EP-1 Problema do Caixeiro Viajante.

Gerado por Doxygen 1.8.17

1 EP-1 Problema do Caixeiro Viajante

1.1 Introdução

Este documento serve a, ao menos, quatro propósitos:

1. Apresentar a implementação de uma solução gulosa para um problema;
2. Especificar a estrutura de entrega dos Exercícios-Programa (EPs);
3. Padronizar a representação de grafos nos EPs;
4. Sugerir algumas ferramentas auxiliares para documentação e visualização.

O programa documentado aqui, compilado e executado sem modificações, aplica uma estratégia gulosa sobre uma variação do problema do Caixeiro Viajante. Nesta, o caixeiro tem um mapa de uma região. O mapa apresenta cidades e estradas entre as cidades. O caixeiro viajante deseja visitar cada cidade apenas uma vez e terminar na cidade em que começou. O problema do caixeiro é encontrar essa rota.

Modelando o mapa com um grafo que representa o mapa, cidades são representadas por vértices e estradas são representadas por arestas. A rota com a característica desejada pelo caixeiro é chamada, em teoria de grafos e em computação, ciclo (ou circuito) hamiltoniano.

A estratégia gulosa implementada inicia por um conjunto vazio de arestas selecionadas, testa se o conjunto de arestas *é solução*. Caso não seja, avalia, na ordem dada pela lista de arestas, se uma aresta *é aceitável*. Caso seja, seleciona essa aresta para fazer parte da solução e volta a avaliar se o conjunto de arestas é solução, até que o conjunto seja solução, ou não haja mais arestas a testar. Os detalhes podem ser vistos no código-fonte.

O programa é modularizado em arquivos que contém definições para diferentes aspectos. O arquivo `grafo.{c,h}` contém as definições das estruturas para representar grafos e das funções, até agora escritas, para lidar com essas estruturas. O arquivo `ep.h` contém a especificação que deve ser implementada para que a solução proposta possa ser avaliada. O arquivo `ep.c` contém um exemplo de implementação da especificação (usando a estratégia gulosa). O arquivo `testador.c` contém um esboço fiel do programa que será usado para testar a solução proposta.

1.1.1 Avisos anteriores

Caros,

O tema do EP é o Problema do Caixeiro Viajante (Traveling Salesman Problem, ou TSP).

Para o EP, serão usados conceitos básicos de teoria de grafos como vértices, arestas e lista de incidência.

https://pt.wikipedia.org/wiki/Teoria_dos_grafos

https://en.wikipedia.org/wiki/Graph_theory

Explicações sobre o TSP podem ser encontradas na wikipedia. Somente o entendimento do problema é essencial. Os verbetes contém formulações e algoritmos complicados sobre o problema e propostas de soluções que não são essenciais para a disciplina, mas podem ser úteis para vocês.

https://pt.wikipedia.org/wiki/Problema_do_caixeiro-viajante

https://en.wikipedia.org/wiki/Travelling_salesman_problem

https://en.wikipedia.org/wiki/Hamiltonian_path_problem

Atenciosamente,

Fábio.

1.2 Especificação da entrega

A solução entregue deve estar totalmente contida no arquivo `ep.c`. Envie somente este arquivo através do e-disciplinas.

Lembre-se de ajustar a função `autor()` e o cabeçalho com seu nome e NUSP.

1.3 Exemplo de compilação do programa

```
fabio@fabio-13Z940-G-BK71P1:~/Documentos/IAA-2021/EP-Hamiltoniano/guloso$ gcc grafo.c ep.c testador.c
```

1.4 Exemplo de execução do programa (estratégia gulosa)

```
fabio@fabio-13Z940-G-BK71P1:~/Documentos/IAA-2021/EP-Hamiltoniano/guloso$ ./a.out
número de vértices=10
número de arestas=20
graph TD
0 ---|383| 1
1 ---|886| 2
2 ---|777| 3
3 ---|915| 4
4 ---|793| 5
5 ---|335| 6
6 ---|386| 7
7 ---|492| 8
8 ---|649| 9
9 ---|421| 0
2 ---|690| 7
9 ---|926| 3
0 ---|172| 6
6 ---|368| 1
7 ---|782| 9
0 ---|123| 2
7 ---|929| 5
2 ---|58| 2
9 ---|393| 7
6 ---|42| 1
Fábio Nakano ; 1048221
Achou solução
Encontrou solução.
número de vértices=10
número de arestas=20
graph TD
0 ---|383| 1
1 ---|886| 2
2 ---|777| 3
3 ---|915| 4
4 ---|793| 5
5 ---|335| 6
6 ---|386| 7
7 ---|492| 8
8 ---|649| 9
9 ---|421| 0
2 -.-|690| 7
9 -.-|926| 3
0 -.-|172| 6
6 -.-|368| 1
7 -.-|782| 9
0 -.-|123| 2
7 -.-|929| 5
2 -.-|58| 2
9 -.-|393| 7
6 -.-|42| 1
número de vértices=10
número de arestas=20
graph TD
3 ---|91| 6
5 ---|729| 6
5 ---|545| 2
6 ---|198| 7
4 ---|784| 5
1 ---|925| 5
8 ---|315| 9
4 ---|750| 3
0 ---|873| 6
2 ---|421| 3
1 ---|373| 2
0 ---|229| 1
4 ---|434| 7
7 ---|324| 8
3 ---|124| 7
2 ---|996| 0
4 ---|336| 7
9 ---|370| 0
5 ---|537| 6
3 ---|919| 4
Fábio Nakano ; 1048221
Não achou solução
Não encontrou solução.
fabio@fabio-13Z940-G-BK71P1:~/Documentos/IAA-2021/EP-Hamiltoniano/guloso$
```

nota: Doxygen aceita markdown como página inicial, mas não lida bem nem com HTML nem com figuras dentro do markdown. Por outro lado, lida bem com figuras incluídas nos comentários. Por isso espalhei as figuras pelos comentários.

2 Índice das estruturas de dados

2.1 Estruturas de dados

Lista das estruturas de dados com uma breve descrição:

3 Índice dos ficheiros

3.1 Lista de ficheiros

Lista de todos os ficheiros documentados com uma breve descrição:

4 Documentação da classe

4.1 Referência à estrutura grafo

Estrutura para representação de um grafo.

`#include <grafo.h>`

Campos de Dados

- int N
- int M
- int ∗ A
- int ∗ S

4.1.1 Descrição detalhada

Estrutura para representação de um grafo.

Os vértices são rotulados por números inteiros no intervalo [0:N-1], sem "pulos".

A lista de arestas é uma matriz [M, 3] onde cada linha corresponde a uma aresta. Dada a linha, o primeiro elemento é o rótulo do vértice de origem, o segundo elemento é o rótulo do vértice de destino e o terceiro elemento é um valor associado à aresta, por exemplo, seu "peso". Espera-se que todas as linhas da lista contenham arestas válidas.

No EP, a aresta pode ser percorrida tanto da origem para o destino quanto do destino para a origem. Em outras palavras, o grafo não é dirigido.

A lista de flags de seleção de arestas é uma lista que indica se a aresta contida na lista de arestas foi ou não selecionada. Se a aresta não foi selecionada, o valor armazenado na posição deve ser 0 (zero), se a aresta foi selecionada, o valor deve ser diferente de 0. Esta lista foi acrescentada por conveniência para retornar a resposta da execução de algoritmos.

4.1.2 Documentação dos campos e atributos

4.1.2.1 A `int* A`

ponteiro para lista de arestas

4.1.2.2 M `int M`

Número de arestas

4.1.2.3 N `int N`

Número de vértices

4.1.2.4 S `int* S`

ponteiro para lista de flags de seleção de aresta

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

- grafo.h

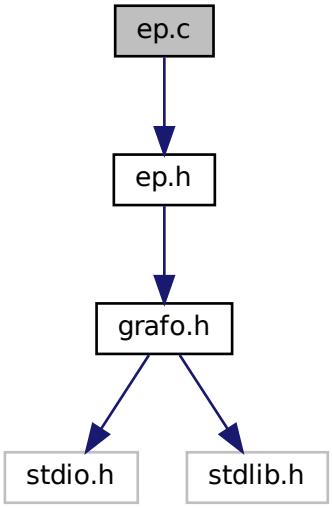
5 Documentação do ficheiro

5.1 Referência ao ficheiro ep.c

Para uso como exemplo, contém uma aplicação da estratégia gulosa para solução do TSP. Para uso como modelo de EP, deve conter sua solução para o EP.

```
#include "ep.h"
```

Diagrama de dependências de inclusão para ep.c:



Funções

- char ∗ autor (void)

  Esta função é importante para que o corretor automático inclua corretamente a autoria do EP. Deve retornar Nome e NUSP separados por ponto-e-virgula, sem pular linhas no meio nem no final.
- int **ehSolucao** (struct grafo ∗G)
- int **ehAceitavel** (struct grafo ∗G, int arestaATestar, int verticeAtual)
- int **aumentaCaminho** (struct grafo ∗G, int arestaAcrescentar, int verticeAtual)
- int **guloso** (struct grafo ∗G, int verticeAtual)
- int iniciaEexecuta (struct grafo ∗G, int verticeInicial)

  Aloca e inicializa variáveis e executa o algoritmo.
- void termina ()

  Libera memória e toma outras ações para limpeza do ambiente de execução.

Variáveis

- int ∗ **grauDoVertice**
- int ∗ **arestaUsada**

5.1.1 Descrição detalhada

Para uso como exemplo, contém uma aplicação da estratégia gulosa para solução do TSP. Para uso como modelo de EP, deve conter sua solução para o EP.

Lembre-se de substituir o nome do autor, abaixo, pelo seu nome e NUSP. (Redundância com o retorno da função autor())

Autor

Fábio Nakano NUSP 1048221

5.1.2 Documentação das funções

5.1.2.1 autor()

```
char* autor (
        void  )
```

Esta função é importante para que o corretor automático inclua corretamente a autoria do EP. Deve retornar Nome e NUSP separados por ponto-e-virgula, sem pular linhas no meio nem no final.

Retorna

Nome e NUSP separados por ponto-e-virgula, sem pular linhas.

5.1.2.2 iniciaEexecuta()

```
int iniciaEexecuta (
        struct grafo * G,
        int verticeInicial )
```

Aloca e inicializa variáveis e executa o algoritmo.

Parâmetros

G	Grafo onde procurar o circuito hamiltoniano e onde, também, armazenar o resultado, como arestas selecionadas, na lista S. Ver grafo.h para detalhes.
verticeInicial	Vértice inicial.

Retorna

retorna 1 caso tenha achado o circuito hamiltoniano e zero caso não tenha achado.

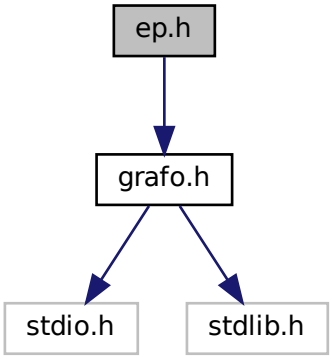
< ATENÇÃO: TEM QUE TER ESTA PARTE PARA A RESPOSTA PODER SER CORRIGIDA!!

5.2 Referência ao ficheiro ep.h

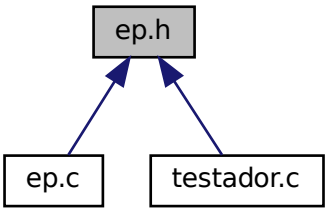
Constantes, estruturas, protótipos de funções essenciais (mandatórias) para o EP.

```
#include "grafo.h"
```

Diagrama de dependências de inclusão para ep.h:



Este grafo mostra quais são os ficheiros que incluem directamente ou indirectamente este ficheiro:



Funções

- char ∗ autor (void)

  Esta função é importante para que o corretor automático inclua corretamente a autoria do EP. Deve retornar Nome e NUSP separados por ponto-e-virgula, sem pular linhas no meio nem no final.
- int iniciaEexecuta (struct grafo ∗G, int verticeInicial)

  Aloca e inicializa variáveis e executa o algoritmo.
- void termina ()

  Libera memória e toma outras ações para limpeza do ambiente de execução.

5.2.1 Descrição detalhada

Constantes, estruturas, protótipos de funções essenciais (mandatórias) para o EP.

Os corpos das funções para o exemplo estão em ep.c. O exemplo implementa uma solução gulosa que tenta encontrar um circuito hamiltoniano em um grafo. O problema é uma variação do Problema do Caixeiro Viajante.

No EP, espera-se que o aluno implemente uma solução por tentativa e erro (backtracking) para esta variação do Problema do Caixeiro Viajante.

Eu queria deixar "o que faz" no header e "como faz" no .c, mas o gerador de documentos (doxygen), quando documenta o header, concatena a documentação documentação do header com a documentação do .c, o que duplica muitas explicações, aumenta o tamanho da documentação, o que desmotiva sua leitura.

Por isso, o que puder ser documentado no header ou no .c será documentado somente no header.

Autor

Fábio Nakano

Data

2021-09-14

5.2.2 Documentação das funções

5.2.2.1 autor() `char* autor ( void )`

Esta função é importante para que o corretor automático inclua corretamente a autoria do EP. Deve retornar Nome e NUSP separados por ponto-e-virgula, sem pular linhas no meio nem no final.

Retorna

Nome e NUSP separados por ponto-e-virgula, sem pular linhas.

5.2.2.2 iniciaEexecuta() `int iniciaEexecuta ( struct grafo * G, int verticeInicial )`

Aloca e inicializa variáveis e executa o algoritmo.

Parâmetros

G	Grafo onde procurar o circuito hamiltoniano e onde, também, armazenar o resultado, como arestas selecionadas, na lista S. Ver grafo.h para detalhes.
verticeInicial	Vértice inicial.

Retorna

retorna 1 caso tenha achado o circuito hamiltoniano e zero caso não tenha achado.

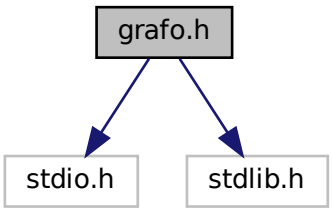
< ATENÇÃO: TEM QUE TER ESTA PARTE PARA A RESPOSTA PODER SER CORRIGIDA!!

5.3 Referência ao ficheiro grafo.h

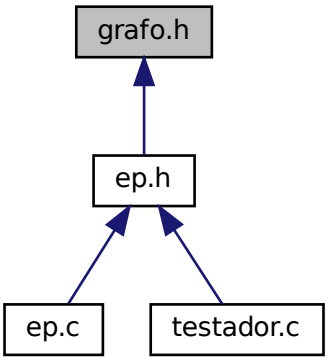
Constantes, estruturas, protótipos de funções para representação, armazenamento e manipulação de grafos.

```
#include <stdio.h>
#include <stdlib.h>
```

Diagrama de dependências de inclusão para grafo.h:



Este grafo mostra quais são os ficheiros que incluem directamente ou indirectamente este ficheiro:



Estruturas de Dados

- struct grafo

  Estrutura para representação de um grafo.

Funções

- struct grafo ∗ criaGrafo (int nV, int nA)

  Aloca estruturas na memória para armazenar um grafo.
- void destroiGrafo (struct grafo ∗g)

  Libera memória ocupada pelo grafo.
- void imprimeGrafo (const struct grafo ∗g)

  Imprime grafo, em formato próprio, na tela.
- void imprimeGrafoMermaid (const struct grafo ∗g, const int ∗destacarAresta)

  Imprime grafo em formato Mermaid, na tela.
- void preencheAleatorio (struct grafo ∗g)

  Preenche a estrutura com um grafo aleatório.
- void trocaRotulo (struct grafo ∗g, int antigo, int novo)

  Troca o rótulo (antigo) de um vértice por outro (novo).
- void preencheHamiltonianoAleatorio (struct grafo ∗g, int mudaOrdem, int mudaRotulo)

  Preenche a estrutura com um circuito Hamiltoniano e completa com arestas geradas aleatoriamente.

5.3.1 Descrição detalhada

Constantes, estruturas, protótipos de funções para representação, armazenamento e manipulação de grafos.

Os corpos das funções estão em grafo.c.

Autor

Fábio Nakano

5.3.2 Documentação das funções

5.3.2.1 criaGrafo()

```
struct grafo* criaGrafo (
        int nV,
        int nA )
```

Aloca estruturas na memória para armazenar um grafo.

Aloca memória para uma estrutura, inicializa a quantidade de vértices e de arestas e aloca a lista de arestas. Esta não é inicializada.

Parâmetros

nV	número de vértices do grafo;
nA	número de arestas do grafo.

Retorna

ponteiro para a estrutura criada.

5.3.2.2 destroiGrafo() `void destroiGrafo ( struct grafo * g )`

Libera memória ocupada pelo grafo.

Libera a memória alocada para a lista de arestas e para a estrutura.

Parâmetros

g	ponteiro para a estrutura que armazena o grafo.

5.3.2.3 imprimeGrafo() `void imprimeGrafo ( const struct grafo * g )`

Imprime grafo, em formato próprio, na tela.

Parâmetros

g	ponteiro para grafo a imprimir

5.3.2.4 imprimeGrafoMermaid() `void imprimeGrafoMermaid ( const struct grafo * g, const int * destacarAresta )`

Imprime grafo em formato Mermaid, na tela.

Mermaid (https://mermaid-js.github.io/) é um gerador de diagramas. Tem um renderizador on-line em https://github.com/mermaid-js/mermaid-live-editor

A idéia de uso é: copiar, ou do console ou de arquivo (gerado através de redirecionamento de saída), e colar no mermaid live editor.

Caso uma lista de arestas a destacar seja fornecida, ela é usada para destacar arestas. A lista deve ter o mesmo comprimento da lista de arestas do grafo. Se o elemento correspondente à aresta contiver 0 então a aresta é desenhada com linha pontilhada, se contiver !=0 então a aresta é desenhada com linha cheia. Caso o ponteiro para a lista valha NULL, entende-se que a lista não foi fornecida e todas as arestas são desenhadas com linha cheia.

Esta função não mostra vértices de grau zero. Não sei se mermaid permite vértices de grau zero.

Parâmetros

g	ponteiro para grafo a imprimir
destacarArestas	ponteiro para lista de arestas a destacar, ou NULL.

5.3.2.5 preencheAleatorio()

```
void preencheAleatorio (
        struct grafo * g )
```

Preenche a estrutura com um grafo aleatório.

Em um grafo criado (estruturas alocadas e de tamanho definido), preenche com a quantidade definida de arestas. Estas recebem origem, destino e peso aleatórios. Origem e destino no intervalo [0:N-1] e peso no intervalo [0:1000]

5.3.2.6 preencheHamiltonianoAleatorio()

```
void preencheHamiltonianoAleatorio (
        struct grafo * g,
        int mudaOrdem,
        int mudaRotulo )
```

Preenche a estrutura com um circuito Hamiltoniano e completa com arestas geradas aleatoriamente.

Em um grafo criado (estruturas alocadas e de tamanho definido), preenche com um circuito hamiltoniano e completa o número de arestas com origem destino e peso aleatórios.

Caso haja menos arestas que o necessário para armazenar o circuito, não preenche e retorna uma mensagem de erro.

As arestas do caminho são geradas sequencialmente: 0-1, 1-2, 2-3, ... O circuito é completado por uma aresta do último para o primeiro vértice.

Na solução do Problema do Caixeiro Viajante por tentativa e erro (backtracking), isto favorece que algoritmos errados gerem soluções corretas. Por isso foram implementadas rotinas para permutar as arestas aleatoriamente e para trocar os rótulos dos vértices aleatoriamente.

Parâmetros

g	Ponteiro para o grafo a preencher;
mudaOrdem	se 0 não muda ordem, se !=0 muda ordem das arestas na lista de arestas (A).
mudaRotulo	se 0 não muda rótulos, se !=0 muda rótulos dos vértices, preservando a forma do grafo (o grafo resultante é isomórfo ao grafo inicial. Ao final, preserva a propriedade de que os rótulos dos vértices estão no intervalo [0:N-1]

Grafo de chamadas desta função:

5.3.2.7 trocaRotulo()

```
void trocaRotulo (
        struct grafo * g,
```

```
int antigo,
int novo )
```

Troca o rótulo (antigo) de um vértice por outro (novo).

Função de apoio para preencheHamiltonianoAleatorio. A troca de um rótulo por outro já existente altera o grafo. A alteração não pode ser revertida pela troca inversa.

Segundo esta especificação do grafo, os vértices são rotulados com números de [0:N-1]. É muito provável que esta função precise ser aplicada várias vezes, com parâmetros diferentes, passando por estados que não obedecem a especificação, para que, ao final da aplicação do conjunto, chegue-se ao ajuste desejado E que segue a especificação.

Parâmetros

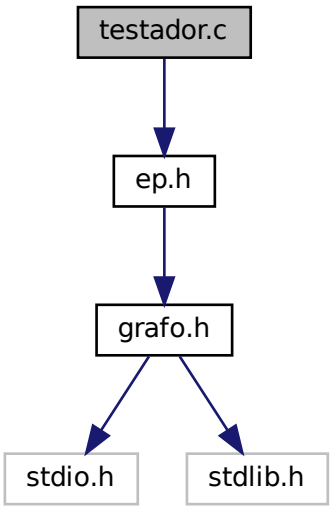
g	Ponteiro para o grafo que contém o vértice;
antigo	rótulo do vértice.
novo	Novo rótulo do vértice.

5.4 Referência ao ficheiro testador.c

Programa para início da execução do algoritmo.

```
#include "ep.h"
```

Diagrama de dependências de inclusão para testador.c:



Macros

- #define **NV** 10
- #define **NA** 20

Funções

- void **main** (void)

5.4.1 Descrição detalhada

Programa para início da execução do algoritmo.

O teste, que será feito para calcular a nota, será feito com uma versão ampliada deste programa. Esta é uma versão de referência para mostrar como os testes serão executados. Note que podem ser feitas várias execuções em sequência.

No EP, espera-se que o aluno implemente uma solução por tentativa e erro (backtracking) para esta variação do Problema do Caixeiro Viajante.

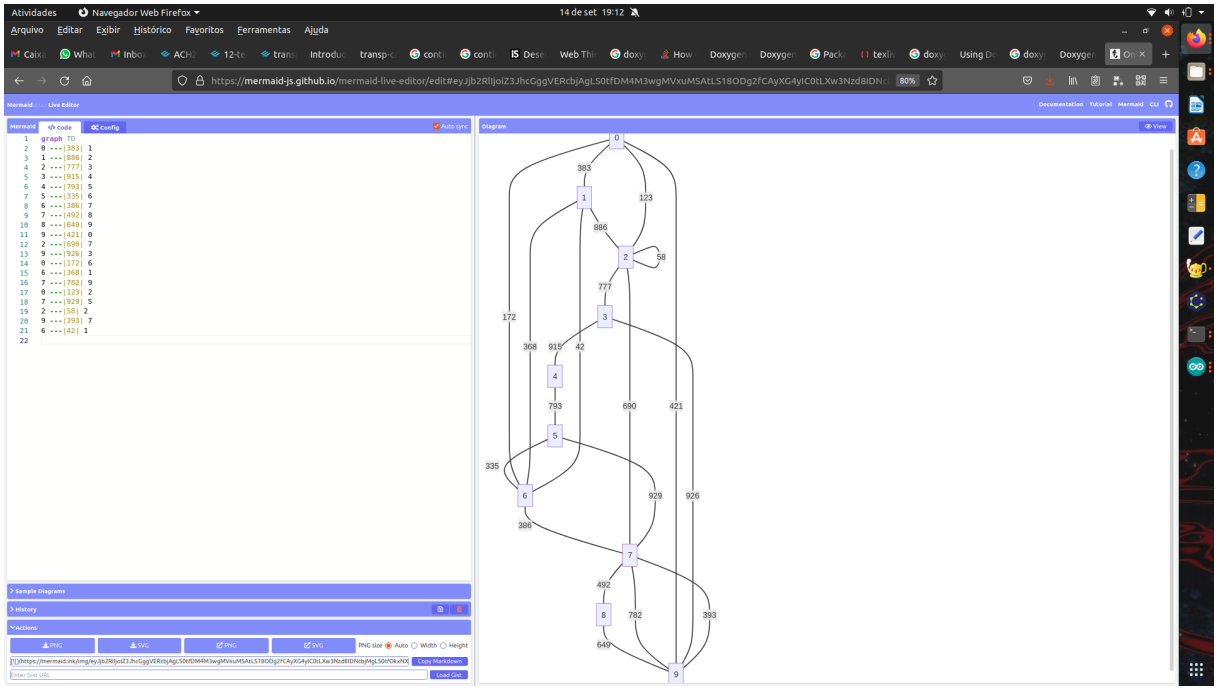


Figura 1 Captura de tela do editor online do Mermaid processando o texto de saída do programa

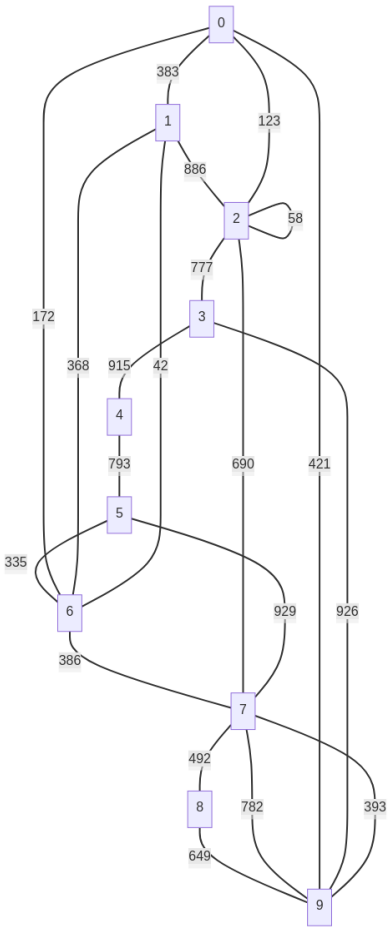


Figura 2 grafo gerado com circuito hamiltoniano e completado com arestas geradas aleatoriamente

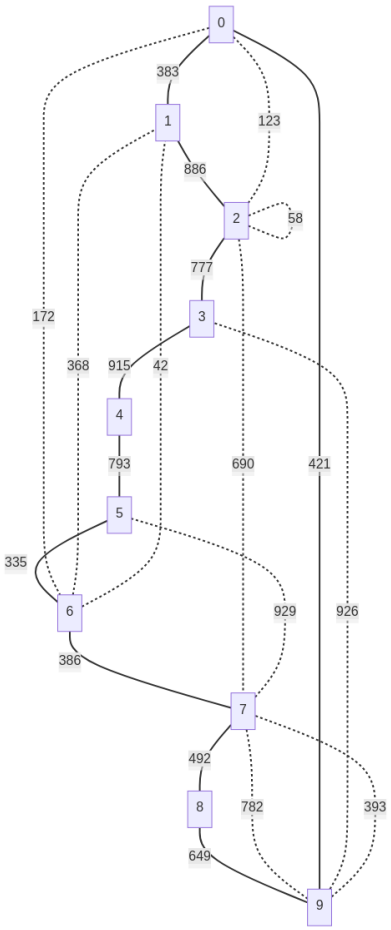


Figura 3 o mesmo grafo, após a execução do algoritmo. Em linha cheia as arestas que fazem parte do circuito hamiltoniano, em pontilhado as arestas que não fazem parte. Neste caso específico, a ordem das arestas e o algoritmo de seleção são tais que o circuito é encontrado

Autor

Fábio Nakano

Data

2021-09-14

Índice