Aula 1 - Buscas em Árvores e Grafos 1

Docupedia Export

Author:Silio Leonardo (CtP/ETS)
Date:11-Oct-2023 14:33

Table of Contents

1 Problemas de Busca

Um problema de busca é um problema onde precisamos encontrar um elemento em uma estrutura de dados. Podemos buscar uma melhor jogada em um jogo como xadrez ou buscar a existência de um elemento de uma árvore binária. Uma IA pode ser implementada como uma forma de busca.
Nos próximos tópicos vamos ver como implementar IA's apenas usando algoritmos de busca. E, é claro, quais algoritmos que podem ser implementados para realizar essa busca.

2 Busca em Profundidade e Busca em Largura

```
using System;
using System.Collections.Generic;

var objA = new TreeState
{
    IsObjective = true
};
var objB = new TreeState
{
    IsObjective = true
};

var state = new TreeState
{
    Children = {
        new TreeState(),
        objB,
        new TreeState
        {
            Children = {
                objA
            }
        }
    }
};

Console.WriteLine(depthFirstSearch(state) == objA);
Console.WriteLine(breadthFirstSearch(state) == objB);

TreeState depthFirstSearch(TreeState state)
{
    var stack = new Stack<TreeState>();
    stack.Push(state);
```

```
    while (stack.Count > 0)
    {
        var current = stack.Pop();

        // Start of Variable Code Here
        if (current.IsObjective)
            return current;
        // End of Variable Code Here

        foreach (var child in current.Children)
            stack.Push(child);
    }

    return null;
}

TreeState breadthFirstSearch(TreeState state)
{
    var queue = new Queue<TreeState>();
    queue.Enqueue(state);

    while (queue.Count > 0)
    {
        var current = queue.Dequeue();

        // Start of Variable Code Here
        if (current.IsObjective)
            return current;
        // End of Variable Code Here

        foreach (var child in current.Children)
            queue.Enqueue(child);
    }

    return null;
}

public class TreeState
```

```
{
    public bool IsObjective { get; set; } = false;
    public List<TreeState> Children { get; } = new();
}
```

3 Dijkstra

O algoritmo de Dijkstra é um algoritmo para busca em grafos com pesos de ordem O(E + V lg V), ou seja, o tempo de execução escala linearmente com a quantidade de arestas e n lg(n) no números de vértices. Ele ajudará a achar o menor caminho de um ponto a outro.

```
Cria um vetor dist de distâncias que mede a distância do ponto inicial ao final
Cria um vetor prev de estados anteriores a cada estado v com valores null
Crie uma fila de prioridade Q = { source : 0 }
Enquanto Q não for vazia:
    Pegue um estado u de Q tal qual a distância é a menor possível (dist(u))
    Remova u de Q

    Para cada vizinho direto v de u:
        d = dist(u, target) + dist(v, u)
        se d < dist(v), ou seja, é um caminho melhor até v:
            dist(v) = d
            prev(v) = u
            adicione v a Q
Retorna dist, prev
```

Note que a distância pode ser tanto uma distância 'física' ou 'geométrica' quanto uma medida que faça sentido para o contexto do problema que se deseja resolver. Como uma busca em árvore atribuindo peso aos esforços para fazer cada jogada em um jogo, por exemplo.

4 A* (A estrela)

O A* é outra algoritmo de busca de menor caminho. Ele recebe uma heurística que ajuda a encontrar o menor caminho a saída. Basicamente criamos uma função h que pode seguir qualquer estratégia. Algumas são melhores que outras é claro. Por exemplo, ao buscar o menor caminho entre duas cidades, levar em conta a distância das cidades a cidade alvo facilita mais o processo do que usar apenas o tamanho das ruas que ligam duas cidades.

```
Cria um vetor dist de distâncias que mede a distância do ponto inicial ao final
Cria um vetor prev de estados anteriores a cada estado v com valores null
Crie uma fila de prioridade Q = { source : 0 }
Enquanto Q não for vazia:
    Pegue um estado u de Q tal qual a distância é a menor possível (dist(u) + h(u))
    Remova u de Q

    Para cada vizinho direto v de u:
        d = dist(u, target) + dist(v, u)
        se d < dist(v), ou seja, é um caminho melhor até v:
            dist(v) = d
            prev(v) = u
            adicione v a Q
Retorna dist, prev
```

Note que quando h(u) = 0 o A* é igual ao Djikstra. O A* é uma optimização do Djikstra, mas o mesmo pode não trazer o melhor resultado. Ele apenas funcionará se:

1. Grafo de custos positivos
2. Heurística Admissível, ou seja h(x) <= dist(x) para todos x.