UNIVERSIDADE FEDERAL DE SANTA CATARINA

CAMPUS UNIVERSITÁRIO – TRINDADE – CAIXA POSTAL 476
CEP. 88040-900 – FLORIANÓPOLIS – SANTA CATARINA
CENTRO TECNOLÓGICO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Engenharia de Componentes de Software

Prof. Dr. Antônio Augusto Fröhlich

Alunos:

Angelo Fábio de Melo

Christopher Viana Lima

POA – Programação Orientada a Aspectos

Neste trabalho vamos apresentar o uso do Paradigma AOP (Aspect-Oriented Programming para complementar o uso de uma classe já existente Grafos.

A classe Grafos tem por função o cálculo da menor distância/caminho entre dois pontos, utilizando o Algoritmo de Dijkstra.

Problema

A classe implementada não leva em consideração se entre dois pontos quaisquer exista algum problema que não permita a passagem, como por exemplo a queda de barreiras, enchentes entre outros.

Ao mesmo tempo, pode-se guardar um caminho requisitado em cache para uma posterior consulta, acelerando o tempo de resposta.

Nesses dois problemas é que entrará a AOP evitando a modificação da classe existente ou a criação de uma nova classe.

Classe Grafos sem Aspectos

Abaixo, temos os métodos GeraCaminho e GetDistancia, originais da classe Grafos.

```
void Grafos::GeraCaminho(char* arquivo, int origem, int destino) {

   ...  // Declaração das variáveis

   dados = fopen(arquivo,"r");
   fscanf(dados, "%d", &codmax);

   ... // Definição do tamanho dos vetores e estruturas

// Carga das distâncias entre os pontos

   while (fscanf(dados, "%d %d %f", &ini, &fim, &custo) == 3) {
      ini--;
      fim--;
      pontos[ini][fim] = custo;
      pontos[fim][ini] = custo;
   }
   fclose(dados);
   prox = -1;

// passa valores para a matriz distância
   for (int conta=0; conta<codmax; conta++) {
      d[conta] = GetDistancia(origem, conta);
      inter[conta] = origem;
      v[conta] = 0;
   }

// calcula tabela de distâncias
   v[origem] = 1;
   marca = -1;
   for (int indice=0; ((indice<codmax-1) && marca!=destino); indice++) {
      valor = 10000001;
      for (int conta=0; conta<codmax; conta++) {
         if ((d[conta]<valor) && (v[conta]!=1) && (prox!=conta)){
            marca = conta;
            valor = d[conta];
         }
      }
      v[marca] = 1;
      seq[indice].cod = indice;
      seq[indice].ini = origem;
      seq[indice].fim = marca;
      seq[indice].custo = valor;
      prox = marca;
      seq[indice].inter = inter[marca];

      for (int conta=0; conta<codmax; conta++) {
         if ((valor+GetDistancia(prox, conta))<d[conta]) {
            inter[conta] = prox;
         }
         d[conta] = MinDist(d[conta],(valor+GetDistancia(prox, conta)));
      }
      passos = indice;
   }

   MontaCaminho(seq, passos);

   ...
}

void Grafos::MontaCaminho(sequencia *pseq, int passos){

   ...  recebe a estrutura pseq para montar o caminho percorrido, passando o mesmo
para a estrutura Caminho_Calculado

}
```

```
double Grafos::GetDistancia(int Inicio, int Fim) {
   return pontos[Inicio][Fim];
}
```

No GeraCaminho, a intenção é fazer com que durante o cálculo do caminho requisitado, seja verificado se o mesmo não esteja dentro do arquivo de cache.

Sua função é abrir o arquivo de cache, se existir o caminho requisitado, o mesmo será carregado diretamente para a estrutura Caminho_Calculado, caso contrário será feito o cálculo e em seguida carregado a estrutura Caminho_Calculado para o arquivo de cache, segue abaixo o fonte do aspecto implementado.

```
advice calls1() : void around(char* arquivo, int origem, int destino) {
   bool existe = true;
   char *cam2 = "caminho";
   char *teste = "caminho";
   int passos, i, f;
   FILE *Cache;
   bool existe_arq = false;
   float custo;
   struct caminho {int vertice; double distancia;};

   printf("Verificando Cache!!!");

   Cache = fopen("c:\\temp\\cache.dat","a+");
   fclose(Cache);

   existe_arq = true;
   Cache = fopen("c:\\temp\\cache.dat","r");
   fscanf(Cache, "%s %d %d %d", cam2, &i, &f, &passos);
   while ((*cam2 != *teste || i != origem || f != destino) && (existe != false)) {
      for (int c=0; c<passos; c++) {
         fscanf(Cache, "%d %f", &i, &custo);
      }
      if (fscanf(Cache, "%s %d %d %d", cam2, &i, &f, &passos)==-1) {
         existe = false;
      }
   }
   if (existe) {
      Grafos::Caminho_Calculado=(struct caminho*)malloc(sizeof(struct caminho)*
passos);
      for (int c=0; c<passos; c++) {
         fscanf(Cache, "%d %f", &i, &custo);
         Grafos::Caminho_Calculado[c].vertice = i;
         Grafos::Caminho_Calculado[c].distancia = custo;
      }
      fclose(Cache);
   } else {
      tjp->proceed();
      if (existe_arq) {
         fclose(Cache);
      }
      Cache = fopen("c:\\temp\\cache.dat","a+");
      fprintf(Cache,"caminho %d %d %d\n ",origem,destino,Grafos::GetTamCaminho());
      for (int n = 0; n < Grafos::GetTamCaminho(); n++) {
```

```
         fprintf(Cache,"%d %f\n",Grafos::Caminho_Calculado[n].vertice,
Grafos::Caminho_Calculado[n].distancia);
      }
      fclose(Cache);
   }
}
```

Na função GetDistancia, o aspecto abrirá o arquivo contendo restrições (pontos sem acesso entre si), caso exista será atribuída uma distância infinita (que fará o cálculo de distância não usar esse trecho no caminho), senão permanecerá com a distância anteriormente atribuída.

```
advice calls2(): double around(int Inicio, int Fim) {

   FILE *restricao;
   bool existe;
   int ini, fim;

   existe = 0;

   restricao = fopen("restricao.dat","a+");
   fclose(Cache);

   restricao= fopen("restricao.dat","r");

   while (fscanf(restricao, "%d %d", &ini, &fim) == 2) {
      if (((ini = Inicio) && (fim = Fim)) || ((fim = Inicio) && (ini = Fim))) {
         existe = 1;
      }
   }
   fclose(restricao);
   if (existe)
      *tjp->result() = 10000000;
   else
      tjp->proceed();
}
```

Abaixo segue a classe Grafos compilada com os Aspectos pelo AspectC++ v0.7.2, disponível em www.aspectc.org.

```
/* This file was generated by the PUMA library, version 1.1. */

#ifndef __ac_h_
#define __ac_h_
class AC {
  public:
    typedef const char* Type;
    typedef int JPType;
    struct Action {void **_args; void *_result; void *_target; void *_that; void
    (*_wrapper)(Action &); inline void trigger () { _wrapper (*this); }};
};
#endif // __ac_h_
#include "AspGrafos.ah"
#line 1 "C:\Fontes\Grafos_Com_Aspect\Grafos.cpp"
//---------------------------------------------------------------------------

#pragma hdrstop
```

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

//---------------------------------------------------------------------------
#pragma package(smart_init)

...

void Grafos::MontaCaminho(sequencia *pseq, int passos){

   ...  // recebe a estrutura pseq para montar o caminho percorrido, passando o
mesmo para a estrutura Caminho_Calculado
   // Não sofre nenhuma modificação
}

struct TJP_GrafosGeraCaminho_F6GrafosvPciiE {
  typedef void Result;
  typedef ::Grafos That;
  typedef ::Grafos Target;
  AC::Action *_action;
  inline void proceed () { _action->trigger (); }
  AC::Action &action() {return *_action;}
};


static inline void
a0_around_AspGrafos_exec_GrafosGeraCaminho_F6GrafosvPciiE(TJP_GrafosGeraCaminho_F6G
rafosvPciiE *tjp, char* arquivo, int origem, int destino) {
   typedef TJP_GrafosGeraCaminho_F6GrafosvPciiE JoinPoint;

      ...   // Código idêntico ao anteriormente mostrado em advice calls1()

}

   ...

void Grafos::__old_GeraCaminho(char* arquivo, int origem, int destino) {

   ...  // A modificação nessa função foi simplesmente a mudança de nome (de
GeraCaminho para __old_GeraCaminho), o código permanece o mesmo.

}

static void __action_exec_GrafosGeraCaminho_F6GrafosvPciiE_0 (AC::Action &action) {
   ((Grafos*)action._target)->::Grafos::__old_GeraCaminho(*((char **)
action._args[0]), *((int*) action._args[1]), *((int*) action._args[2]));
}

static inline void inter_GrafosGeraCaminho_F6GrafosvPciiE_l0(AC::Action &action) {
  void (*_wrapper)(AC::Action &) = action._wrapper;
  action._wrapper = __action_exec_GrafosGeraCaminho_F6GrafosvPciiE_0;
TJP_GrafosGeraCaminho_F6GrafosvPciiE tjp_GrafosGeraCaminho_F6GrafosvPciiE =
{&action};
   a2_before_AspGrafos_exec_GrafosGeraCaminho_F6GrafosvPciiE();
  ((Grafos*)action._target)->Grafos::__old_GeraCaminho (arg0, arg1, arg2);
  action._wrapper = _wrapper;
}

static void __action_exec_GrafosGeraCaminho_F6GrafosvPciiE_1 (AC::Action &action)
 {
  inter_GrafosGeraCaminho_F6GrafosvPciiE_l0(action);
 }

void Grafos::GeraCaminho(char * arg0, int arg1, int arg2) {
     void *args_GrafosGeraCaminho_F6GrafosvPciiE[] = { (void*)&arg0, (void*)&arg1,
(void*)&arg2 };
```

```
  AC::Action tjp_action_GrafosGeraCaminho_F6GrafosvPciiE = {
args_GrafosGeraCaminho_F6GrafosvPciiE, 0,
(TJP_GrafosGeraCaminho_F6GrafosvPciiE::Target*)this,
(TJP_GrafosGeraCaminho_F6GrafosvPciiE::That*)this,
__action_exec_GrafosGeraCaminho_F6GrafosvPciiE_1 };
TJP_GrafosGeraCaminho_F6GrafosvPciiE tjp_GrafosGeraCaminho_F6GrafosvPciiE =
{&tjp_action_GrafosGeraCaminho_F6GrafosvPciiE};


a0_around_AspGrafos_exec_GrafosGeraCaminho_F6GrafosvPciiE(&tjp_GrafosGeraCaminho_F6
GrafosvPciiE);
}

struct TJP_GrafosGetDistancia_F6GrafosdiiE {
  typedef double Result;
  typedef ::Grafos That;
  typedef ::Grafos Target;
  AC::Action *_action;
  inline void proceed () { _action->trigger (); }
  inline Result *result() {return (Result*)_action->_result;}
  AC::Action &action() {return *_action;}
};


static inline void
a1_around_AspGrafos_exec_GrafosGetDistancia_F6GrafosdiiE(TJP_GrafosGetDistancia_F6G
rafosdiiE *tjp, int Inicio, int Fim)
 {
   typedef TJP_GrafosGetDistancia_F6GrafosdiiE JoinPoint;

       ...   // Código idêntico ao anteriormente mostrado em advice calls2()


 }

static inline void a3_after_AspGrafos_exec_GrafosGetDistancia_F6GrafosdiiE()  {
        printf("executou GeraCaminho");
}

double Grafos::__old_GetDistancia(int Inicio, int Fim) {
  return pontos[Inicio][Fim];
}

static void __action_exec_GrafosGetDistancia_F6GrafosdiiE_0 (AC::Action &action)
 {
   *((double*)action._result) = ((Grafos*)action._target)-
>::Grafos::__old_GetDistancia(*((int*) action._args[0]), *((int*)
action._args[1]));
 }

static inline void inter_GrafosGetDistancia_F6GrafosdiiE_l0(AC::Action &action) {
  void (*_wrapper)(AC::Action &) = action._wrapper;
  action._wrapper = __action_exec_GrafosGetDistancia_F6GrafosdiiE_0;
TJP_GrafosGetDistancia_F6GrafosdiiE tjp_GrafosGetDistancia_F6GrafosdiiE =
{&action};
     *((double*)action._result) = ((Grafos*)action._target)-
>Grafos::__old_GetDistancia (arg0, arg1);
   a3_after_AspGrafos_exec_GrafosGetDistancia_F6GrafosdiiE();
  action._wrapper = _wrapper;
}

static void __action_exec_GrafosGetDistancia_F6GrafosdiiE_1 (AC::Action &action)
 {
  inter_GrafosGetDistancia_F6GrafosdiiE_l0(action);
 }

double Grafos::GetDistancia(int arg0, int arg1)
```

```
{
   double result;
  void *args_GrafosGetDistancia_F6GrafosdiiE[] = { (void*)&arg0, (void*)&arg1 };
  AC::Action tjp_action_GrafosGetDistancia_F6GrafosdiiE = {
args_GrafosGetDistancia_F6GrafosdiiE, &result,
(TJP_GrafosGetDistancia_F6GrafosdiiE::Target*)this,
(TJP_GrafosGetDistancia_F6GrafosdiiE::That*)this,
__action_exec_GrafosGetDistancia_F6GrafosdiiE_1 };
TJP_GrafosGetDistancia_F6GrafosdiiE tjp_GrafosGetDistancia_F6GrafosdiiE =
{&tjp_action_GrafosGetDistancia_F6GrafosdiiE};

a1_around_AspGrafos_exec_GrafosGetDistancia_F6GrafosdiiE(&tjp_GrafosGetDistancia_F6
GrafosdiiE);
   return (double)result;
}

...
```

Pode-se notar que os métodos originais foram renomeados para __old_GeraCaminho e __old_GetDistancia, e foram gerados novos métodos com o nome GeraCaminho e GetDistancia, que tratam de chamar os métodos implementados no Aspecto ou o código original da classe Grafos.

Talvez esse exemplo demonstre ser mais fácil implementar uma nova classe do que utilizar AOP, mas se pensar em uma biblioteca de grandes componentes com necessidade de pequenas alterações, o uso de AOP torna-se fundamental.