Universidade Federal do Espírito Santo – Departamento de Informática
Est. de Informação (INF02827) & Est. de Dados (INF01906)
2º Trabalho Prático
Período: 2008/2
Profª Patrícia Dockhorn Costa
Email: pdcosta@inf.ufes.br

Data de Entrega: 29/10/2008 ***Grupos de 2 pessoas***

Este trabalho tem como objetivo praticar o uso de tipos abstratos de dados e estruturas do tipo Lista.

Regras Importantes

- Não é tolerado plágio. Trabalhos copiados serão penalizados com zero.
- A data de entrega é inadiável. Para cada dia de atraso, é retirado um ponto da nota do trabalho.

Material a entregar

- Impresso: Documentação do trabalho, que deve conter:
  - Introdução: descrição do problema a ser resolvido e visão geral sobre o funcionamento do programa.
  - Implementação: descrição sobre a implementação do programa. Deve ser detalhada a estrutura de dados utilizada (de preferência com diagramas ilustrativos), o funcionamento das principais funções utilizadas incluindo pré e pós condições, o formato de entrada e saída de dados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado. Modularize o seu programa como discutido em sala de aula.
  - Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
  - Bibliografia: bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso.

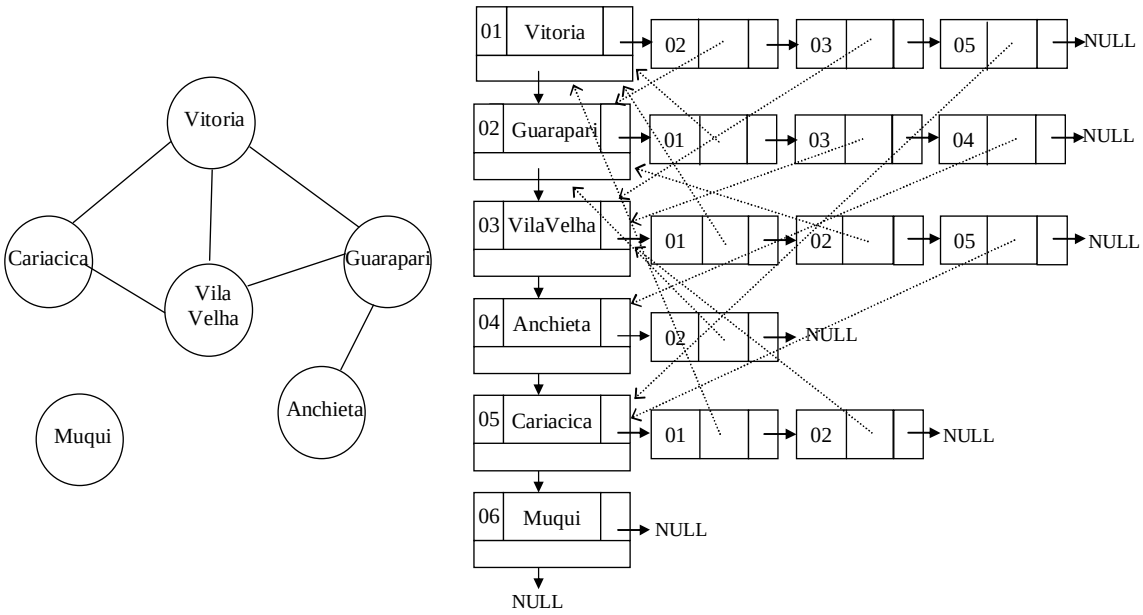
- Por email (pdcosta@inf.ufes.br):
  - O assunto da mensagem deve ser ed:trab2:<nome1>:<nome2>
    - Por exemplo: ed:trab2:<joaosilva>:<mariacosta>
  - Documentação do trabalho (em formato PDF).
  - Todos os arquivos .c e .h criados (exigido código muito bem documentado!).
  - O makefile.
  - Favor nomear os arquivos da seguinte maneira: TadListaVertice.h, TadListaVertice.c, TadListaAresta.h, TadListaAresta.c, TadGrafo.h, TadGrafo.c, Rodovias.c.

Sistema Rodoviário Tabajara

Você foi contrato pelo Governo do Estado do Espírito Santo para implementar um programa que gerencia o sistema rodoviário do estado. De uma maneira geral, para cada cidade cadastrada no sistema, você deverá cadastrar uma lista de cidades que estão diretamente conectadas através de uma rodovia estadual. O sistema rodoviário pode ser visualizado através de uma estrutura chamada *grafo*. Grafos são estruturas compostas de *arestas* e de *vértices*. Vértices são conectados entre si através de arestas que indicam algum tipo de relação entre os vértices. Um grafo pode ser implementado através de listas encadeadas: basicamente, implementa-se uma lista na qual cada célula contém os dados de um vértice. Além disso, cada célula desta lista contém uma lista encadeada indicando quais os vértices que estão ligados a ela.

Considere o seguinte exemplo do sistema rodoviário: a cidade de Vitória está conectada às cidades Guarapari, Cariacica e Vila Velha. Vila Velha está conectada à Guarapari, que está conectada à Anchieta, que por sua vez, não

está conectada a nenhuma outra cidade além de Guarapari. De Vila Velha também podemos chegar a Cariacica através de uma rodovia. De Cariacica não há outras rodovias além da que chega em Vila Velha. Temos também Muqui, *"cidade menina"*, que ainda não está conectada a nenhuma outra cidade. As listas de cidades diretamente conectadas são construídas com células que contém ponteiros para cidades conectadas. A figura a seguir mostra a estrutura do tipo grafo que representa o exemplo citado e a respectiva implementação usando listas encadeadas.



Nesse trabalho, você deverá implementar essa estrutura. Considere que cada cidade é uma célula que contém os campos *nome da cidade* e o *número de habitantes da cidade*. Não existem cidades com o mesmo nome e com nomes duplos. Por exemplo, a cidade de Vila Velha seria cadastrada com o nome "VilhaVelha" no sistema rodoviário tabajara. As seguintes operações devem ser implementadas no seu programa:

- Cadastra (Cidade): Cadastra uma nova cidade no sistema rodoviário, que no momento do cadastramento não está conecta a nenhuma outra cidade. O identificador único da cidade deve ser gerado automaticamente pelo seu programa;
- Remove (Cidade): Remove uma cidade do sistema rodoviário. Todas as conexões desta cidade com outras deverão ser removidas;
- Conecta (Cidade1, Cidade2): Conecta uma cidade a outra, ambas previamente cadastradas no sistema rodoviário. Esta operação seria a construção de uma rodovia que conecta as duas cidades;
- Desconecta (Cidade1, Cidade2): Desconecta uma cidade da outra. Esta operação seria a inutilização de uma rodovia que conecta as duas cidades;
- NoHabitantes (valor): Retorna o número de cidades cadastradas que têm o número de habitantes maior do que *valor*;
- Alcancavel (Cidade1, Cidade2): Verifica se é possível, a partir da Cidade1, alcançar a Cidade2. Retorna SIM caso seja possível e NAO caso contrário;
- ImprimeCidade (Cidade): Imprime as informações de uma cidade (nome e número de habitantes) bem como as informações das cidades que estão conectadas a ela;
- ImprimeSistemaRodoviario (): Imprime as informações de todas as cidades cadastradas no sistema rodoviário, bem como as listas de cidades conectadas.

A implementação da estrutura deverá ser feita utilizando alocação dinâmica de memória (ponteiros). Você deve fazer testes de consistência se essas operações podem ser aplicadas (teste de pré condições) e deve imprimir mensagens de falha (por exemplo, a chamada de função *Remove(SaoMateus)* no exemplo acima deve dar uma mensagem de erro pois a cidade de SaoMateus não está cadastrada no sistema). Imprima a mensagem de erro *"Erro: cidade nao cadastrada"* para esse tipo de erro. Além disso, procure escrever funções auxiliares que

facilitem a implementação das operações acima, evitando a repetição desnecessária de código. Organize o seu sistema usando o conceito de Tipos Abstratos de Dados discutido em sala de aula.

O programa Rodovias.c deverá definir o TipoItemCidade, que define o conteúdo de um vértice do grafo. A definição desse tipo é dada a seguir:

```
//Definição do tipo TipoItemCidade
typedef struct {
      char* NomeCidade;
      int NoHabitantes;
} TipoItemCidade;
```

Entrada e Saída

O seu programa (Rodovias.c) deverá ler os dados de entrada a partir de um arquivo, cujo nome é passado como parâmetro na linha de comando (Faz parte do trabalho descobrir como manipular arquivos e strings em C). Exemplo de execução do programa a partir da linha de comando:

```
Rodovias entrada.txt
```

O arquivo de entrada é basicamente uma lista de comandos (um por linha) em formato texto. O último comando é a palavra FIM, que indica o final do arquivo. O formato a ser usado é exemplificado abaixo:

CADASTRA cidade nohabitantes (ex. CADASTRA Muqui 13000)
CADASTRA cidade nohabitantes (ex. CADASTRA Cachoeiro 250000)
REMOVE cidade (ex. REMOVE SaoMateus)
CONECTA cidade1 cidade2 (ex. CONECTA Muqui Cachoeiro)
DESCONECTA cidade1 cidade2 (ex. DESCONECTA Muqui Alegre)
NOHABITANTES valor (ex. NOHABITANTES 2000)
ALCANCAVEL cidade1 cidade2 (ex. ALCANCAVEL Muqui Cachoeiro)
IMPRIME_CIDADE cidade (ex. IMPRIME_CIDADE Muqui)
IMPRIME_TUDO (ex. IMPRIME_TUDO)
FIM

Os comandos ALVANCAVEL, NOHABITANTES, IMPRIME_DADOS e IMPRIME_TUDO deverão imprimir as informações em um arquivos de saída, a ser criado pelo seu programa. O nome do arquivo de saída deverá ser *saida.txt*. Todas as mensagens de erro também deverão ser impressas no arquivo saida.txt.

O comando NOHABITANTES imprime em uma linha o número de cidades que têm o número de habitantes maior do que *valor*. O comando ALCANCAVEL imprime em uma linha SIM se for possível, a partir da Cidade1, alcançar a Cidade2 e NAO caso contrário.

O comando IMPRIME_DADOS (cidade) deverá imprimir em uma linha as informações daquela cidade (nome da cidade e número de habitantes), separados por espaço. Todas as informações das cidades conectadas a esta cidade serão impressas nas linhas subsequentes (uma por linha), cada linha contendo o nome da cidade e o número de habitantes. Antes das informações das cidades conectadas, inclua uma linha com a string “Cidades conectadas a ” + nome da cidade. Depois de impressa as informações das cidades conectadas, imprima uma linha em branco. No exemplo anterior, a sequência de comandos IMPRIME_DADOS (Muqui), IMPRIME_DADOS (Cachoeiro), imprime as seguintes linhas no aquivo saida.txt:

```
Muqui 13000
Cidades conectadas a Muqui
Cachoeiro 250000
```

```
Cachoeiro 250000
Cidades conectadas a Cachoeiro
Muqui 13000
```

O comando IMPRIME_TUDO deverá imprimir as informações de todas as cidades do Sistema Rodoviário usando o mesmo formato de saída da função IMPRIME_DADOS (cidade).

O Tad Grafo

O seu programa Rodovias precisará de uma estrutura de dados do tipo grafo, que é implementada através de listas encadeadas, usando o Tad Lista de Arestas e o Tad Lista de Vértice (explicados mais adiante).

Definições do Tad Grafo:

```
# include "TadListaAresta.h"
# include "TadListaVertice.h"

// definições para tratamento de erros
# define OK 0  //define que não houve erros
# define ERRVERTICEINEXISTENTE 1
//defina aqui outros tipos de erros se necessário

typedef int codErro; //identificação de erros
typedef void* TipoItem;
typedef int IdVertice;
typedef struct {
      IdVertice Id;
      TipoItem Item;
      TadListaAresta* listadearestas; // um vértice contém uma lista de arestas
} Vertice;

typedef struct {
      IdVertice IdDestino;    // destino da aresta
      Vertice* Destino; // informação adicional (ponteiro para Vertice)
} Aresta;
typedef TadListaVertice TadGrafo;   // um grafo é representado como uma lista de vértices
```

Algumas funcionalidades que deverão fazer parte do Tad Grafo são (você poderá precisar de outras):

- TadGrafo* CriaGrafo (): Cria um grafo vazio;
- int EstaVazio (TadGrafo* g): Verifica se um grafo está vazio (retornando 0 se estiver vazio e 1 caso contrário);
- IdVertice IncluiVertice (TadGrafo* g , TipoItem item): Inclui um vértice que contém o conteúdo item no grafo. Retorna o identificador único do vértice (contendo a cidade), que é gerado nesta função;
- codErro ExcluiVertice (TadGrafo* g, IdVertice id): Exclui um vértice do grafo. O vértice é identificado pelo seu identificador único;
- codErro ConectaVertices (TadGrafo* g, IdVertice id1, IdVertice id2): Conecta dois vértices de um grafo. Cada vértice é identificado pelo seu identificador único;
- TadListaVertice* RetornaListaVertices(TadGrafo* g): Retorna uma lista com todos os vértices de um grafo;
- TadListaAresta* RetornaListaArestas (TadGrafo* g, IdVertice id): Dado um vértice, retorna a lista de vértices conectados a ele;
- int EstaConectado (TadGrafo* g, IdVertice id1, IdVertice id2): Verifica se um vértice está diretamente conectado a outro, usando os identificadores únicos dos vértices;
- int Alcancavel (TadGrafo* g, IdVertice id1, IdVertice id2): Verifica se Vertice1 é alcançável a partir do Vertice2, ou seja, verifica se há um caminho entre estes vértices (usando seus identificadores únicos);
- void DestroiGrafo(TadGrafo* g): Destrói o grafo, liberando a memória alocada.

O Tad Lista de Vértices

Para implementar o grafo, você precisará de um TAD lista de vértices. Os tipos desse Tad são:

```
# include “TadGrafo.h”
typedef struct celula_str_v* Ponteiro;
typedef struct celula_str_v{
      Vertice* Item;
      Ponteiro Prox;
} CelulaVertice;
typedef CelulaVertice TadListaVertice;
```

Algumas funções do TadListaVertice são apresentadas a seguir (você poderá precisar de outras):

- TadListaVertice* CriaListaVertice(): Cria uma lista de vértices vazia;
- void InsereVertice (TadListaVertice* l, Vertice* v): Insere um vértice no final da lista de vértices;
- int TamanhoListaVertice (TadListaVertice* l): Retorna o tamanho da lista de vértices;
- Vertice* BuscaListaVertice (TadListaVertice* l , IdVertice id): Retorna um vertice da lista de vértices dado o seu identificador único;
- Vertice* RetornaVertice (int i): Retorna o i-ésimo vértice da lista;
- TadListaVertice* RetornaProxVertice (TadListaVertice* l): Retorna uma sublista de l cujo primeiro elemento é l->Prox (use ponteiros, não faça cópias);
- void RemoveVertice (TadListaVertice* l, IdVertice id): Remove o vértice da lista de vértices dado o seu identificador;
- void DestroiListaVertice (TadListaVertice* l): Destrói a lista, liberando a memória alocada para cada célula.

O Tad Lista de Arestas

Para implementar o grafo, você precisará de um TAD lista de arestas. Os tipos desse Tad são:

```
# include “TadGrafo.h”
typedef struct celula_str_a* Ponteiro;
typedef struct celula_str_a{
      Aresta* Item;
      Ponteiro Prox;
} CelulaAresta;
typedef CelulaAresta TadListaAresta;
```

As funções do TadListaAresta são análogas às do TadListaVertice.

BOM TRABALHO!