Busca 

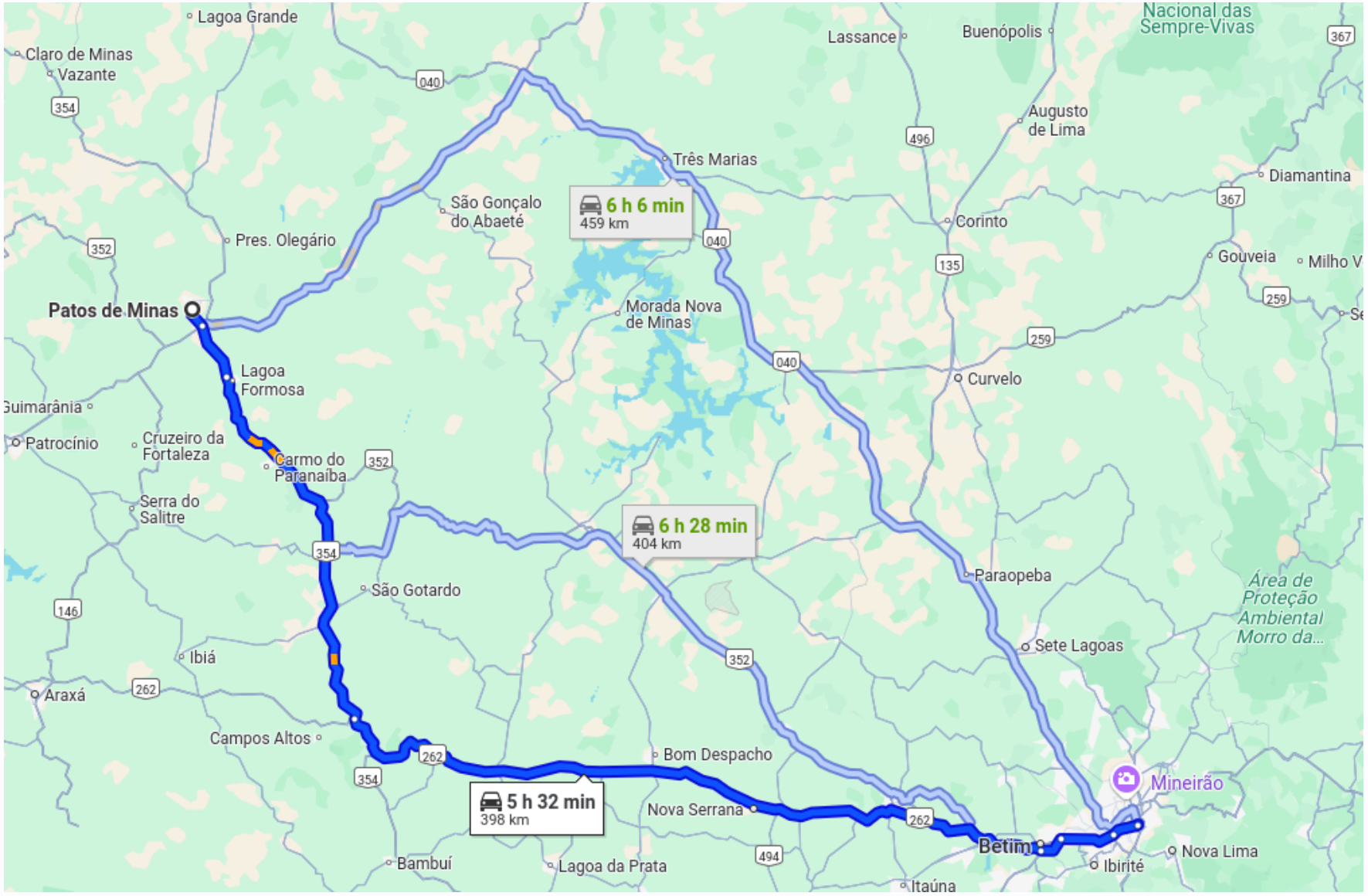
Prof. Me. Alexandre Henrick

Sistemas de Informação - 7º P

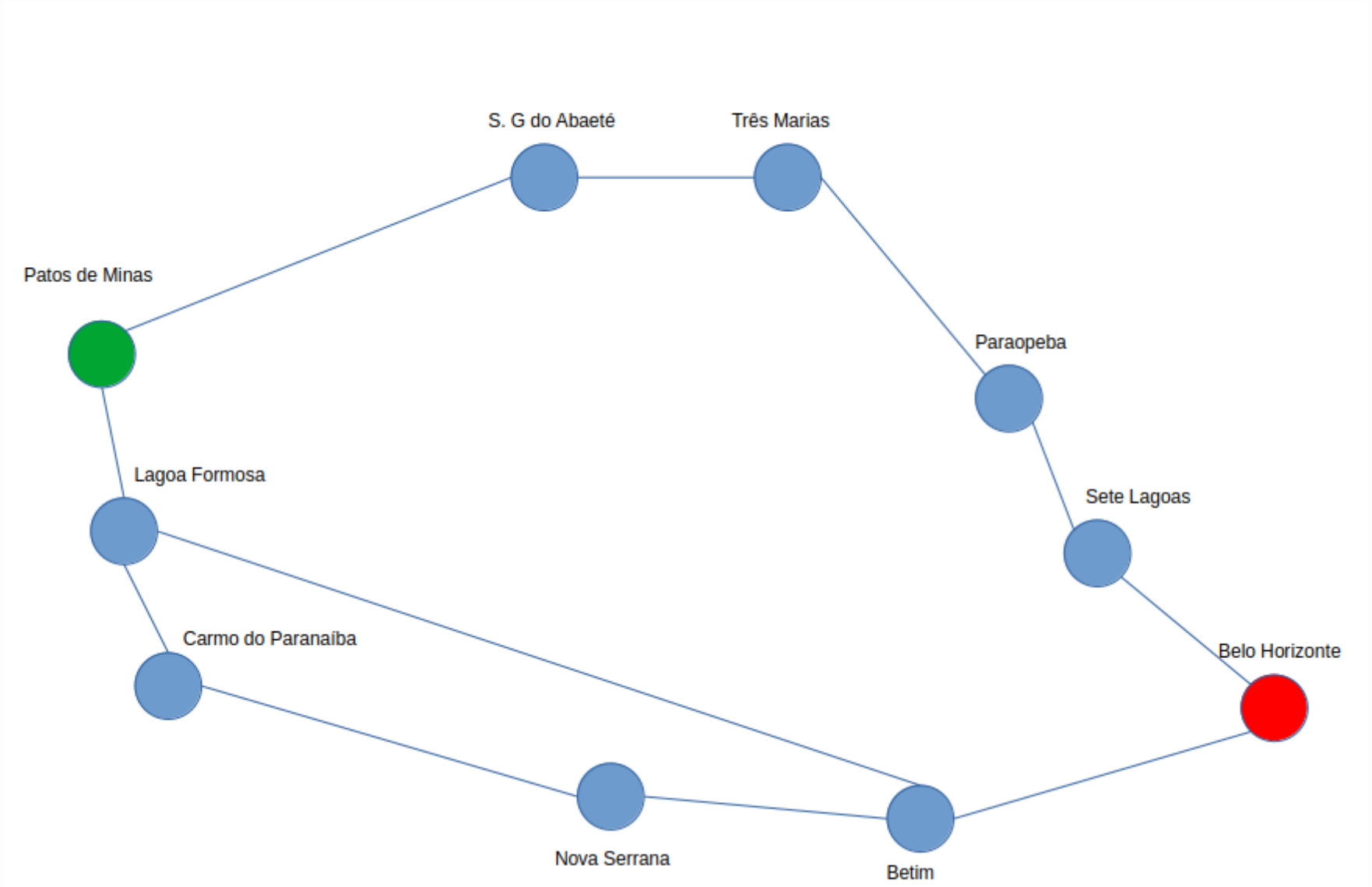
Busca

- Algoritmos que performam busca podem ser considerados **agentes inteligentes**
- Encontram o **caminho** até chegar no objetivo (destino)
- Grafos são estruturas de dados bastante utilizadas em busca

Busca

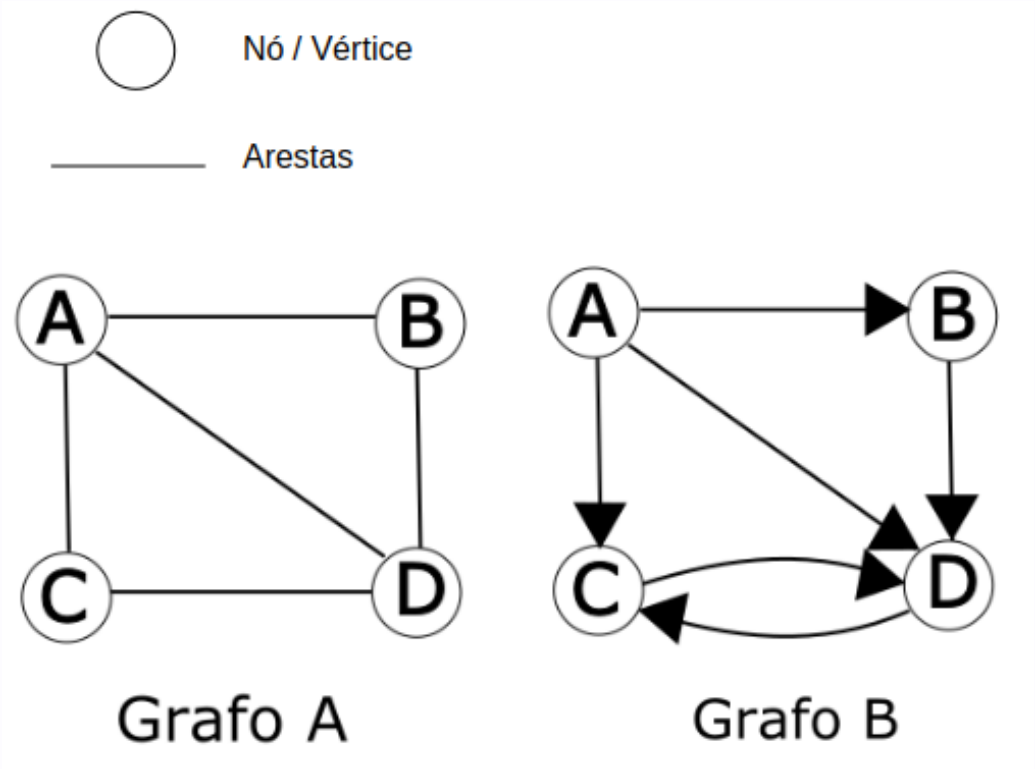

Lagoa Grande
Claro de Minas
Vazante
Lassance
Buenópolis
Nacional das Sempre-Vivas
Augusto de Lima
Três Marias
São Gonçalo do Abaeté
6 h 6 min
459 km
Diamantina
Corinto
Pres. Olegário
Gouveia
Patos de Minas
Morada Nova de Minas
Lagoa Formosa
Curvelo
Guimarânia
Patrocínio
Cruzeiro da Fortaleza
Carmo do Paranaíba
Serra do Salitre
6 h 28 min
404 km
São Gotardo
Paraopeba
Área de Proteção Ambiental Morro da...
Sete Lagoas
Ibiá
Araxá
Campos Altos
Bom Despacho
Mineirão
5 h 32 min
398 km
Nova Serrana
Betim
Nova Lima
Ibirité
Bambuí
Lagoa da Prata
Itaúna

Busca


S. G do Abaeté
Três Marias
Patos de Minas
Paraopeba
Lagoa Formosa
Sete Lagoas
Carmo do Paranaíba
Belo Horizonte
Nova Serrana
Betim

Grafos

- **Grafo A**: Não direcionado
- **Grafo B**: Direcionado



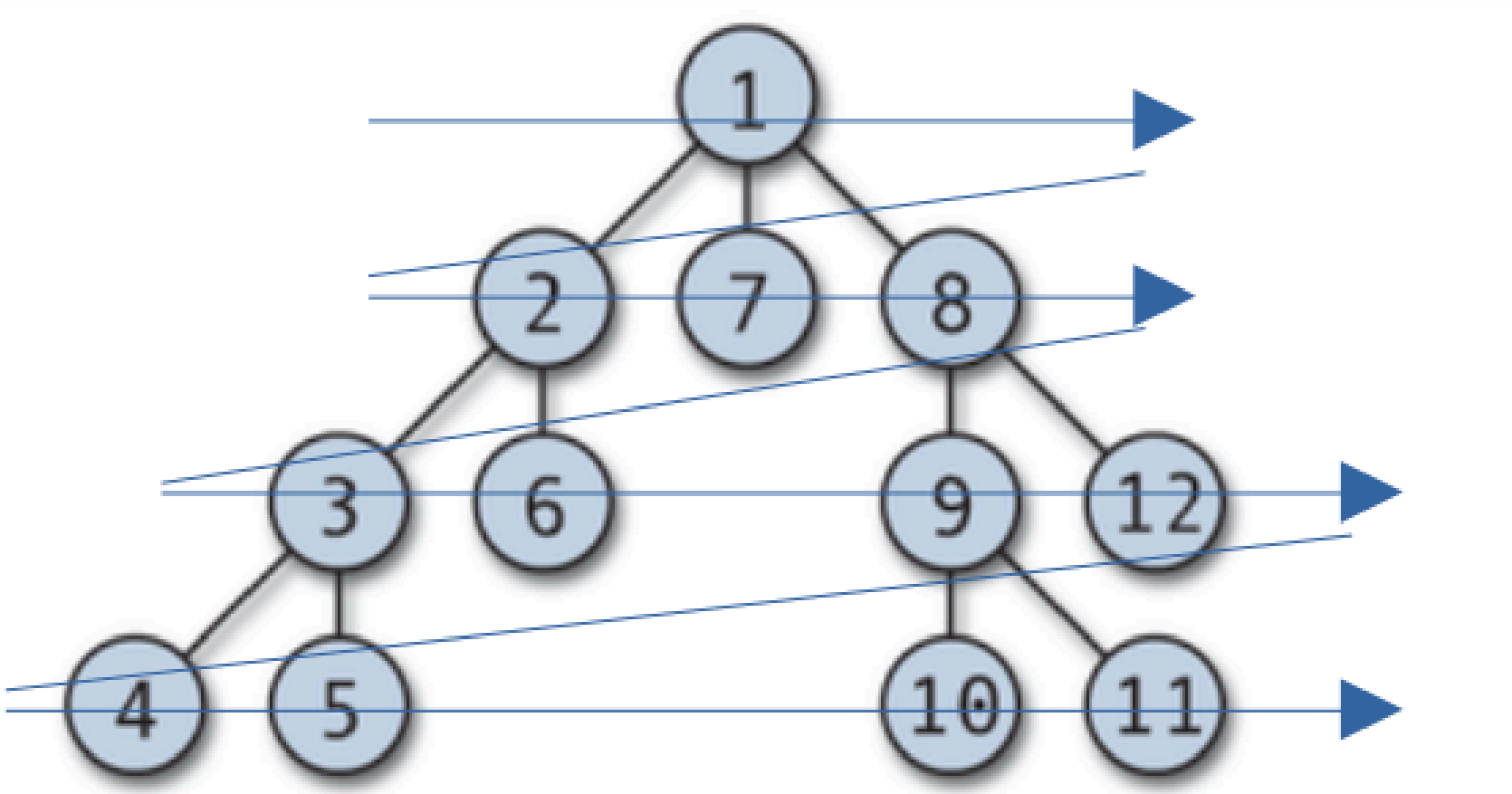
Grafos

```
class Grafo:
    def __init__(self):
        self.adjacencia = {}

    def adicionar_vertice(self, vertice):
        if vertice not in self.adjacencia:
            self.adjacencia[vertice] = []

    def adicionar_aresta(self, vertice1, vertice2):
        if vertice1 in self.adjacencia and vertice2 in self.adjacencia:
            self.adjacencia[vertice1].append(vertice2)
            self.adjacencia[vertice2].append(vertice1)
```

Busca em Largura



[1,2,7,8,3,6,9,12,4,5,10,11]

Busca em Largura

```
def busca_largura(self, vertice):
        # Inicia fila
        fila = deque()

        # Insere a raiz na fila
        if vertice:
            fila.append(vertice)


        while len(fila)>0:

            # Para cada vértice na fila
            for i in range(len(fila)):

                vertice_atual = fila.popleft()
                # Verifica se é o estado desejado
                print(vertice_atual.valor)
                # Insere ao fim da fila o vértice adjacente
                if vertice_atual.esquerda:
                    fila.append(vertice_atual.esquerda)
                # Insere ao fim da fila o vértice adjacente
                if vertice_atual.direita:
                    fila.append(vertice_atual.direita)
```

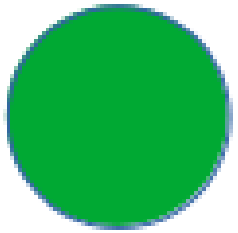
Busca em Largura

Realizando busca em largura no grafo Patos de Minas -> Belo Horizonte

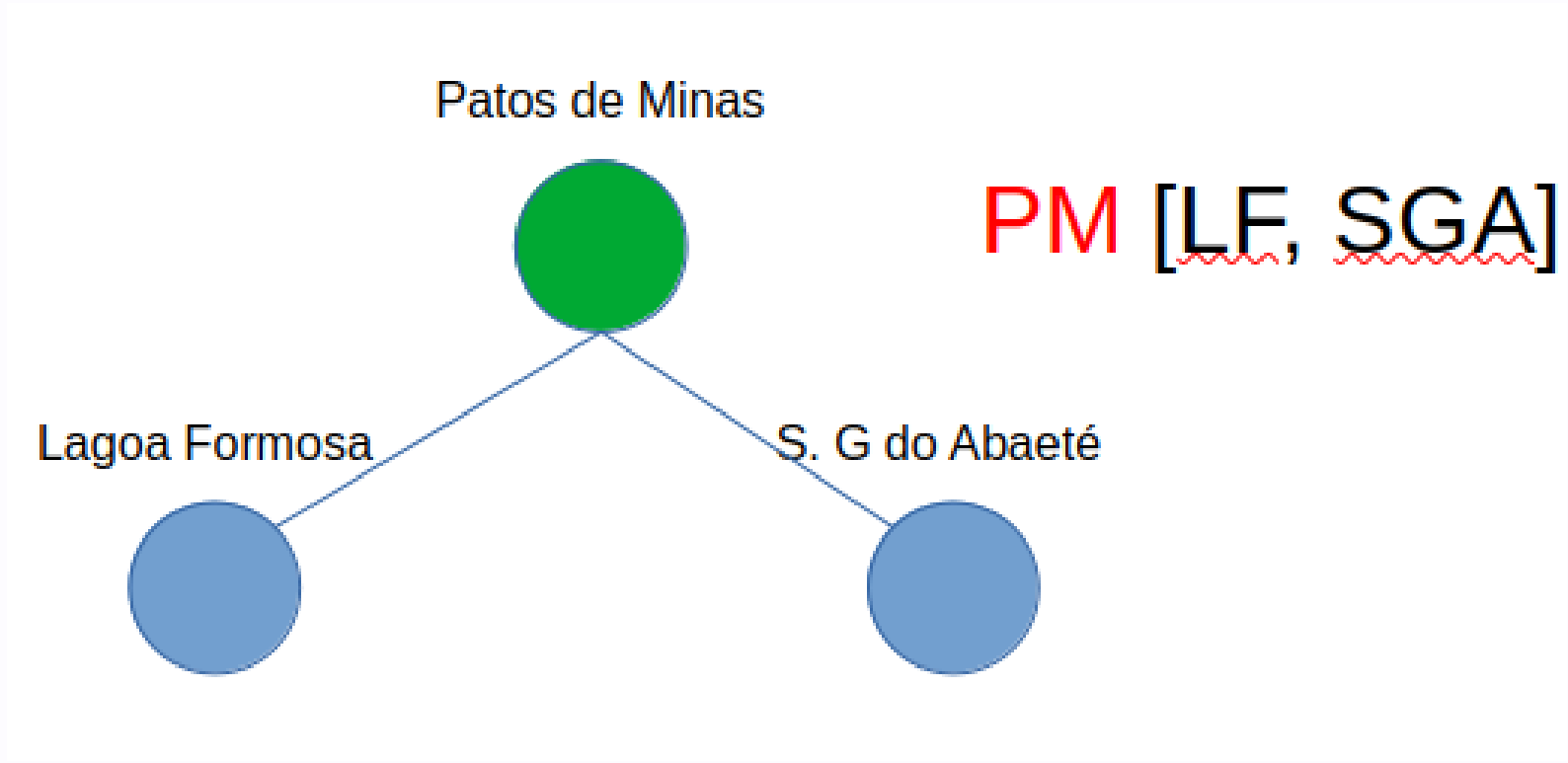
Busca em Largura

[PM]

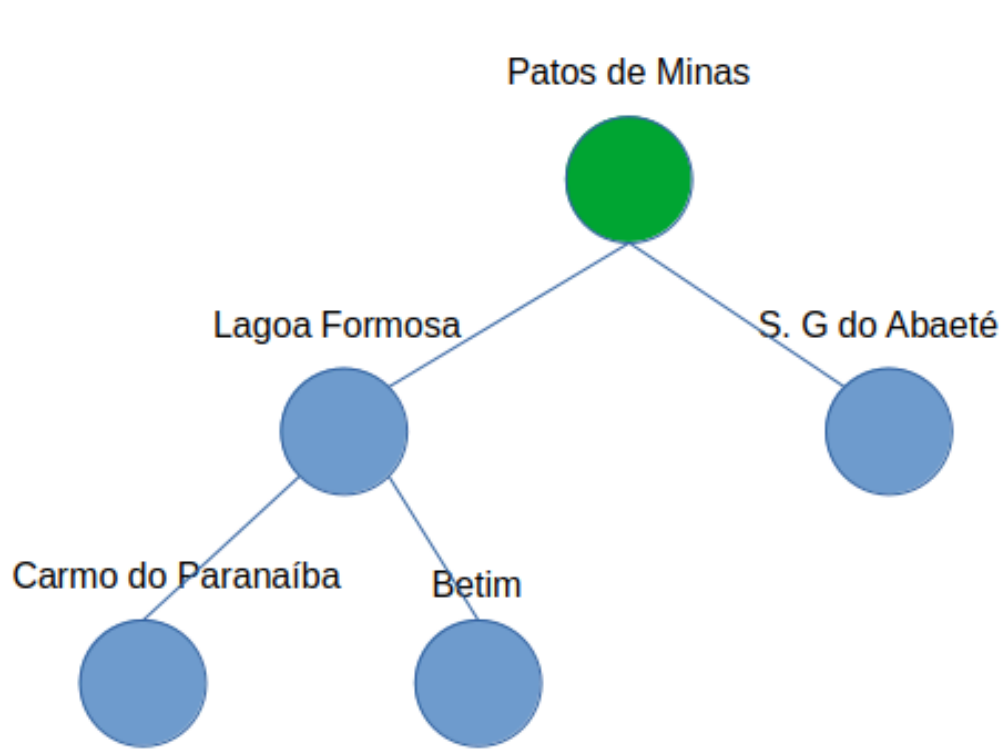
Patos de Minas



Busca em Largura

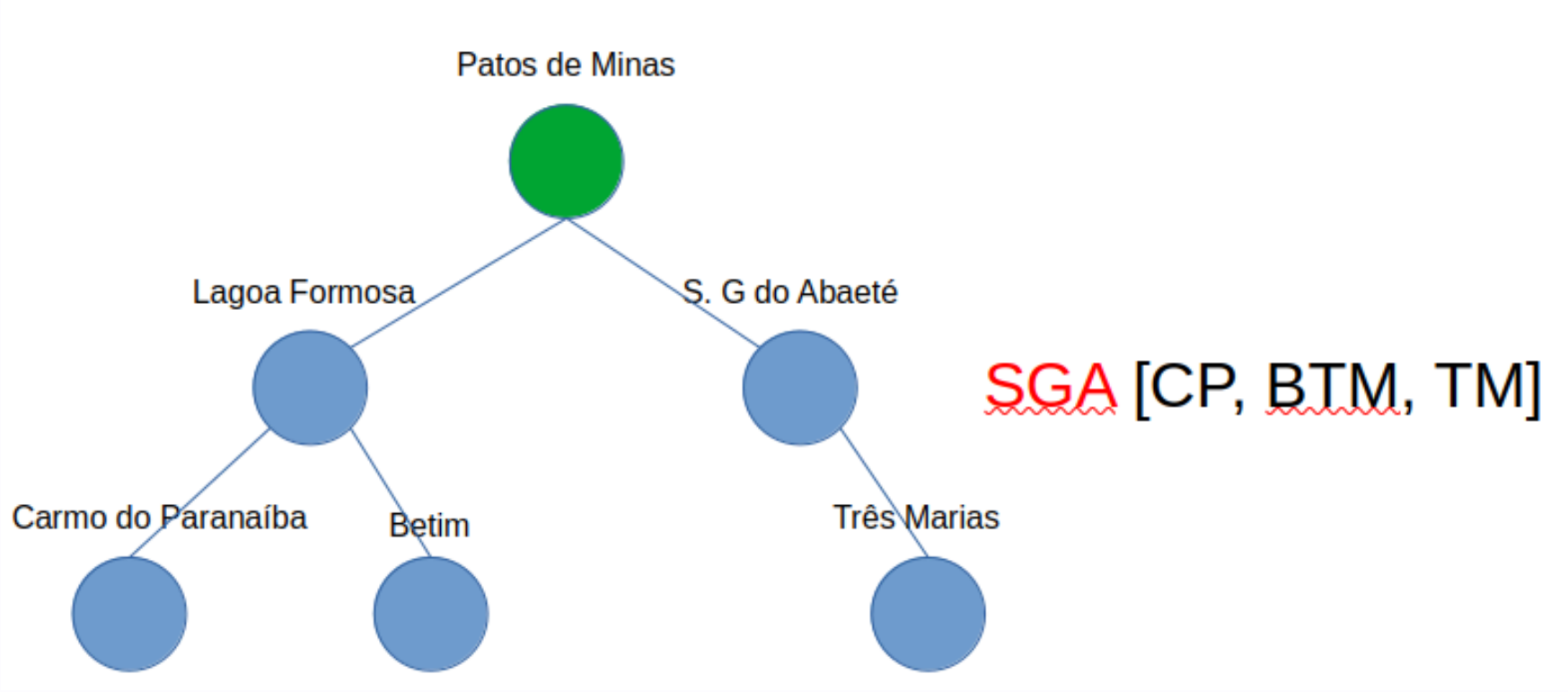


Busca em Largura

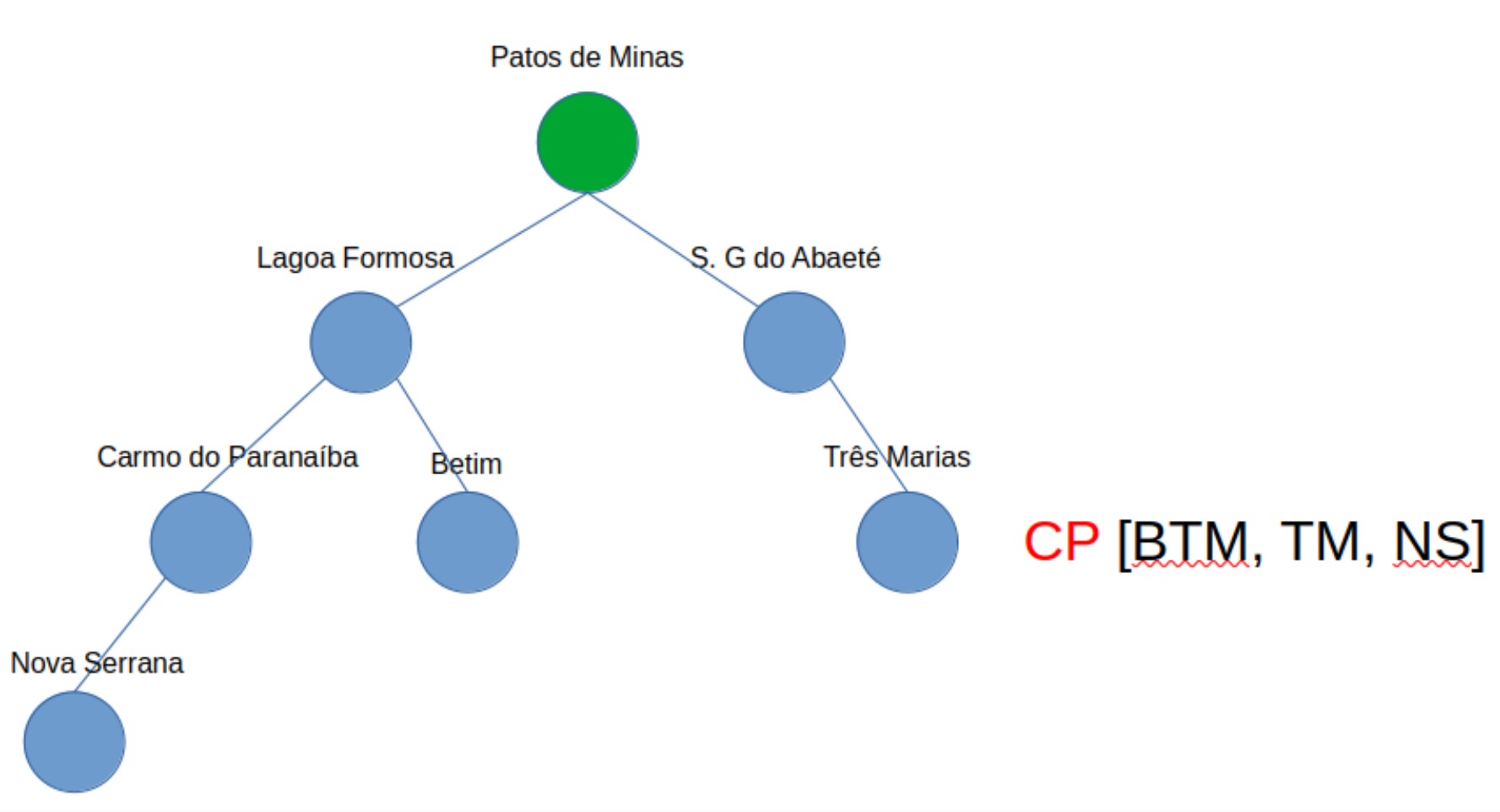


LF [SGA, CP, BTM]

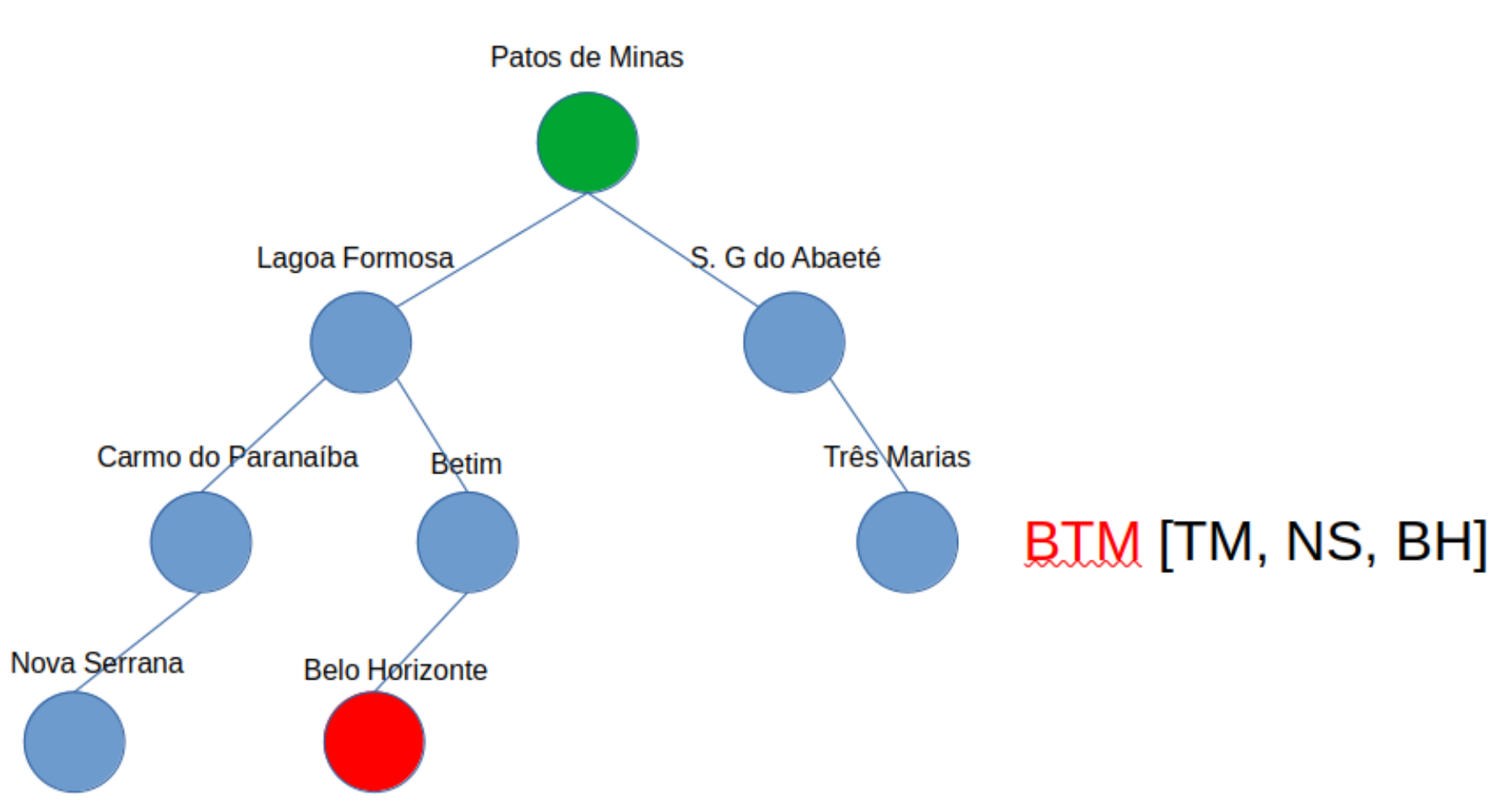
Busca em Largura



Busca em Largura

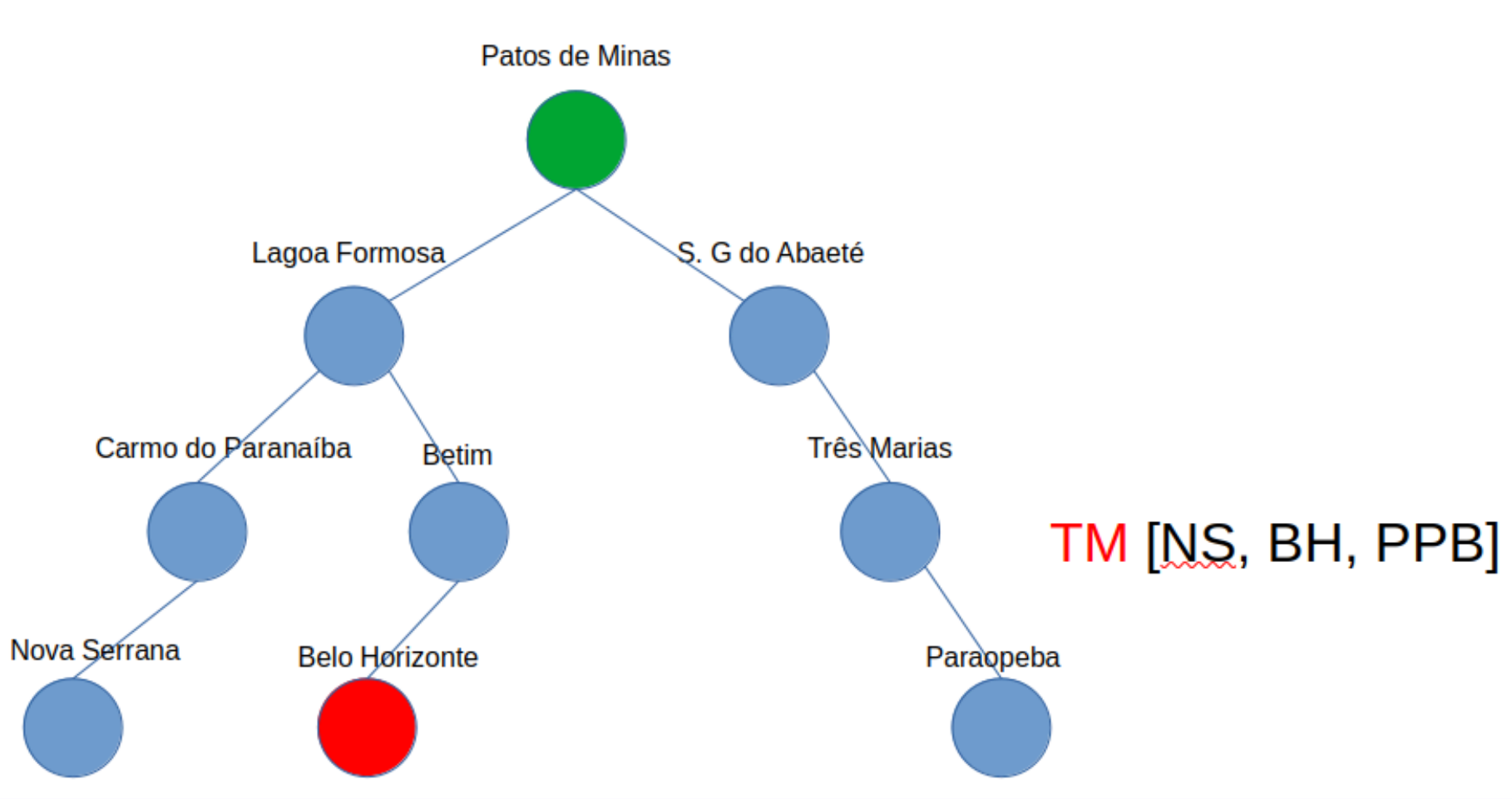


Busca em Largura

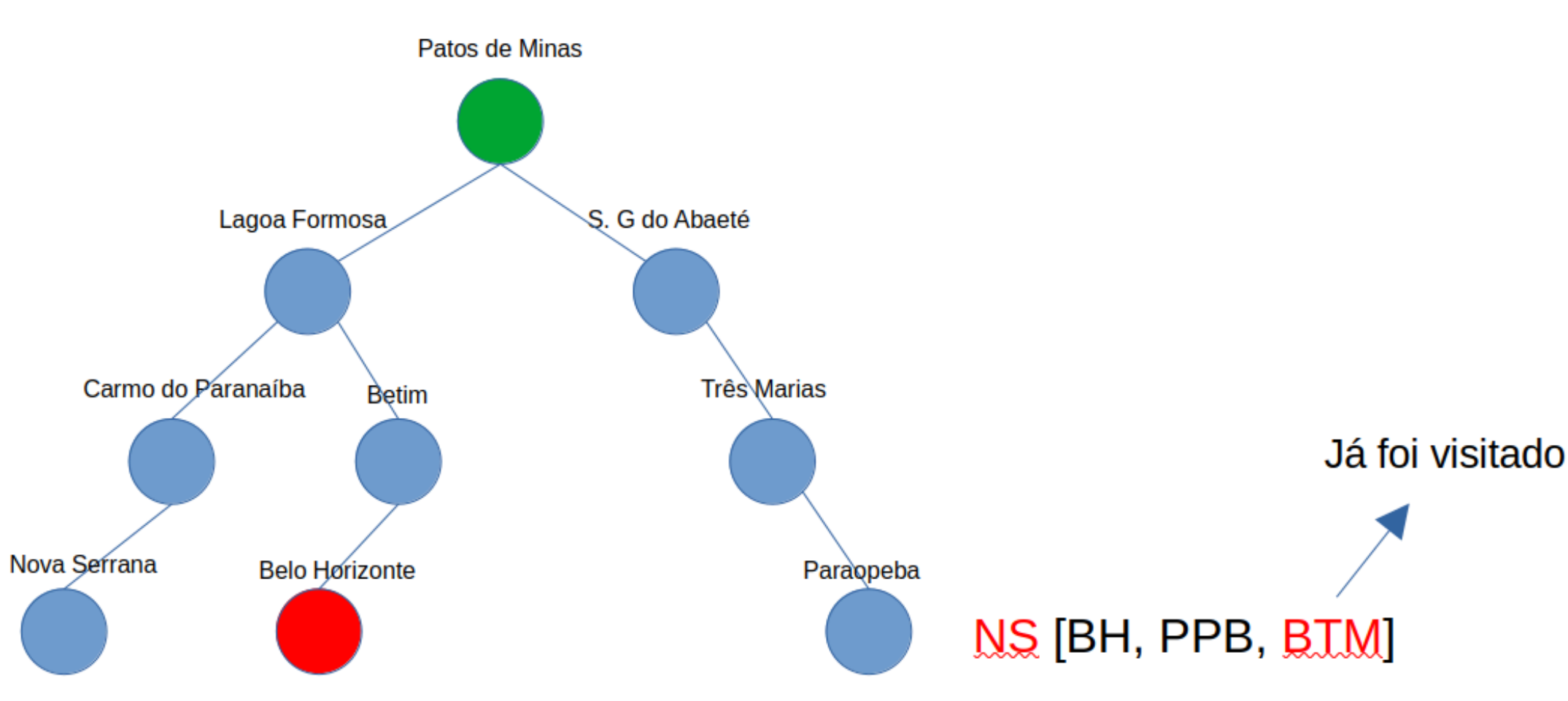


BTM [TM, NS, BH]

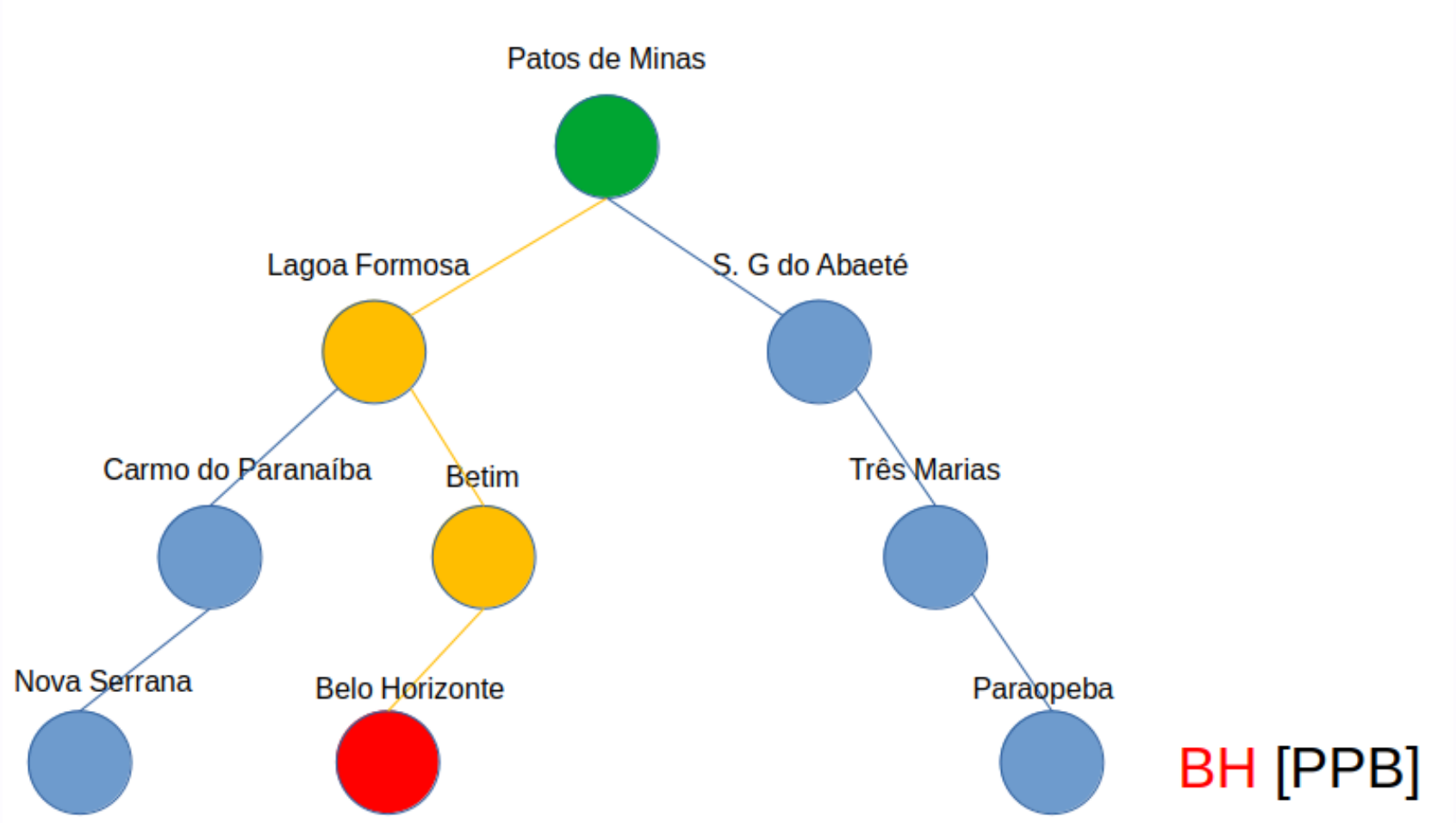
Busca em Largura



Busca em Largura



Busca em Largura



Busca em Largura

- A Busca em Largura é considerada **Completa**, porque sempre encontra a solução, uma vez que percorre todos os níveis
- Também é considerada **Ótima** já que busca por níveis, ou seja, olha sempre para as raizes das sub-árvores. A busca em profundidade, por outro lado, deve primeiramente percorrer todos os nós a esquerda e depois a direita de uma raiz

Busca em Largura

- **Fator de ramificação**: É a grandeza que determina o quanto a árvore cresce. Por exemplo, em uma árvore com fator 10, cada raiz possui 10 filhos. É esse fator que **determina a complexidade temporal** da busca em largura

Exercício IA-1

Busca

- Existem duas categorias de algoritmos de busca em grafos: **Informada** e **Não informada (cega)**
- **Não informada**: Realizamos a busca no grafo sem saber se estamos próximos ou não do destino. Ex: **Busca em Largura**
- **Informada**: A busca é realizada com a informação sobre o quanto estamos perto do destino. Ex: **Busca A*(estrela)**

Agente Baseado em Objetivo

- Problemas de **busca** podem ser resolvidos por agentes baseados em objetivos
- Qualquer algoritmo de busca que, de maneira automatizada, encontra uma solução para o problema, pode ser considerado um agente baseado em objetivo

Agente Baseado em Objetivo

- Encontrar a melhor rota. Qual delas é a menor?
- **Objetivo**: Menor caminho (Minimizar)
- **Custo**: Kilometros
- No problema da melhor rota temos diversas opções, mas qual é a melhor para chegar no objetivo?

A*(estrela)

- Algoritmo para encontrar o **menor caminho entre dois pontos**
- Considerado um algoritmo inteligente, já que encontra o melhor caminho sem intervenção
- Para cada nó que ele percorre, calcula qual será o menor caminho até o destino

A*(estrela)

- Como é calculado o menor caminho?
- O A* precisa calcular o custo de cada caminho possível
- Para isso usamos uma **função heurística**
- A função heurística **quantifica**, de maneira simplificada, o caminho de um nó qualquer até o destino
- Por esse motivo é conhecido como um algoritmo de **busca informada/best-first search**

O que é uma heurística?

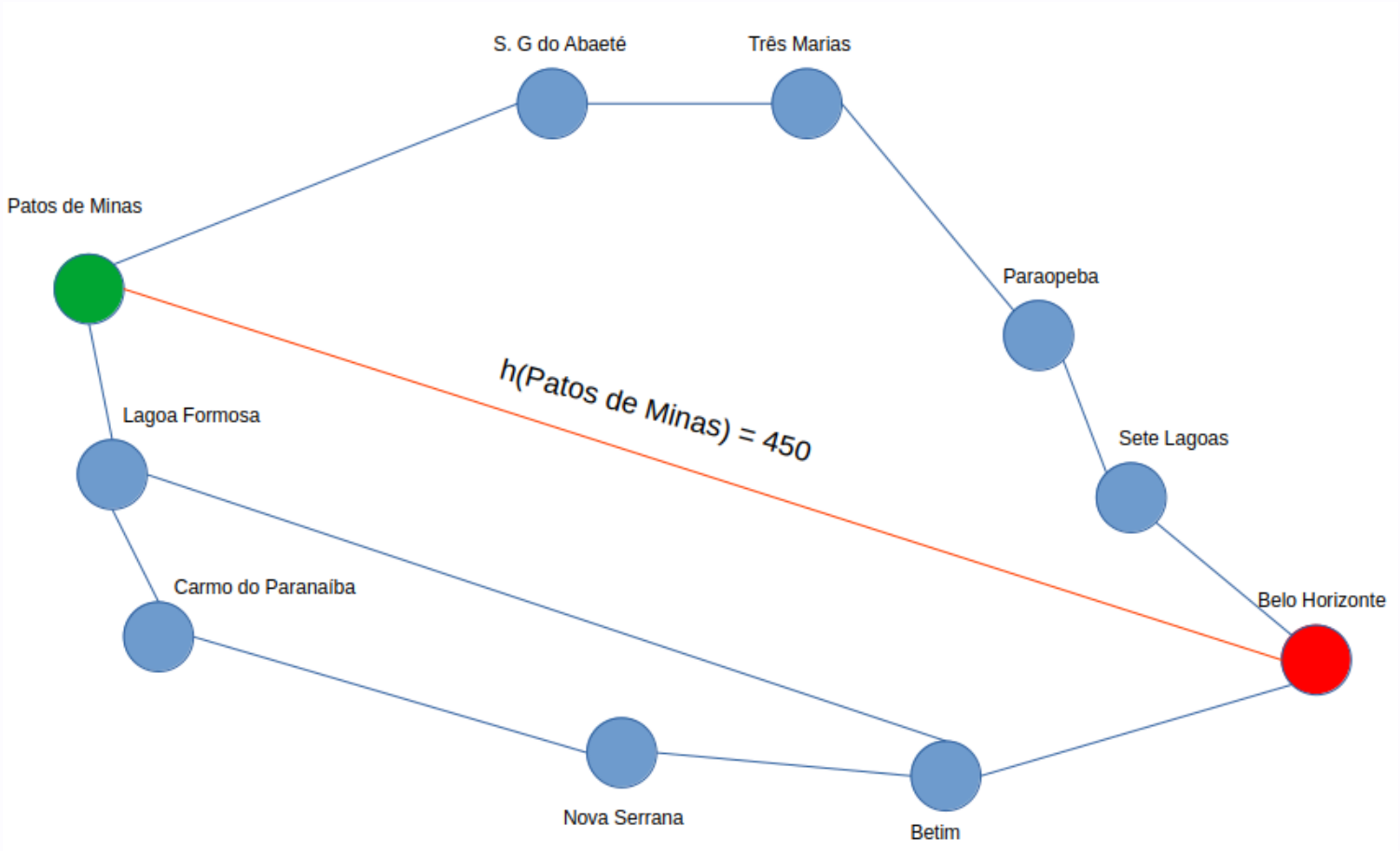
- Heurística são regras simples que nos permitem a chegar em um bom resultado.
- Heurística pode ser utilizada em diversos exemplos do nosso dia-a-dia
- Exemplo: Ao comprar uma maçã, selecione aquela que tem sua aprte inferior "achatada". Isso é uma regra simples, ou conhecimento adquirido ao longo dos anos
- Outro exemplo de heurística: A menor distância entre dois pontos é uma reta

A*(estrela) - Função de Custo

- $f(n) = g(n) + h(n)$,
- onde n é o próximo caminho (nó), $g(n)$ o custo acumulado até n e $h(n)$ a função heurística que calcula o menor custo de n até o destino.

A*(estrela)

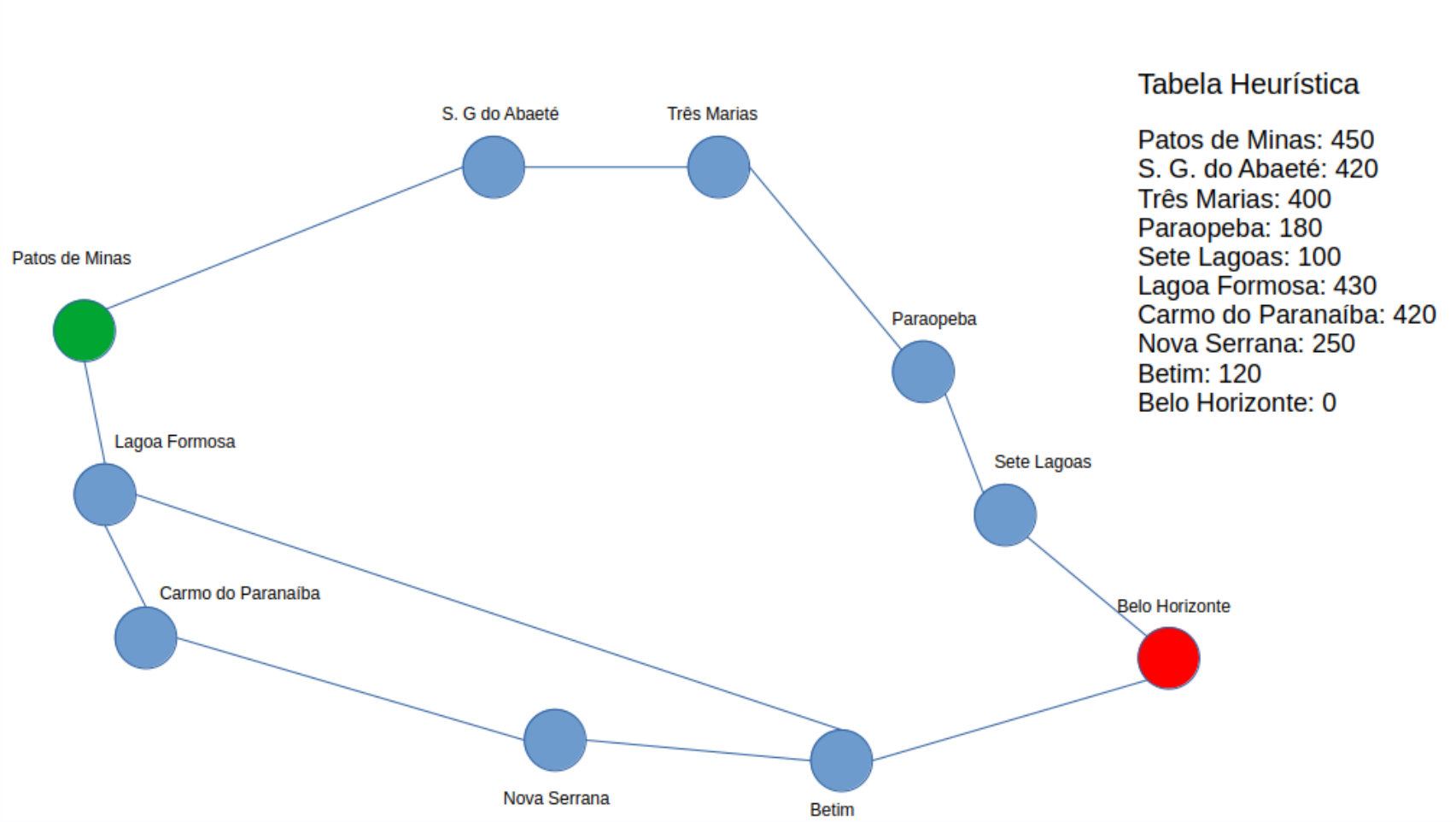
- Exemplo de heurística para buscar menor caminho: Distância em linha reta



A*(estrela)

- Esse cálculo é feito iterativamente, já que a cada nó que percorremos existem n caminhos possíveis
- Olhar apenas para o menor caminho a cada nó nem sempre vai nos retornar o menor caminho total
- Em alguns casos precisamos escolher o maior caminho em um nó para obter o menor caminho total

A*(estrela)


S. G do Abaeté
Três Marias
Patos de Minas
Paraopeba
Lagoa Formosa
Sete Lagoas
Carmo do Paranaíba
Belo Horizonte
Nova Serrana
Betim
Tabela Heurística
Patos de Minas: 450
S. G. do Abaeté: 420
Três Marias: 400
Paraopeba: 180
Sete Lagoas: 100
Lagoa Formosa: 430
Carmo do Paranaíba: 420
Nova Serrana: 250
Betim: 120
Belo Horizonte: 0

A*(estrela)

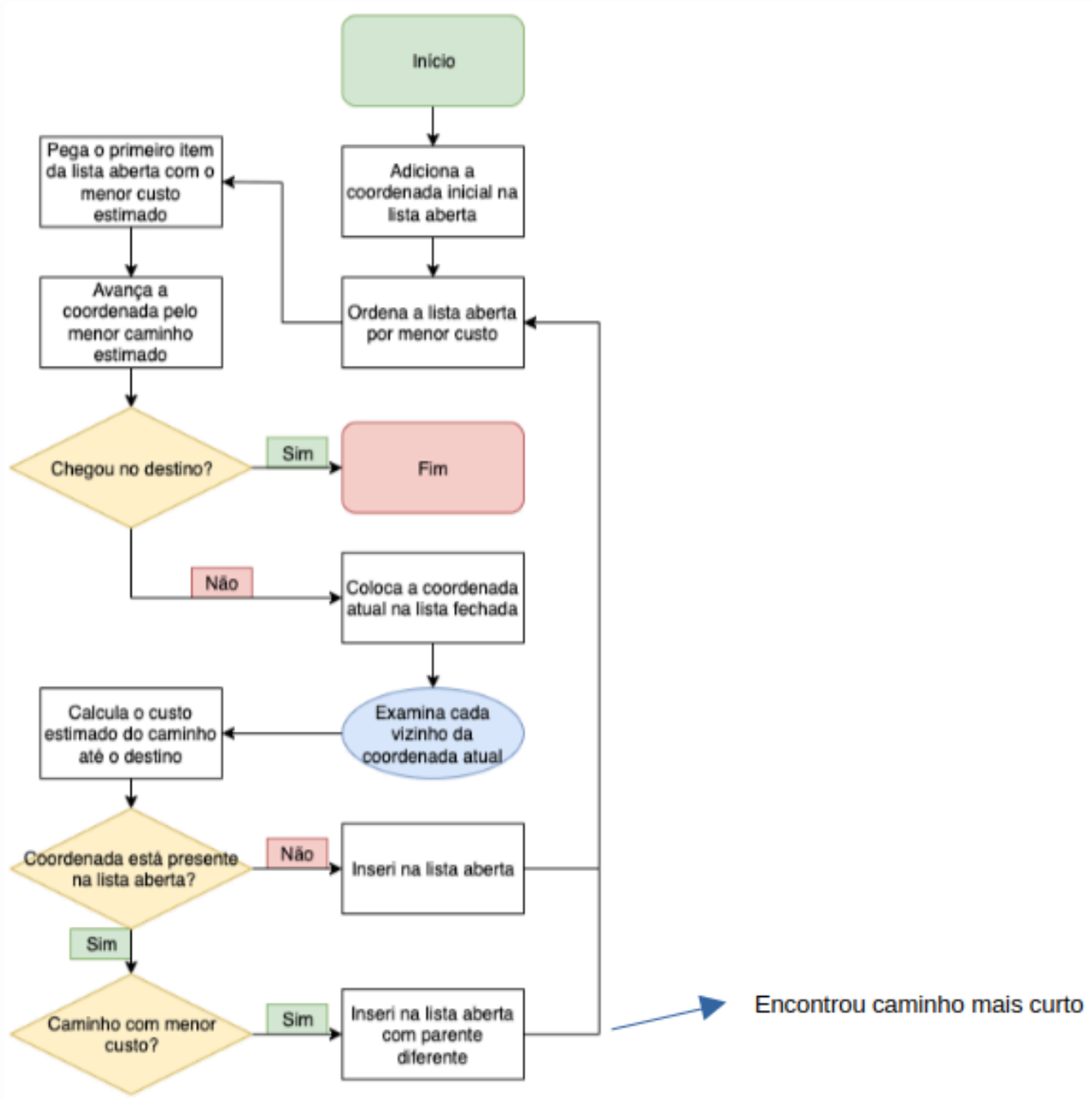
- **Lista Aberta**: Uma lista que contém os vértices que ainda não tiveram o menor custo calculado. O vértice adjacente (nó filho) com menor custo é adicionado a lista aberto com o vértice atual como "parente".

A*(estrela)

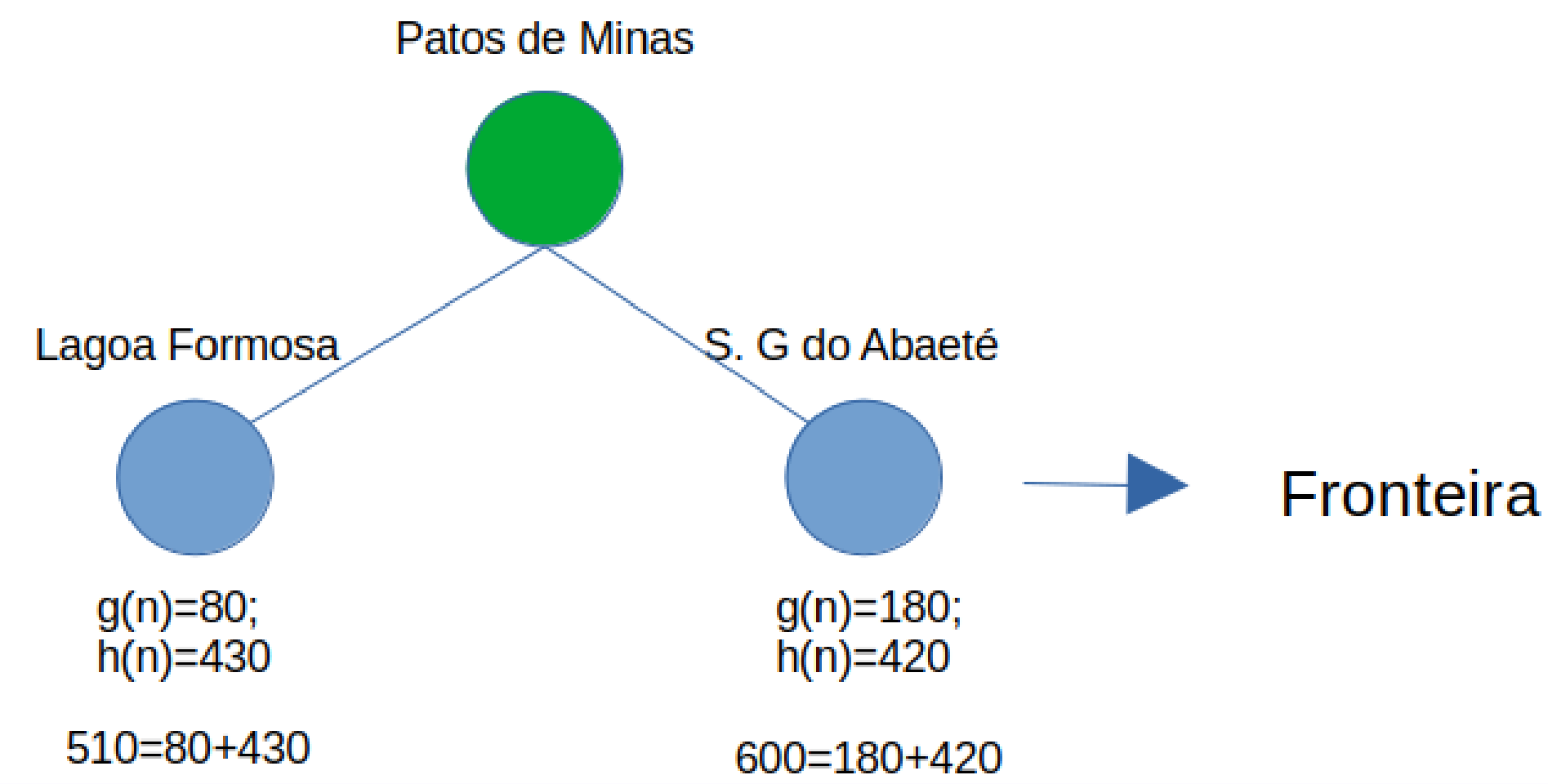
Lista aberta						
A direção x + 1 é o ponto mais próximo do destino. Por essa razão, ela é adicionada à lista aberta com o "Ponto atual" como parente.		**Direção y + 1**				
	Direção x - 1	**Ponto atual**	**Direção x + 1**			**Destino**
		Direção y - 1				

A*(estrela)

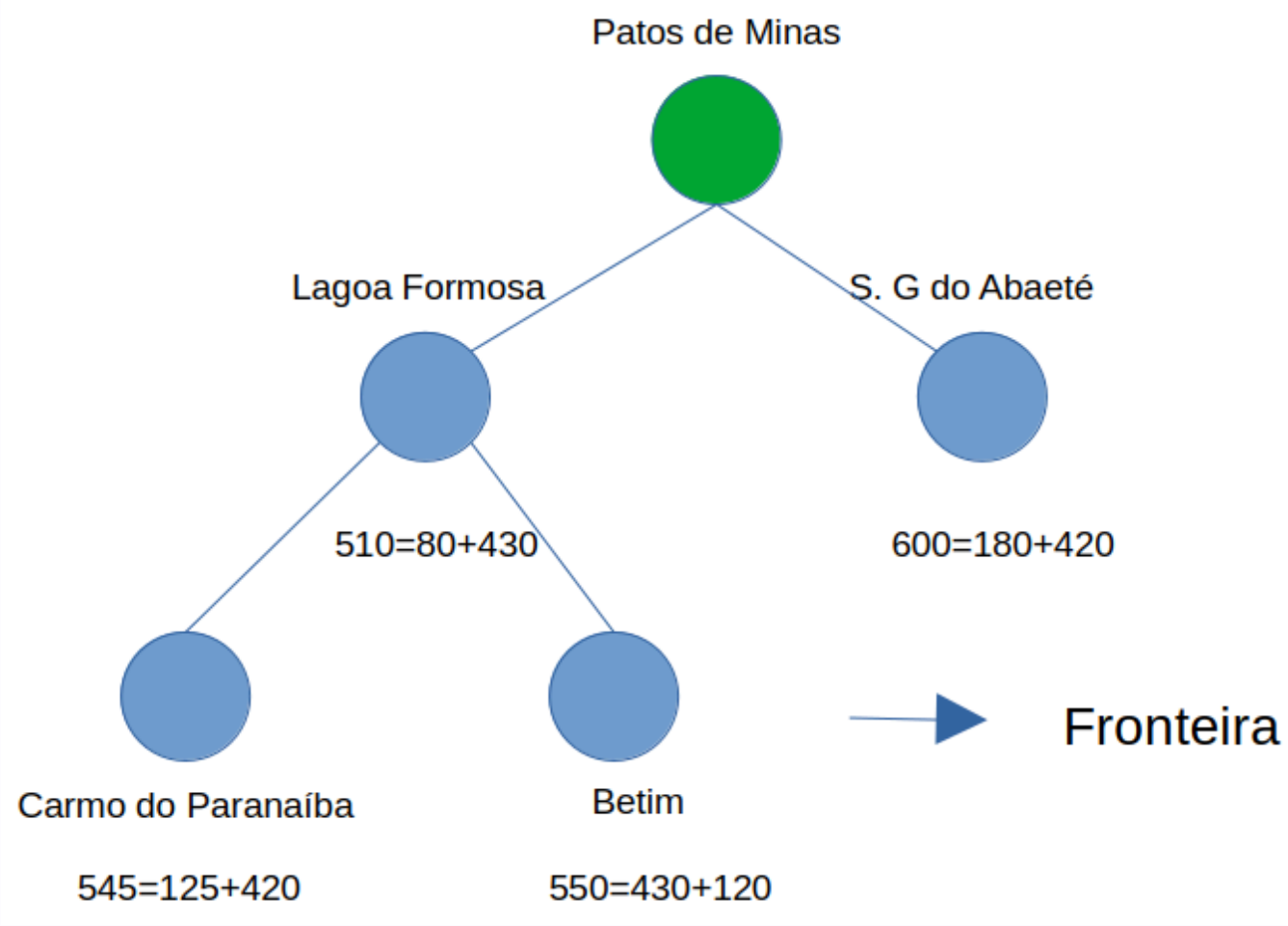
- **Lista Fechada**: Vértices onde todos os seus vizinhos já foram calculados.


Início
Adiciona a coordenada inicial na lista aberta
Ordena a lista aberta por menor custo
Pega o primeiro item da lista aberta com o menor custo estimado
Avança a coordenada pelo menor caminho estimado
Chegou no destino?
Sim
Fim
Não
Coloca a coordenada atual na lista fechada
Examina cada vizinho da coordenada atual
Calcula o custo estimado do caminho até o destino
Coordenada está presente na lista aberta?
Não
Inseri na lista aberta
Sim
Caminho com menor custo?
Sim
Inseri na lista aberta com parente diferente
Encontrou caminho mais curto

A*(estrela)


Patos de Minas
Lagoa Formosa
S. G do Abaeté
Fronteira
g(n)=80;
h(n)=430
510=80+430
g(n)=180;
h(n)=420
600=180+420

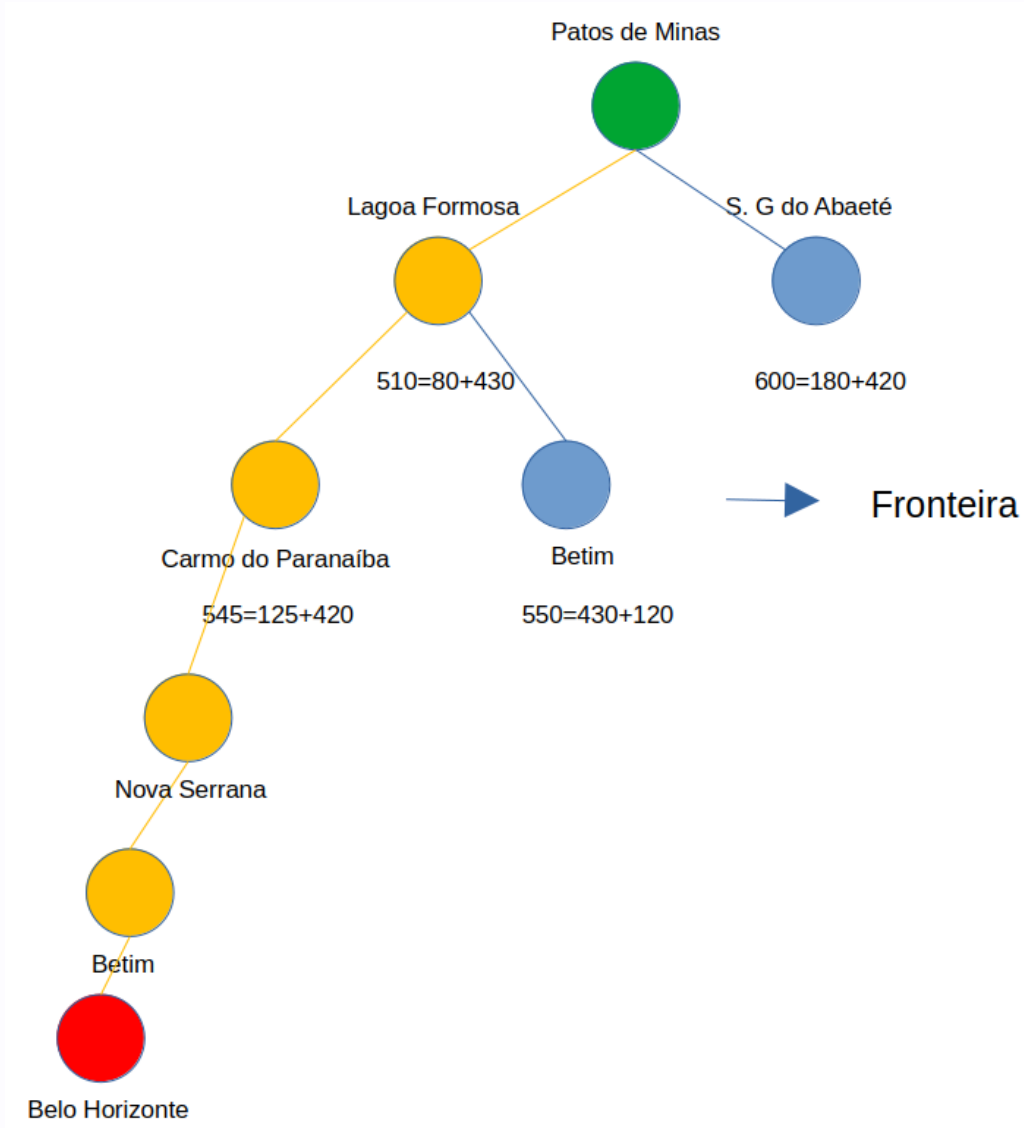
A*(estrela)


Patos de Minas
Lagoa Formosa
S. G do Abaeté
510=80+430
600=180+420
Carmo do Paranaíba
Betim
545=125+420
550=430+120
Fronteira

A*(estrela)

- Perceba que a cada iteração escolhemos o menor valor da nossa fronteira
- Isto é, dos nós analisados qual obteve menor valor na função $f(n)$
- Agora de Carmo do Paranaíba para Belo Horizonte temos apenas um caminho

A*(estrela)


Patos de Minas
Lagoa Formosa
S. G do Abaeté
510=80+430
600=180+420
Carmo do Paranaíba
Betim
Fronteira
545=125+420
550=430+120
Nova Serrana
Betim
Belo Horizonte

A*(estrela)

- **Atenção**: Precisamos sempre expandir os nós com menor valor de $f(n)$. Mesmo tendo chegado ao destino, verifique o nó com **menor custo**. Isso porque podemos ter **n** caminhos diferentes até o destino.

A*(estrela)

- A heurística da linha reta é **admissível** porque sabemos que a menor distância entre dois pontos é uma reta
- No entanto é importante lembrar que ela pode ser imprecisa

A*(estrela)

- O exemplo apresentado anteriormente é simplificado e para fins didáticos
- Problemas de busca podem ter complexidade muito superior, principalmente quando o número de possibilidades é grande, podendo ser **impossível encontrar a melhor solução**, por questões de memória ou tempo

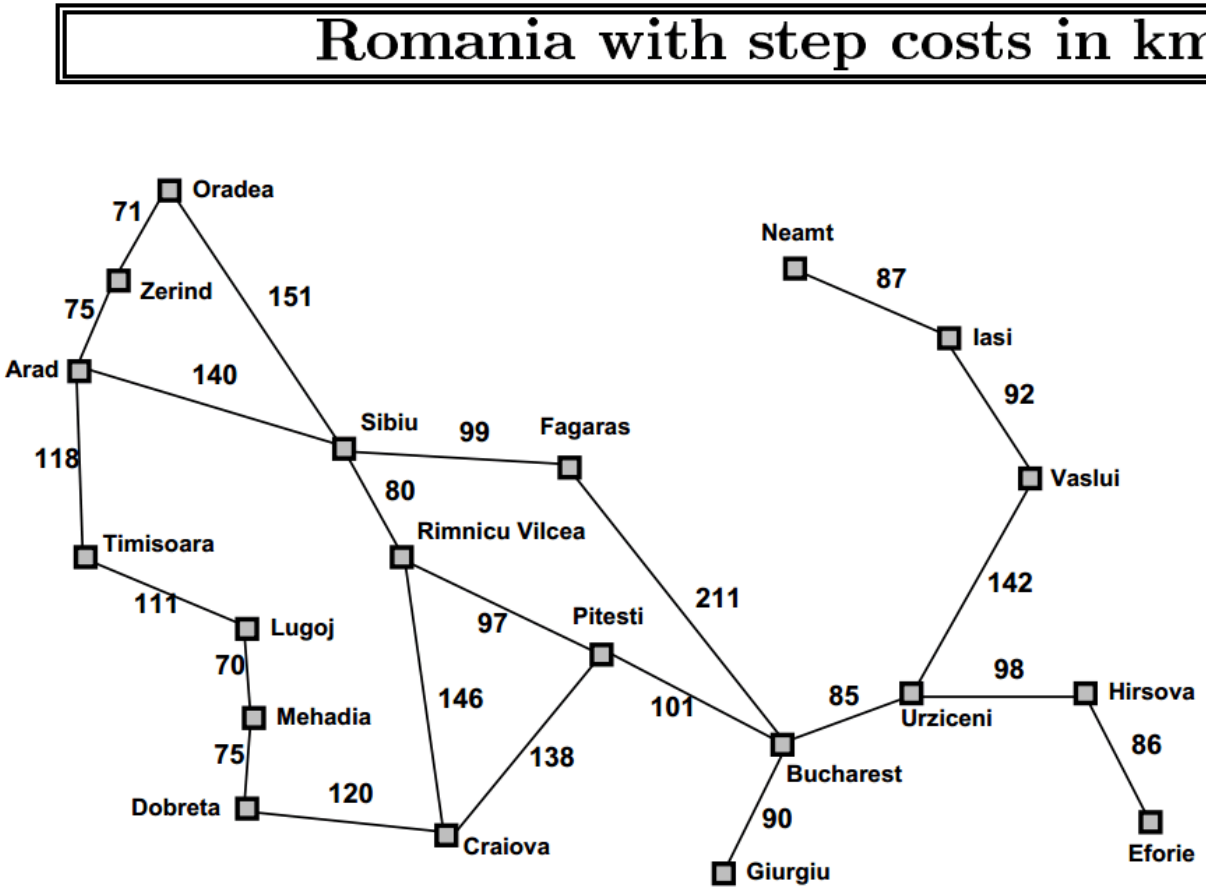
A*(estrela)

- O código do A* foi compartilhado no plano de aula juntamente com esses slides

A*(estrela)

- No mapa do próximo slide, faça uma execução manual (dry run) do A* para encontrar o menor caminho
- A origem é a cidade de **Arad** e o destino é **Bucharest**

A*(estrela)



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Referência

- Russell, S. J. 1., & Norvig, P. (1995). Artificial intelligence: a modern approach. Englewood Cliffs, N.J., Prentice Hall.
- https://dicionariotec.com/posts/algoritmo-a-a-estrela