Universidade do Estado do Rio de Janeiro
Departamento de Informática e Ciência da Computação
Instituto de Matemática e Estatística

Otimização em Grafos

Profª. Igor Machado Coelho

Implementação dos Algoritmos em Grafos

apresentados em sala de aula

Aluno: Eduardo Fernandes Dias Martins
Matrícula: 201410336611

Implementação dos algoritmos em grafos vistos em sala de aula.

Linguagem de Programação Utilizada: **Python versão 3.7.3**

Repositório para Controle de Versão: **GitHub - Eduardo Martins**

Exemplo do Formato do Grafoutilizado:

```
{
   "nome":"GRAFO_ALEATORIO_GRUPO_EDUARDO",
   "vertices":[
      "1",
      "2",
      "3"
   ],
   "arestas":[
      [
         "1",
         "2"
      ],
      [
         "2",
         "3"
      ]
   ]
}
```

CRIAR GRAFOS DE TESTE: m=?, n=5, n=6, n=7, n=8, n=9, n=10, n=20, n=50, n=100, n=200, n=500, n=1000.

Relatório

Os experimentos realizados neste relatório foram executados em um computador com as seguintes especificações:

- **Processador**: Intel Core i7 7700HQ 7ª geração
- **Memória RAM**: 16GB DDR4 SDRAM
- **Placa de Video**: NVIDIA GeForce GTX1050Ti 4GB GDDR5
- **Sistema Operacional**: Windows 10 Home

Os algoritmos implementados foram baseados nos slides *5, 6, 9, 10, 11, 12, 13, 17, 26, 27, 57 e 62* apresentados na disciplina *Otimização em Grafos* no período **2019.1**

A unidade de medida utilizada no tempo de execução dos algoritmos citados é microsegundo e os grafos utilizados para os testes possuem como número de vértices(*4, 8, 16, 32, 64, 128, 256, 512, 1024 e 2048 respectivamente*).

As estruturas utilizadas para representar computacionalmente um grafo são:

- **Lista de Adjacências**: Para cada vértice do grafo, uma lista de todos os outros vértices com os quais ele tem uma aresta
- **Matriz de Adjacências**: Dado um grafo G com n vértices, podemos representá-lo em uma matriz n x n A(G)=[aij]

Buscas em Grafos

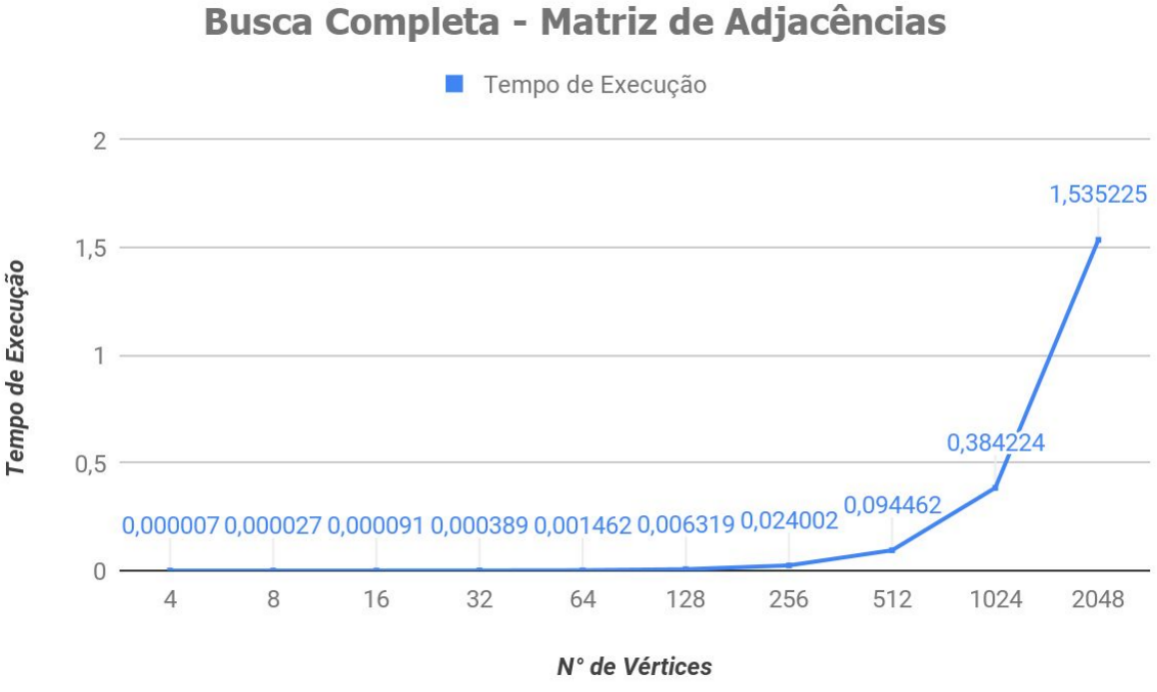
A busca visa resolver um problema básico, dado um grafo G deseja-se obter um processo de como percorrer pelos vértices e arestas do mesmo. Se G é uma árvore, existem três tipos de percursos:

- *Percurso Pre-Ordem*
- *Percurso Em-Ordem*
- *Percurso Pos-Ordem*

Seja **G** um grafo conexo em que todos os seus vértices se encontram desmarcados. No passo inicial, marca-se um vértice arbitrariamente escolhido. No passo geral, seleciona-se algum vértice *v* que esteja marcado e seja incidente a alguma aresta **(v, w)** ainda não selecionada. A aresta **(v, w)** torna-se então selecionada e o vértice *w* marcado. O processo termina quando todas as arestas de G tiverem sido selecionadas. Este tipo de caminhamento é denominado busca no grafo G (*Teoria Computacional de Grafos, Jayme Luiz Szwarcfiter, Fabiano S. Oliveira e Paulo E. D. Pinto*)

Foram apresentadas nos *slides 5 e 6* algoritmos de busca simples que visam resolver o problema enunciado anteriormente. Segue abaixo um comparativo do tempo de execução da *Busca Completa*(Slide 6) implementada com *Lista de Adjancências* e *Matriz de Adjacências*.

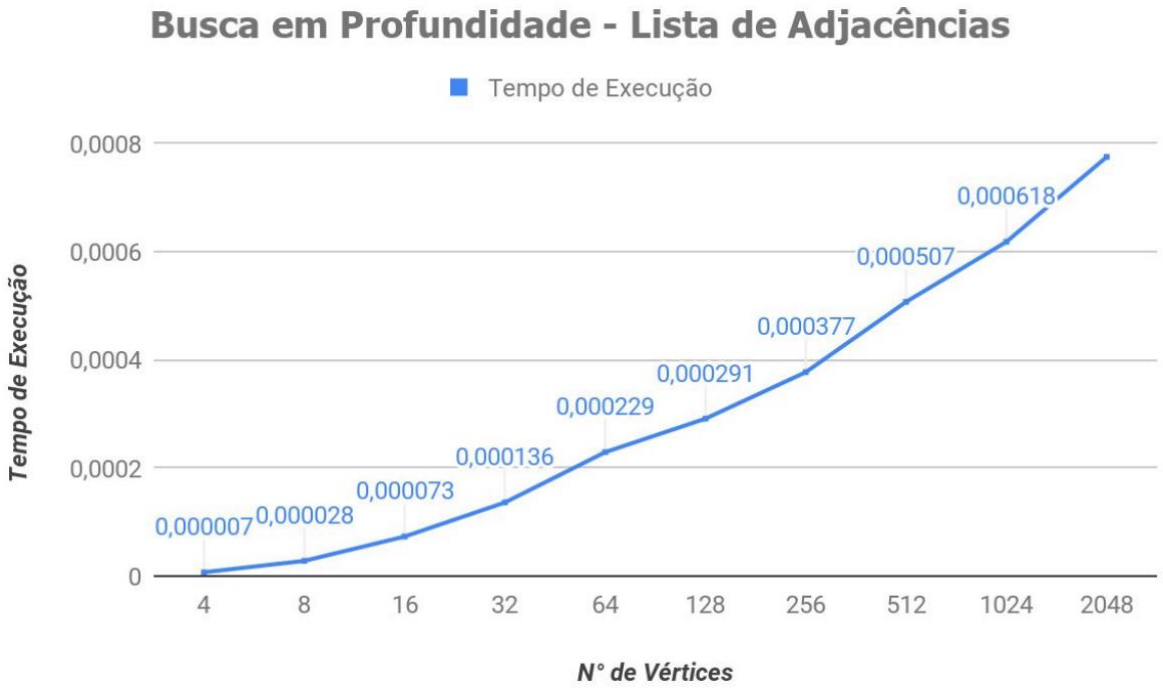

Busca Completa - Lista de Adjacências
Tempo de Execução
Tempo de Execução
0,00125
0,001
0,00075
0,0005
0,00025
0
0,0000050,0000060,0000080,0000150,0000340,0000520,0001090,0002340,0004380,001021
4
8
16
32
64
128
256
512
1024
2048
N° de Vértices

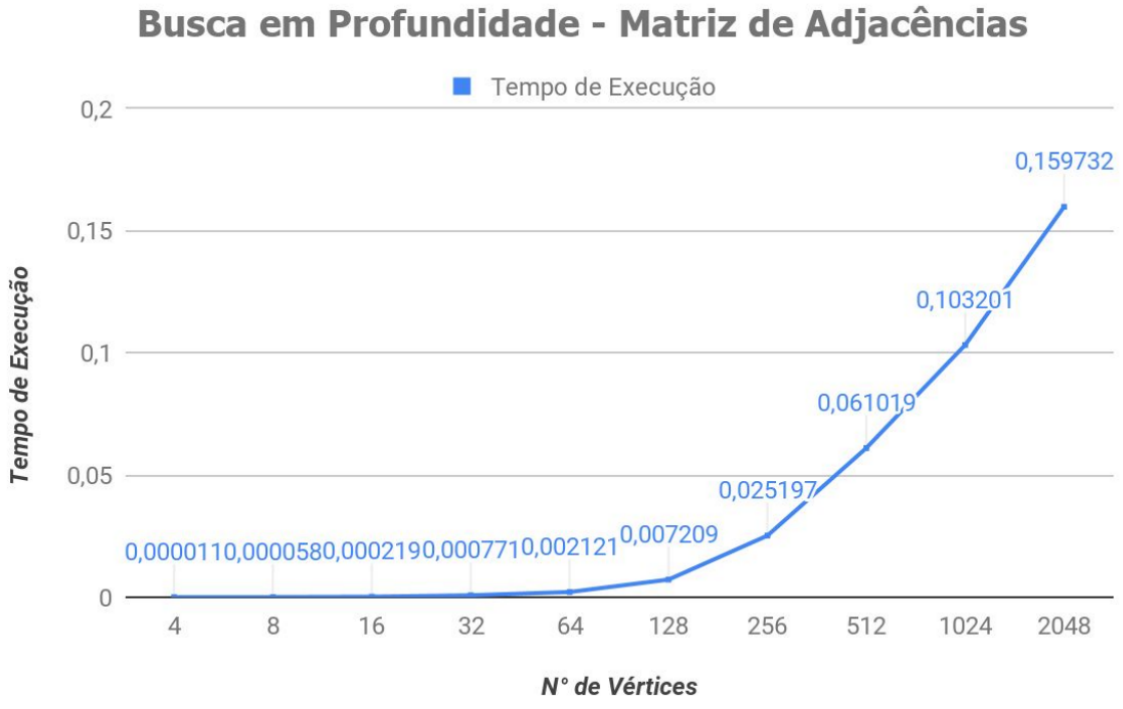

Busca Completa - Matriz de Adjacências
Tempo de Execução
Tempo de Execução
2
1,5
1
0,5
0
0,000007 0,000027 0,000091 0,000389 0,001462 0,006319 0,024002 0,094462 0,384224 1,535225
4
8
16
32
64
128
256
512
1024
2048
N° de Vértices

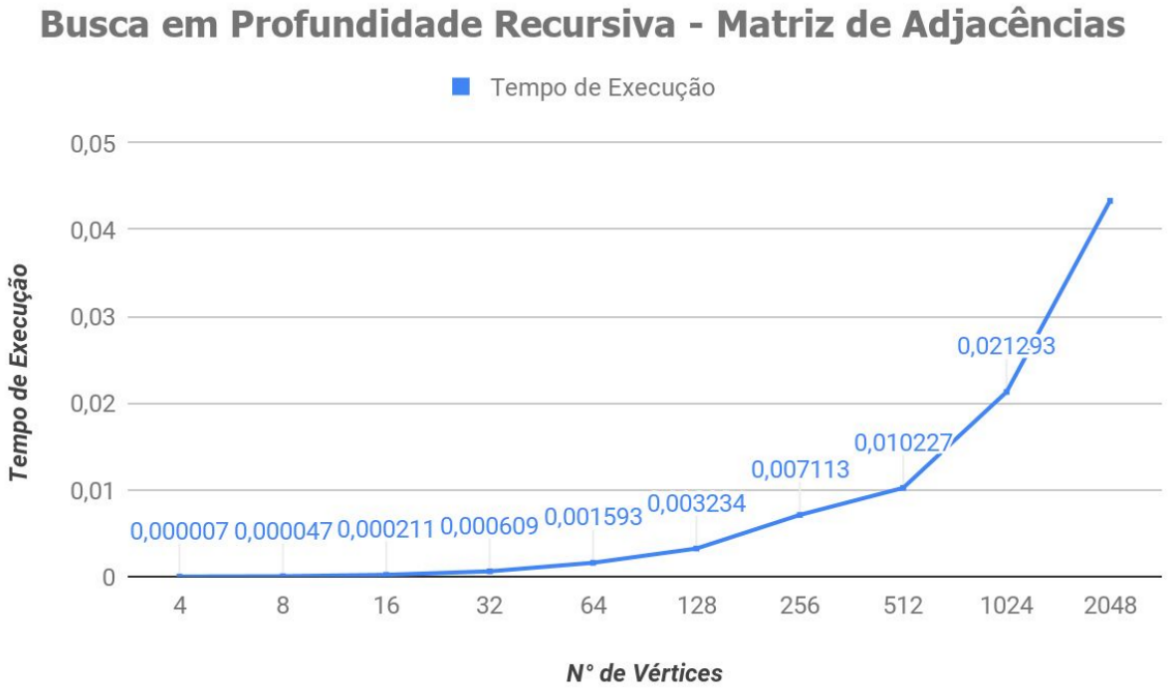
Busca em Profundidade

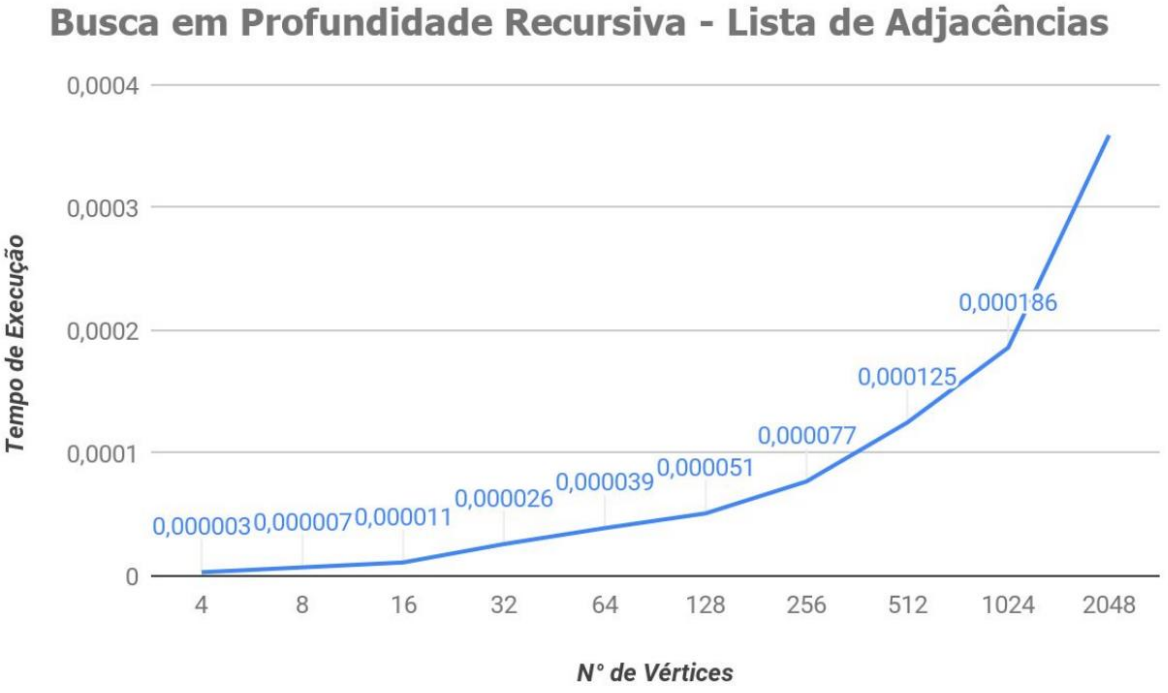
Uma busca é dita em *profundidade* quando o critério de escolha de vértice marcado obedecer “dentre todos os vértices marcados e incidentes a alguma aresta ainda não explorada, escolher aquele mais recentemente alcançado na busca”. Com isso, a escolha do vértice marcado torna-se única e sem ambiguidade.

Abaixo, um comparativo do tempo de execução da Busca em Profundidade(*Slide 26 e 27*) implementada com *Lista de Adjancências* e *Matriz de Adjacências*, a implementação utiliza uma **Pilha**(*Estrutura de dados auxiliar*) em sua implementação e também possui uma *versão recursiva*.





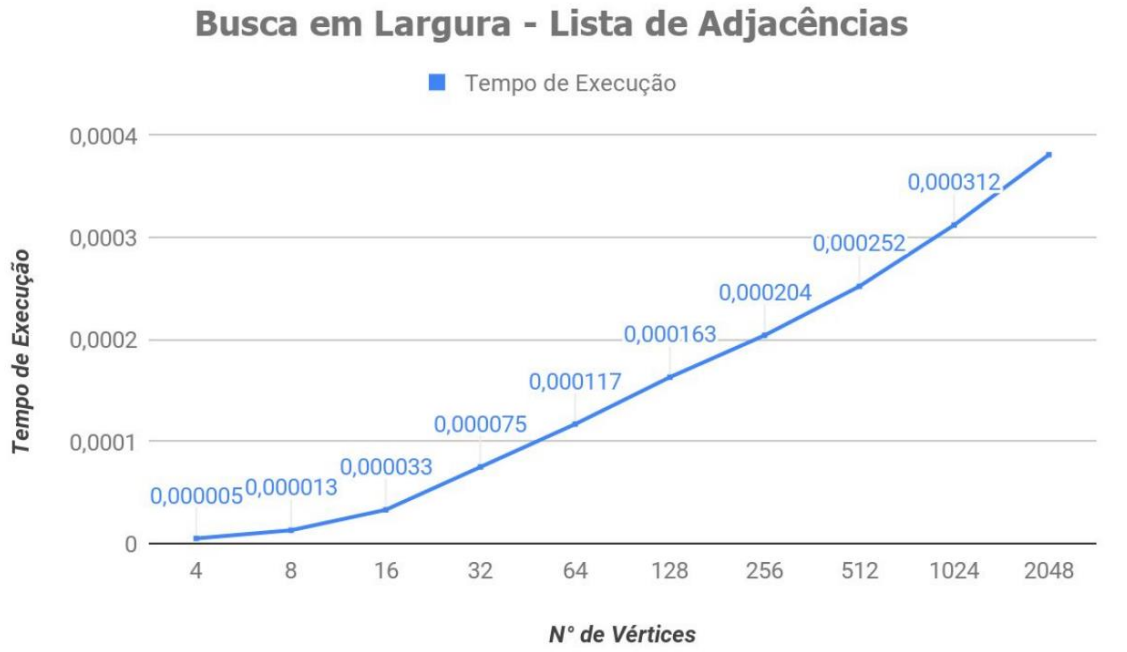

Busca em Profundidade Recursiva - Matriz de Adjacências
Tempo de Execução
0,05
0,04
0,03
0,02
0,01
0
Tempo de Execução
0,000007
0,000047
0,000211
0,000609
0,001593
0,003234
0,007113
0,010227
0,021293
4
8
16
32
64
128
256
512
1024
2048
N° de Vértices

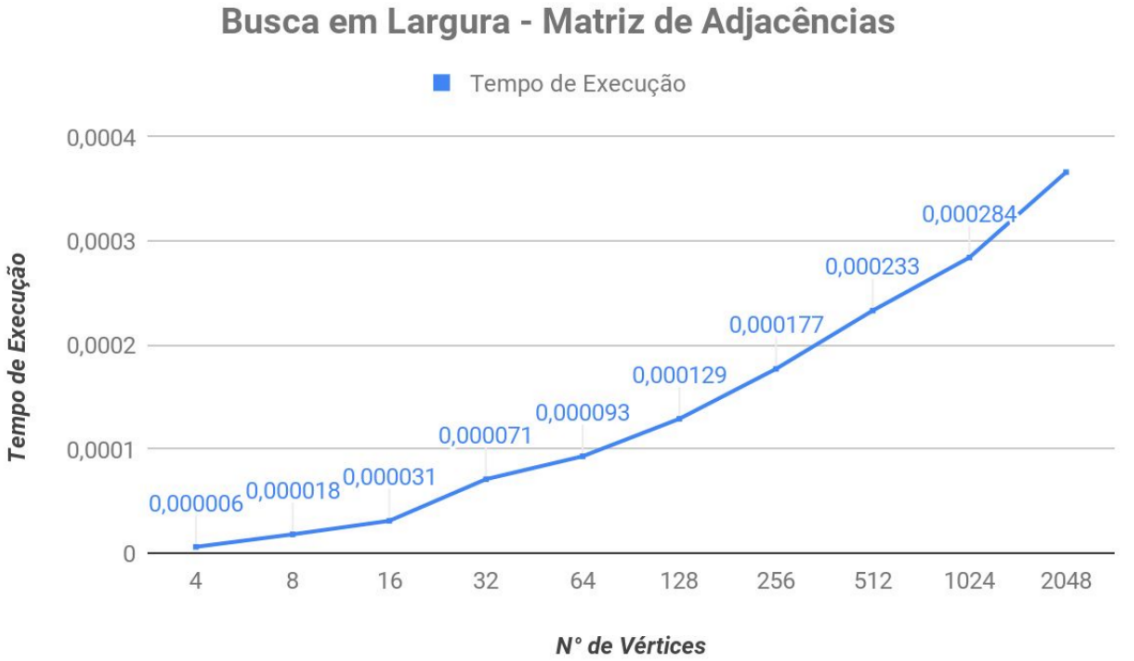

Busca em Profundidade Recursiva - Lista de Adjacências
0,0004
0,0003
0,0002
0,0001
0
Tempo de Execução
0,000003
0,000007
0,000011
0,000026
0,000039
0,000051
0,000077
0,000125
0,000186
4
8
16
32
64
128
256
512
1024
2048
N° de Vértices

Busca em Largura

Uma busca é dita em *largura* quando o critério de escolha de vértice marcado obedece “dentre todos os vértices marcados e incidentes a alguma aresta ainda não explorada, escolher aquele menos recentemente alcançado na busca”. A implementação utiliza uma **Fila**(*Estrutura de dados auxiliar*) em sua implementação e possui semelhança com a busca em profundidade em relação ao critério de escolha de arestas, que neste caso também é arbitrário.

Abaixo, um comparativo do tempo de execução da *Busca em Largura*(*Slide 57*) implementada com *Lista de Adjancências* e *Matriz de Adjacências*.



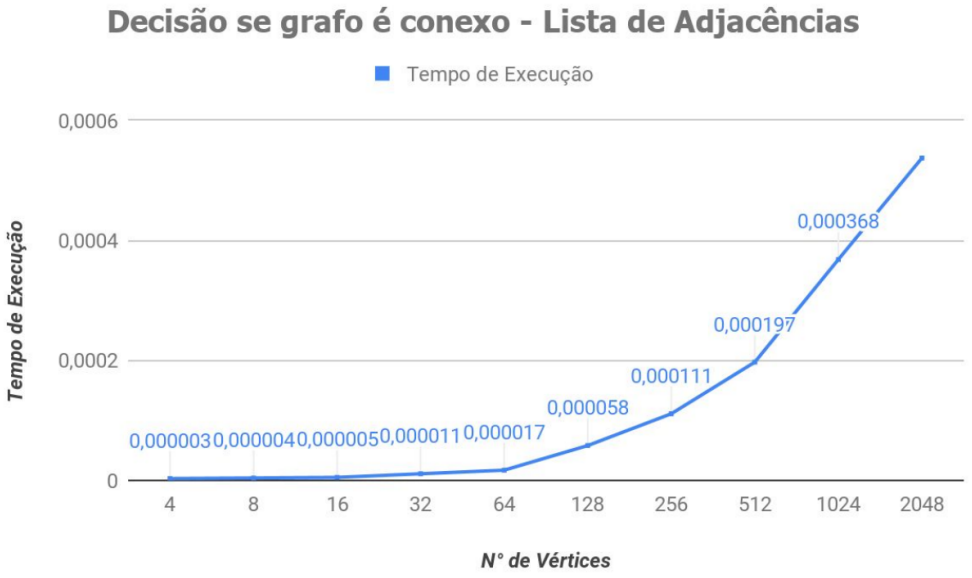


Algoritmos em Grafos

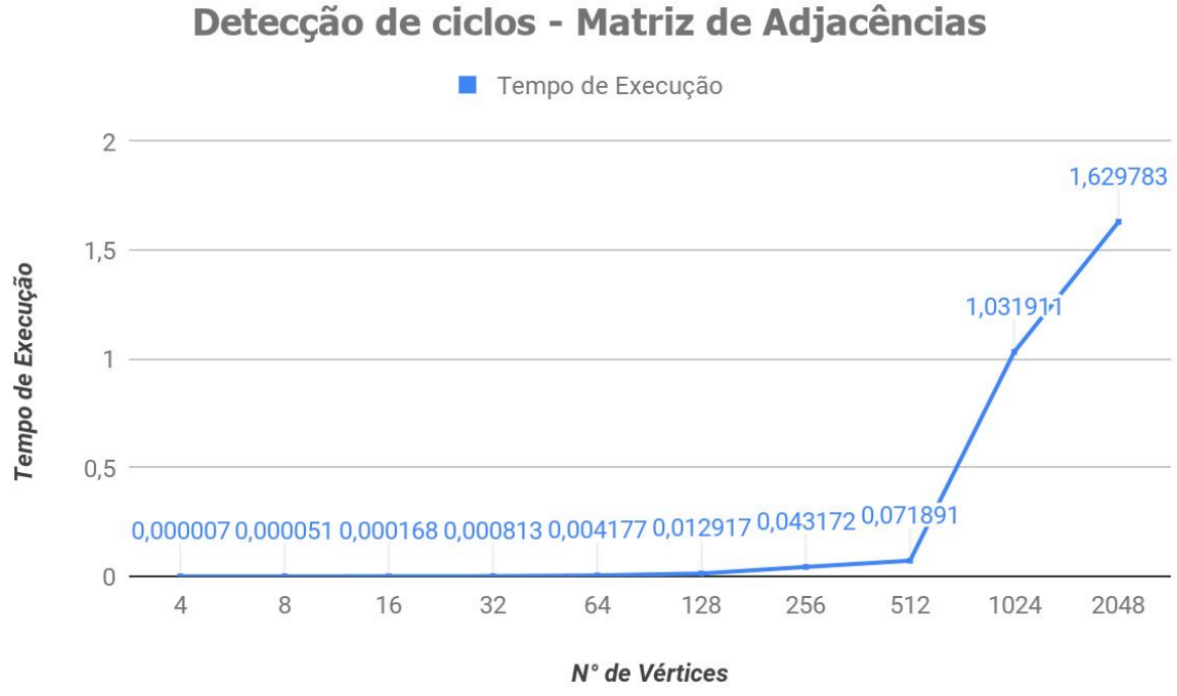
Seguindo com outras técnicas, um grafo pode ser visualizado através de uma representação na qual seus vértices correspondem a pontos distintos do plano em posições arbitrárias, enquanto que cada aresta **(v, w)** é associada a uma linha arbitrária unindo os pontos correspondentes. Com isso, houve a necessidade de verificar algumas características de um determinado grafo como por exemplo:

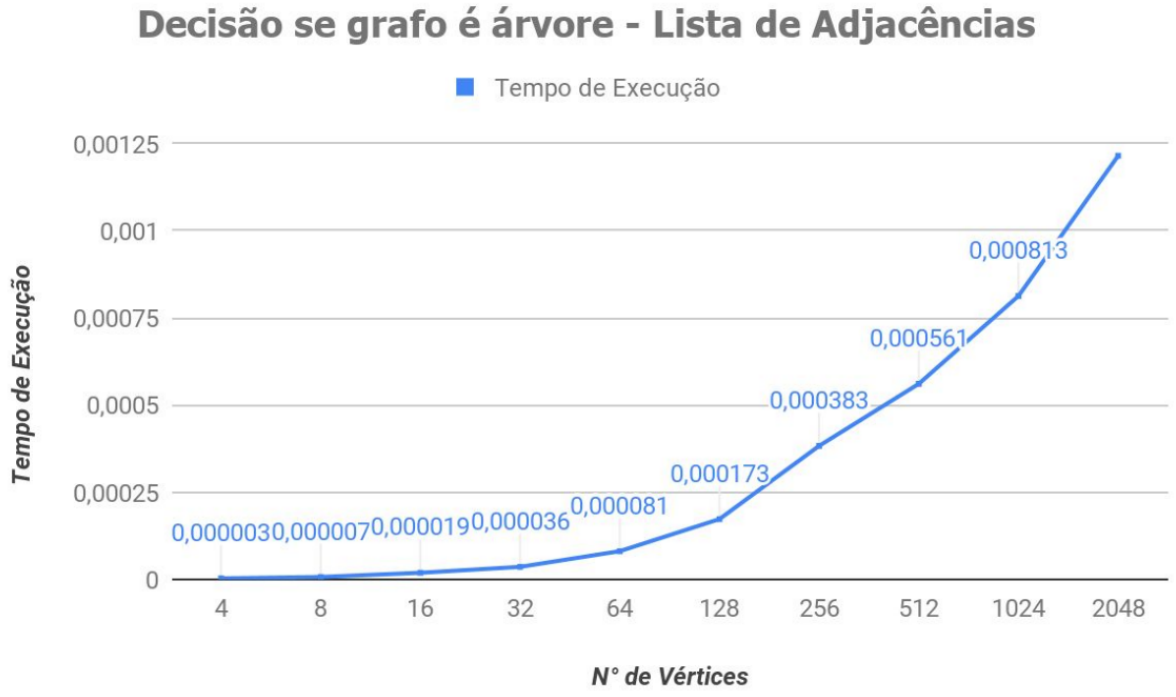
- *Seja G um grafo, G é conexo ?*
- *Seja G um grafo, G é cíclico ?*
- *Seja G um grafo, G é uma árvore ?*
- *Dado um grafo G (árvore), como obter sua floresta geradora ?*

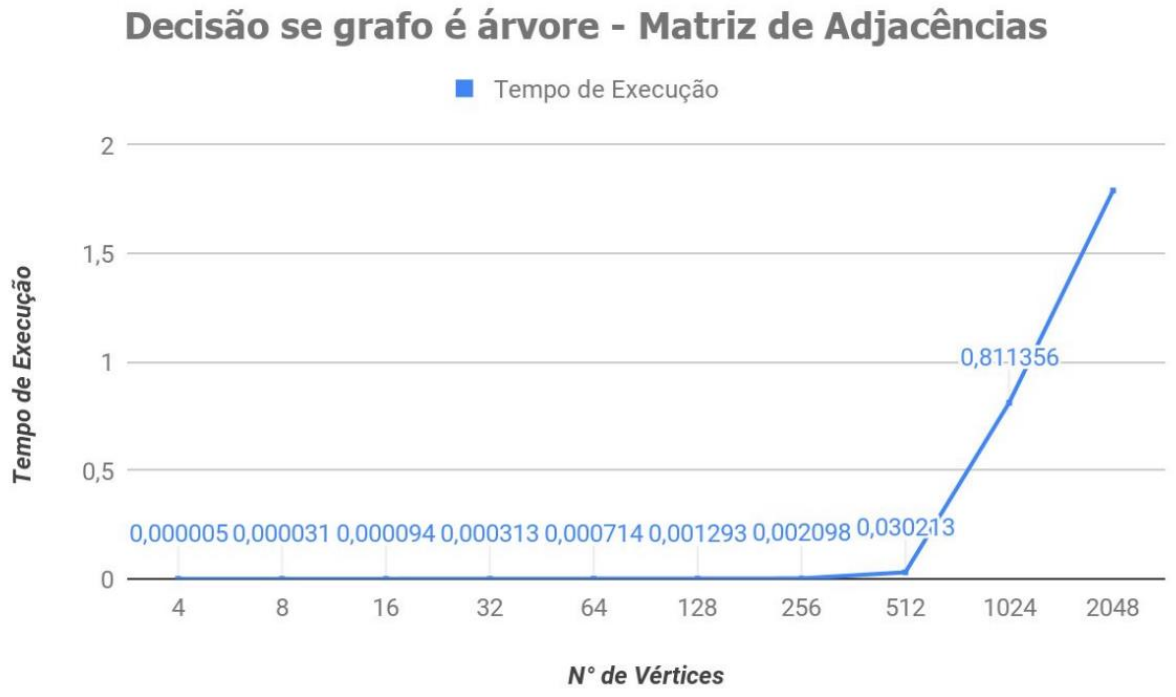
Para isto, foram apresentados nos *slides 9, 10, 11, 12, 13 e 17* algoritmos para resolução das questões citadas acima. Abaixo, um comparativo do tempo de execução dos algoritmos citados anteriormente, implementados com *Lista de Adjancências* e *Matriz de Adjacências*.




Detecção de ciclos - Lista de Adjacências
Tempo de Execução
0,00125
0,001
0,00075
0,0005
0,00025
0
Tempo de Execução
0,0000030,0000040,0000110,0000170,000041
0,000088
0,000123
0,000347
0,000781
4
8
16
32
64
128
256
512
1024
2048
N° de Vértices


Detecção de ciclos - Matriz de Adjacências
Tempo de Execução
2
1,5
1
0,5
0
Tempo de Execução
0,000007 0,000051 0,000168 0,000813 0,004177 0,012917 0,043172 0,071891
1,031911
1,629783
4
8
16
32
64
128
256
512
1024
2048
N° de Vértices


Decisão se grafo é árvore - Lista de Adjacências
Tempo de Execução
0,00125
0,001
0,00075
0,0005
0,00025
0
0,0000030,000007
0,000019
0,000036
0,000081
0,000173
0,000383
0,000561
0,000813
4
8
16
32
64
128
256
512
1024
2048
Tempo de Execução
N° de Vértices


Decisão se grafo é árvore - Matriz de Adjacências
Tempo de Execução
2
1,5
1
0,5
0
0,000005
0,000031
0,000094
0,000313
0,000714
0,001293
0,002098
0,030213
0,811356
4
8
16
32
64
128
256
512
1024
2048
Tempo de Execução
N° de Vértices


Floresta Geradora - Lista de Adjacências
Tempo de Execução
2
1,5
1
0,5
0
Tempo de Execução
0,000043 0,000127 0,000517 0,002616 0,006899
0,177521
0,321289
0,682199
1,051078
4
8
16
32
64
128
256
512
1024
2048
N° de Vértices


Floresta Geradora - Matriz de Adjacências
Tempo de Execução
3
2
1
0
Tempo de Execução
0,000017 0,000098 0,000391 0,001019 0,006117 0,014201
0,071017
0,310442
0,981707
4
8
16
32
64
128
256
512
1024
2048
N° de Vértices

Fluxo Máximo

Uma *rede* é um *digrafo* **D**(V,E) em que a cada aresta e está associado um número real positivo c(e) denominado capacidade da aresta e. Suponha que **D** possua dois vértices especiais e distintos *s*, *t* chamados de *origem* e *destino* com respectivas propriedades:

- **s** *é uma fonte que alcança todos os vértices*
- **t** *é um sumidouro alcançado também por todos*

Um fluxo *f* de *s* à *t* em **D** é uma função que a cada aresta e associa um número real não negativo *f(e)* satisfazendo às seguintes condições:

1. $0 \leq f(e) \leq c(e)$, para toda aresta pertencente ao conjunto de arestas de D
2. $\sum f(w1, v) = \sum f(v, w2)$, *para todo vértice* $v \neq s, t$

No repositório do GIT, onde encontram-se todas as implementações citadas neste relatório, possui a implementação do *Algoritmo de Fluxo Máximo – Ford-Fulkerson* apresentado nos slides da disciplina de *Otimização em Grafos* e também no livro *Teoria Computacional de Grafos* (*Jayme Luiz Szwarcfiter, Fabiano S. Oliveira e Paulo E. D. Pinto*). Não houve a coleta do tempo de execução do algoritmo.

Conclusão

A partir de um comparativo do tempo de execução, conclui-se que implementações que utilizam uma lista de adjacências como representação de um grafo qualquer são mais performáticas(dentro das condições enunciadas no inicio deste documento) em modo geral, tendo em vista que implementações que utilizam matriz de adjacências como forma de representação possuem peculiaridades que podem tornar-la performaticas em alguns tipos específicos de problemas.

Contudo, assintoticamente a complexidade de tempo de ambas representações para estes algoritmos são similares e o que difere de fato é a complexidade de espaço(Espaço em memória que cada representação do grafo utiliza).

Referências Bibliográficas

- *Grafos e algoritmos computacionais - SZWARCFITER, Jayme Luiz*
- *Teoria Computacional de Grafos - SZWARCFITER, Jayme Luiz*, Fabiano S. Oliveira e Paulo E. D. Pinto
- *Estruturas de Dados e Seus Algoritmos - SZWARCFITER, Jayme Luiz*, Lilian Markenzon