UNIVERSIDADE FEDERAL DE MINAS GERAIS

Departamento de Ciência da Computação

Trabalho Prático 1

Estrutura de Dados

Ordenador Universal

Nome: Anny Caroline Almeida Marcelino

Matrícula: 2024024046

Email: annycaroline@ufmg.br

Sumário

1. Introdução

A empresa Zambs deseja lançar um **Tipo de Abstrato de Dados Ordenador Universal** com a proposta de selecionar o algoritmo de ordenação mais eficiente para diferentes vetores.

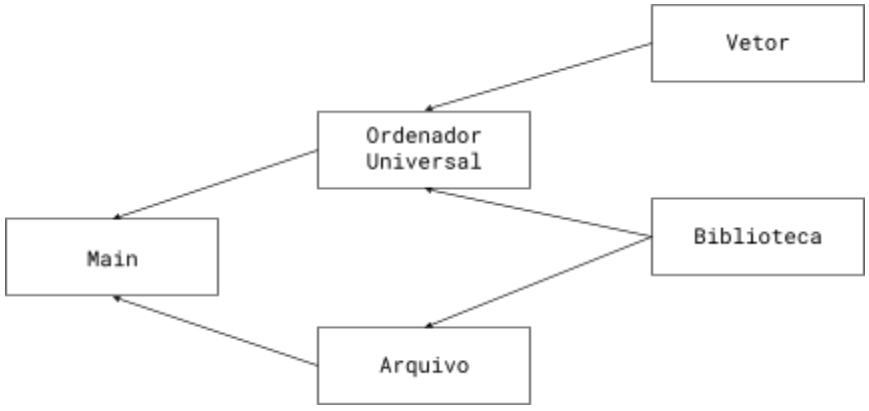
O TAD é capaz de decidir entre os algoritmos **insertion sort** e **quicksort com mediana de três**, com base em dois parâmetros :

- **Grau de ordenação inicial do vetor**, medido pelo número de "quebras". Em que o Insertion sort tende a ser mais eficiente para vetores com menos quebras.
- **Tamanho da partição no quicksort**, que define até que ponto é vantajoso usar o quicksort.

O processo de escolha destes parâmetros é feito por meio da análise comparativa dos custos há cada iteração para diferentes configurações de ordenação. Esse custo é calculado por uma função, que combina três métricas: número de comparações, movimentações e chamadas, de modo que os coeficientes usados nessa função são definidos previamente. Ao final de cada iteração, os resultados são registrados e utilizados para refinar a tomada de decisão do TAD.

2. Implementação

A implementação do sistema segue o planejamento realizado com base no fluxograma a seguir.



Fluxograma da implementação com TADs

2.1. Vetor

A classe template **Vetor<T>** implementa um vetor estático genérico.

Atributos

- **T* array:** Ponteiro para o array que armazena os elementos do tipo T.
- **Estatistica est:** Objeto usado para registrar estatísticas durante as operações realizadas no vetor.

Construtores e Destrutor

- **Vetor(int tamanho_):** Aloca dinamicamente o vetor com o tamanho especificado.
- **Vetor(const Vetor<T>& copia):** Construtor de cópia. Cria um novo vetor com os mesmos valores de outro objeto da classe Vetor.
- **~Vetor():** Destrutor. Libera a memória alocada dinamicamente para o vetor.

Funções

- **T& operator[](int i):** Sobrecarga do operador de índice. Permite acessar e modificar diretamente os elementos do array.
- **int newShuffle(int numShuffle, int seed):** Aplica um número definido de quebras ao vetor.
- **void Copia(const Vetor<T>& original):** Copia o conteúdo de outro vetor para o vetor atual, desde que ambos tenham o mesmo tamanho. **int getTamanho():** Retorna o tamanho do vetor.

Observações de Implementação

A classe foi projetada a fim de possibilitar melhor manipulação dos dados, como acesso direto aos elementos do array, armazenamento das estatística do vetor e suporte a cópias. O uso de templates permite a criação de vetores de qualquer tipo de dado primitivo.

2.2. Biblioteca

Conjunto de funções auxiliares e estruturas que não pertencem diretamente a uma classe específica, mas oferecem suporte à lógica geral.

Estruturas

Faixa: Armazena os dados de custo de uma execução.

Atributos

- **int limiar:** valor do limiar utilizado na execução.
- **double custo:** custo total da execução (baseado em coeficientes).
- **int posc:** número da iteração de definição do limiar na execução (numMPS, numQUB)

Métodos

- **void registra(int l, double c, int m):** Armazena os valores do limiar, custo e posc.

Parametros: Estrutura usada para armazenar os parâmetros de configuração lidos do arquivo.

Atributos

- **int seed:** semente para aleatoriedade.
- **double limiarCusto:** limiar usado para decisão de troca de algoritmo.
- **double coef_cmp, coef_mov, coef_call:** coeficientes para definição do custo.
- **int tam:** tamanho do vetor.

Métodos

- **template <typename T> void SetParametro(int posc, T valor):** Armazena os valores dos parâmetros.

Estatistica: Estrutura para armazenar o número de comparações, movimentações e chamadas de função.

Atributos

- **int cmp:** número de comparações realizadas.
- **int move:** número de movimentações de dados.
- **int calls:** número de chamadas de função.

Métodos

- **void reset():** Zera todos os contadores.
- **void addcmp(int num):** Incrementa o contador de comparações.
- **void addmove(int num):** Incrementa o contador de movimentações.
- **void addcalls(int num):** Incrementa o contador de chamadas de função.

Funções

- **swap(T *xp, T yp, Estatistica est):** Troca dois elementos de lugar no vetor e registra 3 movimentações.
- **insertionSort(T v, int l, int r, Estatistica est):** Algoritmo de ordenação no intervalo [l,r], em que cada elemento é comparado com os anteriores até encontrar sua posição correta. Nesse processo, os elementos maiores que ele são deslocados para a direita.
- **mediana(T a, T b, T c):** Retorna o valor mediano entre três valores.
- **partition3(T *A, int l, int r, int *i, int j, Estatistica est):** Realiza o particionamento do vetor em três regiões e utiliza a mediana para definir o pivô.

- **quickSort(T A, int l, int r, int limiar, Estatistica est):** Algoritmo de ordenação que dependendo do tamanho da partição utiliza quickSort com partição 3 ou insertionSort para ordenar.

2.3. Ordenador Universal

A classe **OrdenaUniversal<T>** implementa um sistema de ordenação genérico que utiliza limiares para decidir o algoritmo de ordenação que melhor otimiza o custo. Ela gerencia limiares de partição de de quebra, calculando-os através das estatísticas de execução.

Construtor e Destrutor

- **OrdenaUniversal(Parametros* pd_, Vetor<T>* vetor):**Inicializa o objeto com os parâmetros lidos de um arquivo e o vetor a ser ordenado. Os limiares são inicialmente indefinidos.
- **~OrdenaUniversal():** Destrutor padrão.

Métodos

- **double custo(const Estatistica* est):** Calcula o custo total de uma ordenação com base nas estatísticas e os coeficientes definidos nos parâmetros.
- **void ordenadorUniversal(T* vetor, int limiarParticao, int tam, int limiarQuebras, int quebras, Estatistica* est):** Função principal que executa a ordenação, decidindo qual algoritmo usar com base nos limiares.
- **void orderna(T* vetor, int limiar, int parametro, Estatistica* est, char tipo):** Executa apenas um algoritmo de ordenação da função ordernadorUniversal.
- **void determinaLimiarParticao():** Define o limiar de partição, com base na comparação dos custos de vetores ordenados com diferentes tamanhos de partição usando o quicksort.
- **void calculaNovaFaixa(int& minMPS, int& maxMPS, int& passoMPS, int posclimiar, float& diff_custo, int hist_add, const Faixa* est_custo):** Ajusta os valores mínimo, máximo e passo da faixa de testes com base nos registros de estatísticas de vetores ordenados anteriormente.
- **void determinaLimiarQuebra():** Define o limiar de quebra, com base na comparação dos custos de vetores ordenados com diferentes quantidade de quebras usando o insertion sort.
- **void calculaNovaQuebra(int& minQUB, int& maxQUB, int& passoQUB, int posclimiar, float& diff_custo, int hist_add, const Faixa* est_custo):** O mesmo funcionalidade do **calculaNovaFaixa**.
- **int numQuebras():** Retorna o número atual de quebras no vetor.
- **string Escrita(const string& texto):** Retorna a string passada como parâmetro.
- **string RegistraEstatistica(const string& tag, int numLimiar, double custo, const Estatistica& est):** Gera uma string com as estatísticas completas de uma execução formatadas.
- **string RegistraEstConlusoes(const string& tag, const string& taglim, int numLimiar, int limiar, double diff):** Gera uma linha de conclusão com o

limiar escolhido e a diferença percentual de custo entre os métodos comparados, formatada.

Atributos

- **Parametros* paramet:** Ponteiro para os parâmetros carregados do arquivo.
- **int limiarParticao:** Limiar para escolha entre quicksort e insertion sort com base no tamanho da partição.
- **int limiarQuebras:** Limiar para escolha entre algoritmos com base no número de quebras no vetor.
- **int numMPS, numQUB:** Contadores nos testes de partição e quebras.
- **int quebras:** Número atual de quebras quantificadas no vetor.
- **Vetor<T>* vet:** Ponteiro para o vetor genérico que será ordenado.

2.4. Arquivo

A classe **Arquivo** gerencia operações básicas de leitura e escrita em arquivos de texto, com foco no manuseio de parâmetros de configuração armazenados em arquivos.

Construtor e Destrutor

- **Arquivo(string caminho_, bool novo_arq):** abre o arquivo no caminho especificado para leitura. Se o arquivo não existir e o parâmetro novo_arq for verdadeiro, cria um arquivo vazio.
- **~Arquivo():** garante o fechamento do arquivo ao destruir o objeto.

Funções

- **void leParametros(int n_paramt, Parametros* pt_):** Lê 7 linhas do arquivo e armazena os valores lidos na struct Parametros.
- **fstream& getArquivo():** retorna a referência ao objeto fstream associado ao arquivo aberto.
- **bool verificaArquivo():** verifica se o arquivo está aberto.
- **void fechaArquivo():** fecha o arquivo, caso esteja aberto.
- **void Registra(string caminho_, ostringstream& oss):** abre o arquivo no e grava o conteúdo, adicionando uma nova linha.

3. Análise de Complexidade

insertionSort: Complexidade de espaço: O(1). Complexidade de tempo: Melhor caso: O(n), para vetores ordenados/quase ordenados. Pior caso: $O(n^2)$, para vetores inversamente ordenados.

quickSort: Complexidade de espaço: O(1). Complexidade de tempo: Melhor caso: O(n log n), quando as partições são bem balanceadas e o pivô divide o vetor aproximadamente ao meio. Pior caso: $O(n^2)$, para vetores inversamente ordenados.

ordenaUniversal: Complexidade de espaço: O(1). Complexidade de tempo: Melhor caso: O(n), o número de quebras é menor que o limiar de quebras, utiliza o insertionSort para ordenar. Pior caso: O(n log n), utiliza o quicksort para ordenar.

determinaLimiarParticao: Complexidade de espaço: O(n), faz uma cópia do vetor. Complexidade de tempo: O(n log n) * O(n), o produto da complexidade do quicksort e da cópia do vetor.

determinaLimiarQuebra: Complexidade de espaço: O(n), faz uma cópia do vetor. Complexidade de tempo: O(n) * O(n), o produto da complexidade do insertion e da cópia do vetor.

Copia Vetor: Complexidade de espaço: O(1). Complexidade de tempo: O(n), percorre todos elementos para copiá-los.

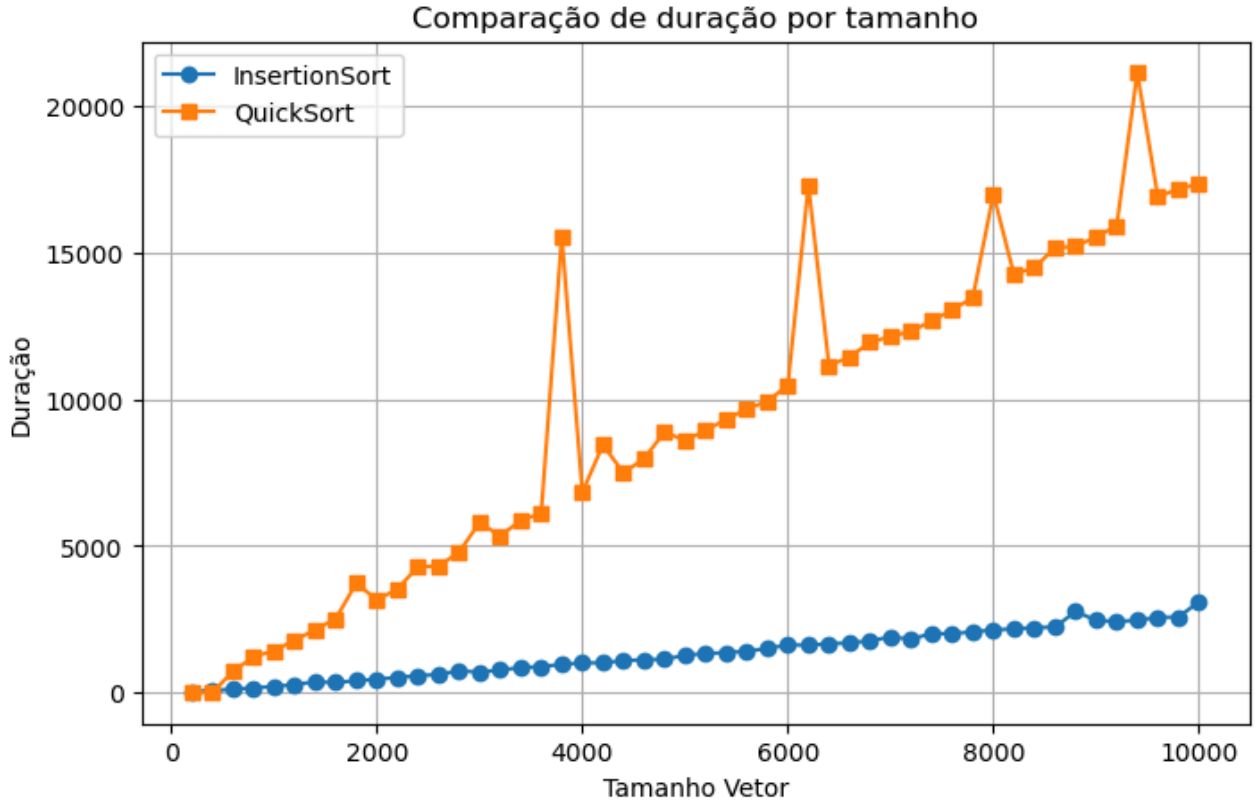
Construtor Cópia Vetor: Complexidade de espaço: O(n), aloca o vetor. Complexidade de tempo: O(n), percorre todos elementos para copiá-los.

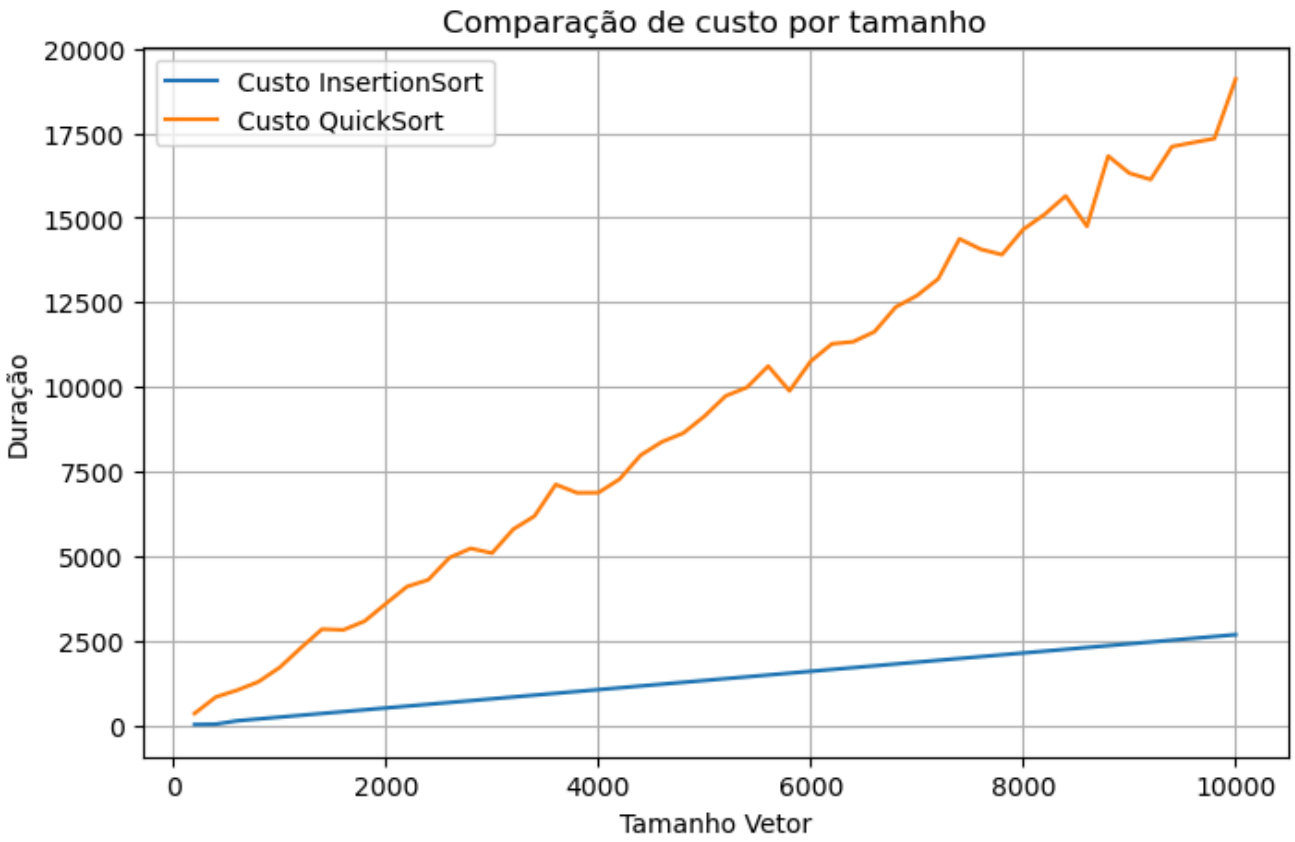
4. Análise Experimental

Coeficientes definidos por meio de regressão linear múltipla, das variáveis de duração e das estatísticas: comparações, movimentações e chamadas de função.

a = -0.0440 b = 0.0451 c = 18.5731

Para vetores de tamanho entre 200 a 10000 e duração calculada em microssegundo.





***custo calculado através dos coeficientes definidos na regressão linear.*