Modelo Físico Estrutural Auto Verificável

Assertivas

Exemplo Físico

Aplicação: Labirinto

Autores: João Fernando MacDowell, Lucas Hardman e Rafael Azevedo

Versão: 3.0

int \* identidadeGrafo

int identidadeVertices

LIS\_tppLista pOrigemGrafo

LIS\_tppLista pVertices

void ( \*ExcluirValor ) ( void \* pValor )

int idTipo

int numeroElementos

void \* pConteudo

struct tagElemLista \* pAnt

struct tagElemLista \* pProx

void \* pConteudo

struct tagElemLista \* pAnt

struct tagElemLista \* pProx

void \* pConteudo

struct tagElemLista \* pAnt

struct tagElemLista \* pProx

void \* pConteudo

struct tagElemLista \* pAnt

struct tagElemLista \* pProx

tpElemLista \* pOrigemLista

tpElemLista \* pFimLista

tpElemLista \* pElemCorr

int numElem

void (\*ExcluirValor)

(void \* pValor)

tpElemLista \* pOrigemLista

tpElemLista \* pFimLista

tpElemLista \* pElemCorr

int numElem

void (\*ExcluirValor)

(void \* pValor)

tpElemLista \* pOrigemLista

tpElemLista \* pFimLista

tpElemLista \* pElemCorr

int numElem

void (\*ExcluirValor)

(void \* pValor)

tpElemLista \* pOrigemLista

tpElemLista \* pFimLista

tpElemLista \* pElemCorr

int numElem

void (\*ExcluirValor)

(void \* pValor)

int identidade;

LIS\_tppLista

pVerticeDestino;

LIS\_tppLista pArestas

void \* pConteudo

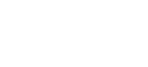
int identidade

int idTipo

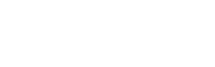
GRF\_tppGrafo pGrafoCabeca

Modelo Físico Estrutural

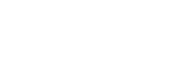
:



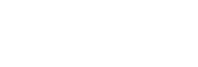
Grafo



Vértices



Vértice Cabeça



Arestas

Cabeça

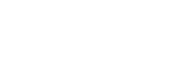
Origens



#



#



Vértice

Aresta

Assertivas Estruturais:

**Lista:**

* Seja *lis* um ponteiro para a estrutura cabeça da lista.
* Seja *no* um ponteiro para a estrutura nó da lista.
* Se *no->pProx != NULL*, então *no->pProx->pAnt == no.*
* Se *no->pAnt != NULL*, então *no->pAnt->pProx == no.*
* Se *lis->numElem == 0*, então:
* *lis->pElemCorr == NULL*;
* *lis->pOrigemLista == NULL*;
* *lis->pFimLista == NULL.*
* Se *lis->numElem > 0*, então *lis->ElemCorr != NULL.*
* Se *lis->numElem == 1*, então:
* *(lis->pElemCorr == lis->pOrigemLista ) && (lis->pElemCorr == lis->pFimLista ).*

**Grafo:**

* Seja *grf* um ponteiro para a estrutura cabeça do grafo.
* Se *grf->pOrigemGrafo != NULL*, então *grf->pVertices != NULL*.
* Se um grafo existe, então (*grf->pVertices != NULL)*

&& (*grf->pOrigemGrafo != NULL).*

* Se uma aresta de identidade A aponta para uma cabeça de vértice, então essa cabeça de vértice possui um vértice que possui exatamente uma aresta de identidade -A que aponta para o vértice que tinha a aresta de identidade A.
* Uma cabeça de vértice é uma lista que possui sempre um único elemento que armazena uma estrutura vértice. Toda estrutura vértice possui uma lista que armazena estruturas aresta.
* Um vértice possui um e um somente ciclo. Se um vértice possui uma aresta que aponta para a cabeça desse próprio vértice, então nenhuma outra aresta desse vértice aponta para a cabeça desse próprio vértice
* Se *pGrafo->pVertices* possui pelo menos um elemento, então *pGrafo->pOrigemGrafo* possui pelo menos um elemento.
* Se *pGrafo->pVertices* possui um único elemento, então esse elemento é uma origem e está em *pGrafo->pOrigemGrafo*.
* Um vértice não pode ter duas arestas com a mesma identidade (número inteiro).
* Cabeças de vértices podem ser apontados uma e uma somente vez por cada aresta de qualquer vértice do grafo.
* Toda aresta aponta para uma cabeça de lista que é cabeça de vértice.
* Se existe uma cabeça de lista verticeCabeca que não pertence à lista pGrafo->pOrigemGrafo,

então *verticeCabeca->ElementoCorrente->vertice != NULL && verticeCabeca->ElementoCorrente->vertice->pArestas != NULL.*

* Se existe uma cabeça de vértice, existe uma estrutura vértice apontada pelo único elemento dela e existe uma lista de arestas apontada por essa estrutura vértice.
* Se um vértice possui uma aresta em sua lista de arestas que aponte para uma cabeça de vértice, então o vértice dessa cabeça de vértice tem uma aresta que possui ponteiro para o vértice referido no início dessa assertiva.
* Se um vértice existe, pVertice->pGrafoCabeca != NULL e deve ser ponteiro para o grafo ao qual vértice pertence.
* Se grafo vazio, é verdade que (pGrafo->numeroElementos == 0).
* Seja pVertice um ponteiro para struct vértice.

Então, é verdade que (pVertice->idTipo == pGrafo->idTipo).

**Labirinto:**

* Seja pLabirinto um ponteiro para a estrutura labirinto.
* Seja pLabirinto->tabuleiro != NULL, então pLabirinto != NULL.
* Seja pLabirinto->solução != NULL, então pLabirinto != NULL e pLabirinto->tabuleiro != NULL.
* Se um labirinto existe, então pLabirinto->tabuleiro != NULL e pLabirinto->idVerticeCorrente >= 0 de modo que pLabirinto>idVerticeCorrente seja um inteiro que represente um vértice do grafo pLabirinto->tabuleiro e que seja identificador do vértice corrente do pLabirinto->tabuleiro.

Exemplo Físico:

