Projeto e Análise de algoritmos

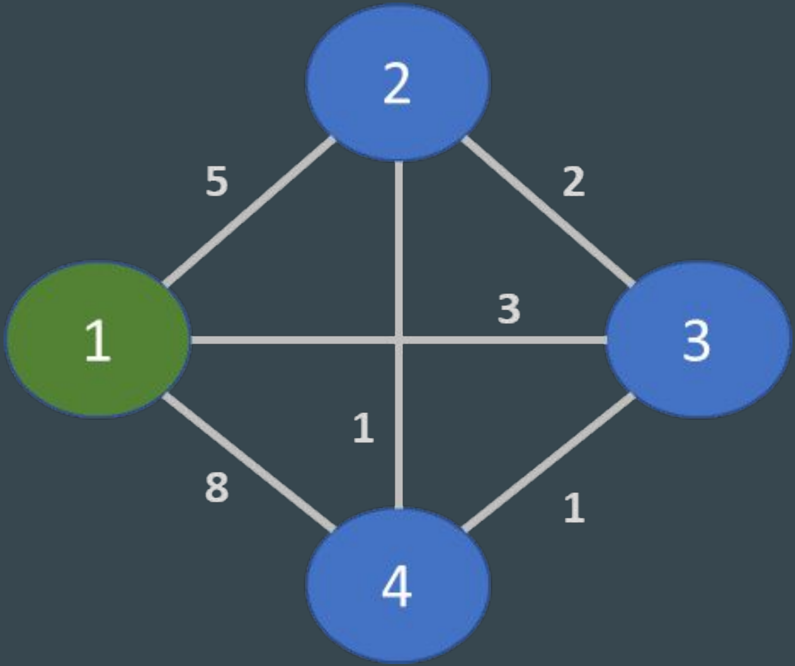
Entrega 2 - Problema do Caixeiro Viajante

Carlos Meneses, Gabriel Reis, Paulo Araujo

Descrição do Problema

- O problema do caixeiro-viajante tenta determinar a menor rota para percorrer todos os nós de um grafo conectado, retornando ao nó de origem.
- Ele é um problema de otimização NP-difícil inspirado na necessidade dos vendedores em realizar entregas em diversos locais no menor tempo possível.

Algoritmo de Força Bruta



Array de Permutação

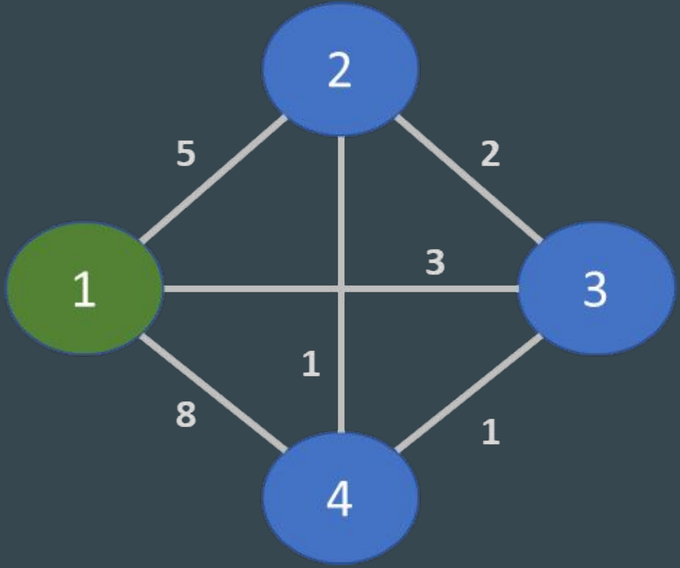
1	2	3	4

2	3	4
2	4	3
3	2	4
3	4	2
4	2	3
4	3	2

Todas as rotas possíveis

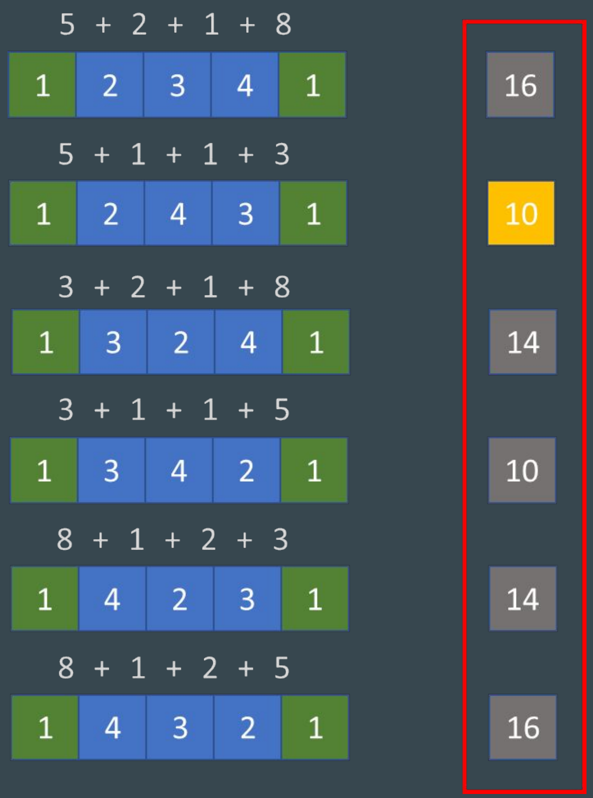
1	2	3	4	1
1	2	4	3	1
1	3	2	4	1
1	3	4	2	1
1	4	2	3	1
1	4	3	2	1

Calcular o custo de cada rota e salva no vetor

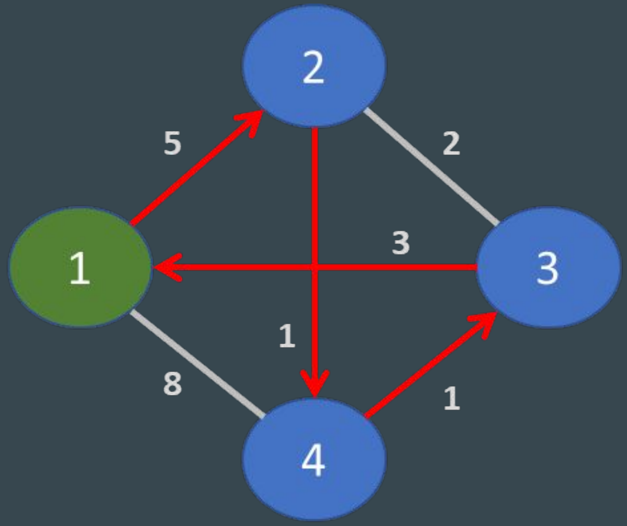


5 + 2 + 1 + 8						
1	2	3	4	1		16
5 + 1 + 1 + 3						
1	2	4	3	1		10
3 + 2 + 1 + 8						
1	3	2	4	1		14
3 + 1 + 1 + 5						
1	3	4	2	1		10
8 + 1 + 2 + 3						
1	4	2	3	1		14
8 + 1 + 2 + 5						
1	4	3	2	1		16

Buscar o índice do menor custo



Resultado

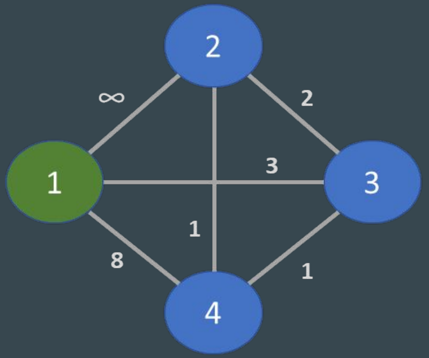


5 + 1 + 1 + 3

1	2	4	3	1

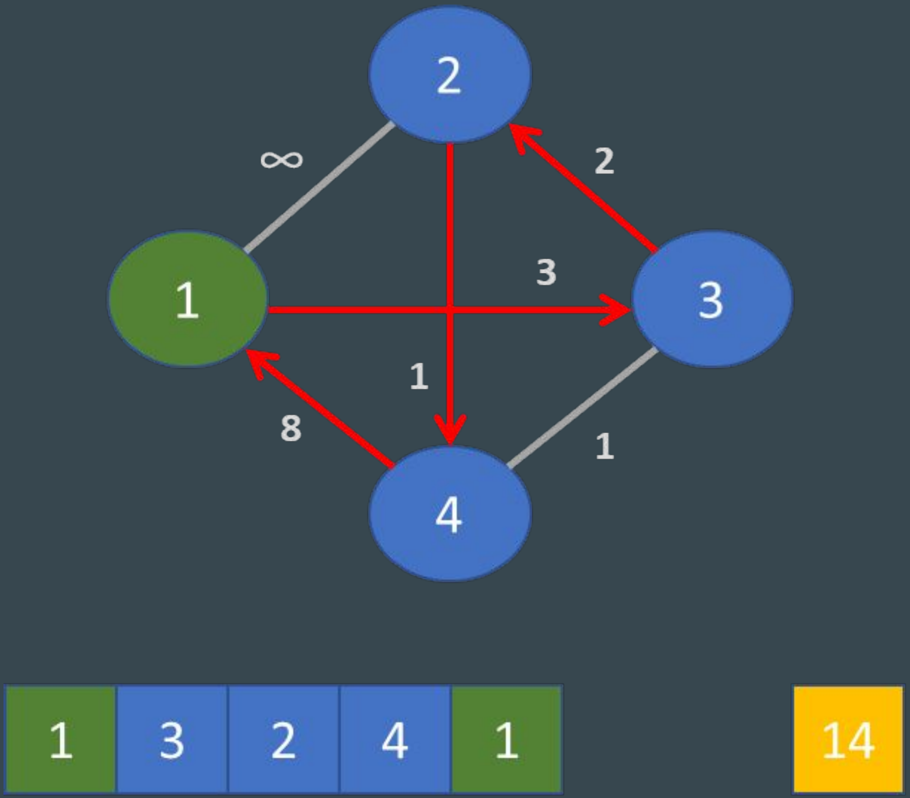
10

E se uma aresta não existir?



$\infty + 2 + 1 + 8$						
1	2	3	4	1		∞
$\infty + 1 + 1 + 3$						
1	2	4	3	1		∞
$3 + 2 + 1 + 8$						
1	3	2	4	1		14
$3 + 1 + 1 + \infty$						
1	3	4	2	1		∞
$8 + 1 + 2 + 3$						
1	4	2	3	1		14
$8 + 1 + 2 + \infty$						
1	4	3	2	1		∞

Resultado



Algoritmo e Complexidade

1. Array de nós para permutação - O(n-1)
2. Permutar o array - O(n-1!)
3. Adicionar nó inicial no início e fim do array - O(n-1!)
4. Calcular o custo de cada rota - O(n-1!)
5. Buscar menor custo - O(n-1!)

Array de nós para permutação - O(n-1)

```
function createPermutationArray(init, end) {
  const length = end - init + 1; // O(1)
  const array = new Array(length).fill(init); // O(n-1)
  array.forEach((value, index) => { // O(n-1)
    array[index] = value + index; // O(1)
  });
  return array; // O(1)
}
```

Permutar o array - O(n!)

```
function permuteList(array) { // 7 * O(1) + 2 * O(n-1) + T(n-1) = T(n-1) + O(n-1) = O(n-1!)
  if (array.length == 1) { // O(1)
    return [array]; // O(1)
  }

  let result = []; // O(1)

  for (let i = 0; i < array.length; i++) { // O(n-1)
    const currentElement = array[i]; // O(1)
    const remainingElements = array.slice(0, i).concat(array.slice(i + 1)); // O(1)
    const remainingPermutations = permuteList(remainingElements); // T(N-1)

    for (let j = 0; j < remainingPermutations.length; j++) { // O(n-1)
      result.push([currentElement].concat(remainingPermutations[j])); // O(1)
    }
  }

  return result; // O(1)
}
```

Adicionar nó inicial no início e fim do array - O(n-1!)

```
function addInitialNode(array) {
  return array.map((list) => { // O(n-1)
    return [1, ...list, 1]; // O(1)
  });
}
```

Calcular o custo de cada rota - O(n-1!)

```
function calcRoutesValues(permutations, grafh) {
  let permutationsValues = []; // O(1)

  for (let i = 0; i < permutations.length; i++) { // O(n-1)
    const permutation = permutations[i]; // O(1)
    let value = 0; // O(1)

    for (let j = 0; j < permutation.length - 1; j++) { // O(n-2)
      const origin = permutation[j]; // O(1)
      const destiny = permutation[j + 1]; // O(1)
      value += grafh[origin - 1][destiny - 1]; // O(1)
    }

    permutationsValues.push(value); // O(1)
  }

  return permutationsValues; // O(1)
}
```

Buscar menor custo - O(n-1!)

```
function calcMinValueIndex(array) {
  let min = array[0]; // O(1)
  let minValueIndex = 0; // O(1)

  for (let i = 1; i < array.length; i++) { // (O(n-1))
    if (array[i] < min) { // O(n-1)
      min = array[i]; // O(1) -> O(n-1)
      minValueIndex = i; // O(1) -> O(n-1)
    }
  }

  return minValueIndex; // O(1)
}
```

COMPLEXIDADE DO ALGORITMO DE FORÇA BRUTA

- Complexidade de tempo: O(n-1!)

- Onde n é o número de vértices no grafo. Isso ocorre porque o algoritmo gera todas as permutações possíveis do conjunto de vértices com exceção do ponto inicial.