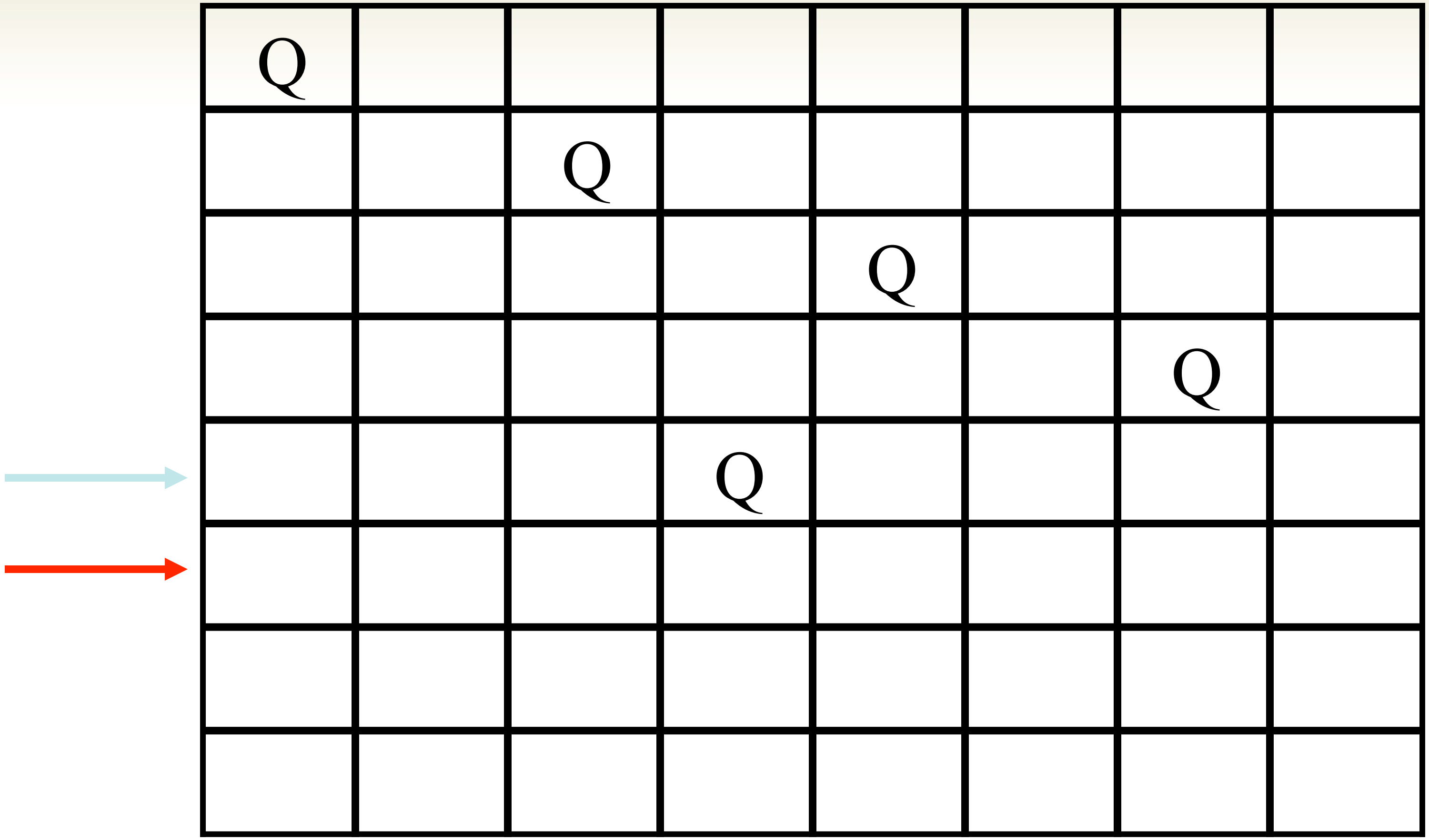
O Problema das Oito Rainhas

Q							
		Q					
				Q			
						Q	
	Q						

O Problema das Oito Rainhas

Q							
		Q					
				Q			
						Q	
			Q				

O Problema das Oito Rainhas



O Problema das Oito Rainhas

Q							
		Q					
				Q			
						Q	

O Problema das Oito Rainhas

Q							
		Q					
				Q			

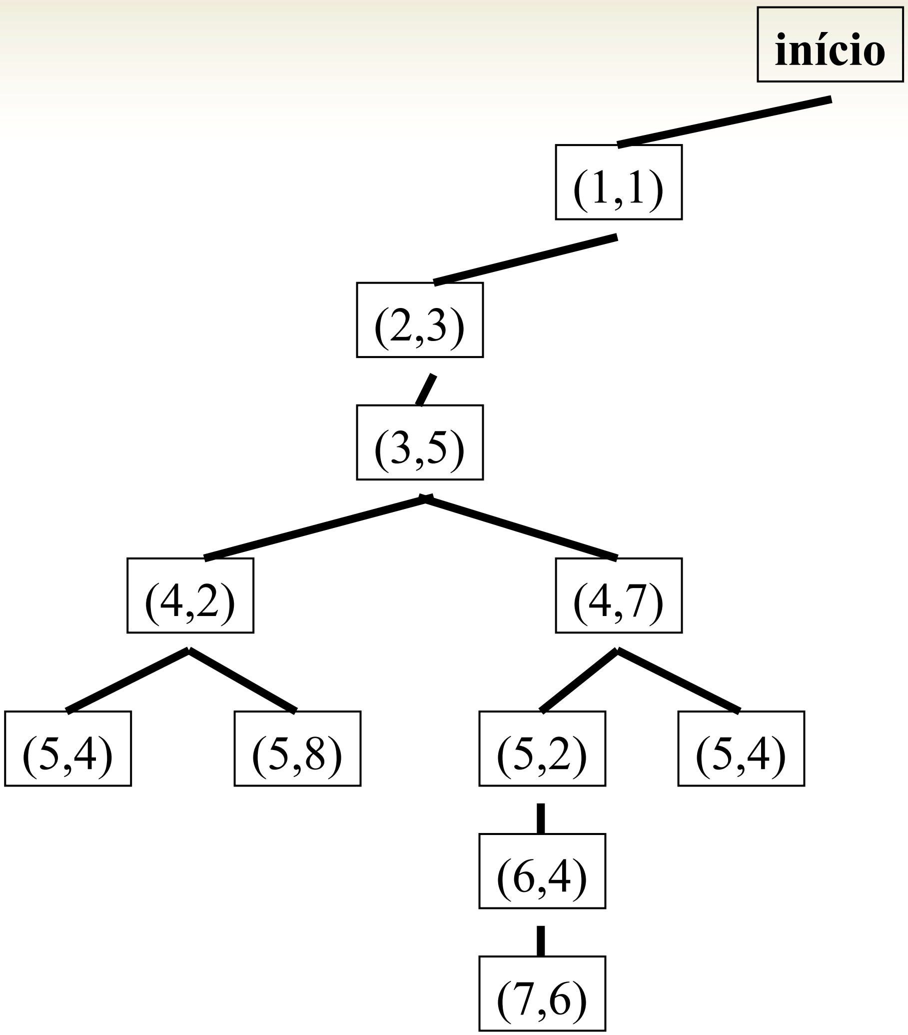
O Problema das Oito Rainhas

Q							
		Q					
				Q			
							Q

O Problema das Oito Rainhas

Q							
		Q					
				Q			
							Q

O Problema das Oito Rainhas

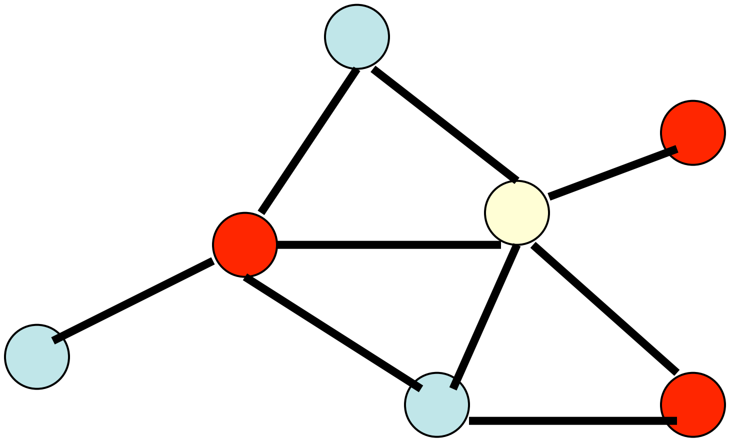


APLICAÇÃO DE BACKTRACKING

PROBLEMA DA 3-COLORAÇÃO

Problema da 3-Coloração

- Dado um grafo G(V, E), encontrar uma 3-coloração de G, ou seja, definir uma função cor $V \rightarrow \{1, 2, 3\}$ de forma que:
  - Se (u, v) é uma aresta em E, então cor(u) $\neq$ cor(v).



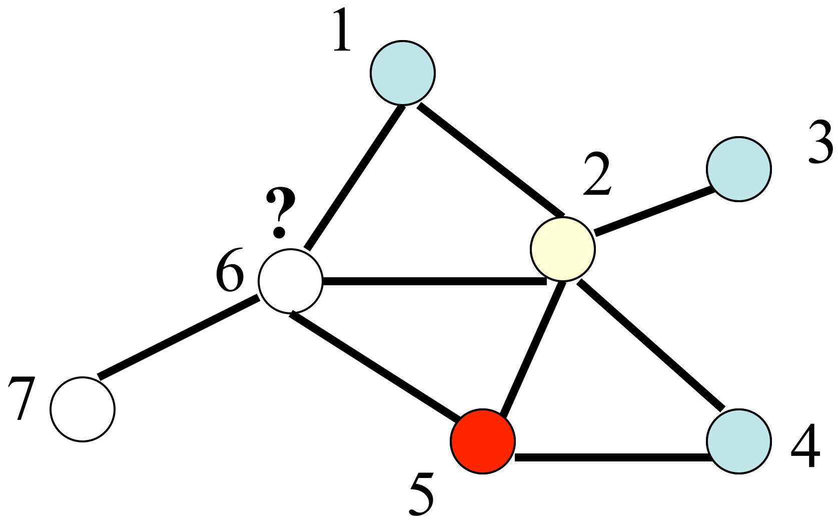
Algoritmo para Problema da 3-Coloração

```
para v ← 1 até n faça
    cor(v) ← 0;
cor(1) ← 1;
se (3-Colorir(1)) então
    imprima  cor
senão
    imprima "Impossível 3-colorir"
```

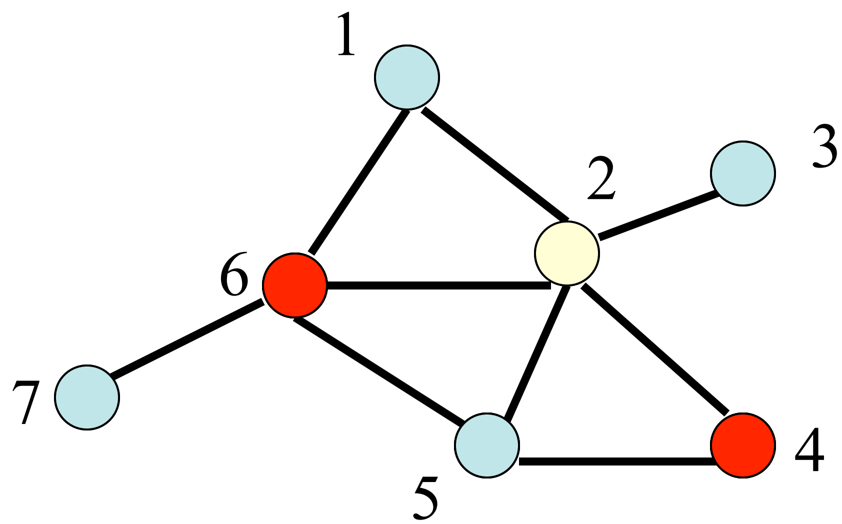
Algoritmo para Problema da 3-Coloração

```
procedimento 3-Colorir(v):
    se v = n então retorne (suc)
    tome o vértice z ← v+1
    se z não tem vizinhos w com cor(w) = 1 então
        cor (z) ← 1;
        se (3-Colorir (z)) então retorne (suc)
    se z não tem vizinhos w com cor(w) = 2 então
        cor (z) ← 2;
        se (3-Colorir (z)) então retorne (suc)
    se z não tem vizinhos w com cor(w) = 3 então
        cor (z) ← 3;
        se (3-Colorir (z)) então retorne (suc)
    cor (z) ← 0;
    retorne (fail); // backtrack feito pela recursão
```

Problema da 3-Coloração



O Problema da 3-Coloração



APLICAÇÃO DE BACKTRACKING

BRANCH-AND-BOUND

Branch-and-Bound

- Uma outra técnica muito usada que está relacionada com Backtracking é a técnica chamada **Branch-and-Bound**.
- Branch-and-Bound é uma técnica de exploração mais sofisticada, que procura explorar opções (branch), mas colocando um limite quantitativo (bound), com o objetivo de evitar buscas em espaços menos promissores.

Branch-and-Bound

- Exemplo:
  - Análise das possíveis seqüências de lances na implementação de um jogo, explorando os lances que parecem melhores, uma vez que o número total de possibilidades é muito grande.

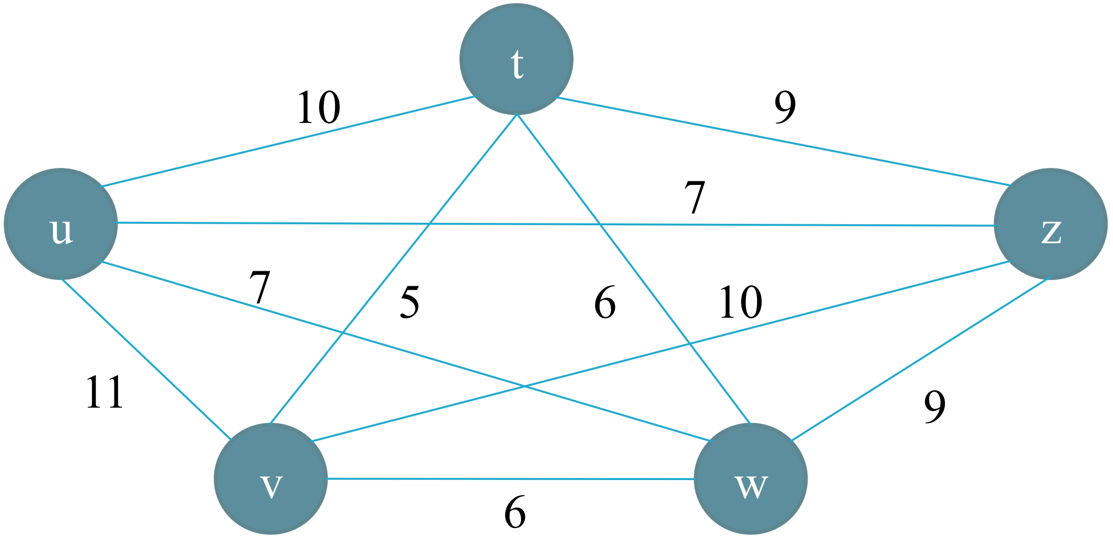
- Esta técnica é muito utilizada também na resolução de modelos de **Programação Linear Inteira**, podendo ser combinada com outras técnicas, gerando, por exemplo:
  - Branch-and-cut
  - Branch-and-price
  - Branch-and-lift

APLICAÇÃO DE BACKTRACKING

CAIXEIRO VIAJANTE

Problema do Caixeiro Viajante

- O problema do caixeiro viajante consiste em minimizar o custo de um caixeiro viajante que deseja percorrer **n** cidades, visitando cada cidade apenas uma vez, e retornar para casa.



Problema do Caixeiro Viajante

Usando **<u>forca bruta</u>**:

Se calcularmos um bilhão de solucoes por segundo para o problema do caixeriro com 30 cidades demoraria 8 quadrilhoes de anos para achar a melhor solucao.

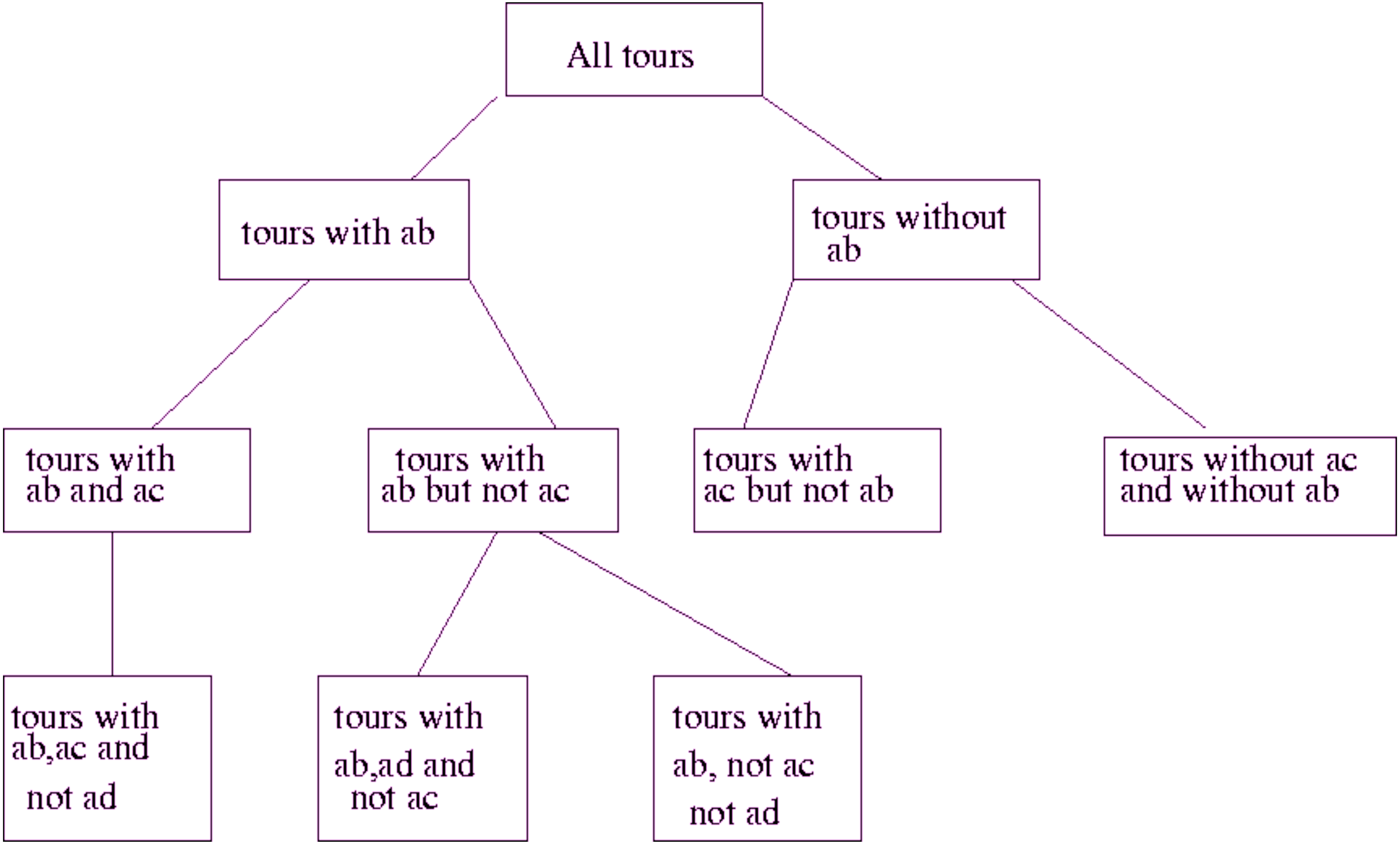
melhor solucao.
de anos para achar a

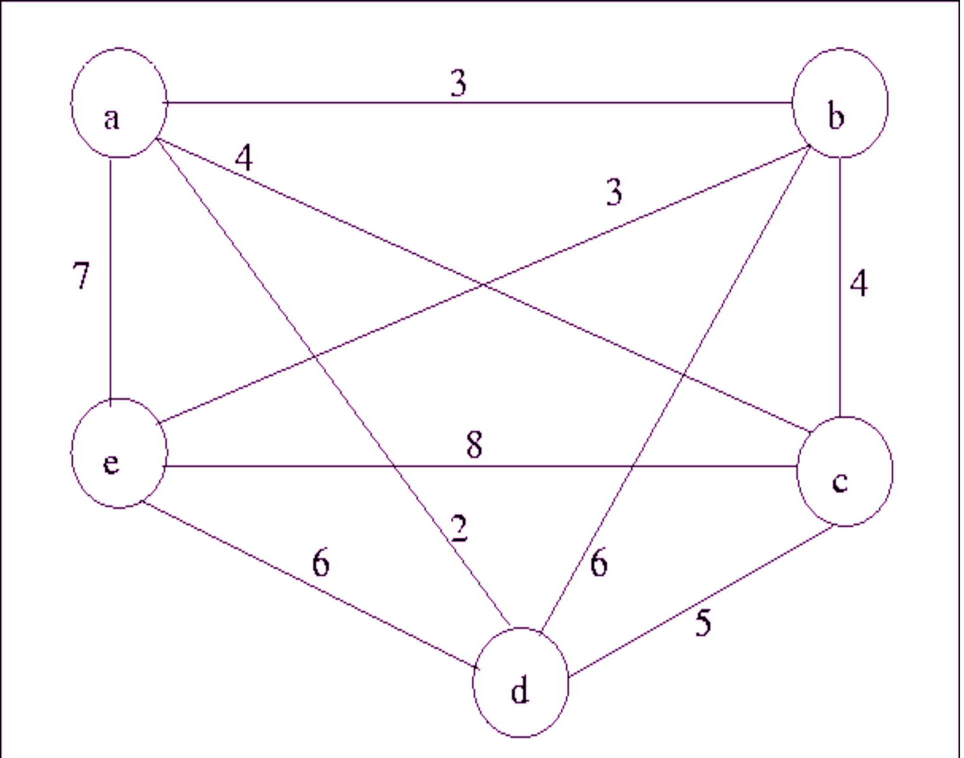
Problema do Caixeiro Viajante

- Construindo a árvore de soluções:
  - Um nó que não é folha representa todas as viagens que começam no caminho armazenado pelo nó.
  - Cada folha representa uma viagem ou caminho abandonado.
- O algoritmo deve expandir cada nó promissor e parar quando todos os nós promissores tiverem sido expandidos.
- Durante o procedimento devem ser abandonados todos os nós não promissores.

Problema do Caixeiro Viajante

- Nó promissor: o limite inferior do nó é menor do que o atual tamanho da viagem
- Nó não promissor: o limite inferior do nó não é menor do que o atual tamanho da viagem


All tours
tours with ab
tours without ab
tours with ab and ac
tours with ab but not ac
tours with ac but not ab
tours without ac and without ab
tours with ab,ac and not ad
tours with ab,ad and not ac
tours with ab, not ac not ad



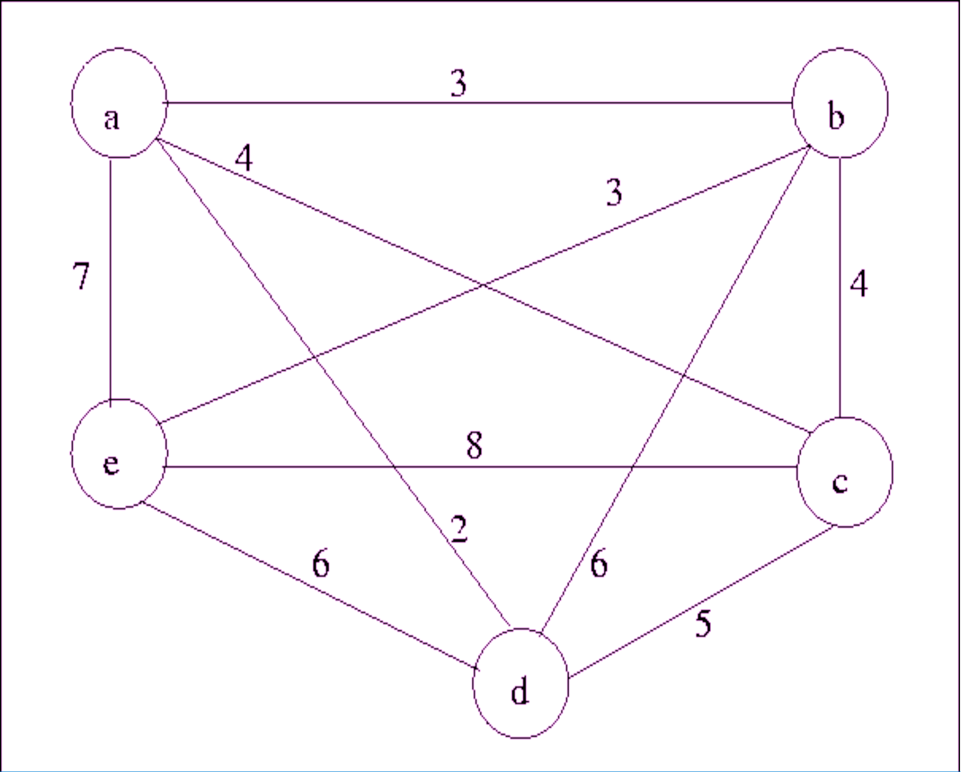
	A	B	C	D	E
A	-	3	4	2	7
B	3	-	4	6	3
C	4	4	-	5	8
D	2	6	5	-	6
E	7	3	8	6	-

Limite Inferior para cada nó: **Nenhum caminho pode custar mais que metade do total do custo dos vertices incidentes de menor custo no nó**

Caso A - Sem restricoes

a	(a, d), (a, b)	5
b	(a, b), (b, e)	6
c	(c, b), (c, a)	8
d	(d, a), (d, c)	7
e	(e, b), (e, f)	9

ABCDE = (5 + 6 + 8 + 7 + 9) = 17.5



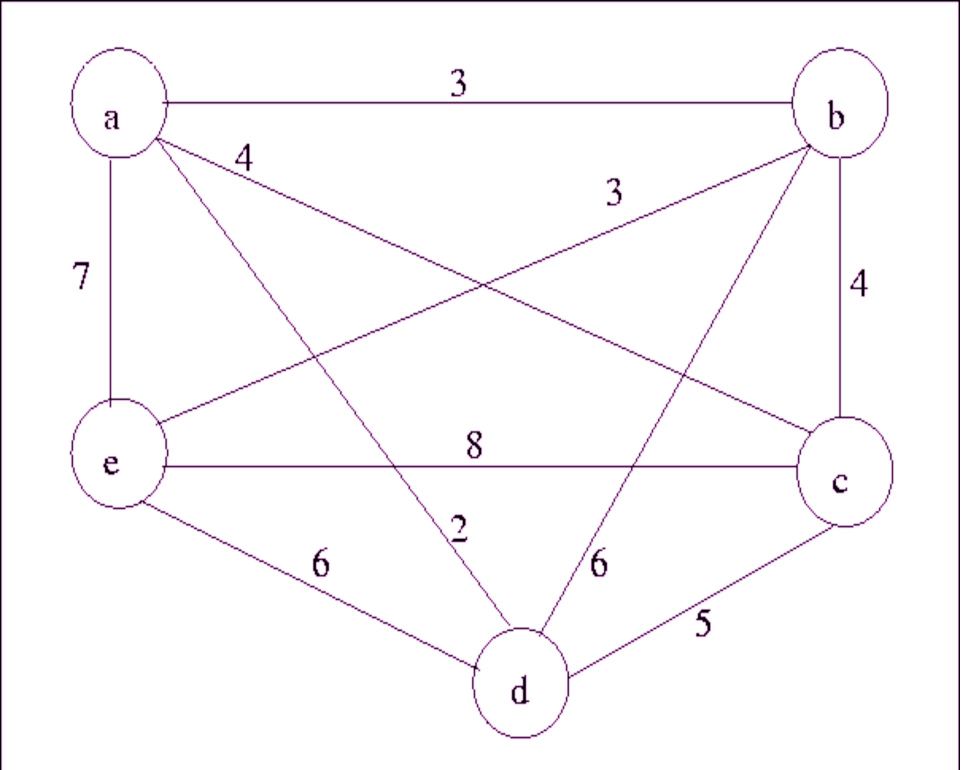
	A	B	C	D	E
A	-	3	4	2	7
B	3	-	4	6	3
C	4	4	-	5	8
D	2	6	5	-	6
E	7	3	8	6	-

Limite Inferior para cada nó: **Nenhum caminho pode custar mais que metade do total do custo dos vertices incidentes de menor custo no nó**

Caso B – Com ab

a	(a, d), (a, b)	5
b	(a, b), (b, e)	6
c	(c, b), (c, a)	8
d	(d, a), (d, c)	7
e	(e, b), (e, f)	9

ABCDE = (5 + 6 + 8 + 7 + 9) = 17.5



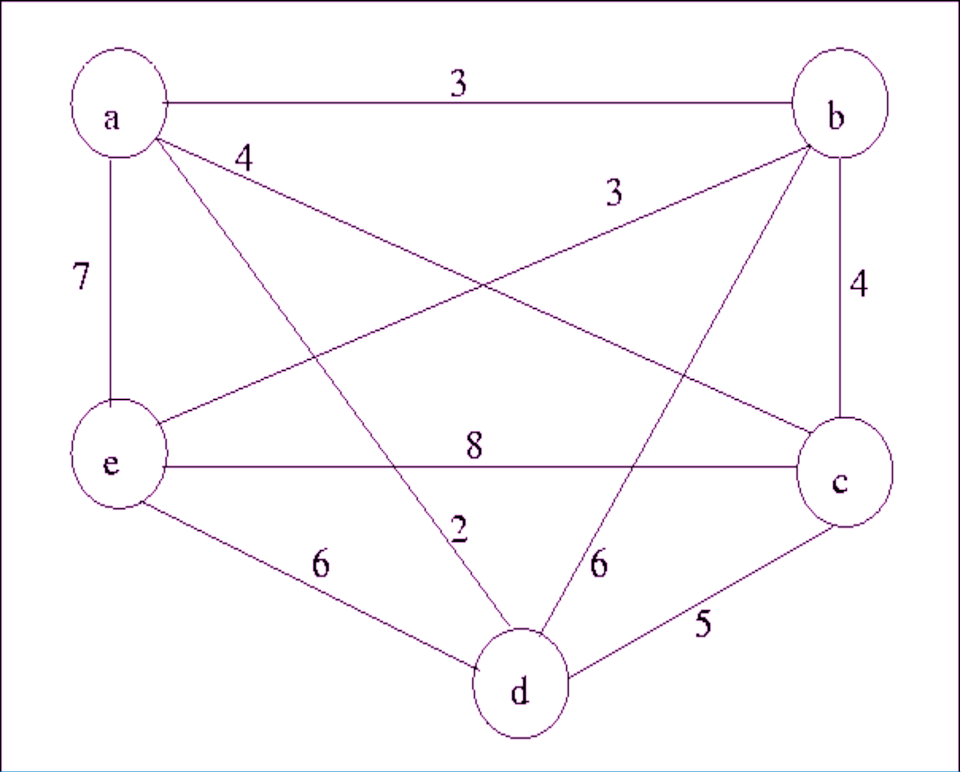
	A	B	C	D	E
A	-	3	4	2	7
B	3	-	4	6	3
C	4	4	-	5	8
D	2	6	5	-	6
E	7	3	8	6	-

Limite Inferior para cada nó: **Nenhum caminho pode custar mais que metade do total do custo dos vertices incidentes de menor custo no nó**

Caso C - Calculo sem AB

a (a,d), (a,c) 6
b (b,c), (b,e) 7
c (c,b), (c,a) 8
d (d,a), (d,c) 7
e (e,b), (e,f) 9

ABCDE = (6 + 7 + 8 + 7 + 9) = 18.5



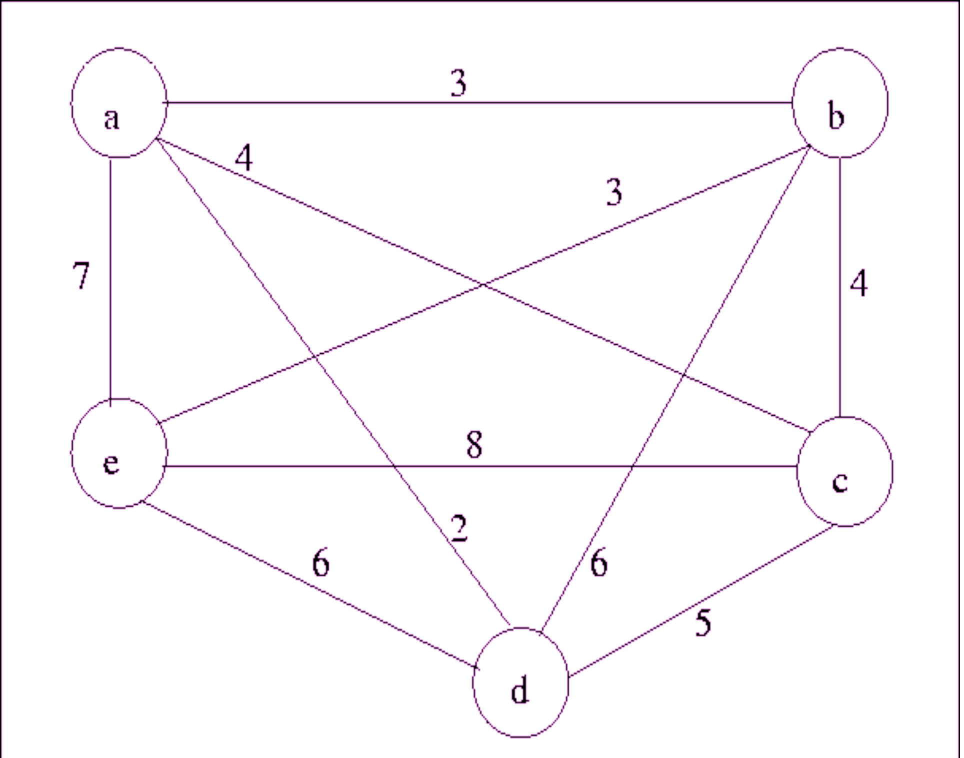
	A	B	C	D	E
A	-	3	4	2	7
B	3	-	4	6	3
C	4	4	-	5	8
D	2	6	5	-	6
E	7	3	8	6	-

Limite Inferior para cada nó: **Nenhum caminho pode custar mais que metade do total do custo dos vertices incidentes de menor custo no nó**

Caso D - com ab ac sem ad ae

a (a,b), (a,c) 7
B (a,b), (b,e) 6
c (c,b), (c,a) 8
d (d,b), (d,c) 11
e (e,b), (e,f) 9

ABCDE = (7 + 6 + 8 + 11 + 9) = 20.5



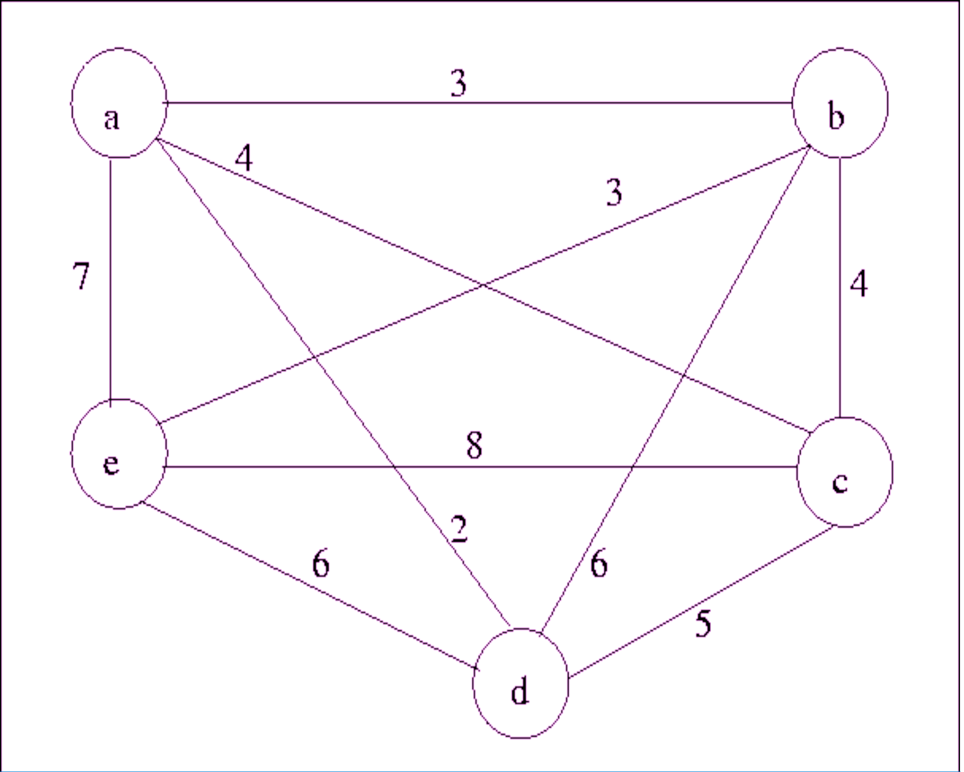
	A	B	C	D	E
A	-	3	4	2	7
B	3	-	4	6	3
C	4	4	-	5	8
D	2	6	5	-	6
E	7	3	8	6	-

Limite Inferior para cada nó: **Nenhum caminho pode custar mais que metade do total do custo dos vertices incidentes de menor custo no nó**

Caso E - Com ab, Sem ac

a	(a, d), (a, b)	5
b	(a, b), (b, e)	6
c	(c, b), (c,d)	9
d	(d, a), (d, c)	7
e	(e, b), (e, f)	9

ABCDE = (5 + 6 + 9 + 7 + 9) = 18



	A	B	C	D	E
A	-	3	4	2	7
B	3	-	4	6	3
C	4	4	-	5	8
D	2	6	5	-	6
E	7	3	8	6	-

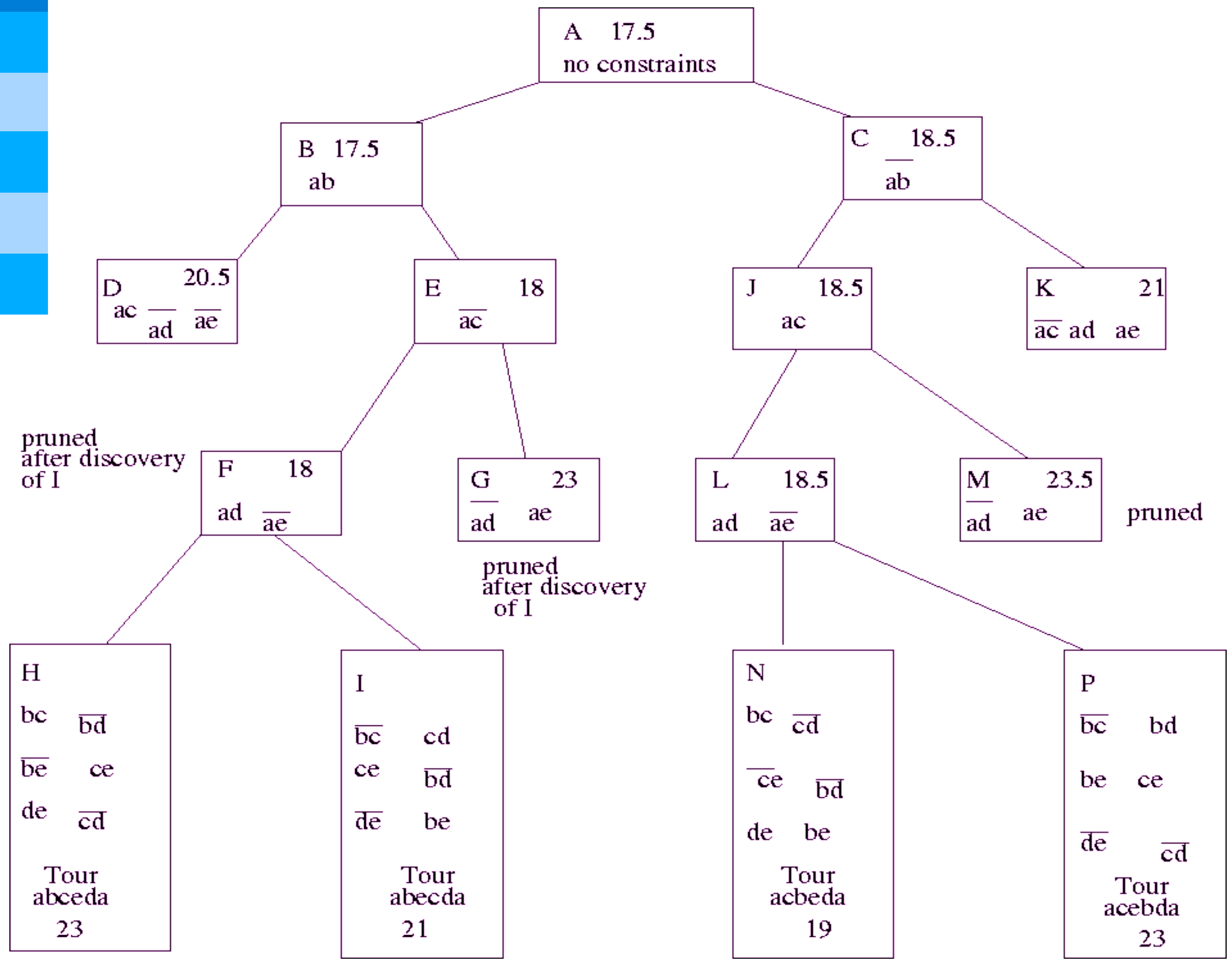
Limite Inferior para cada nó: **Nenhum caminho pode custar mais que metade do total do custo dos vertices incidentes de menor custo no nó**

Caso F- Com ab, ad sem ac, ae

a	(a, d), (a, b)	5
b	(a, b), (b, e)	6
c	(c, b), (c, d)	9
d	(d, a), (d, c)	7
e	(e, b), (e, f)	9

ABCDE = (5 + 6 + 8 + 7 + 9)
=18

	A	B	C	D	E
A	-	3	4	2	7
B	3	-	4	6	3
C	4	4	-	5	8
D	2	6	5	-	6
E	7	3	8	6	-



Algoritmo: Caixeiro Viajante com Backtracking

```
Algorithm TSP_Backtrack (A, , lengthSoFar , minCost)

n ← length[A] // numero de elementos em A
if l = n
  then minCost ← lengthSoFar + distance[A[n], A[1]]
  else for i ← l+1 to n
      do Swap A[l+1], A[i]
        newLength ← lengthSoFar + distance[A[ l ], A[ l+ 1]]
         if newLength minCost // solução não promissora
           then skip
        else minCost ← min(minCost, TSP_Backtrack(A, l+1,
                                   newLength, minCost))
           Swap A[l+1], A[i] // desfaz a ação
return minCost
```

Perguntas?