

```
    check = false;
```

```
}
```

```
if (automatosGerados[i].nome == "backtrack" && !check)
```

```
{
```

```
    indexFix = i -
```

```
    check = true;
```

```
}
```

```
if (check && indexFix > 0)
```

```
{
```

```
    automatosGerados[i] = automatosGerados[automatosGerados.length - indexFix]; // copia o conteudo de indexFix;
```

```
    indexFix--;
```

```
}
```

```
}
```

Inteligência Artificial

# ALGORITMO DE PROFOUNDIDADE PARA VIAGEM

**Equipe:** Eduardo Tavares, Guilherme Andrade,  
Nadianne Galvão, Rafael Rezende,  
Valdir Santana & Verenilson da Silva

```
string log = "";
```

```
foreach (Automaton aut in automatosGerados)
```



```
    check = false;
```

```
}
```

```
if (automatosGerados[i].nome != "backtrack")
```

```
{
```

```
    indexFix = i - 1;
```

```
    check = true;
```

```
}
```

```
if (check && indexFix > 0)
```

```
{
```

```
    automatosGerados[i] = novoAutomaton(automatosGerados[indexFix]); // caminho[indexFix];
```

```
    ix--;
```



**Implementação realizada em C# usando a engine Unity para visualização e interação com o usuário.**



# OBJETIVO:

Encontrar um caminho de viagem entre duas cidades em um conjunto de autômatos.

terminal da unity pra ver se ta tudo certo

# AUTÔMATOS:

Representa as conexões possíveis entre as localidades e permite determinar a sequência de transições necessárias para alcançar a localidade de destino a partir da localidade de origem.

```
    check = false;
```

```
}
```

```
if (automatosGerados[i].nome != "backtrack")
```

```
{
```

```
    ind
```

```
    che
```

```
}
```

```
if (che
```

```
{
```

```
    aut
```

```
    ind
```

```
}
```

```
/ aqui sem
```

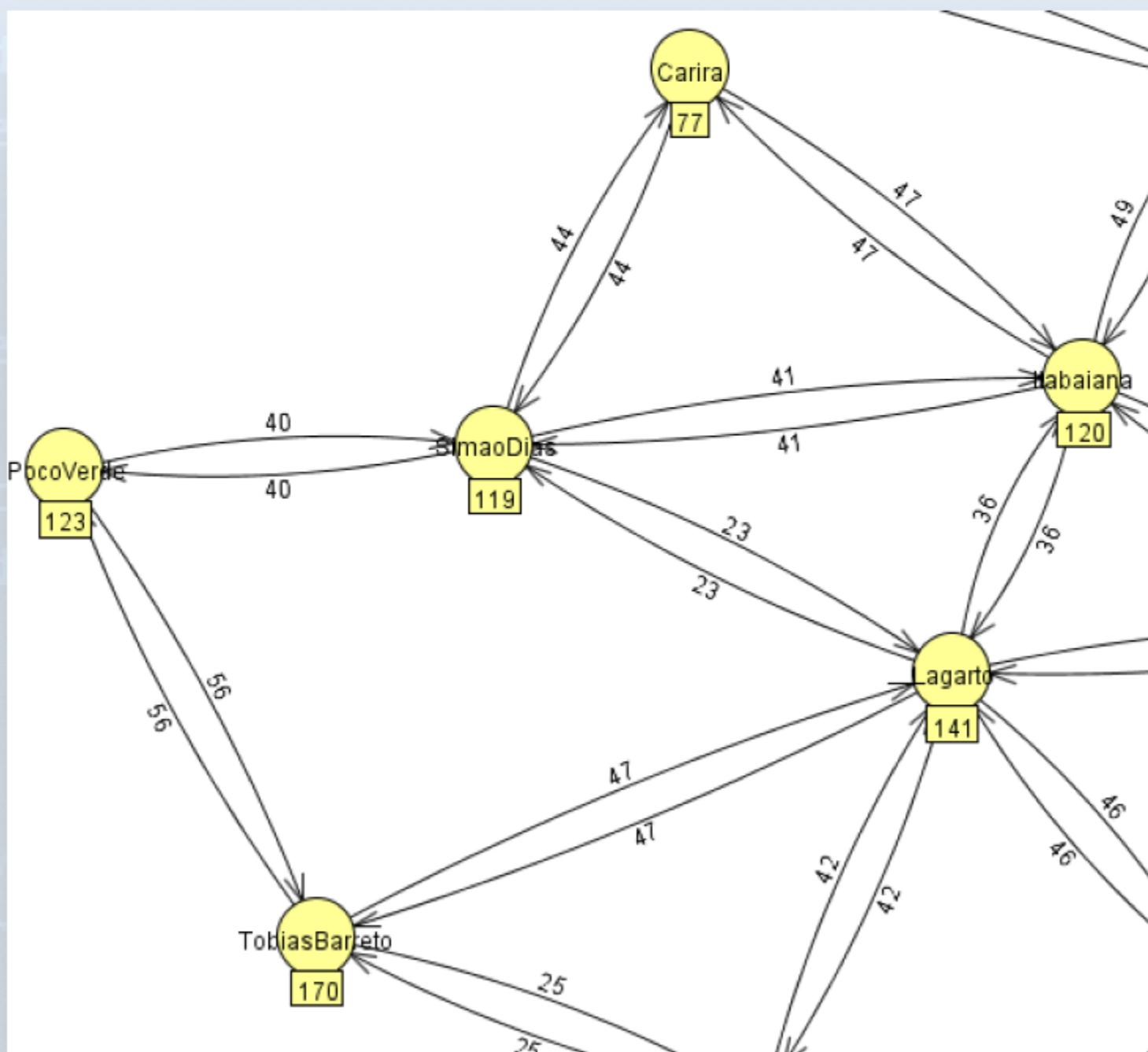
```
if (debugPr
```

```
{
```

```
    string log = "
```

```
foreach (Automaton aut in automatosGerados)
```

# AUTÔMATOS:



```
<state id="0" name="SimaoDias">#13;
  <x>369.755</x>#13;
  <y>396.0</y>#13;
  <label>119</label>#13;
</state>#13;
```

```
<transition>#13;
  <from>0</from>#13;
  <to>5</to>#13;
  <read>44</read>#13;
</transition>#13;
```

## ALGORITMO DE PROFOUNDIDADE PARA VIAGEM

```
    check = false;
}
if (automatosGerados[0] == null)
{
    indexFix = i - 2;
    check = true;
}

if (check && indexFix >= 0)
{
    automatosGerados[indexFix] = null;
    indexFix--;
}
}

// aqui serve só pra printar
if (debugPrint)
{
    string log = "";
```

```
namespace AutomatonNameSpace
{
    using System.Collections.Generic;
    public class Automaton
    {
        public int id;
        public float x, y, label;
        public string nome;
        public List<int[]> transition;
        public bool visitado;
        public Automaton(int id, string nome, float x, float label) //cria um novo automato
        {
            this.id = id;
            this.nome = nome;
            this.x = x;
            this.y = y;
            this.label = label;
            transition = new();
            visitado = false;
        }
        public Automaton(Automaton automaton) //recebe uma classe dela mesma para copiar
        {
            id = automaton.id;
            nome = automaton.nome;
            x = automaton.x;
            y = automaton.y;
            label = automaton.label;
            transition = automaton.transition;
            visitado = automaton.visitado;
        }
    }
}
```

# ALGORITMO

## de busca em profundidade

Explora todos os vértices de um grafo seguindo uma ramificação específica até alcançar o objetivo ou até não haver mais vértices a explorar.



ALGORITMO DE  
PROFOUNDIDADE  
PARA VIAGEM

```
    check = false;
```

```
}
```

```
if (automatosGerados[i].nome != "backtrack")
```

```
{
```

```
    indexFix = i - 2;
```

```
    check = true;
```

É aplicado quando a busca em profundidade atinge um ponto em que não é possível avançar para nenhum estado não visitado a partir do estado atual.

```
}
```

```
// aqui serve só pra printar no terminal da unity pra ver se ta tudo certo
```

```
if (debugPrint)
```

```
{
```

```
    string log = "";
```



# LER ARQUIVO

```
check = false;
}

if (a
{
    i
    c
}
}

if (c
{
    a
    i
}
}

if (debug
{
    string log = "";
    foreach (Automaton aut in automatosGerados)
    {
        log += "Automato " + aut.id + " - ";
        foreach (Transition tran in aut.transitions)
        {
            log += "Transição " + tran.id + " - ";
            log += "Estado de Origem: " + tran.from + " - ";
            log += "Estado de Destino: " + tran.to + " - ";
            log += "Leitura: " + tran.read + " - ";
            log += "Peso: " + tran.weight + "\n";
        }
    }
    Console.WriteLine(log);
}
}

public List<Automaton> automatos = new();
readonly XmlDocument xmlDoc = new();
public void LerArquivoXML(string path){
    //aqui o xml abre o "caminho/diretorio" do arquivo xml
    xmlDoc.Load(path);
    //aqui no estados pega TODAS as tags "state" e guarda
    XmlNodeList estados = xmlDoc.SelectNodes("//state");
    CultureInfo culture = CultureInfo.InvariantCulture;
    //nesse for, para cada tag "state", cria uma entidade "Automato" mas sem transições
    foreach (XmlNode estado in estados) {
        automatos.Add(new Automaton(
            int.TryParse(estado.Attributes["id"].Value, out int parseId) ? parseId : 0,
            estado.Attributes["name"].Value,
            float.TryParse(estado.SelectSingleNode("x").InnerText, NumberStyles.Float, culture, out float x) ? (float)Math.Round(x, 2) : 0,
            float.TryParse(estado.SelectSingleNode("y").InnerText, NumberStyles.Float, culture, out float y) ? (float)Math.Round(y, 2) : 0,
            float.TryParse(estado.SelectSingleNode("label").InnerText, NumberStyles.Float, culture, out float label) ? (float)Math.Round(label, 2) : 0
        ));
    }

    XmlNodeList transition = xmlDoc.SelectNodes("//transition");
    foreach(XmlNode tran in transition)
    {
        foreach(Automaton auto in automatos )
        {
            if (auto.id == (int.TryParse(tran.SelectSingleNode("from").InnerText,out int f) ? f : 0))
            {
                auto.transition.Add(new int[] { int.Parse(tran.SelectSingleNode("to").InnerText), int.Parse(tran.SelectSingleNode("read").InnerText)} );
            }
        }
    }
}

//aqui se
if (debug
{
    string log = "";
    foreach (Automaton aut in automatosGerados)
    {
        log += "Automato " + aut.id + " - ";
        foreach (Transition tran in aut.transitions)
        {
            log += "Transição " + tran.id + " - ";
            log += "Estado de Origem: " + tran.from + " - ";
            log += "Estado de Destino: " + tran.to + " - ";
            log += "Leitura: " + tran.read + " - ";
            log += "Peso: " + tran.weight + "\n";
        }
    }
    Console.WriteLine(log);
}
}
```

## ALGORITMO DE PROFUNDIDADE PARA VIAGEM

```

    if(automatosGerados[i].nome != "backtrack")
    {
        check = false;
    }
    if (automatos == null)
    {
        indexFix = 0;
        check = true;
    }
    if (check && i < automatos.Count)
    {
        automatos[i].visitado = false;
        indexFix++;
    }
    //aqui serve só para debug
    if (debugPrint)
    {
        string log = "i = " + i + " | ";
        log += "origem = " + origem + " | ";
        log += "destino = " + destino + " | ";
        log += "automatos = " + automatos + " | ";
        log += "main = " + main + " | ";
        log += "automatosGerados = " + automatosGerados + " | ";
        log += "check = " + check + " | ";
        log += "indexFix = " + indexFix + " | ";
        log += "automatoStart = " + automatoStart + " | ";
        log += "automatoStart.visitado = " + automatoStart.visitado + " | ";
        log += "automatosGerados.Add(new Automaton(automatoStart));";
        ProfundidadeLookForWay(new Automaton(automatoStart));
    }
}

```

```

 1 referência
public class Profundidade
{
    public string origem, destino;
    public bool chegou;
    List<Automaton> automatos = new();
    Automaton automatoStart;
    public List<Automaton> automatosGerados;
    public Main main;
    1 referência
    public Profundidade(string origem, string destino, List<Automaton> automatos, Main main)
    {
        this.origem = origem;
        this.destino = destino;
        this.automatos = automatos;
        this.main = main;
        automatosGerados = new List<Automaton> ();
    }
    1 referência
    public void RodarAutomato()
    {
        chegou = false;
        foreach(Automaton automaton in automatos)
        {
            if(automaton.nome == origem)
            {
                automatoStart = automaton;
                automatoStart.visitado = true;
                automatosGerados.Add(new Automaton(automaton));
                break;
            }
        }
        ProfundidadeLookForWay(new Automaton(automatoStart));
    }
}

```

## ALGORITMO DE PROFOUNDIDADE PARA VIAGEM

```
public void ProfundidadeLookForWay(Automaton automatoSwitch)
{
    if(automatoSwitch.nome == destino)
    {
        chegou = true;
        return;
    }
    foreach(int[] tran in automatoSwitch.transition)
    {
        if(chegou)
        {
            break;
        }
        foreach(Automaton automato in automatos)
        {
            if (chegou)
            {
                break;
            }
            if (tran[0] == automato.id && automato.visitado == false)
            {
                automato.visitado = true;
                automatosGerados.Add(new Automaton(automato));
                ProfundidadeLookForWay(new Automaton(automato));
            }
        }
        if (chegou == false)
        {
            automatoSwitch.nome = "backtrack";
            automatosGerados.Add(new Automaton(automatoSwitch));
        }
    }
}
```

ALGORITMO DE  
PROFOUNDIDADE  
PARA VIAGEM

```
[  
    if (automatosGerados[i].nome != "backtrack")  
        check = false;  
    }  
}  
if (  
{  
    ProfundidadeLookForWay(new Automaton(automatoStart));  
    bool checkBackTrack = false;  
    for(int i = 0, indexFix=0; i < automatosGerados.Count; i++)  
    {  
        if (automatosGerados[i].nome != "backtrack")  
        {  
            checkBackTrack = false;  
        }  
        if (automatosGerados[i].nome == "backtrack" && !checkBackTrack)  
        {  
            indexFix = i - 2;  
            checkBackTrack = true;  
        }  
        if (checkBackTrack && indexFix >= 0)  
        {  
            automatosGerados[i] = new Automaton(automatosGerados[indexFix]); // caminho[indexFix];  
            indexFix--;  
        }  
    }  
}  
if (de  
[  
    string log = "";
```