Graph

Generated by Doxygen 1.6.1

Mon May 17 22:56:33 2010

Contents

Chapter 1

Todo List

Member grafo::dijkstra(int src, int dst) Fazer o método gravar as arestas usadas no vetor prev_aresta.

Member grafo::print() Fazer o método imprimir c e f, e colocar uma opção de imprimir as arestas inseridas para fluxos ou não.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

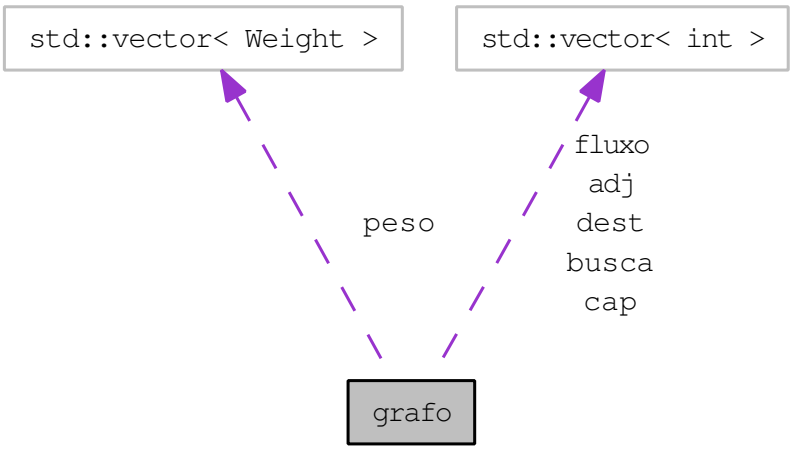
Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 grafo Struct Reference

Collaboration diagram for grafo:



Public Member Functions

- void inic (int n)

  Inicializador do grafo.

- void aresta (int src, int dst, Weight p, int c)

  Insere uma aresta no grafo.

- void print ()

  Imprime o grafo.

- void dfs (int src)

  Busca em profundidade.

- void bfs (int n)

  Busca em largura.

- void dijkstra (int src, int dst)

  Algoritmo de Dijkstra.

- int bellman_ford (int src)

  Bellman-Ford.

- int edmonds_karp (int src, int dst)

  Algoritmo de fluxo máximo de Edmonds-Karp.

- int bfs_fluxo (int src, int dst)

  Busca em largura, modificada para o ser usada por Edmonds-Karp.

- int inverso (int a)

  Retorna a aresta inversa.

Public Attributes

- vector< int > dest
- vector< int > fluxo
- vector< int > cap
- int M [VT]
- vector< int > adj [VT]
- int nadj [VT]
- int nvt
- int nar
- int vis [VT]
- int dist [VT]
- int prev [VT]
- vector< Weight > peso
- vector< int > busca
- int prev_busca [VT]
- int prev_aresta [VT]

4.1.1 Member Function Documentation

4.1.1.1 void grafo::aresta (int *src*, int *dst*, Weight *p* = 0, int *c* = 0)

Insere uma aresta no grafo.

Parameters:

← ***src*** Vértice origem.

← ***dst*** Vértice destino.

← ***p*** Peso da aresta. Default 0.

← ***c*** Capacidade da aresta. Default 0.

Note:

Para usar os algoritmos de fluxo, é necessário descomentar uma parte do código desse método. A parte está indicada em seu código. Isso faz com que o grafo insira a aresta desejada e, além dela, uma aresta inversa, usada internamente pelos algoritmos de fluxo, para que o fluxo "possa ser mandado de volta". Essa aresta tem capacidade 0. Para problemas onde o grafo é não direcionado, o valor de sua capacidade pode ser alterado para a capacidade da aresta normal, e, dessa forma, não é necessário fazer duas chamadas a este método na função main.

4.1.1.2 int grafo::bellman_ford (int *src*)

Bellman-Ford. Acha a menor distância até cada nó. Funciona em grafos com arestas cujos pesos são negativos.

Parameters:

← ***src*** Nó origem.

→ ***prev*** Predecessor de cada nó.

→ ***dist*** Distância até cada nó.

Returns:

1 se existe pelo menos um ciclo negativo no grafo, 0 caso contrário.

Note:

Limpa automaticamente os vetores prev e busca antes de executar o algoritmo.

4.1.1.3 void grafo::bfs (int *n*)

Busca em largura. Veja os comentários da busca em profundidade. Os mesmos se aplicam aqui.

Note:

Não limpa vis, prev_busca, prev_aresta e o vector busca internamente.

4.1.1.4 int grafo::bfs_fluxo (int *src*, int *dst*)

Busca em largura, modificada para o ser usada por Edmonds-Karp.

Parameters:

← ***src*** Nó origem.

← ***dst*** Nó destino.

→ ***vis*** Nós que podem ser alcançados nesse grafo residual. Determina o Min-Cut ao final do algoritmo de Edmonds-Karp.

Returns:

Capacidade do caminho encontrado.

Note:

Limpa o vetor vis. Atualiza os vetores prev_busca e prev_aresta para que sejam usados por Edmonds-Karp. Esses vetores não precisam ser limpados.

Here is the caller graph for this function:

4.1.1.5 void grafo::dfs (int *src*)

Busca em profundidade.

Parameters:

← ***src*** Nó inicial.

→ ***busca*** Ordem em que os nós são visitados pela primeira vez.

→ ***prev_busca*** Predecessor de cada nó.

→ ***prev_aresta*** Aresta usada pelo predecessor de cada nó para chegar nele.

→ ***vis*** Nós que foram visitados por essa busca.

Note:

Não limpa vis, prev_busca, prev_aresta e o vector busca internamente.

4.1.1.6 void grafo::dijkstra (int *src*, int *dst* = `-1`)

Algoritmo de Dijkstra. O algoritmo para quando encontra a menor distância até o nó destino, ou acha a menor distância até cada nó, caso dst == -1.

Parameters:

← ***src*** Nó origem.

← ***dst*** Nó destino. Default -1.

→ ***prev*** Predecessor de cada nó. -1 indica que não existe predecessor.

→ ***vis*** Nós visitados.

→ ***dist*** Distância total até cada nó, partindo-se de src.

Note:

Não limpa o vetor vis, e limpa prev e dist antes de executar o algoritmo.

Todo

Fazer o método gravar as arestas usadas no vetor prev_aresta.

4.1.1.7 int grafo::edmonds_karp (int *src*, int *dst*)

Algoritmo de fluxo máximo de Edmonds-Karp. Acha o fluxo máximo do grafo. Para isso, realiza várias buscas em largura para encontrar caminhos que adicionam fluxo.

Parameters:

← ***src*** Nó origem.
← ***dst*** Nó final.
→ ***fluxo*** Fluxo passando em cada aresta.
→ ***vis*** Determina o Min-Cut do grafo.

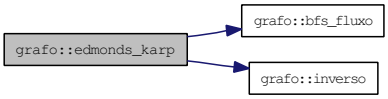
Returns:

Valor do fluxo máximo no grafo.

Note:

O método limpa o vetor fluxo antes de executar o algoritmo. Utiliza os vetores prev_busca e prev_-aresta, mas como parte da implementação, por isso não são marcados como parâmetros de saída aqui. vis não é alterado por esse método diretamente, e sim pela busca para fluxos.

Here is the call graph for this function:



4.1.1.8 void grafo::inic (int *n* = 0)

Inicializador do grafo. Após esse método ser executado, o grafo tem n nós e nenhuma aresta.

Parameters:

← ***n*** Numero de vertices que o grafo tem. Deve ser menor ou igual a VT. Default 0.

4.1.1.9 int grafo::inverso (int *a*) `[inline]`

Retorna a aresta inversa. Método usado por algoritmos de fluxo. Retorna a aresta inversa de uma dada aresta.

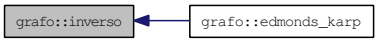
Parameters:

← ***a*** Índice de uma aresta.

Returns:

Índice da aresta inversa.

Here is the caller graph for this function:



4.1.1.10 void grafo::print ()

Imprime o grafo. Imprime na forma {dst, p, c, f}, onde p é o peso da aresta, c, sua capacidade e f, o fluxo nela.

Todo

Fazer o método imprimir c e f, e colocar uma opção de imprimir as arestas inseridas para fluxos ou não.

4.1.2 Member Data Documentation

4.1.2.1 vector<int> grafo::adj[VT]

Armazena o índice de cada aresta.

4.1.2.2 vector<int> grafo::busca

Armazena o resultado de uma busca.

4.1.2.3 vector<int> grafo::cap

Capacidade de cada aresta.

4.1.2.4 vector<int> grafo::dest

Nós de destino de uma dada aresta.

4.1.2.5 int grafo::dist[VT]

Distância até cada vértice. Usado por Dijkstra e Bellman-Ford.

4.1.2.6 vector<int> grafo::fluxo

Fluxo em cada aresta.

4.1.2.7 int grafo::M[VT]

Usado para achar a capacidade dos caminhos na busca em largura de fluxos.

4.1.2.8 int grafo::nadj[VT]

Número de arestas saindo de cada vértice.

4.1.2.9 int grafo::nar

Número de arestas no grafo.

4.1.2.10 int grafo::nvt

Número de vertices no grafo. Esse numero dever ser menor ou igual a VT.

4.1.2.11 vector<Weight> grafo::peso

Peso de cada aresta.

4.1.2.12 int grafo::prev[VT]

Predecessor de cada vértice. Usado por Dijkstra e Bellman-Ford.

4.1.2.13 int grafo::prev_aresta[VT]

Aresta usada para chegar aos vértices em uma busca.

4.1.2.14 int grafo::prev_busca[VT]

Predecessores de cada vértice em uma busca.

4.1.2.15 int grafo::vis[VT]

Nós visitados. 1 indica que o nó foi visitado e 0 indica que não foi nos métodos que o utilizam.

The documentation for this struct was generated from the following file:

- /home/troll/prog/algoricmc/algorithms/graph/src/graph.cpp

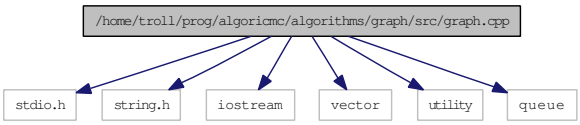
Chapter 5

File Documentation

5.1 /home/troll/prog/algoricmc/algorithms/graph/src/graph.cpp File Reference

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <vector>
#include <utility>
#include <queue>
```

Include dependency graph for graph.cpp:



Classes

- struct grafo

Typedefs

- typedef int Weight

Variables

- const int VT = 10
- const int AR = VT ∗ VT
- const int INFINITY = 99999

5.1.1 Typedef Documentation

5.1.1.1 typedef int Weight

5.1.2 Variable Documentation

5.1.2.1 const int AR = VT * VT

Numero máximo de arestas que o grafo pode comportar.

5.1.2.2 const int INFINITY = 99999

Valor de infinito, usado por Dijkstra e outros algoritmos.

5.1.2.3 const int VT = 10

Numero máximo de vértices que o grafo pode comportar. Pode estourar a memoria estática.

Index