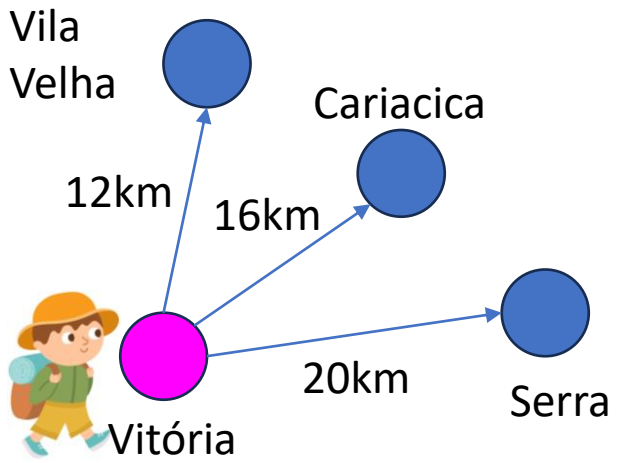
Estruturas de Dados

Especificação do Trabalho 1

Informações Gerais

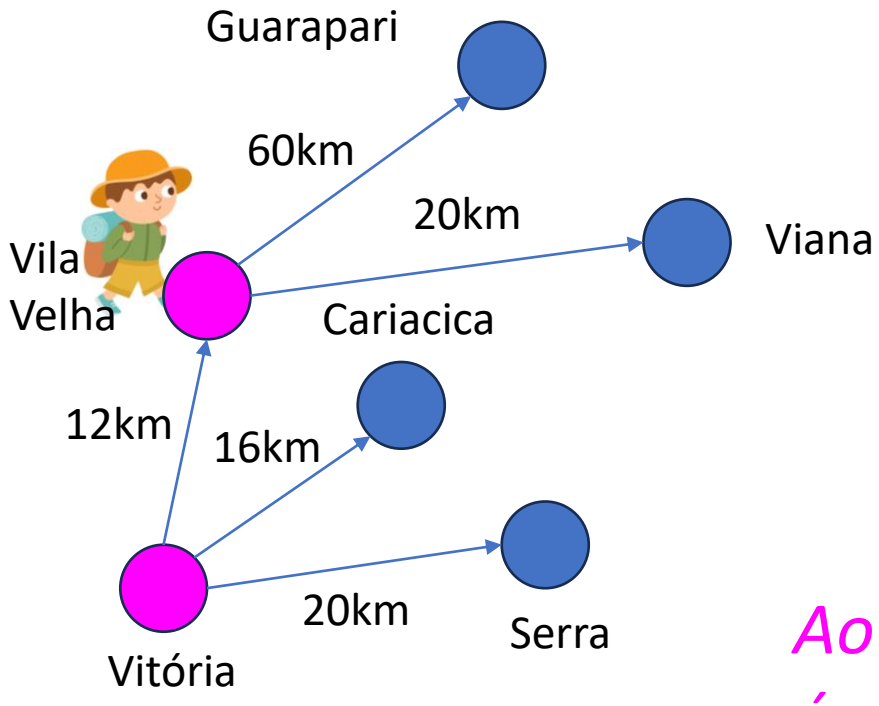
- **Data Limite de Entrega**: **04/08/2024 (23:59).**
- **Pontuação**: 10 pontos (**25% da nota** do semestre).
- **Formato de Entrega**: Os arquivos produzidos no trabalho devem ser compactados em formato .zip e submetidos na tarefa do ambiente virtual de aprendizagem (AVA).
- Os trabalhos devem ser desenvolvidos **individualmente**.
- **Importante**: Trabalhos entregues após a data limite sem justificativa com comprovação documental (atestado médico, etc.), ou que não estiverem de acordo com a especificado receberão nota zero.

Contexto: Leopoldo ama viajar e conhecer lugares, mas não possui um mapa nem um celular. Para tentar chegar em um local objetivo (por exemplo, Iconha), sempre que ele chega em uma nova cidade, ele pergunta para os moradores quais são as cidades vizinhas e a distância para estas cidades.



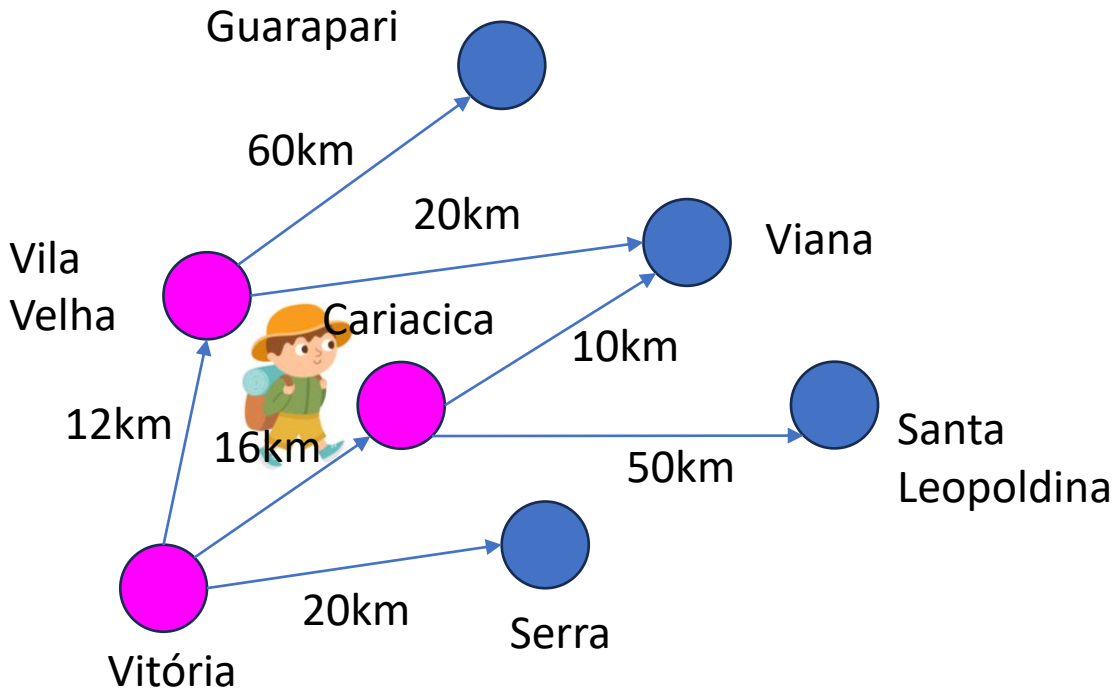
Ao chegar em vitória, Leopoldo descobre que as cidades vizinhas são Vila Velha a 12km, Cariacica a 16km e Serra a 20km.

Contexto: Leopoldo ama viajar e conhecer lugares, mas não possui um mapa nem um celular. Para tentar chegar em um local objetivo (por exemplo, Iconha), sempre que ele chega em uma nova cidade, ele pergunta para os moradores quais são as cidades vizinhas e a distância para estas cidades. Usando uma **estratégia que discutiremos em breve**, ele escolhe uma cidade dentre as possibilidades para visitar.



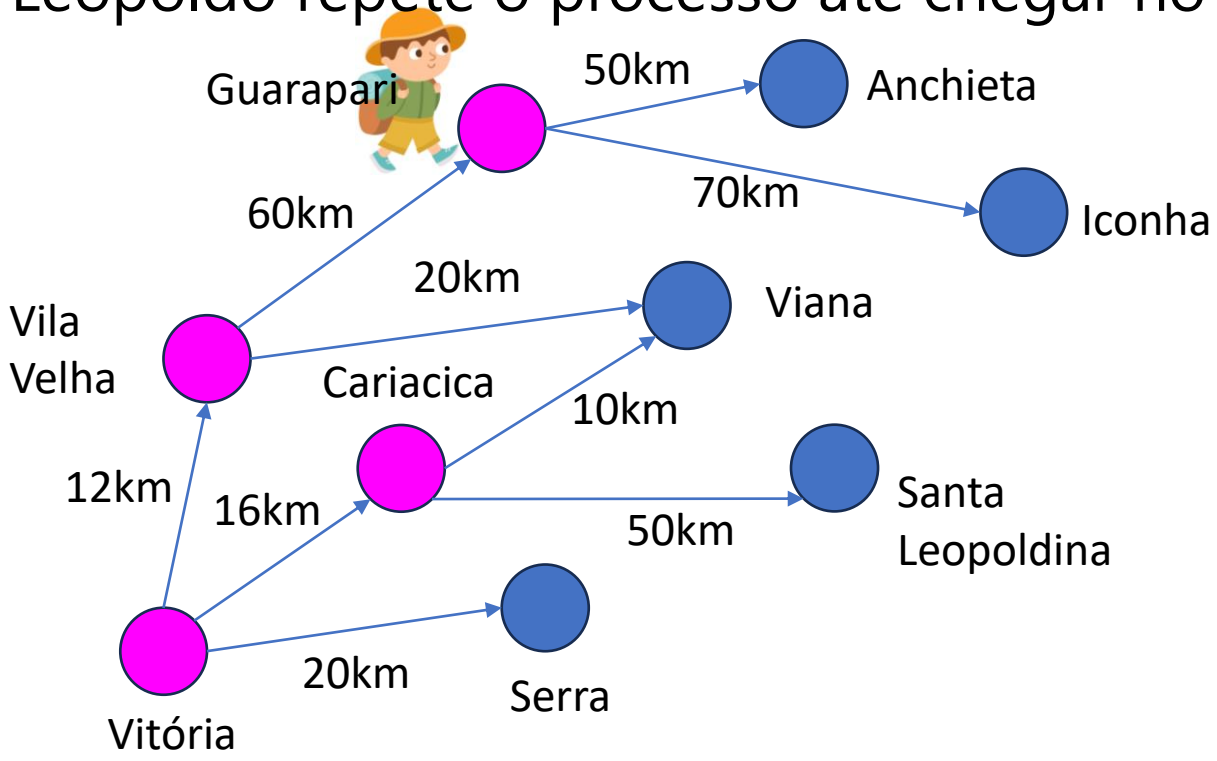
Ao visitar Vila Velha, Leopoldo descobre que é capaz de chegar em Guarapari e Viana.

Contexto: Leopoldo ama viajar e conhecer lugares, mas não possui um mapa nem um celular. Para tentar chegar em um local objetivo (por exemplo, Iconha), sempre que ele chega em uma nova cidade, ele pergunta para os moradores quais são as cidades vizinhas e a distância para estas cidades. Usando uma **estratégia que discutiremos em breve**, ele escolhe uma cidade dentre as possibilidades para visitar. Leopoldo repete o processo até chegar no objetivo.



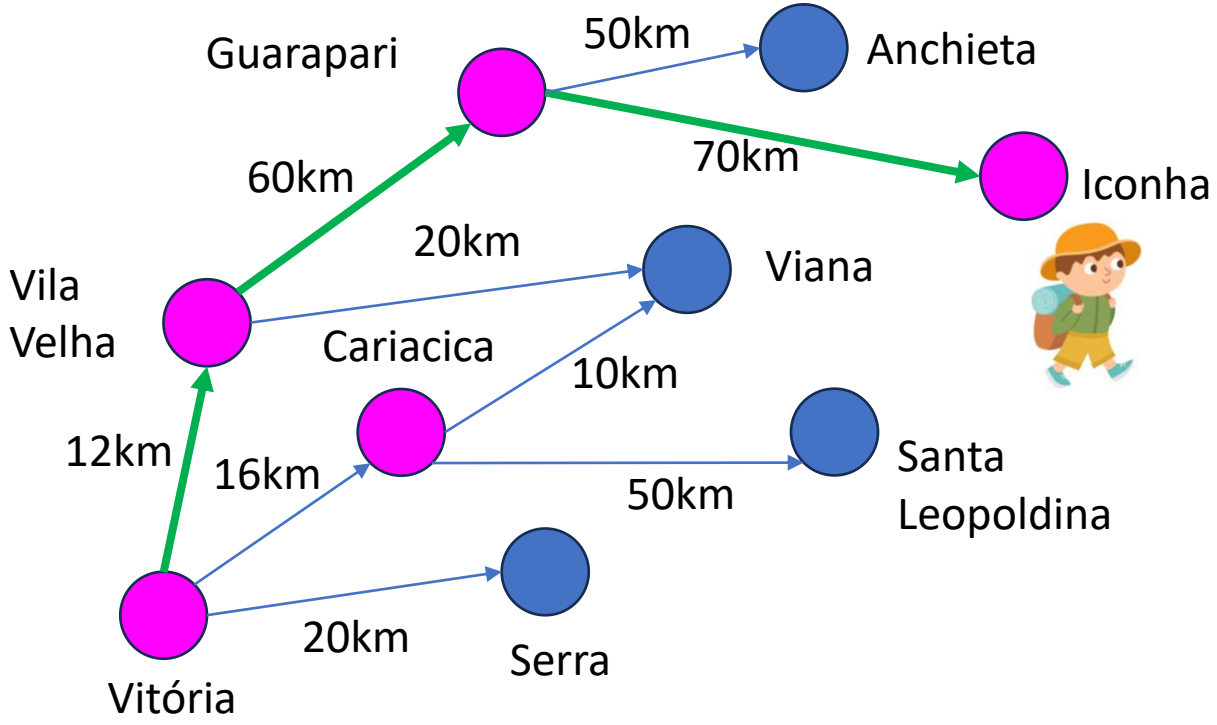
Ao visitar Cariacica, Leopoldo descobre outra rota para chegar em Viana e uma rota para Santa Leopoldina.

Contexto: Leopoldo ama viajar e conhecer lugares, mas não possui um mapa nem um celular. Para tentar chegar em um local objetivo (por exemplo, Iconha), sempre que ele chega em uma nova cidade, ele pergunta para os moradores quais são as cidades vizinhas e a distância para estas cidades. Usando uma **estratégia que discutiremos em breve**, ele escolhe uma cidade dentre as possibilidades para visitar. Leopoldo repete o processo até chegar no objetivo.



Ao visitar Guarapari, descobre caminhos para Anchieta e Iconha.

Contexto: Leopoldo ama viajar e conhecer lugares, mas não possui um mapa nem um celular. Para tentar chegar em um local objetivo (por exemplo, Iconha), sempre que ele chega em uma nova cidade, ele pergunta para os moradores quais são as cidades vizinhas e a distância para estas cidades. Usando uma **estratégia que discutiremos em breve**, ele escolhe uma cidade dentre as possibilidades para visitar. Leopoldo repete o processo até chegar no objetivo.

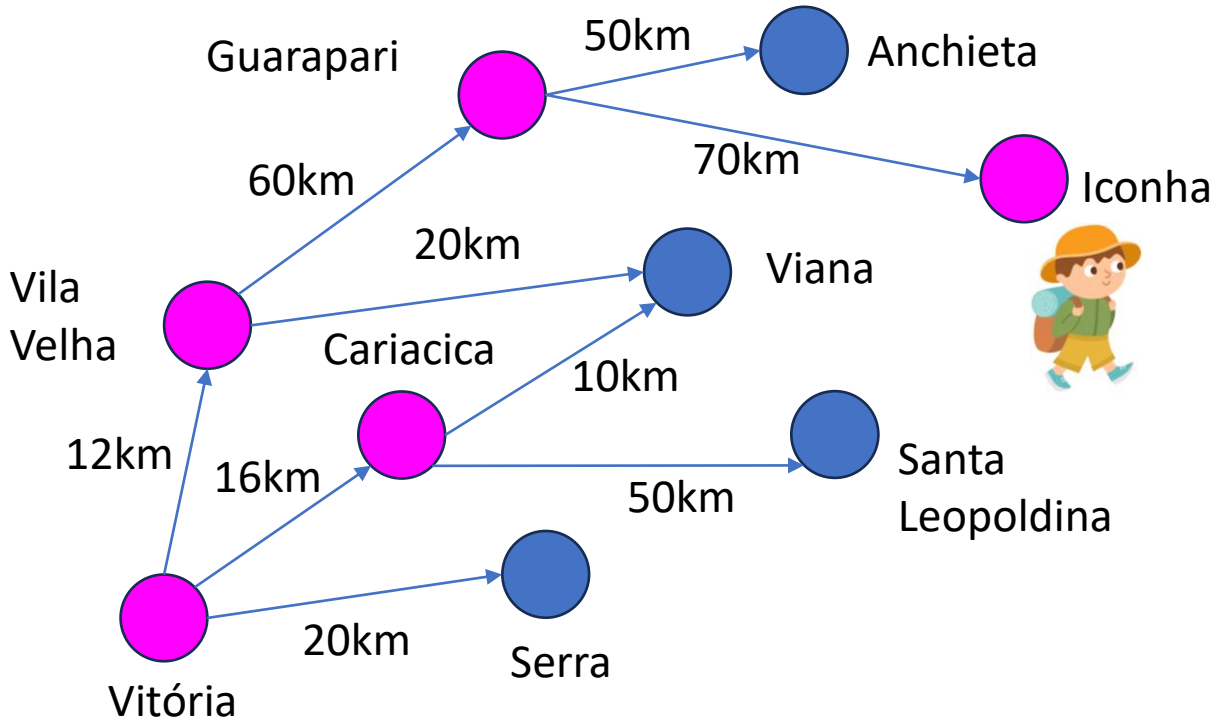


Note que embora Leopoldo tenha visitado Cariacica, esta cidade não faz parte do trajeto!

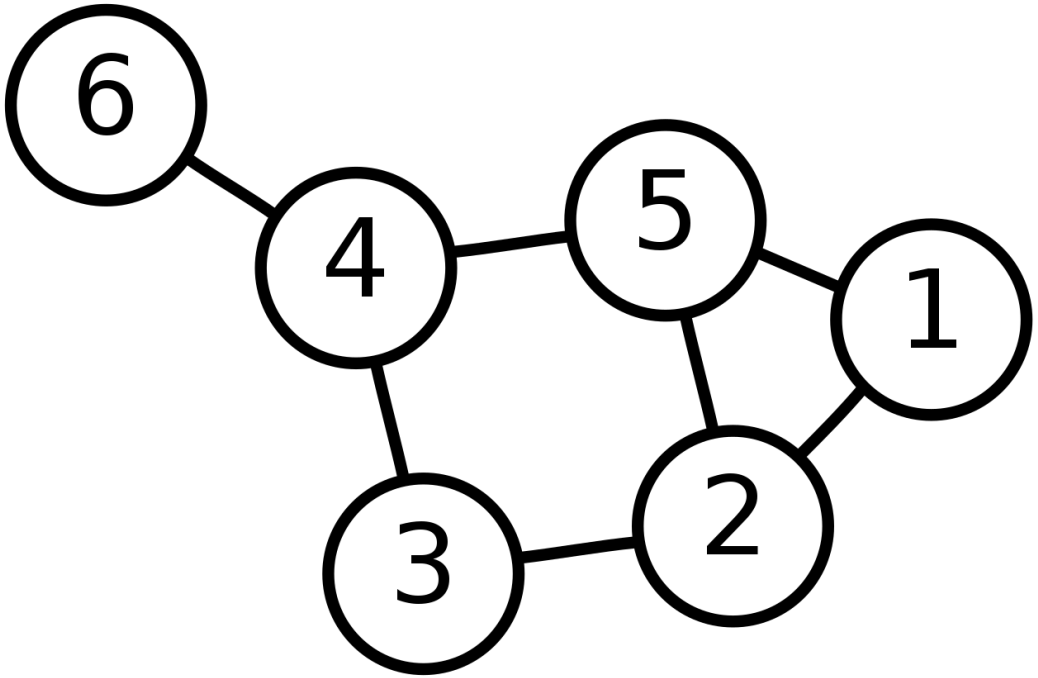
Ao visitar Iconha, Leopoldo descobre que chegou ao objetivo e que um caminho possível de Vitória para Iconha é Vitória -> Vila Velha -> Guarapari -> Iconha (com 142km).

Terminologia

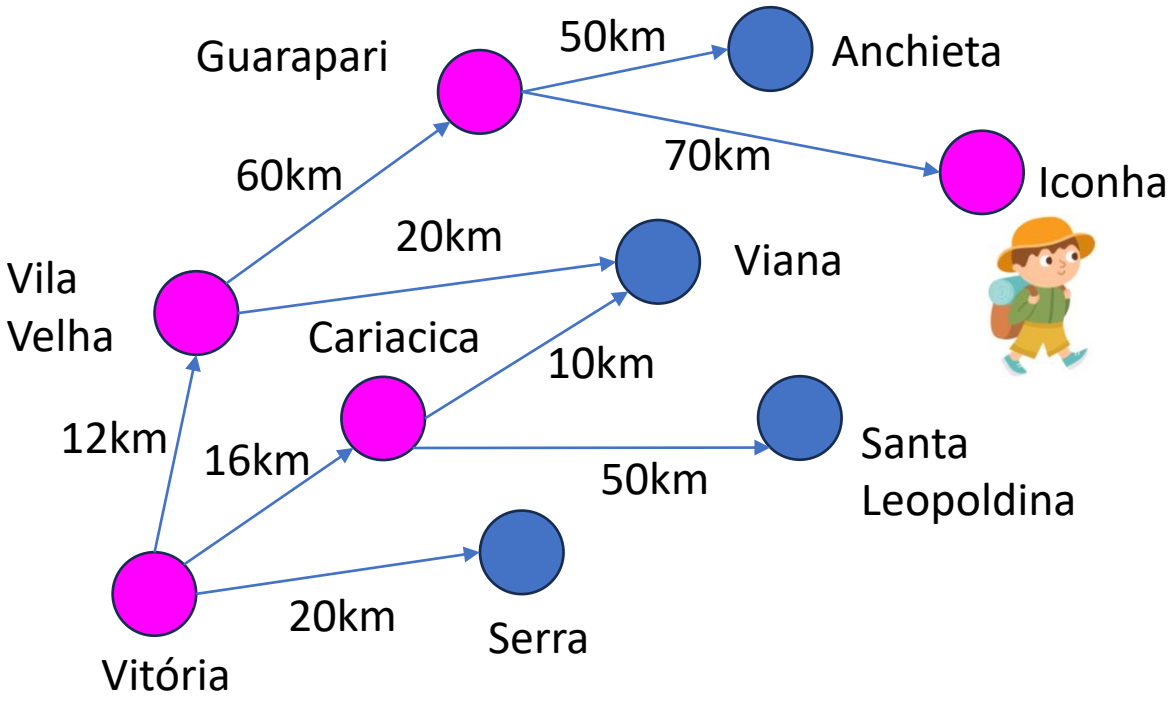
- Grafos
- Busca em grafos
- Fronteira



Grafos



- Um grafo é definido por um conjunto de vértices conectados por arestas direcionadas ou não.
- Dados podem ser "anexados" em nós e arestas (e.g., distância entre dois nós).
- A Teoria dos Grafos é uma subárea da Teoria da Computação que estuda como modelar e resolver problemas usando grafos.
- Grafos possuem aplicações nas mais diversas áreas da computação (e.g., Pesquisa Operacional, IA, redes sociais, etc.).



algoritmo de Busca em Grafos

fronteira <- [cidade inicial]

visitados <- []

enquanto existem cidades na fronteira:

C <- **selecione uma cidade*** da fronteira

se C é o alvo: interrompa o loop

se C não foi visitado:

para cada cidade vizinha D:

dist <- distância entre C e D

adicione na fronteira a cidade D junto com as informações de que podemos chegar em D via C usando um caminho de tamanho dist:

aresta=(cidade=D, origem=C, km=dist).

adicione C na lista de visitados.

se C é o alvo: **retorne o caminho**** e o seu tamanho.

caso contrário, acabaram as possibilidades. Informe que é impossível alcançar o alvo.

Algoritmos para Selecionar Cidades da Fronteira

- **Busca em Profundidade (*Depth-First Search* - DFS)**: A última cidade adicionada na fronteira é explorada primeiro.
- **Busca em Largura (*Breadth-First Search* - BFS)**: A cidade adicionada há mais tempo na fronteira é explorada primeiro.
- **Busca de custo uniforme (*Uniform-Cost Search*)**: A cidade com menor **custo para a origem** na fronteira é explorada primeiro.
- **Busca A* (lê-se "a estrela" ou "*a star*" em inglês)**: Similar à Uniform-Cost Search, mas uma função de custo especial (veremos em breve) é utilizada.

Regras

- Os algoritmos de busca em largura e profundidade **não** poderão ser recursivos. Para implementar os algoritmos, serão utilizadas filas e pilhas. A estrutura de dados **array circular** deve ser usada para implementar estas estruturas.
- A estrutura de dados **heap** deverá ser utilizada na implementação da busca de custo uniforme e da busca A*.
- Todas as estruturas deverão ser opacas e genéricas.
- Não é permitido usar arrays alocados dinamicamente diretamente. Deve ser usado o tipo Vector.

O usuário do programa deverá **digitar como entrada** o nome de um arquivo contendo, nesta ordem:

- O algoritmo a ser utilizado (BFS, DFS, UCS ou A*).
- Dois inteiros representando os índices das cidades de origem e destino na lista de **m** cidades (ver a seguir).
- Um número inteiro **m** indicando o número de cidades.

m vezes (uma para cada cidade):

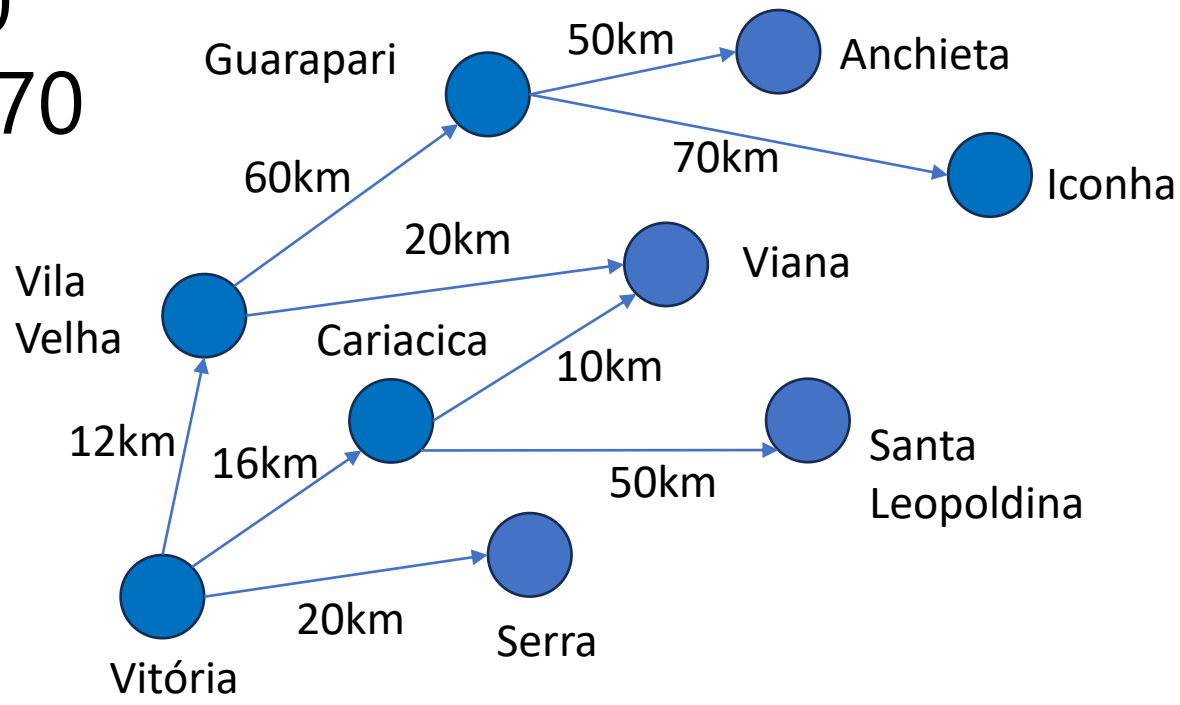
- O nome da cidade (uma palavra, sem espaços ou acentos)
- Um par de floats (x, y) indicando a posição GPS da cidade.
- Um número **n** indicando o número de cidades vizinhas, seguido de **n** inteiros representando os índices das vizinhas na sequência de **m** cidades e a distância da rodovia que conecta das acidades.

UCS
0 5
9
0 Vitoria 0 0 3 1 16 2 20 3 12
1 Cariacica -16 0 3 0 16 7 10 8 50
2 Serra 16 16 1 0 20
3 VilaVelha 0 15 3 0 12 7 20 4 60
4 Guarapari 10 25 3 3 60 5 50 6 70
5 Anchieta 25 40 1 4 50
6 Iconha 20 55 1 4 70
...

Vitoria está em (0, 0)

Vitoria tem 3 cidades vizinhas:

- 1 (Cariacica) a 16 km
- 2 (Serra) a 20km
- 3 (VilaVelha) a 12km

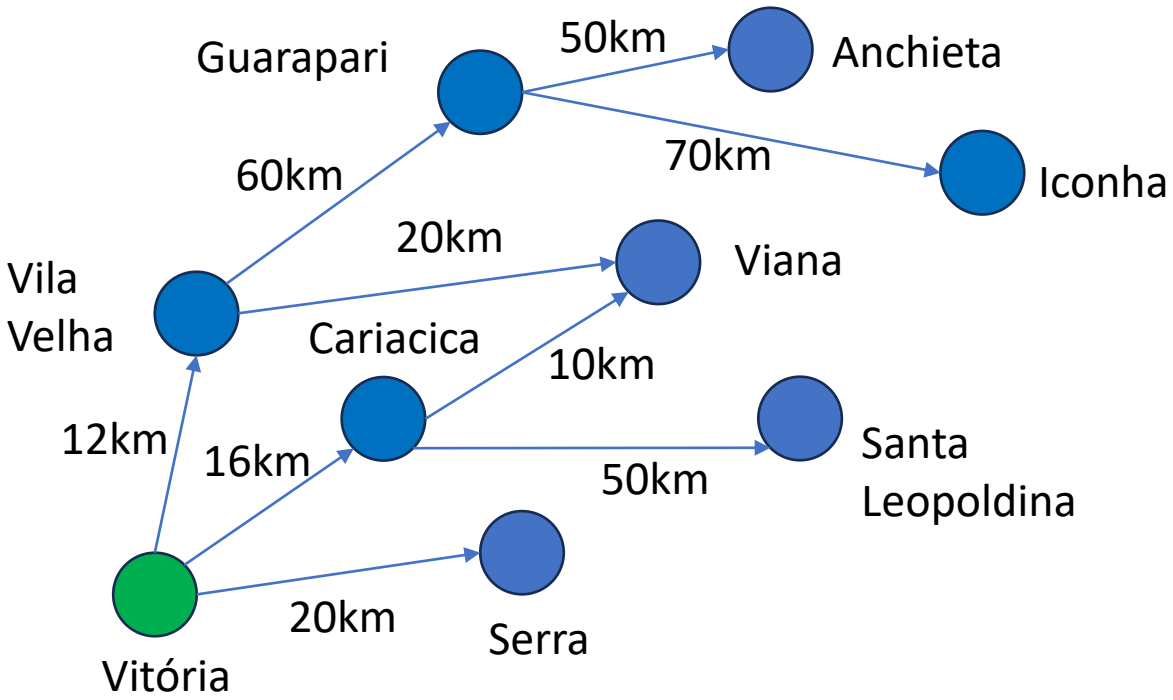


Saída do Programa

- A sequência de cidades que levam o agente da origem para o destino ou uma mensagem dizendo que é impossível chegar na cidade alvo.
  - Será impossível no caso de grafos desconectados.
- O custo total do caminho com 2 casas após a vírgula.
- O número de cidades visitadas.

DFS

A última cidade adicionada na fronteira é explorada primeiro.

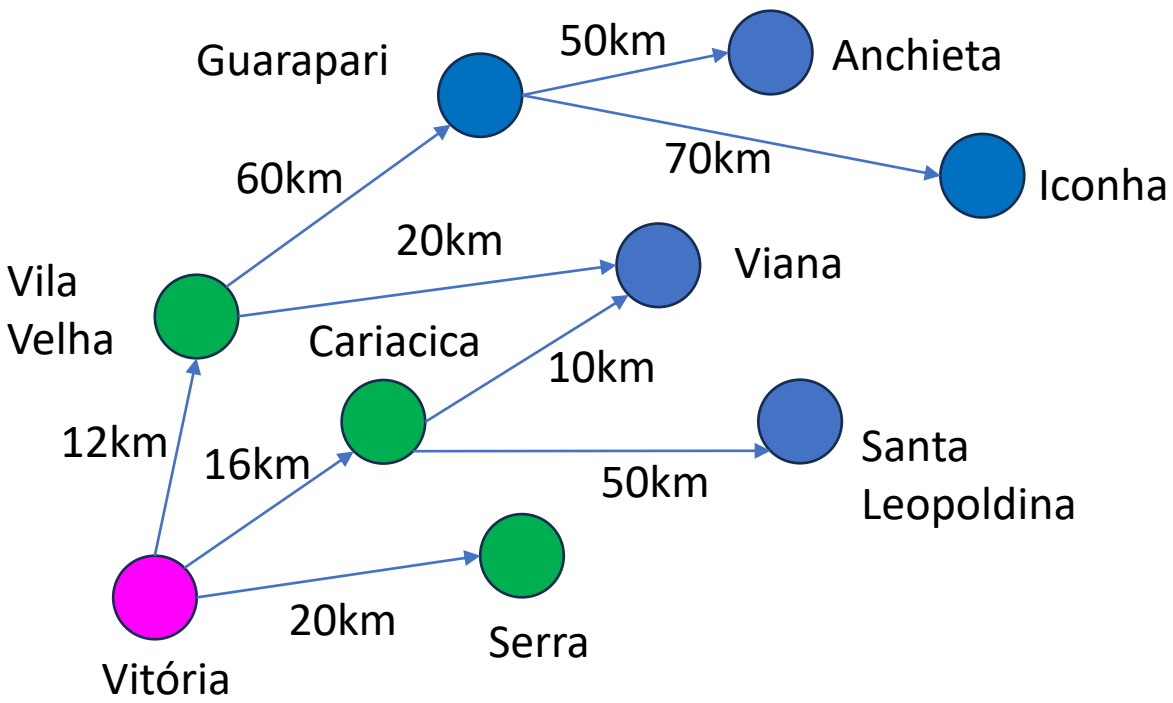


Fronteira = [Vitoria]

Visitados = []

DFS

A última cidade adicionada na fronteira é explorada primeiro.

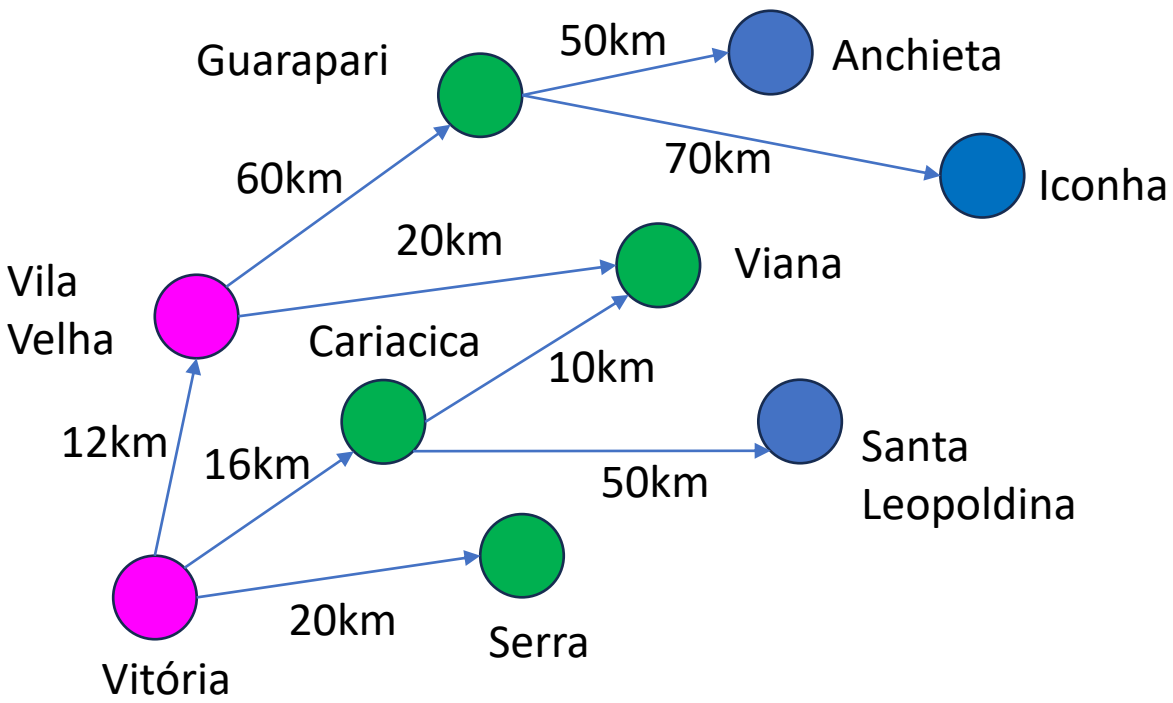


Fronteira = [Cariacica, Serra, VilaVelha]

Visitados = [Vitoria]

DFS

A última cidade adicionada na fronteira é explorada primeiro.

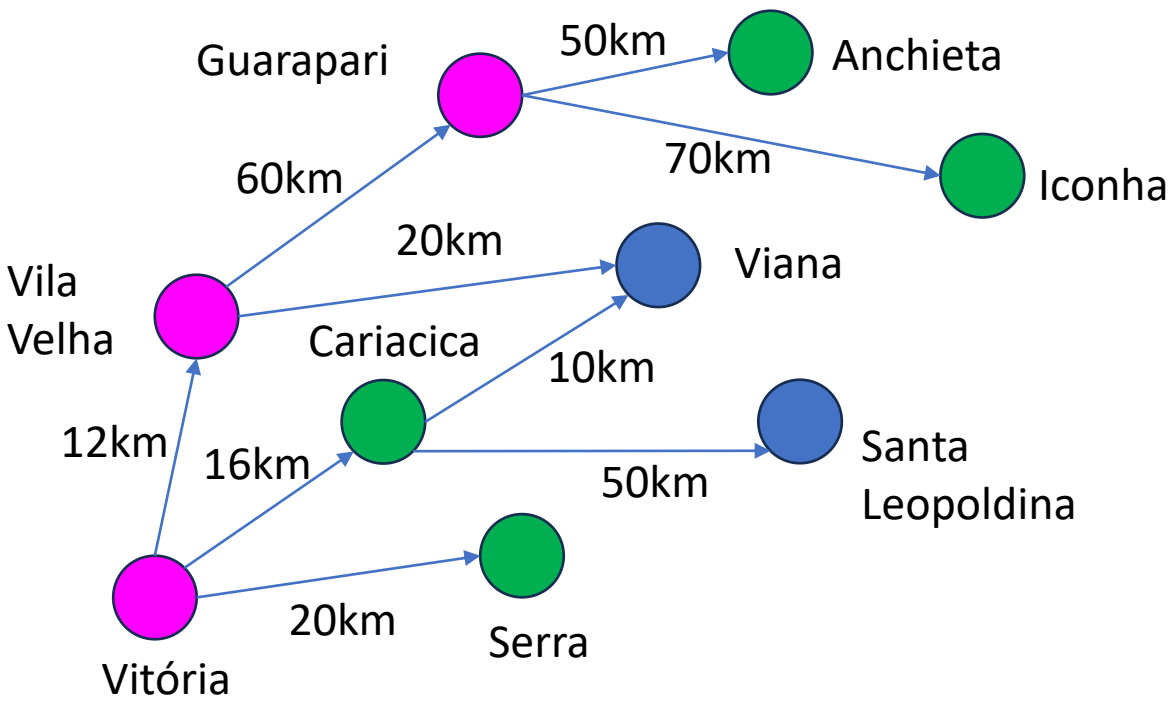


Fronteira = [Cariacica, Serra, Viana, Guarapari]

Visitados = [Vitoria, VilaVelha]

DFS

A última cidade adicionada na fronteira é explorada primeiro.

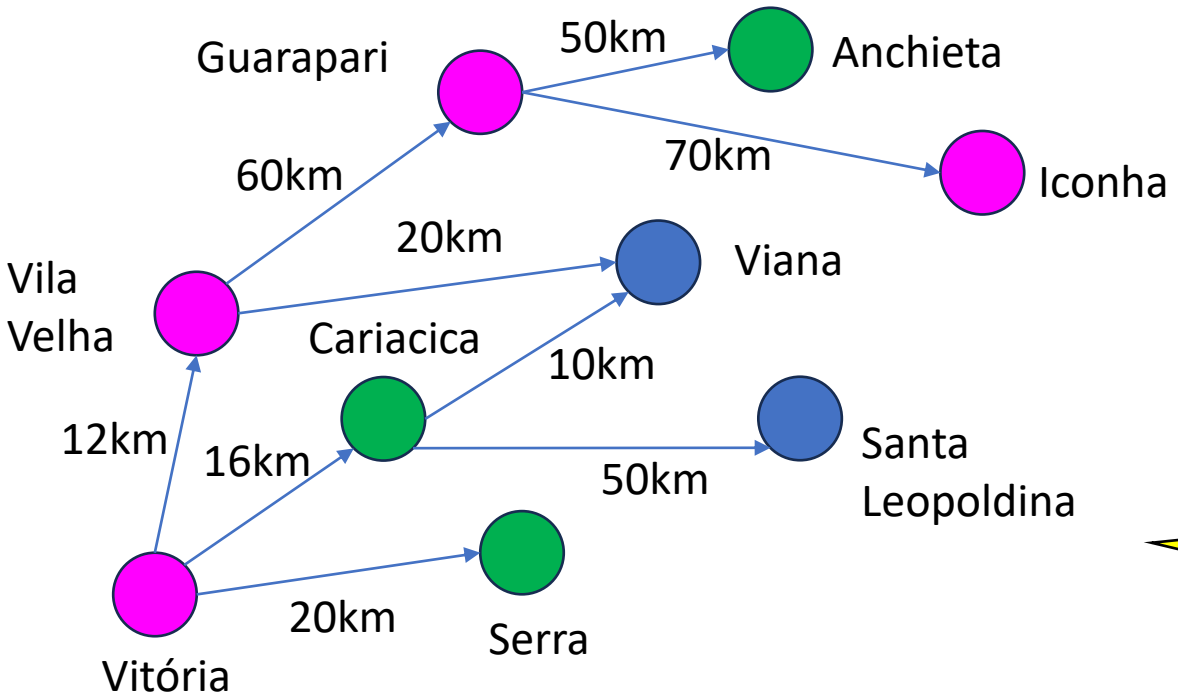


Fronteira = [Cariacica, Serra, Anchieta, Iconha]

Visitados = [Vitoria, VilaVelha, Guarapari]

DFS

A última cidade adicionada na fronteira é explorada primeiro.



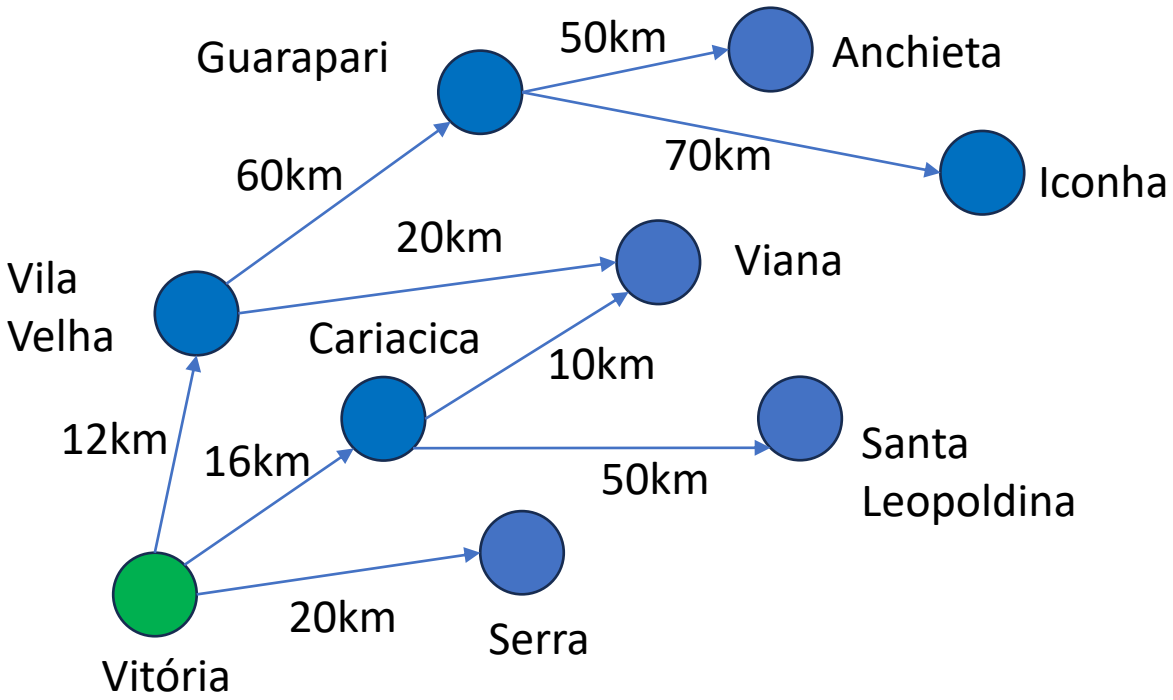
Fronteira = [Cariacica, Serra, Anchieta]

Visitados = [Vitoria, VilaVelha, Guarapari, Iconha]

Alvo Encontrado!

BFS

A cidade adicionada há mais tempo na fronteira é explorada primeiro.

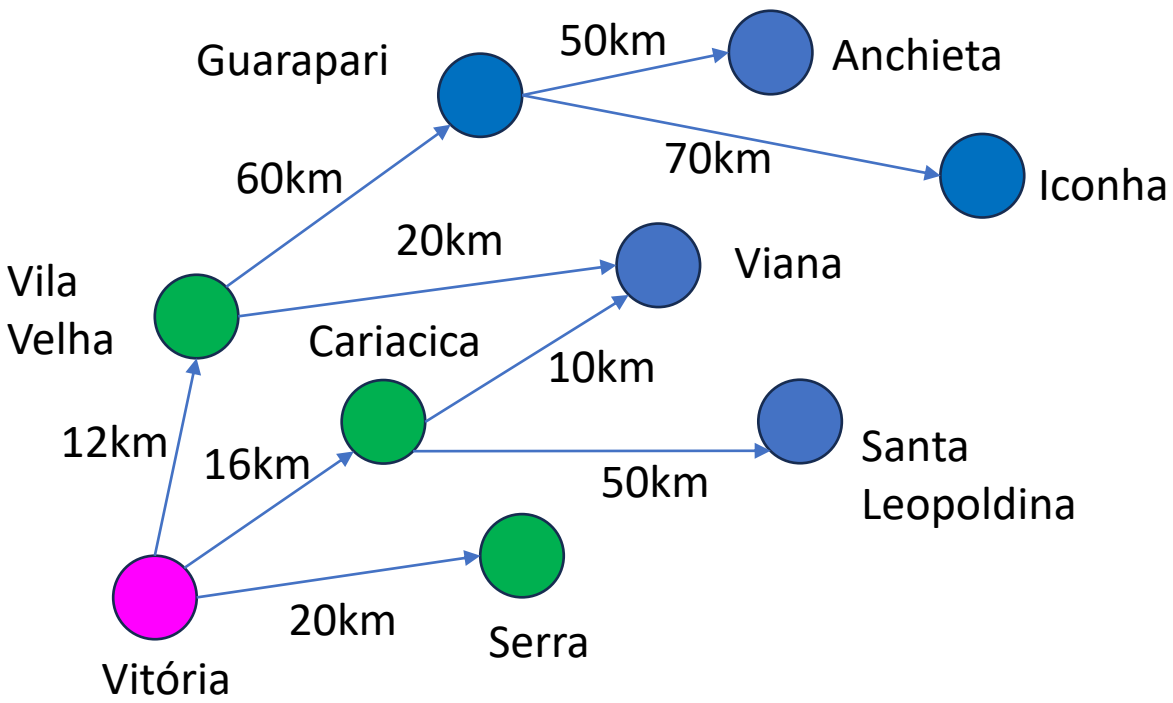


Fronteira = [Vitoria]

Visitados = []

BFS

A cidade adicionada há mais tempo na fronteira é explorada primeiro.

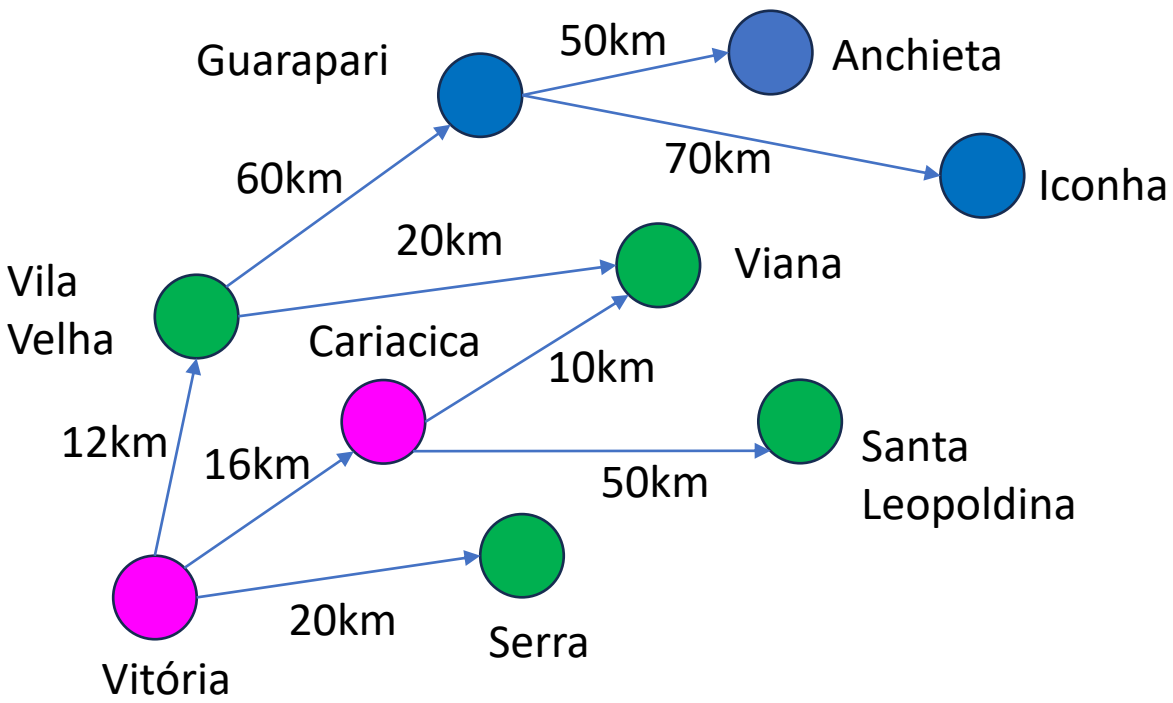


Fronteira = [Cariacica, Serra, VilaVelha]

Visitados = [Vitoria]

BFS

A cidade adicionada há mais tempo na fronteira é explorada primeiro.

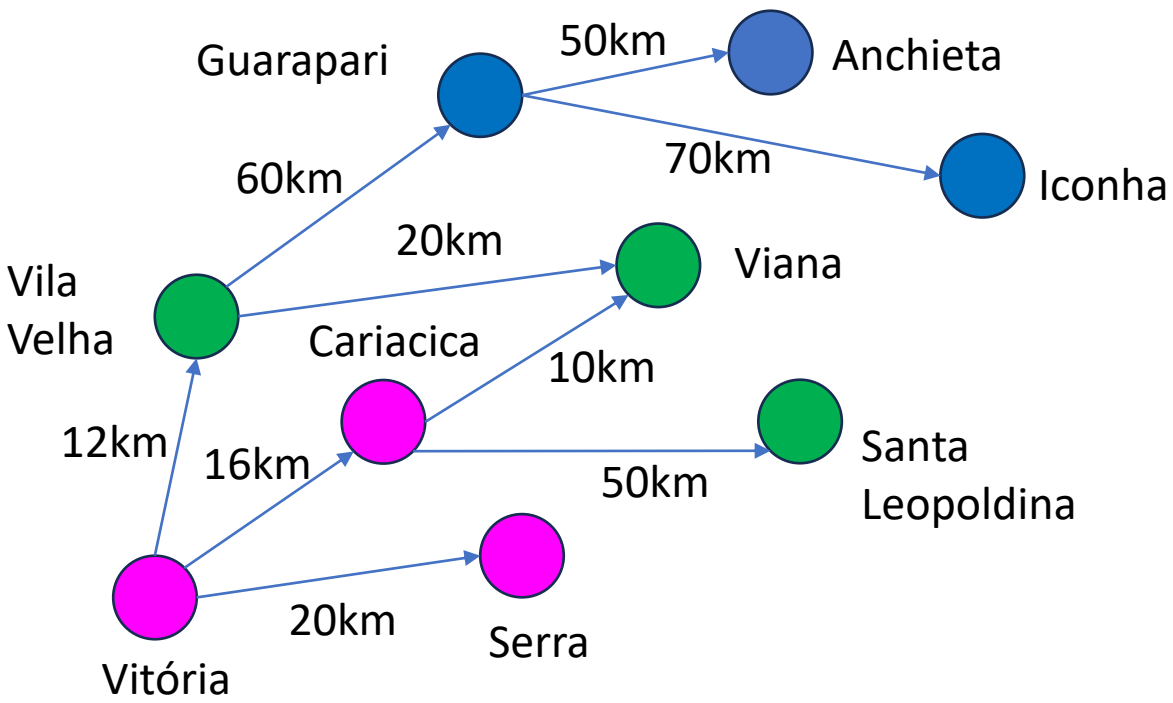


Fronteira = [Serra, VilaVelha, Viana, SantaLeopoldina]

Visitados = [Vitoria, Cariacica]

BFS

A cidade adicionada há mais tempo na fronteira é explorada primeiro.

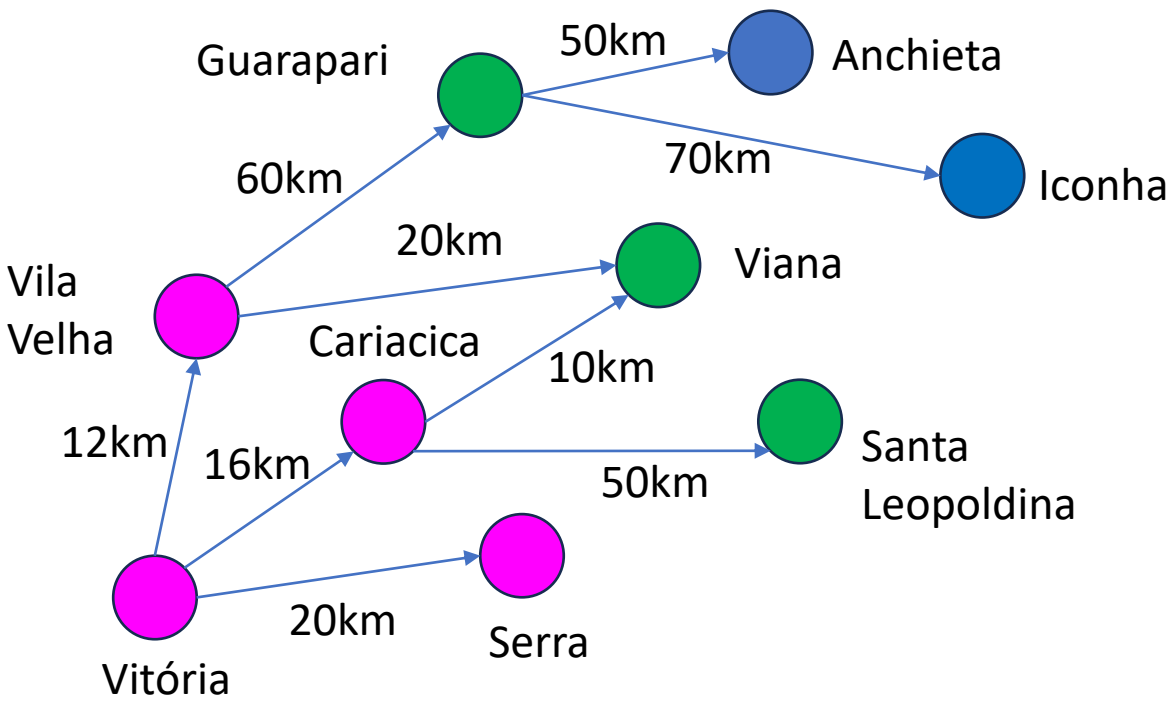


Fronteira = [VilaVelha, Viana, SantaLeopoldina]

Visitados = [Vitoria, Cariacica, Serra]

BFS

A cidade adicionada há mais tempo na fronteira é explorada primeiro.

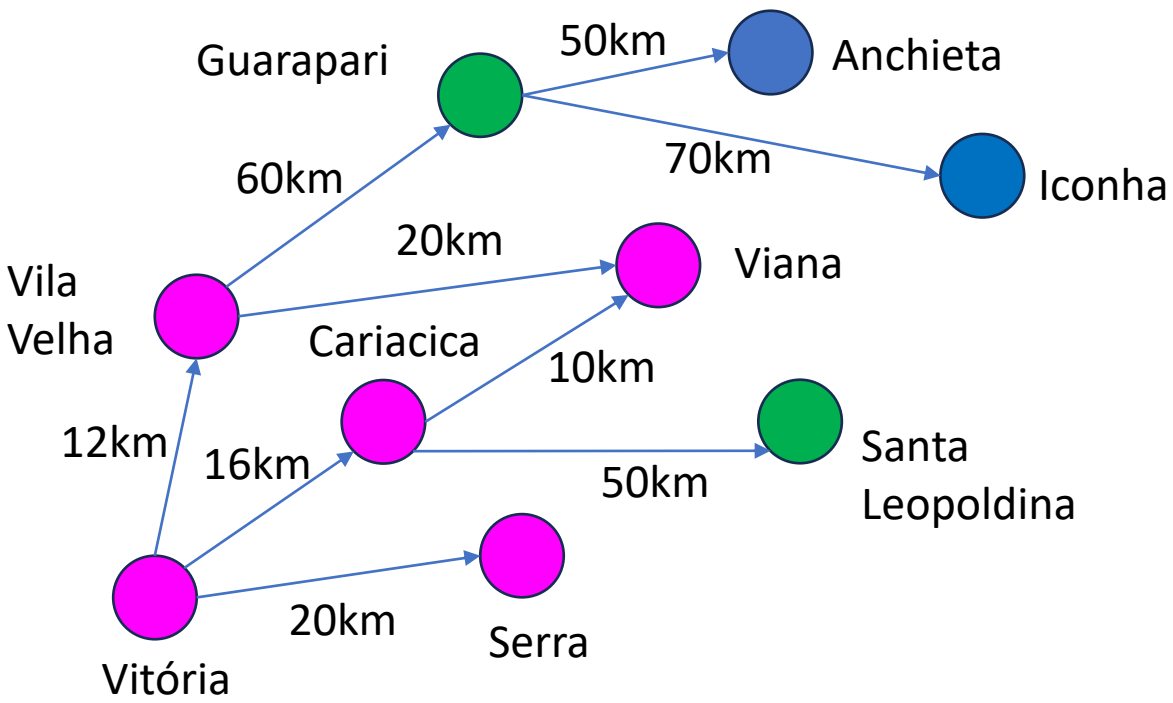


Fronteira = [Viana, SantaLeopoldina, Guarapari]

Visitados = [Vitoria, Cariacica, Serra, VilaVelha]

BFS

A cidade adicionada há mais tempo na fronteira é explorada primeiro.

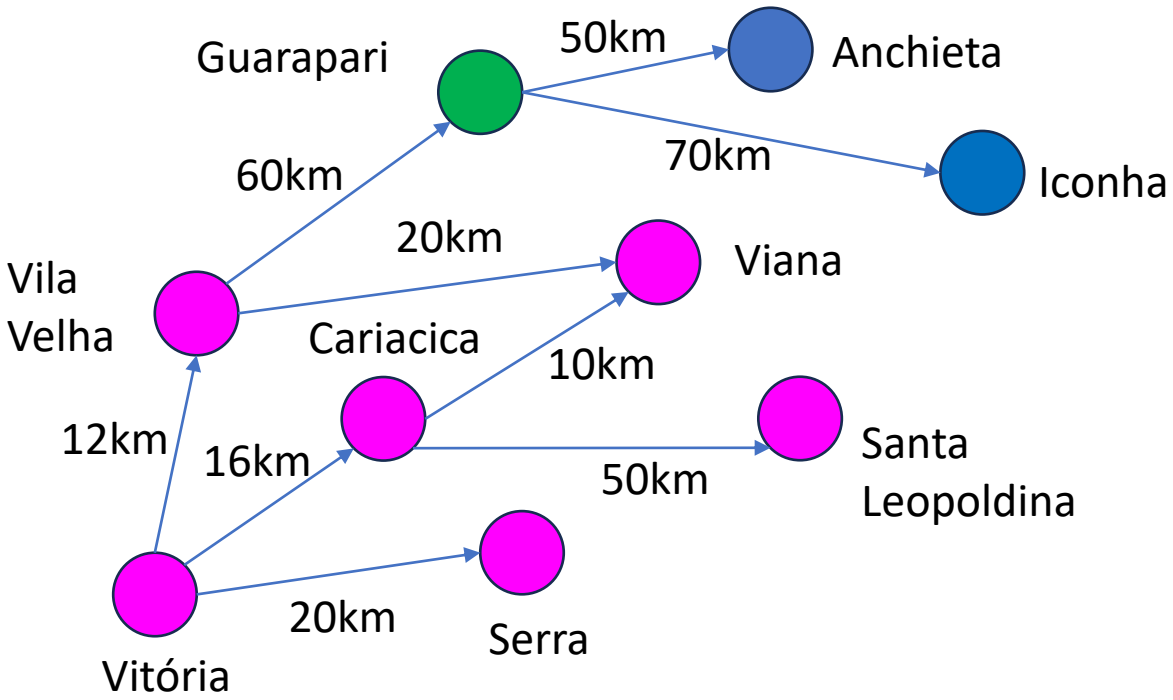


Fronteira = [SantaLeopoldina, Guarapari]

Visitados = [Vitoria, Cariacica, Serra, VilaVelha, Viana]

BFS

A cidade adicionada há mais tempo na fronteira é explorada primeiro.

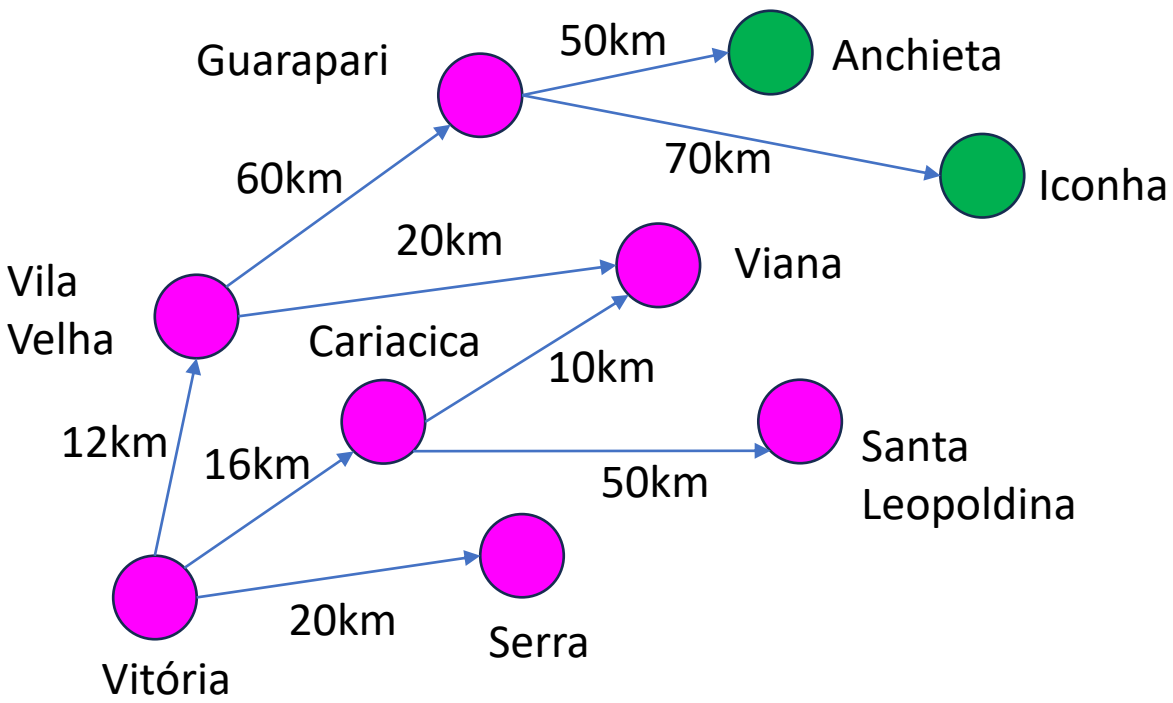


Fronteira = [Guarapari]

Visitados = [Vitoria, Cariacica, Serra, VilaVelha, Viana, SantaLeopoldina]

BFS

A cidade adicionada há mais tempo na fronteira é explorada primeiro.

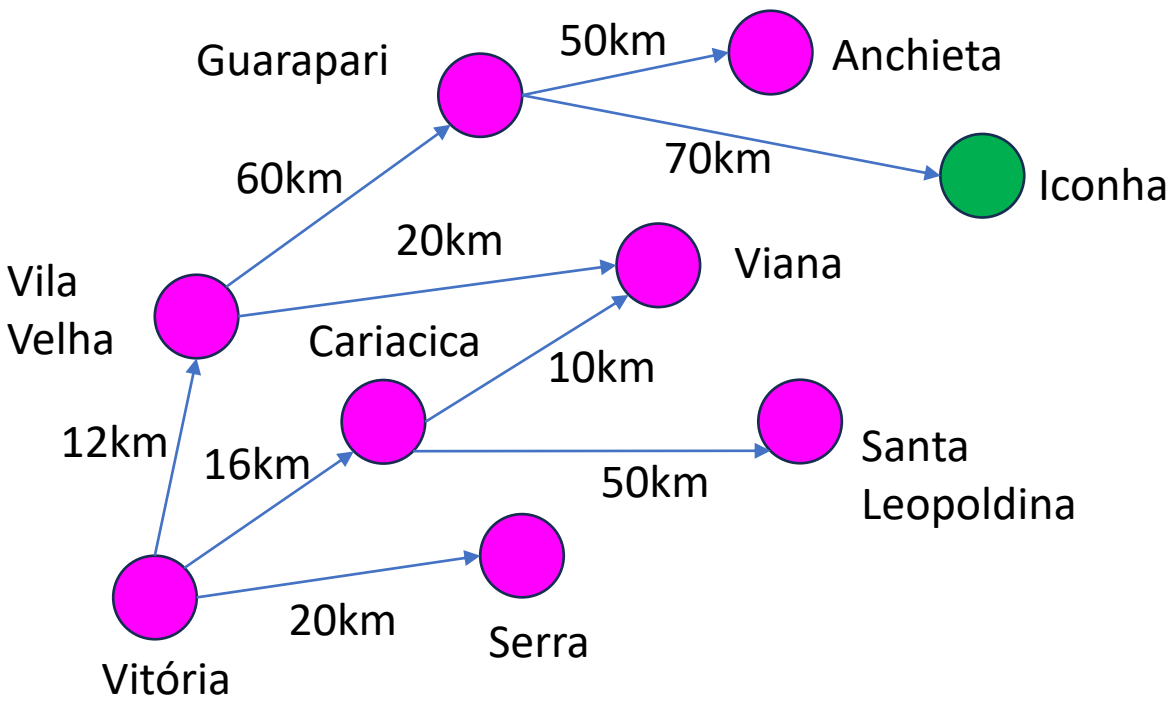


Fronteira = [Anchieta, Iconha]

Visitados = [Vitoria, Cariacica, Serra, VilaVelha, Viana, SantaLeopoldina, Guarapari]

BFS

A cidade adicionada há mais tempo na fronteira é explorada primeiro.

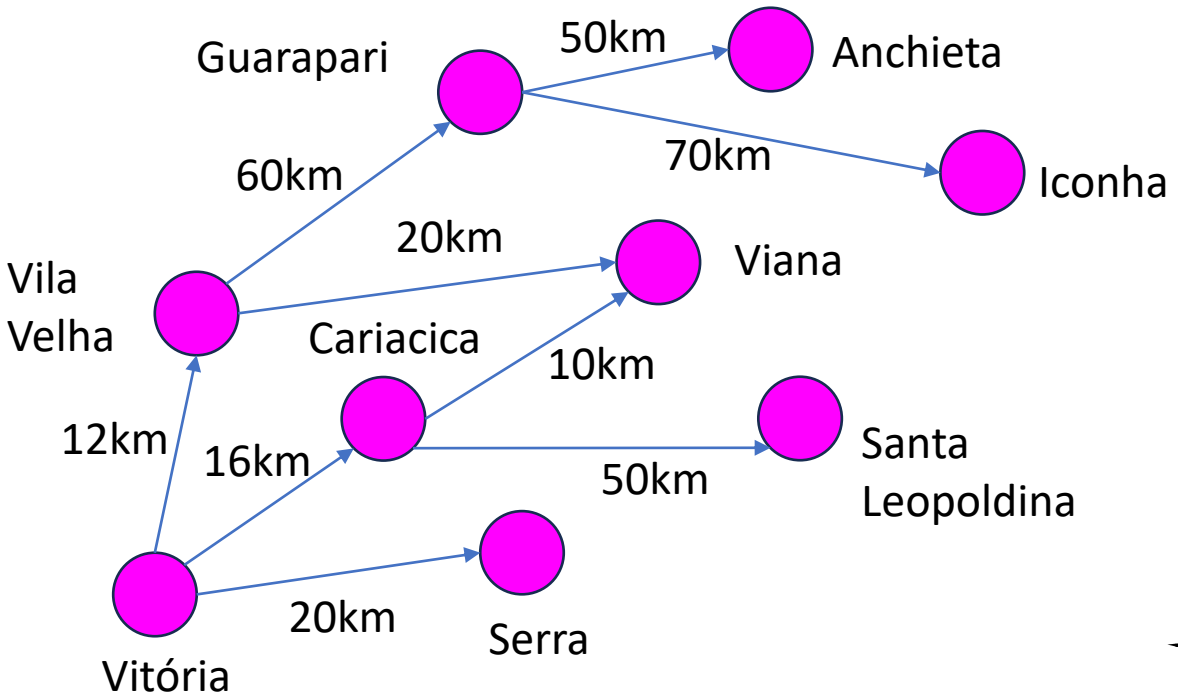


Fronteira = [Iconha]

Visitados = [Vitoria, Cariacica, Serra, VilaVelha, Viana, SantaLeopoldina, Guarapari, Anchieta]

BFS

A cidade adicionada há mais tempo na fronteira é explorada primeiro.



Fronteira = []

Visitados = [Vitoria, Cariacica, Serra, VilaVelha, Viana, SantaLeopoldina, Guarapari, Anchieta, Iconha]

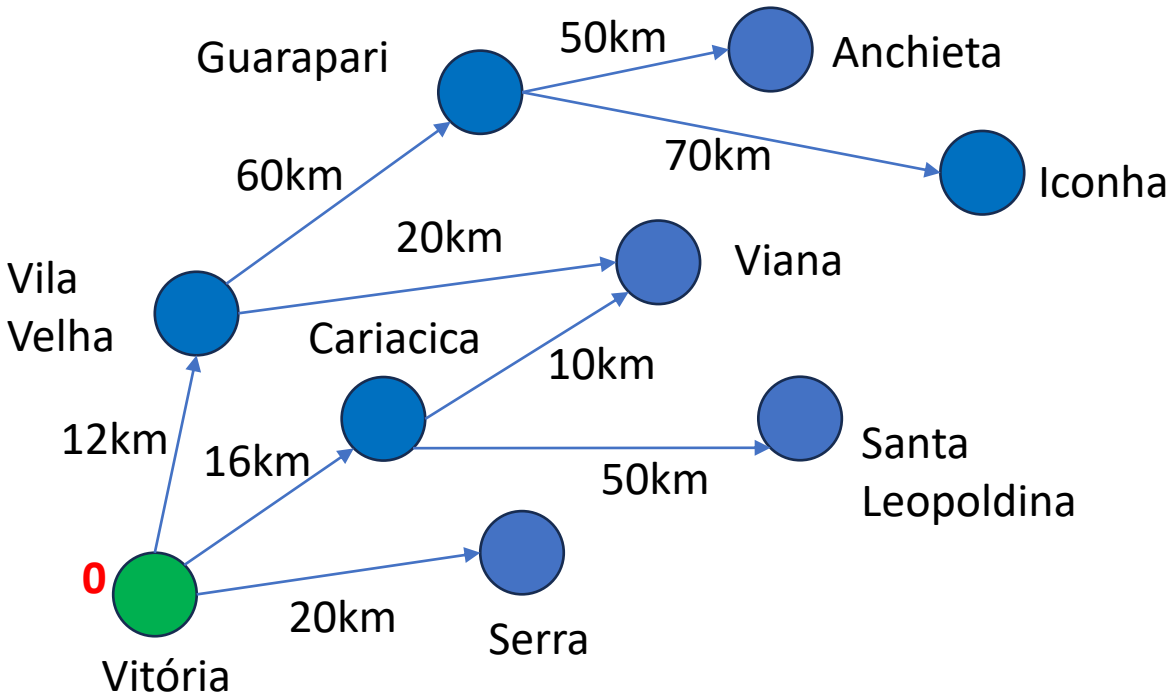
Alvo Encontrado!

DFS vs BFS

- Existem situações em que o DFS encontrará o alvo primeiro e outras em que o BFS será o primeiro.
- A implementação recursiva do DFS requer menos memória que o BFS por manter apenas um caminho armazenado.
- O BFS tem garantia de retornar o caminho com menor número de cidades (mas não necessariamente com menor custo total em KM).

UCS

A cidade com menor custo para a origem na fronteira é explorada primeiro.

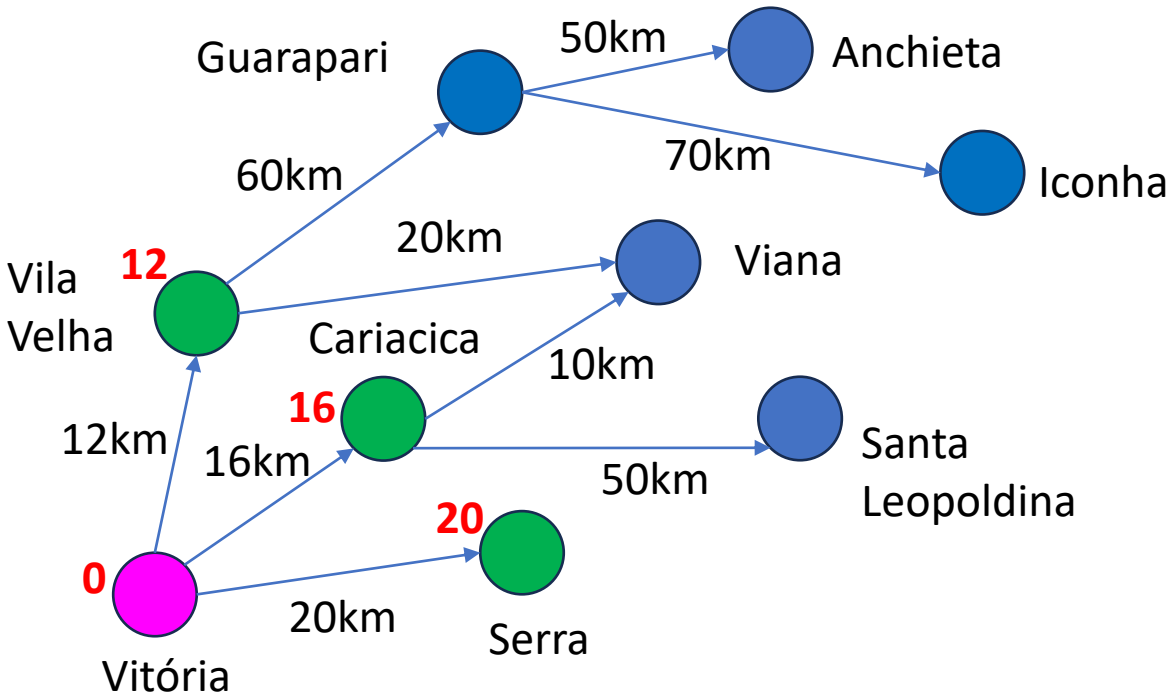


Fronteira = [Vitoria]

Visitados = []

UCS

A cidade com menor custo para a origem na fronteira é explorada primeiro.

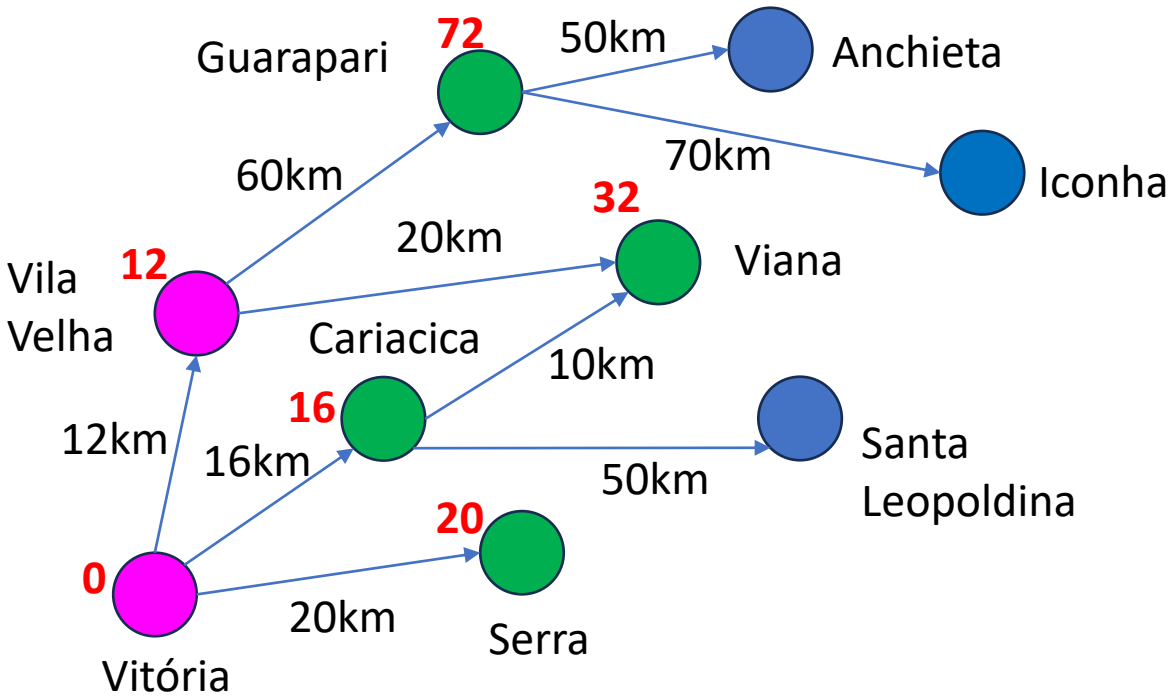


Fronteira = [Cariacica, Serra, VilaVelha]

Visitados = [Vitoria]

UCS

A cidade com menor custo para a origem na fronteira é explorada primeiro.

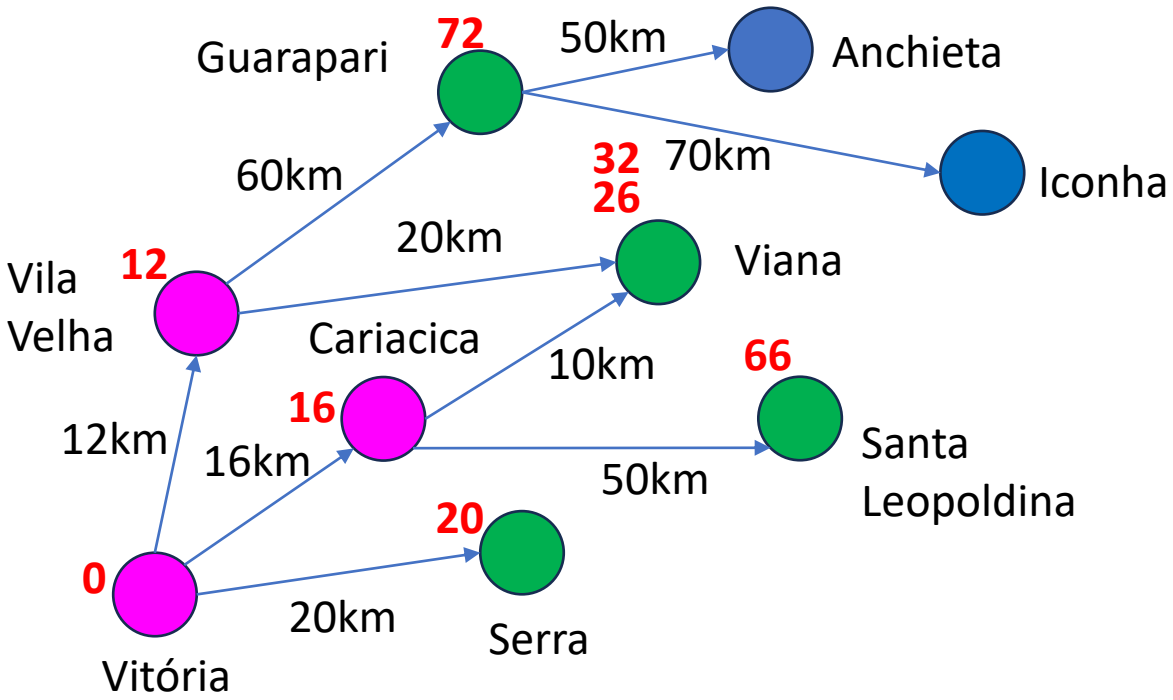


Fronteira = [Cariacica, Serra, Guarapari, Viana]

Visitados = [Vitoria, VilaVelha]

UCS

A cidade com menor custo para a origem na fronteira é explorada primeiro.

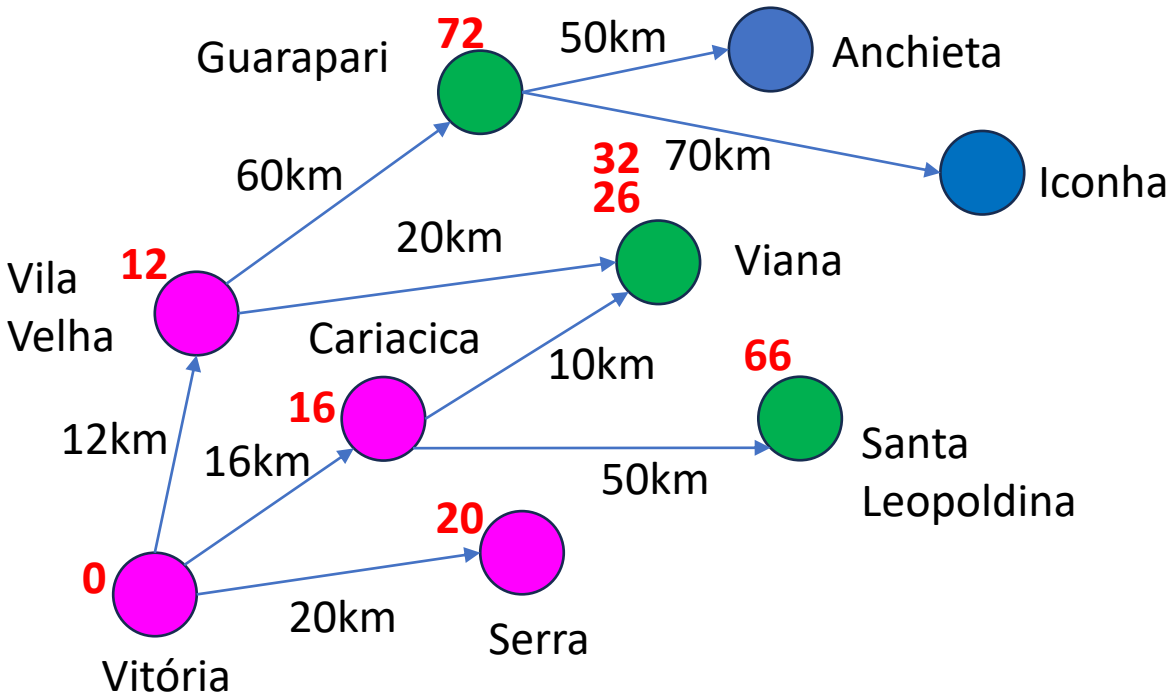


Fronteira = [Serra, Guarapari, Viana, SantaLeopoldina]

Visitados = [Vitoria, VilaVelha, Cariacica]

UCS

A cidade com menor custo para a origem na fronteira é explorada primeiro.

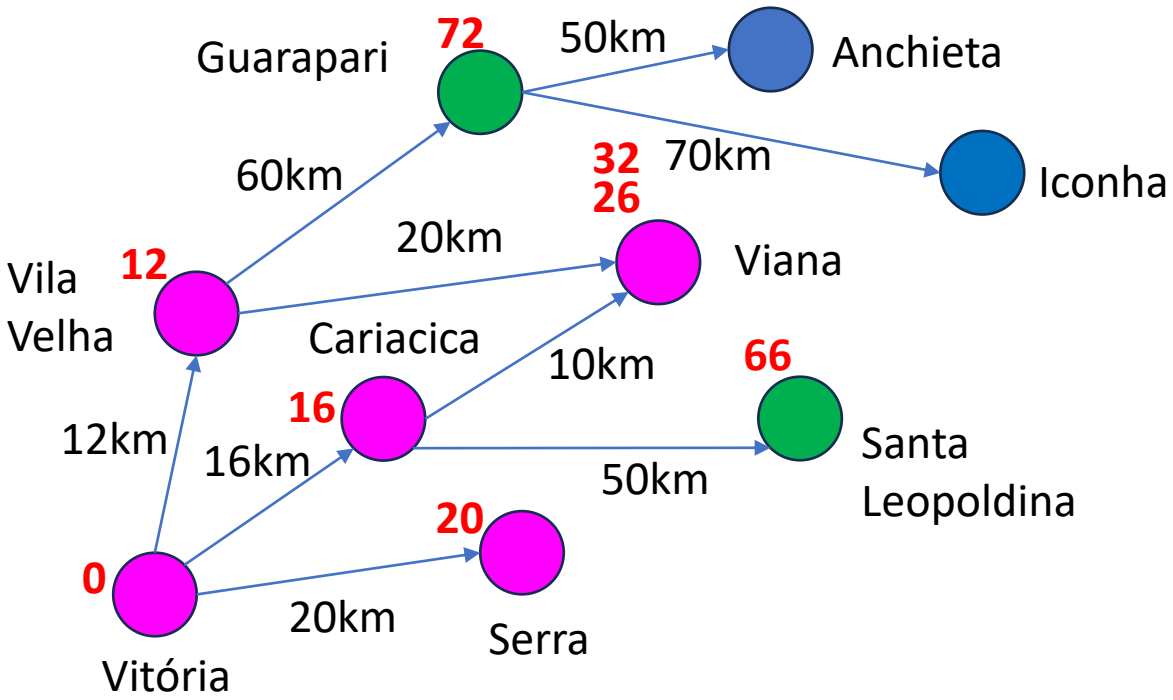


Fronteira = [Guarapari, Viana, SantaLeopoldina]

Visitados = [Vitoria, VilaVelha, Cariacica, Serra]

UCS

A cidade com menor custo para a origem na fronteira é explorada primeiro.

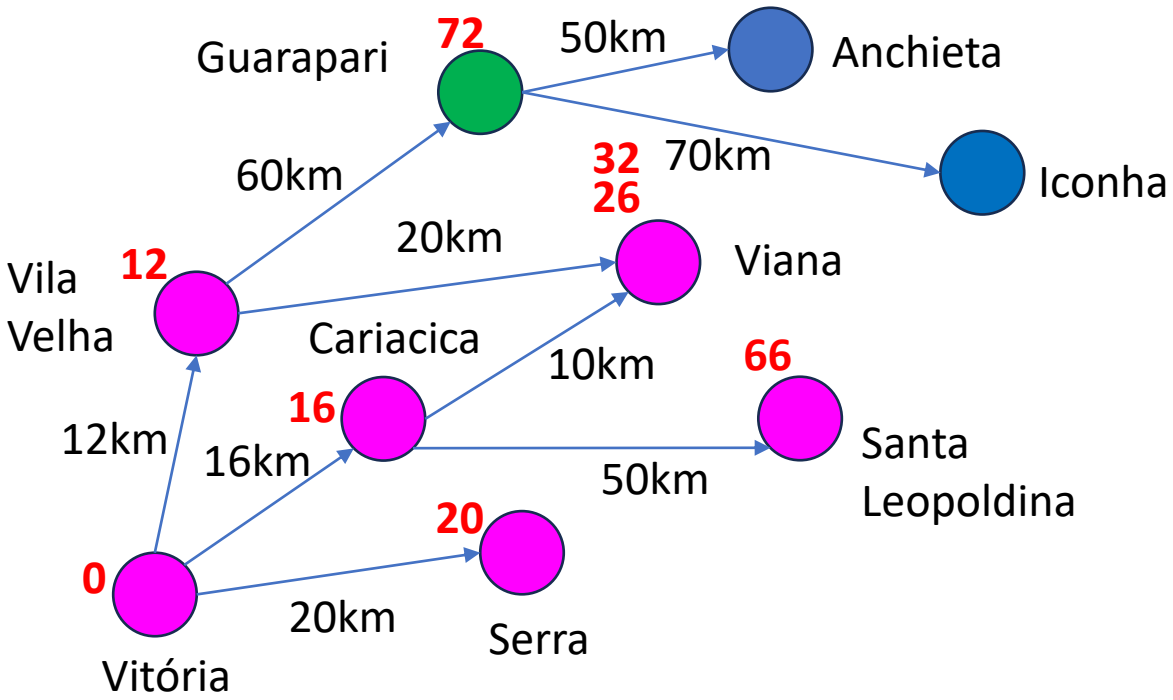


Fronteira
= [Guarapari, SantaLeopoldina]

Visitados =
[Vitoria, VilaVelha, Cariacica, Serra, Viana]

UCS

A cidade com menor custo para a origem na fronteira é explorada primeiro.

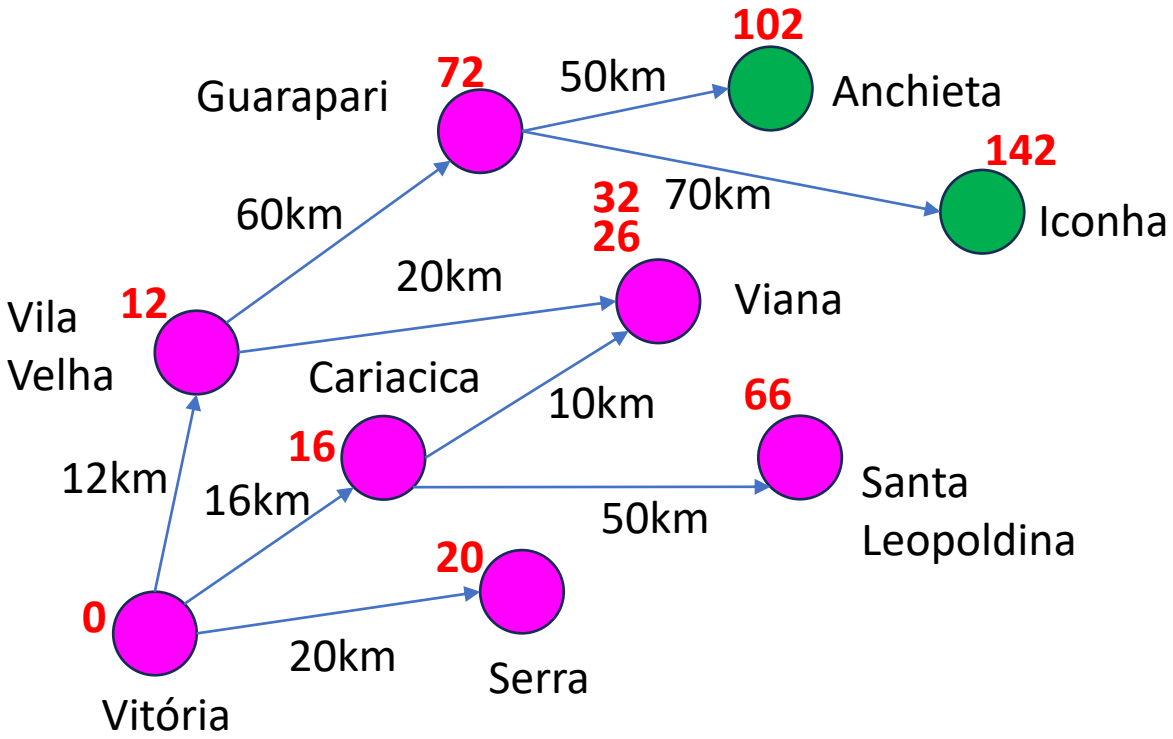


Fronteira = [Guarapari]

Visitados =
[Vitoria, VilaVelha, Cariacica, Serra, Viana, SantaLeopoldina]

UCS

A cidade com menor custo para a origem na fronteira é explorada primeiro.

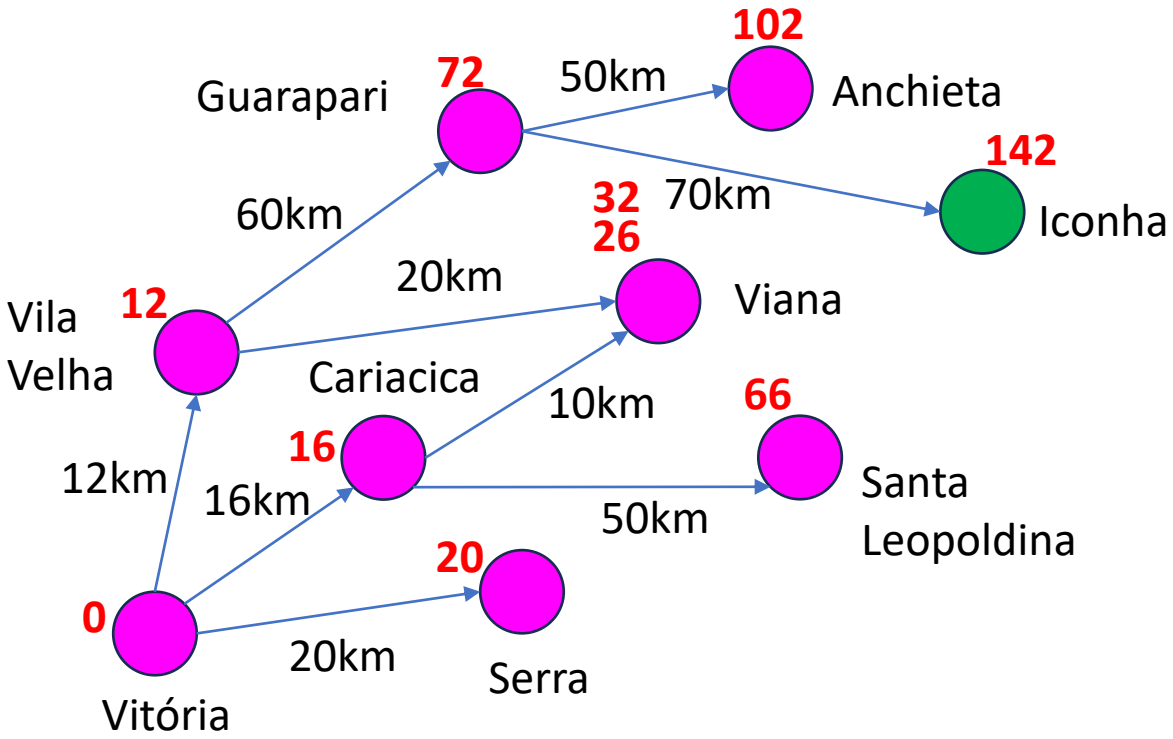


Fronteira = [Anchieta, Iconha]

Visitados =
[Vitoria, VilaVelha, Cariacica, Serra, Viana, SantaLeopoldina, Guarapari]

UCS

A cidade com menor custo para a origem na fronteira é explorada primeiro.

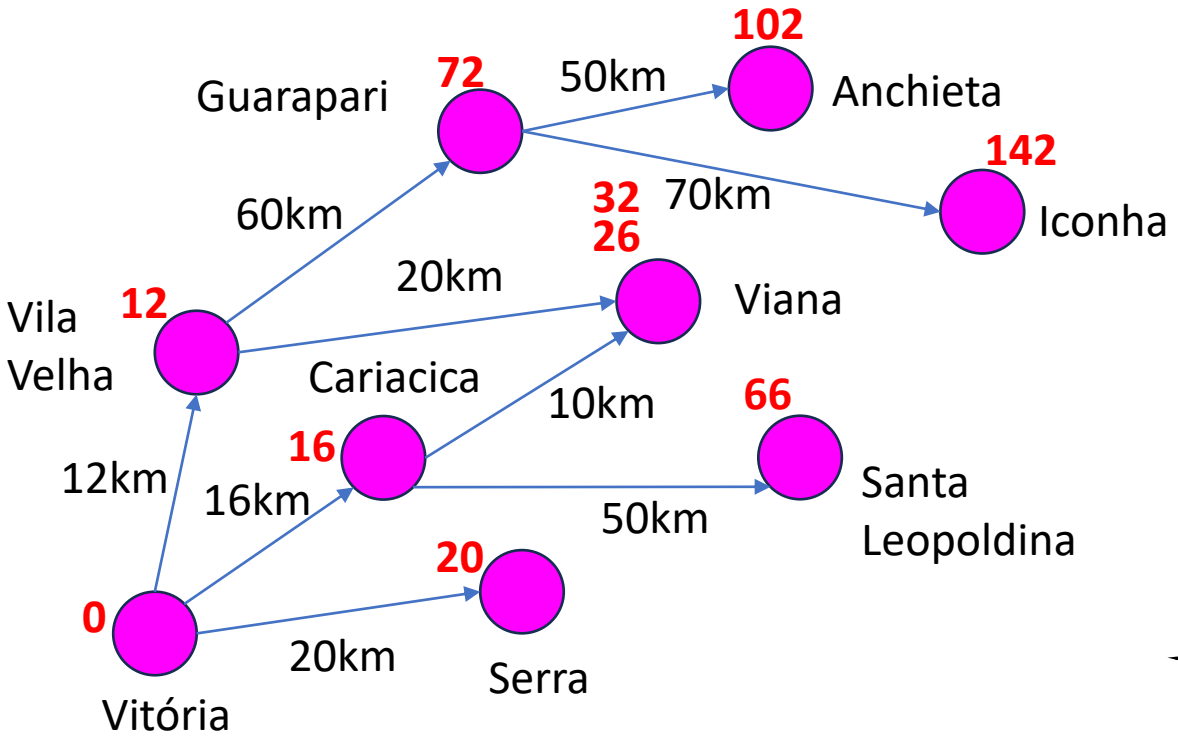


Fronteira = [Iconha]

Visitados =
[Vitoria, VilaVelha, Cariacica, Serra, Viana, SantaLeopoldina, Guarapari, Anchieta]

UCS

A cidade com menor custo para a origem na fronteira é explorada primeiro.



Fronteira = []

Visitados =
[Vitoria, VilaVelha, Cariacica, Serra, Viana, SantaLeopoldina, Guarapari, Anchieta, Iconha]

Alvo Encontrado!

UCS

- Explora primeiro os caminhos mais curtos em relação à origem.
- Garante que o caminho retornado é o mais curto. Se existisse uma alternativa melhor, ela teria sido explorada primeiro.
- Tem o mesmo comportamento do algoritmo de Dijkstra, mas evita manter o grafo completo na memória. Portanto, é uma alternativa melhor para grafos grandes.

Algoritmo A* (A-estrela ou *A-star*)

Nós com menor valor de f(n) são expandidos primeiro:

$$f(n) = g(n) + h(n) \ .$$

Distância total da origem até o nó n (mesmo do UCS)

Custo estimado do nó n até o destino (usaremos a distância euclidiana)

h(.) é chamada de função heurística e incentiva que nós sejam explorados primeiro se eles "parecem" levar rápido ao alvo. Na prática, nós não sabemos o custo de um nó para o destino, então usamos uma função para estimar este valor.

Algoritmo A* (A-estrela ou *A-star*)

Se a função heurística for **admissível**, isto é, se ela nunca superestima o custo real para atingir o alvo, é garantido que o A* retornará o caminho de menor custo.

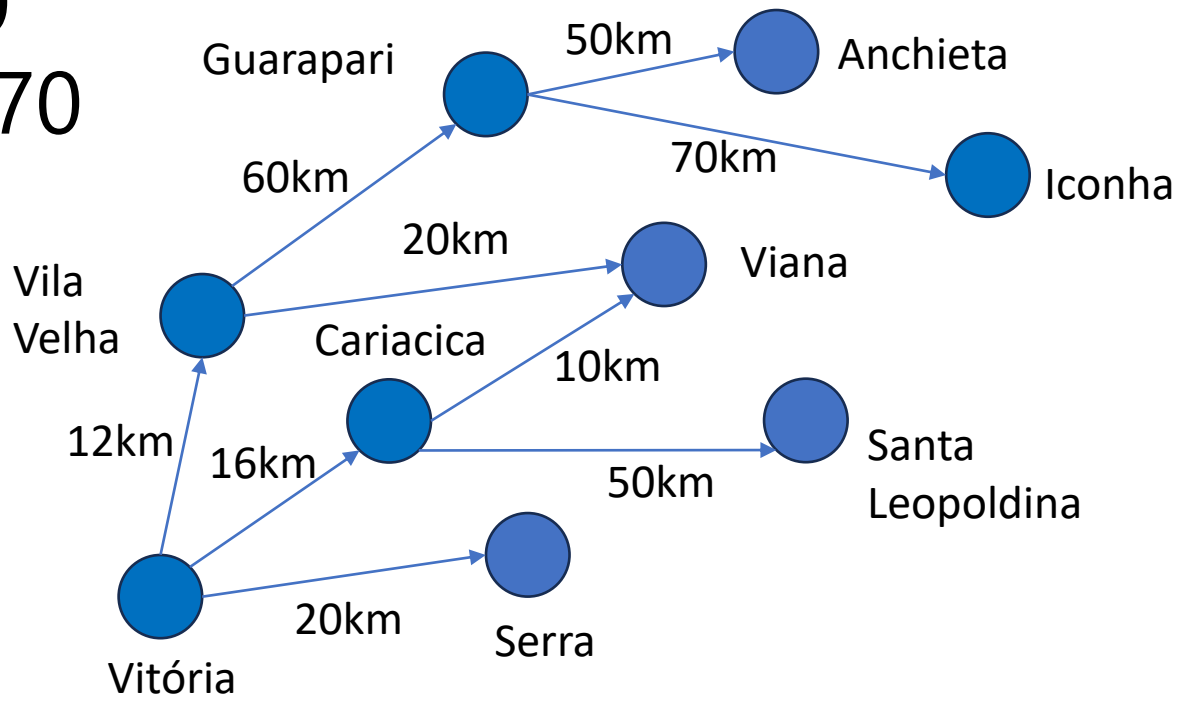
UCS

Vitoria está em (0, 0)

Vitoria tem 3 cidades vizinhas:

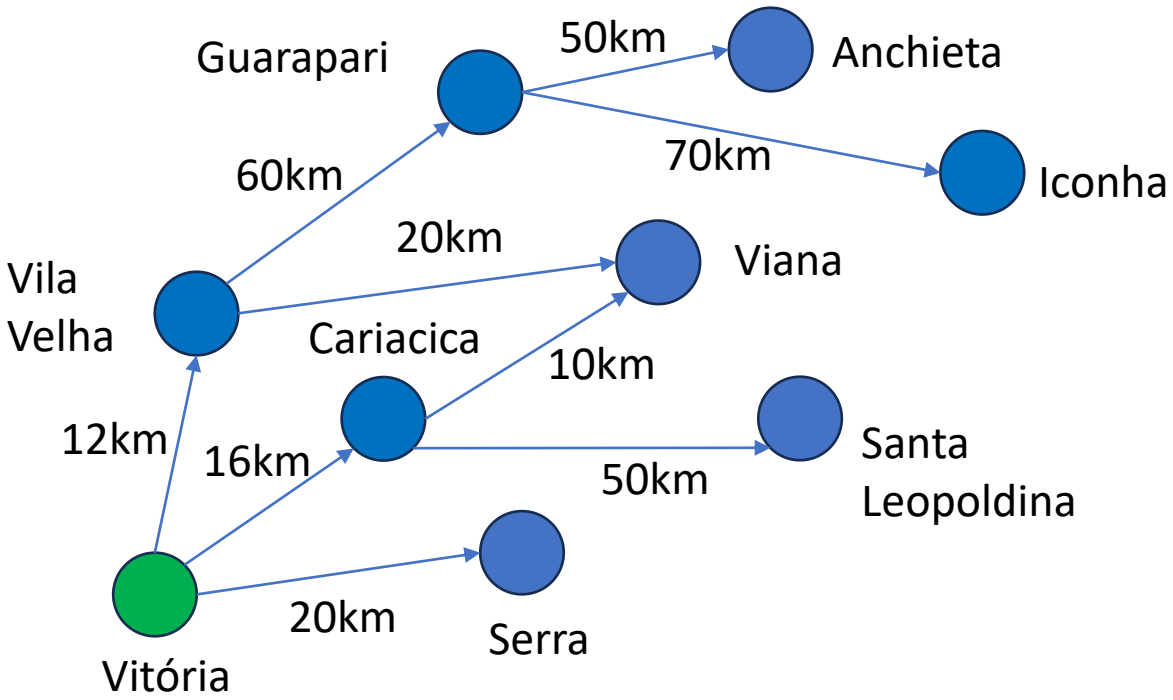
- 1 (Cariacica) a 16 km
- 2 (Serra) a 20km
- 3 (VilaVelha) a 12km

```
0 5
6
0 Vitoria 0 0 3 1 16 2 20 3 12
1 Cariacica -16 0 3 0 16 7 10 8 50
2 Serra 16 16 1 0 20
3 VilaVelha 0 15 3 0 12 4 60 7 20
4 Guarapari 10 25 3 3 60 5 50 6 70
5 Anchieta 25 40 1 4 50
6 Iconha 20 55 1 4 70
...
```



A*

A cidade com menor custo para a origem + estimativa de distância para o alvo é explorada primeiro.



Fronteira = [Vitoria]

Visitados = []

UCS vs A*

A função heurística guia a busca por regiões mais promissoras do espaço de busca, o que pode reduzir o número de nós visitados.

Recuperação de Caminhos

- Vizinhos devem ser adicionados na fronteira na ordem em que aparecem no arquivo.
- Para recuperar o caminho, sempre que um nó for visitado, armazene o nó pai (de onde ele veio).
  - Por exemplo, pode ser criado um array de inteiros do mesmo tamanho do número de cidades, em que cada posição armazena o índice do pai.
  - Se o array for inicializado com um índice inválido de pai (-1, por exemplo), ele pode ser usado para verificar se um nó já foi visitado ou não.
- Nós que já foram visitados não devem ser visitados novamente.
- Para recuperar o caminho: partindo do nó destino, salte para o nó pai iterativamente enquanto não chegar na origem. O caminho é a sequência de nós invertida.

Custo Total para a Origem no UCS e no A*

- Armazene em cada nó o custo para a origem.
- O nó inicial tem custo 0.
- Sempre que um vizinho por adicionado na fronteira, calcule o custo até a origem como:

  custo do pai até a origem (armazenado no nó pai) + *custo do pai até o nó*

- Se for encontrado outro caminho até um nó que já está na fronteira, adicione ele novamente na fronteira. Se existir mais de um caminho para o mesmo nó, o menor deve ser explorado primeiro. Lembre-se que os nós pai nos dois caminhos poderão ser diferentes.