Universidade Estadual do Oeste do Paraná
Bacharelado em Ciência da Computação

Atividade Prática 2 - Documentação

Professor:
Leonardo Medeiros

Alunos:
Davi Marchetti Giacomel
Giuliano Girlando Assenza de Albuquerque
Jaqueline Cavaller Faino

Cascavel, Paraná
25/07/2022

Módulos (e sua interdependência)

- Grafo

  A implementação de grafos e seus principais conceitos consta com os seguintes módulos:

  1. **node.h** (TADs e assinaturas de funções para o tipo de nó utilizado para o grafo)
  2. **node.c** (implementação das funções assinadas em node.h)
  3. **list.h** (TADs e assinaturas de funções para a Estrutura de Dados *Lista Simplesmente Encadeada*, a qual foi frequentemente utilizada na implementação de grafo)
  4. **list.c** (implementação das funções assinadas em list.h)
  5. **grafo.h** (TADs e assinaturas de funções para a Estrutura de Dados *Grafo*)
  6. **grafo.c** (implementação das funções assinadas em grafo.h)
  7. **main.c** (arquivo principal que se aproveitará dos métodos implementados nos módulos anteriores, a fim de demonstrar seu funcionamento)

- Fila com herança de Lista Encadeada Simples

A implementação do TAD *Fila* com herança de *Lista Encadeada Simples* em .c++ foi desenvolvida com os seguinte módulos:

  1. **List.hpp** (classe com getters, setters e métodos específicos de manipulação de lista)
  2. **Node.hpp** (classe simples, com getters e setters para todos os atributos e três construtores, que inicializam nenhum, um ou os três atributos da classe por parâmetro)
  3. **Queue.hpp** (Utiliza os atributos e métodos herdados da classe List. implementa apenas o método dequeue, enqueue e printQueue, que na prática são removeFirst, append e printList)
  4. **main.cpp** (código principal utilizado para demonstrar as funcionalidades implementadas anteriormente)

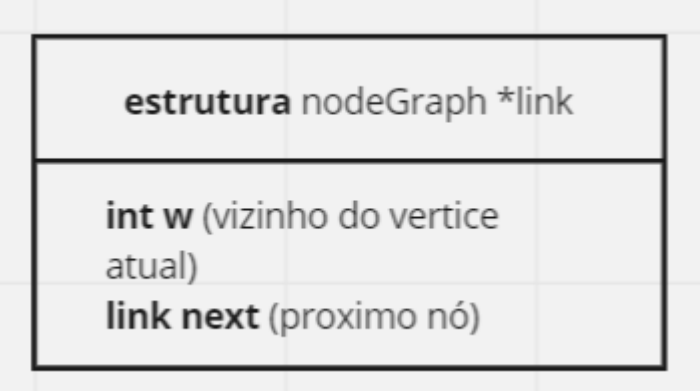
- Método de ordenação ShellSort

A implementação do método de ordenação *ShellSort* foi desenvolvido com os seguintes módulos:

  1. **array.util.h** (TADs e assinaturas de funções para o tipo *vetor*)
  2. **array.util.c** (implementação das funções assinadas em array.util.h, incluindo o próprio método de ordenação *ShellSort*)
  3. **main.c** (arquivo principal que utiliza os métodos implementados nos arquivos anteriores, a fim de demonstrar seu funcionamento)

TADs e Estruturas

- Grafo
    1. node.h



Funções:

link newNode(**vertice** w, **link** next); ***vertice = typedef int vertice*

Cria um novo nó de acordo com a estrutura

2. list.h

***tdata = typedef int tdata*

Funções:

void initList(**list** *L);

Inicia a lista passada como parâmetro

void deleteList(**list** *L);

Libera o espaço de memória ocupado pela lista passada como parâmetro

void printList(**list** L);

Imprime o conteúdo da lista passada como parâmetro

void insertLeft(**tdata** x, **list** *L); **tdata = typedef int tdata*

Insere o elemento x à esquerda da lista passada como parâmetro

void insertRight(**tdata** x, **list** *L);

Insere o elemento x à direita da lista passada como parâmetro

bool emptyList(**list** L);

Verifica se a lista passada como parâmetro está vazia

int removeLeft(**list** *L);

Remove o elemento mais à esquerda da lista passada como parâmetro (e o retorna)

int removeRight(**list** *L);

Remove o elemento mais à direita da lista passada como parâmetro (e o retorna)

node* searchList(**tdata** x, **list** L);

Pesquisa um dado x na lista passada como parâmetro e caso o encontre retorna o nó que contém esse dado

void insertPos(**tdata** x, **unsigned** pos, **list** *L);

Insere na posição especificada em *pos* o dado x na lista passada como parâmetro

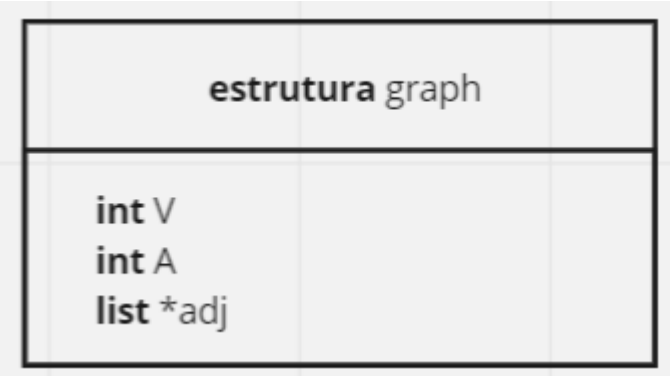
int removePos(**unsigned** pos, **list** *L);

Remove o nó localizado em *pos* na lista passada como parâmetro

void removeSpec(**tdata** x, **list** *L);

Procura na lista o dado x e o remove-o, se existir, da lista passada como parâmetro

3. grafo.h



*typedef struct graph *Graph* (ponteiro da estrutura graph)

Funções:

Graph graphInit(**int** V);

Inicializa um grafo com V vértices e retorna-o

void graphInserirAresta(**Graph** G, **vertice** v, **vertice** w);

Insere uma aresta no grafo G entre os vértices v e w

void graphExcluirAresta(**Graph** G, **vertice** v, **vertice** w);

Exclui a aresta entre os vértices v e w no grafo G

void reachR(**Graph** G, **vertice** v, **int** *visited);

Função recursiva auxiliar para a função graphReach

bool graphReach(**Graph** G, **vertice** s, **vertice** t);

Verifica se o vértice t está ao alcance do vértice s do grafo G.

list* buscaProfundidade(**Graph** G);

Função que visita todos os vértices do grafo G e enumera-os na ordem em que são visitados.

list* buscaLargura(**Graph** G, **vertice** s);

Algoritmo de busca no grafo G, onde o vértice inicial é s, onde é visitado o vértice s, em seguida seus vizinhos, depois os vizinhos dos vizinhos, e assim por diante.

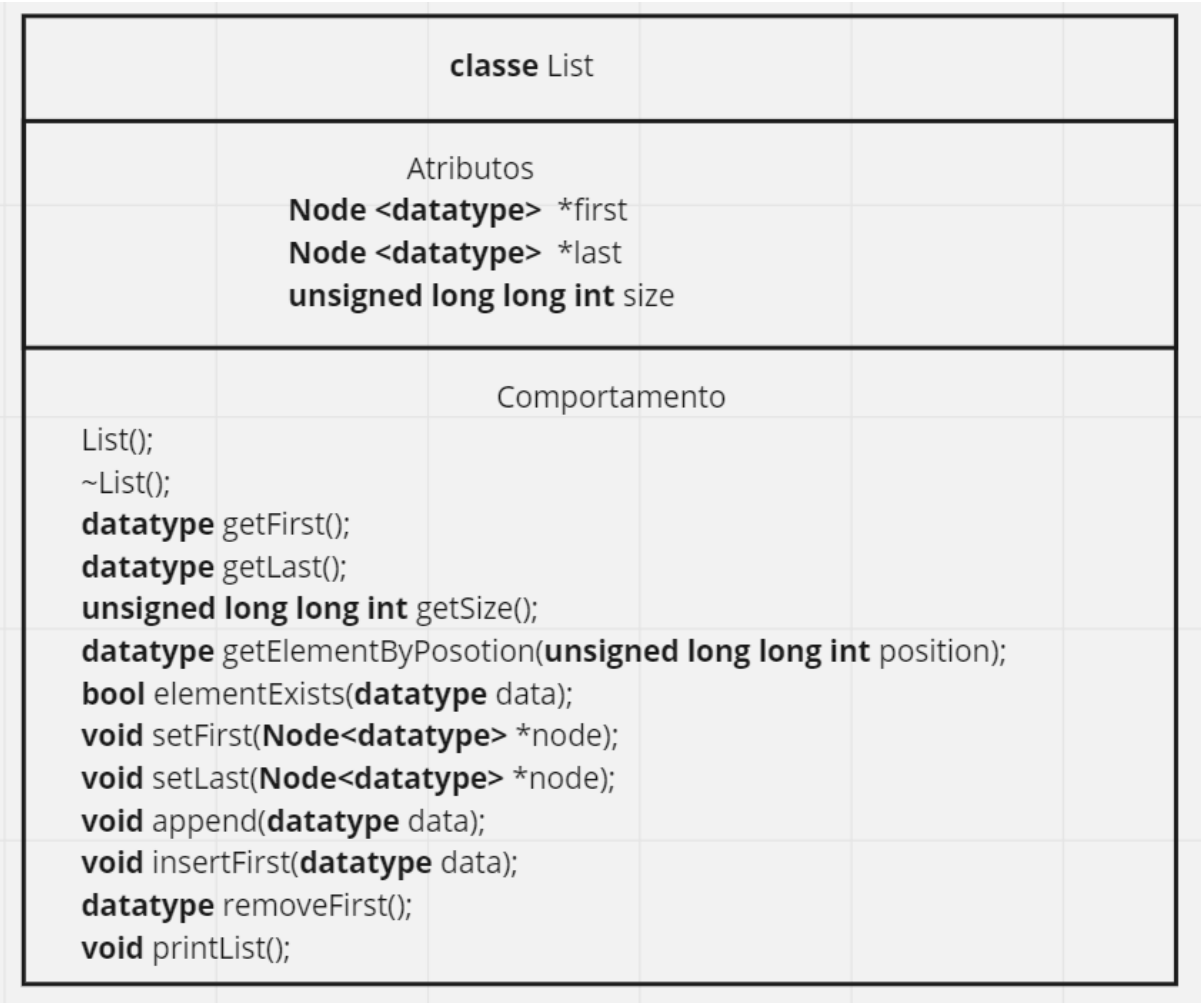
void graphImprime(**Graph** G);

Função que imprime todas as listas de adjacência do grafo G.

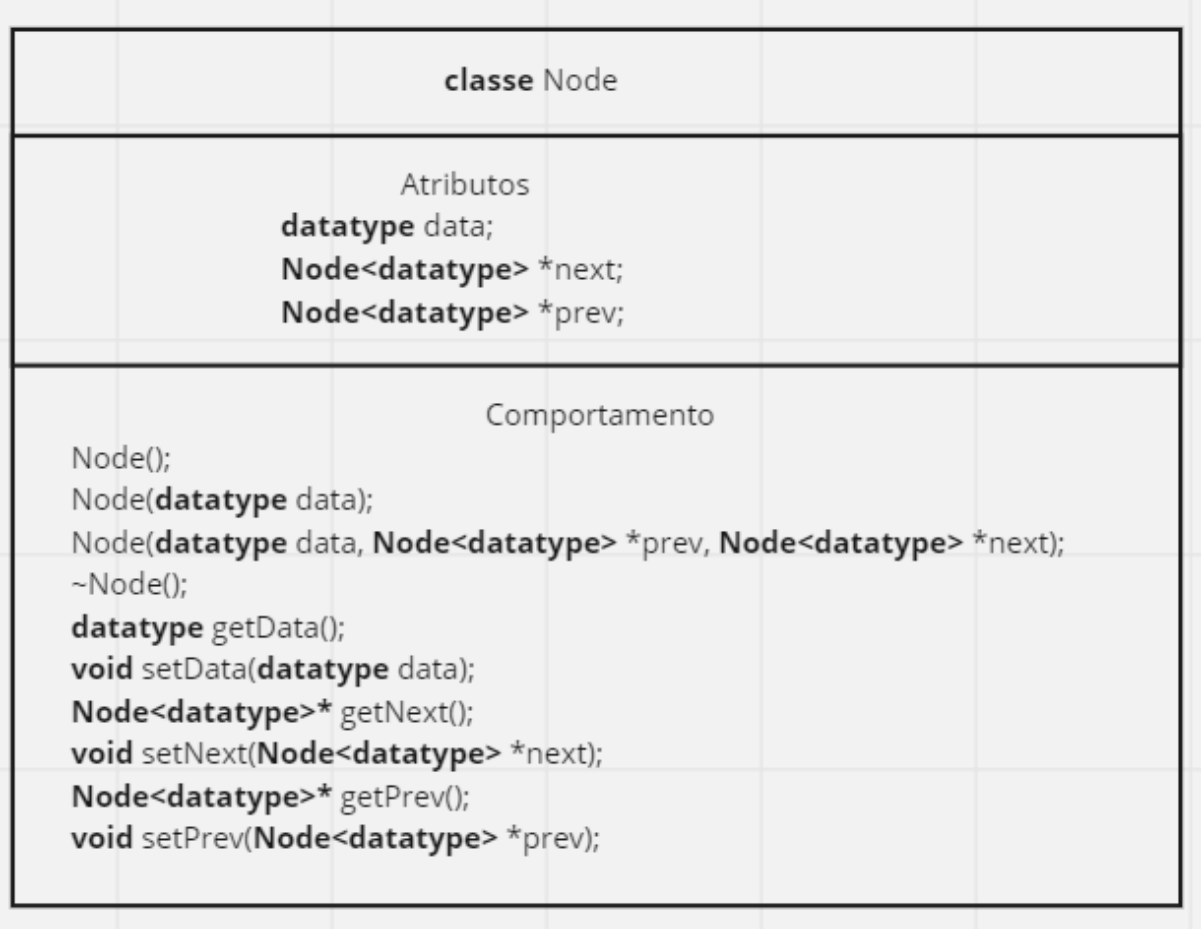
void graphDeleta(**Graph** G);

Função que libera o espaço de memória ocupado pelo grafo G.

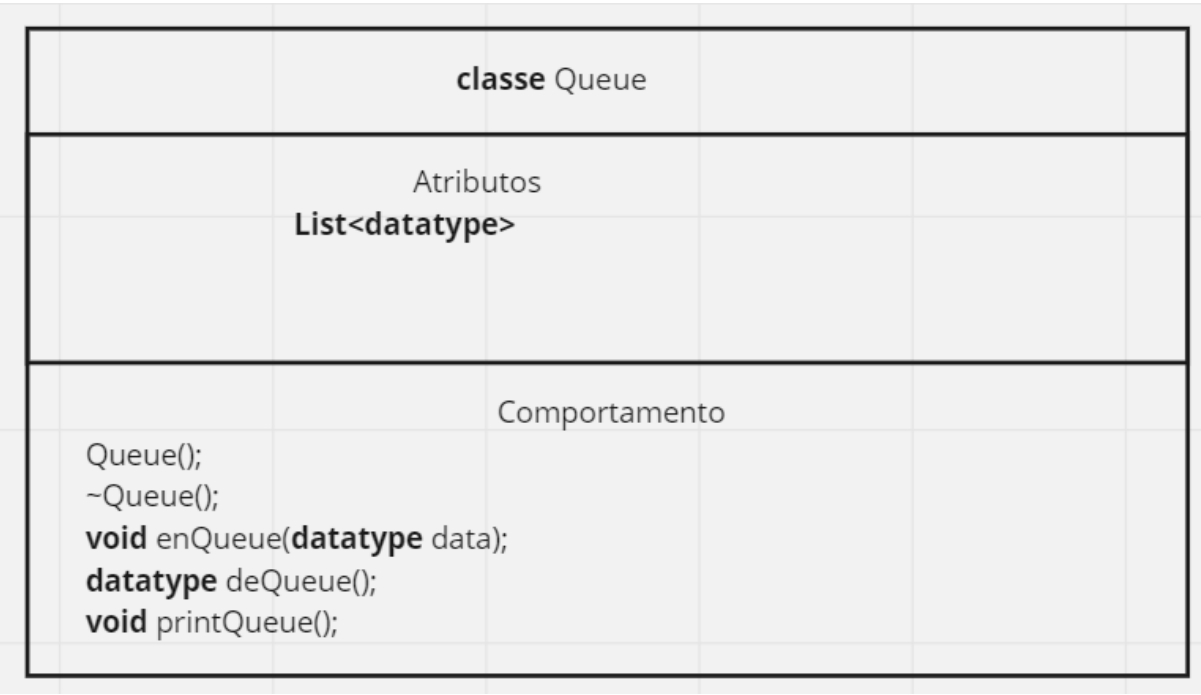
- Fila com herança de Lista Encadeada Simples
    1. List.hpp



2. Node.hpp



3. Queue.hpp



- Método de ordenação ShellSort

1. array.util.h
   Funções:
   int* allocateArray(**int** *array, **int** size);
   Aloca um espaço na memória para o vetor *array de tamanho size.

   void destructArray(**int** *array);
   Libera o espaço na memória ocupado por *array;

   void printArray(**int** *array, **int** size);
   Imprime na saída padrão os elementos de um array entre colchetes com os elementos separados por vírgula

   void shellSort(**int** *a, **int** n);
   Ordenação através do método shellSort

   void readArray(**int** *array, **int** n);
   Lê valores do vetor pelo terminal

Entradas

- Grafo

  O primeiro dado solicitado pelo programa principal é a quantidade de vértices do grafo, pois a partir disso cria as listas de adjacência necessárias para cada vértice.

  Posteriormente, é apresentado um menu de opções, as quais são:

  1. 1 = Inserir Aresta (parâmetros: dois vértices entre os quais a aresta será criada)
  2. 2 = Excluir Aresta (parâmetros: dois vértices cuja aresta será excluída)
  3. 3 = Imprimir Grafo
  4. 4 = Busca em Profundidade
  5. 5 = Busca em Largura (parâmetro: vértice pelo qual iniciará a busca em largura).
  6. 100 = Sair do menu e do programa

- Fila com herança de Lista Encadeada Simples
  Inicialmente, não há entradas de usuário para este módulo, uma vez que os testes estão sendo realizados diretamente no código (*hard-coded*).

- Método de ordenação ShellSort

É solicitado como entrada, inicialmente, o tamanho do vetor que será alocado na memória. Em seguida, solicita que o usuário preencha os espaços do vetor (o que pode ser feito com um arquivo.txt se especificado no terminal).

Saídas

- Grafo

  As saídas para as possíveis entradas, listadas anteriormente, são:

  1. 1 = Inserir Aresta (não possui saída explícita, mas insere os dados na lista de adjacência dos vértices informados)
  2. 2 = Excluir Aresta (também não possui saída explícita, mas exclui a aresta dos vértices informados)
  3. 3 = Imprimir Grafo (imprime as listas de adjacência do grafo)
  4. 4 = Busca em Profundidade (imprime a enumeração da ordem em que os vértices foram visitados)
  5. 5 = Busca em Largura (imprime a ordem em que os vértices foram descobertos a partir do vértice informado)
  6. 100 = "até mais!" e finaliza o programa.

- Fila com herança de Lista Encadeada Simples

  Primeiramente, a saída imprime a Fila (após inserções feitas no código), e a cada retirada de elemento na fila, é informado "dequeued: %valor", onde %valor é o valor do elemento retirado da fila.

- Método de ordenação ShellSort

  Após a leitura do vetor e seus elementos, o programa imprime o vetor desordenado (na ordem que o usuário informou), executa a ordenação *shellSort* e imprime, novamente, o vetor **ordenado**.

Explicação

- Grafo

O programa irá manipular o grafo (inicializado com a quantidade de vértices informada pelo usuário) de acordo com as opções escolhidas pelo usuário (opções listadas nos tópicos de Entrada e Saída).

- Fila com herança de Lista Encadeada Simples

O programa aplica os métodos básicos da estrutura de dados Fila, com base na estrutura Lista, inicialmente em modo *hard-coded*.

- Método de ordenação ShellSort
  O programa irá ler um vetor de tamanho **n**, sendo **n** informado pelo usuário assim como os elementos do vetor, e irá ordená-lo com o método de ordenação *ShellSort*.

Utilização (do programa)

- Grafo
  Para utilizar o programa basta executar *make run* (em sistemas Linux), o qual possui entrada e saída automáticas por arquivo, ou executar o arquivo gerado pelo make run e informar as entradas solicitadas e/ou escolher as opções do menu apresentado.

- Fila com herança de Lista Encadeada Simples
  Para compilar e executar o programa, enviando a saída para um arquivo nomeado saida.txt, basta executar o comando *make run* (em sistemas Linux) na pasta onde se encontram os códigos deste método.

- Método de ordenação ShellSort
  Para utilizar o programa basta executar *make run* (em sistemas Linux), o qual possui entrada e saída automáticas por arquivo, ou executar o arquivo gerado e informar as entradas solicitadas.

Complexidade

- Grafo
  1. Busca em Profundidade
     a. Cada arco é examinado apenas uma vez. Portanto, considerando que o grafo contém V vértices e A arcos, a Busca em Profundidade tem complexidade O(**V** + **A**) no pior caso.
  2. Busca em Largura
     a. Durante o laço *while* principal do método, cada arco do grafo é percorrido no máximo uma vez. Ou seja, no pior caso, o tempo de execução das iterações é proporcional ao valor de **A** arestas do grafo, enquanto o restante do método leva tempo proporcional ao valor de **V** vértices do grafo.

Portanto, a complexidade da Busca em Largura seria equivalente a O(**V** + **A**) no pior caso, sendo o grafo representado com listas de adjacência.

- Fila com herança de Lista Encadeada Simples
  A complexidade dos métodos de uma Fila são de complexidade extremamente simples, representada de forma:
  T(n) = O(1).

- Método de ordenação ShellSort
  1. O próprio método de ordenação *ShellSort* possui a seguinte complexidade, em relação ao número de comparações para a sequência de Knuth:
     a. Conjectura 1: $C(n) = O(n^{1,25})$
     b. Conjectura 2: $C(n) = O(n * (ln\ n)^2)$
     c. Obs.: sendo **n** o número de comparações.

Listagem dos testes executados

- Grafo
  - Ao realizar o *make run*, automaticamente é executado um teste utilizando como entrada o arquivo entradaTeste.txt, e enviado a saída no arquivo saida.txt. O teste se trata de um grafo não ponderado e não direcionado de 10 vértices. Na saída consta a impressão do grafo e suas listas de adjacência, busca em profundidade no grafo a partir do vértice 0 e busca em largura também a partir do vértice 0.

- Fila com herança de Lista Encadeada Simples
  - São inseridos quatro elementos quaisquer, e depois é tentado retirar 6 vezes, ao ponto que, quando não há o que retirar, o programa notifica um aviso e não manipula a fila. Os resultados do teste são direcionados diretamente para o arquivo saida.txt.

- Método de ordenação ShellSort
  - Ao realizar o *make run*, automaticamente é executado um teste utilizando como entrada o arquivo teste.txt, e é enviada a saída para o arquivo saida.txt. Onde é realizado um teste de melhor e pior caso, respectivamente, e imprime o tempo levado para cada um deles.