Graph

Generated by Doxygen 1.7.1

Mon Aug 16 2010 20:25:20

Contents

Chapter 1

Todo List

Member Graph::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

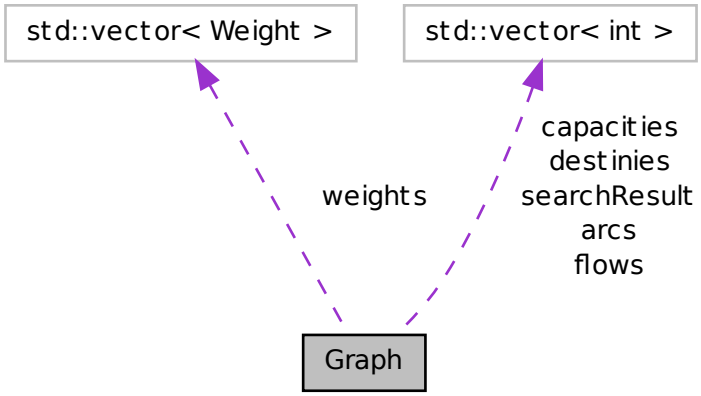
Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 Graph Struct Reference

Collaboration diagram for Graph:



Public Member Functions

- void initialize (int nNodes)

  Inicializador do Grafo.

- void insertArc (int srcNode, int dstNode, Weight weight, int capacity)

  Insere uma aresta no grafo.

- void print ()

  Imprime o grafo.

- void dfs (int srcNode)

  Busca em profundidade.

- int bfs (int srcNode, int dstNode)

  Busca em largura.

- void dijkstra (int srcNode, int dstNode)

  Algoritmo de Dijkstra.

- int bellmanFord (int srcNode)

  Bellman-Ford.

- int edmondsKarp (int srcNode, int dstNode)

  Algoritmo de fluxo máximo de Edmonds-Karp.

- int reverse (int arc)

  Retorna a aresta inversa.

Public Attributes

- vector< int > destinies
- vector< int > flows
- vector< int > capacities
- int pathCapacities [MAX_NODES]
- vector< int > arcs [MAX_NODES]
- int nNodes
- int visited [MAX_NODES]
- int distances [MAX_NODES]
- vector< Weight > weights
- vector< int > searchResult
- int previousNodes [MAX_NODES]
- int usedArcs [MAX_NODES]

4.1.1 Member Function Documentation

4.1.1.1 int Graph::bellmanFord (int *srcNode*)

Bellman-Ford.

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

Parameters

`[in]` ***srcNode*** Nó origem.

`[out]` ***previousNodes*** Predecessor de cada nó.

`[out]` ***usedArcs*** Arestas usada pelo predecessor de cada nó para chegar nele.

`[out]` ***distances*** Distância até cada nó.

Returns

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

Note

Inicializa "previousNodes" e "distances" antes de executar.

4.1.1.2 int Graph::bfs (int *srcNode*, int *dstNode* = −1)

Busca em largura.

Veja os comentários da busca em profundidade. Os mesmos se aplicam aqui. Também implementa algumas funcionalidades usadas por edmondsKarp. Elas estão comentadas no código.

Parameters

[in] ***dstNode*** Nó destino. Default -1 (nenhum nó destino, visita todos os nós).

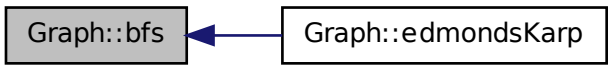
Returns

Usado apenas pelo edmondsKarp. Retorna o valor da menor capacidade do caminho achado.

Note

Não inicializa "visited", "previousNodes", "usedArcs" e nem "searchResult" internamente. edmondsKarp faz as inicializações necessárias antes de chamar este método.

Here is the caller graph for this function:



4.1.1.3 void Graph::dfs (int *srcNode*)

Busca em profundidade.

Parameters

[in] ***srcNode*** Nó inicial.

[out] ***searchResult*** Ordem em que os nós são visitados pela primeira vez. Não é inicializado internamente.

[out] ***previousNodes*** Predecessor de cada nó. Não é inicializado internamente.

[out] ***usedArcs*** Arestas usada pelo predecessor de cada nó para chegar nele. Não é inicializado internamente.

[out] ***visited*** Nós que foram visitados por essa busca. Não é inicializado internamente.

4.1.1.4 void Graph::dijkstra (int *srcNode*, int *dstNode* = `-1`)

Algoritmo de Dijkstra.

Se ("dstNode" == -1), então o algoritmo para após ter calculado a distância mínima do nó origem para cada nó; caso contrário, o algoritmo retorna assim que achar a distância mínima até o nó destino.

Parameters

`[in]` ***srcNode*** Nó origem.

`[in]` ***dstNode*** Nó destino. Default -1.

`[out]` ***previousNodes*** Predecessor de cada nó. -1 indica que não existe predecessor.

`[out]` ***usedArcs*** Arestas usada pelo predecessor de cada nó para chegar nele.

`[out]` ***visited*** Nós visitados. Não é inicializado internamente.

`[out]` ***distances*** Distância total até cada nó, partindo-se de srcNode.

Note

Inicializa "previousNodes" e "distances" internamente antes de executar o algoritmo.

4.1.1.5 int Graph::edmondsKarp (int *srcNode*, int *dstNode*)

Algoritmo de fluxo máximo de Edmonds-Karp.

Acha o fluxo máximo no grafo. Para isso, realiza várias buscas em largura para encontrar caminhos que adicionam fluxo.

Parameters

`[in]` ***srcNode*** Nó origem.

`[in]` ***dstNode*** Nó final.

`[out]` ***flows*** Fluxo passando em cada aresta.

`[out]` ***visited*** Determina o Min-Cut.

`[out]` ***previousNodes*** É usada pela busca em largura. [out] usedArcs É usada pela busca em largura.

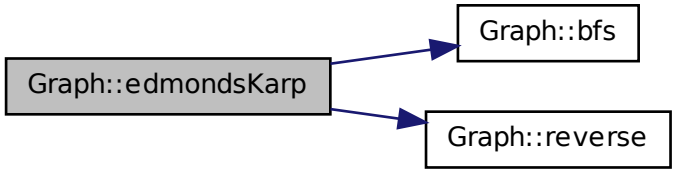
Returns

Valor do fluxo máximo no grafo.

Note

O método já faz as inicializações necessárias antes de executar o algoritmo. Não esqueça de descomentar as linhas de código da busca em largura e da inserção de arestas para usar esse método.

Here is the call graph for this function:



4.1.1.6 void Graph::initialize (int *nNodes* = *0*)

Inicializador do Grafo.

Limpa o grafo inteiro e cria um grafo com nNodes nós.

Parameters

[in] ***nNodes*** Numero de vertices que o Graph tem. Deve ser menor ou igual a MAX_NODES. Default 0.

4.1.1.7 void Graph::insertArc (int *srcNode*, int *dstNode*, Weight *weight* = *0*, int *capacity* = *0*)

Insere uma aresta no grafo.

Parameters

[in] ***srcNode*** Vértice origem.

[in] ***dstNode*** Vértice destino.

[in] ***weight*** Weight da aresta. Default 0.

[in] ***capacity*** 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 NÃO é direcionado, não é preciso descomentar o código (só testei isso em alguns programas, mas eles passaram).

4.1.1.8 void Graph::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.

4.1.1.9 int Graph::reverse (int *arc*) [inline]

Retorna a aresta inversa.

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

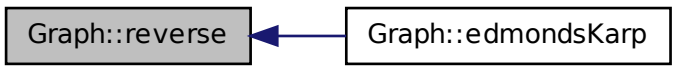
Parameters

[in] ***arc*** Índice de uma aresta.

Returns

Índice da aresta inversa.

Here is the caller graph for this function:



4.1.2 Member Data Documentation

4.1.2.1 vector<int> Graph::arcs[MAX_NODES]

Arestas de cada nó. Armazena o índice de cada aresta.

4.1.2.2 vector<int> Graph::capacities

Capacidade de cada aresta.

4.1.2.3 vector<int> Graph::destinies

Nós de destino de uma dada aresta.

4.1.2.4 int Graph::distances[MAX_NODES]

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

4.1.2.5 vector<int> Graph::flows

Fluxo em cada aresta.

4.1.2.6 int Graph::nNodes

Número de vertices no Graph. Esse numero dever ser menor ou igual a MAX_NODES.

4.1.2.7 int Graph::pathCapacities[MAX_NODES]

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

4.1.2.8 int Graph::previousNodes[MAX_NODES]

Predecessores de cada nó. Usado pelas buscas e pelo dijkstra.

4.1.2.9 vector<int> Graph::searchResult

Armazena o resultado de uma busca.

4.1.2.10 int Graph::usedArcs[MAX_NODES]

Aresta usada para chegar aos nós em uma busca. Verifique previousNode antes de ler um valor daqui.

4.1.2.11 int Graph::visited[MAX_NODES]

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

4.1.2.12 vector<Weight> Graph::weights

Peso de cada aresta.

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

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

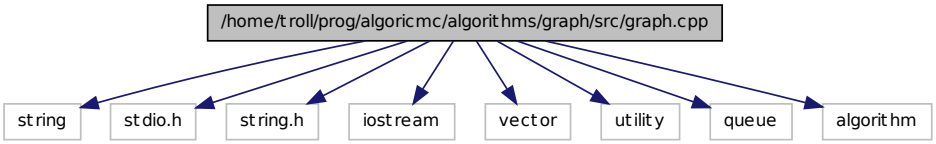
Chapter 5

File Documentation

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

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

Include dependency graph for graph.cpp:



Classes

- struct Graph

Typedefs

- typedef int Weight

Variables

- const int MAX_NODES = 10000
- const int INFINITY = 100000

5.1.1 Typedef Documentation

5.1.1.1 typedef int Weight

5.1.2 Variable Documentation

5.1.2.1 const int INFINITY = 100000

Valor de infinito, usado por Dijkstra e outros algoritmos.

5.1.2.2 const int MAX_NODES = 10000

Numero máximo de nós que o Graph pode comportar. NOTA: Pode estourar a memoria estática.

Index