­**UNIVERSIDADE FEDERAL DE MINAS GERAIS**

**DCC | SISTEMAS DE INFORMAÇÃO | ALGORITMOS I**

**DOCUMENTAÇÃO**

**TRABALHO PRÁTICO 1 | BLACKJACK**

**Rômulo Rafael da Silva, 2012055308**

**Setembro / 2019**

Sumário

[1. DESCRIÇÃO DO PROBLEMA 3](#_Toc20606882)

[2. DESCRIÇÕES TÉCNICAS 4](#_Toc20606883)

[2.1. HARDWARE 4](#_Toc20606884)

[2.2. SOFTWARE 4](#_Toc20606885)

[2.3. LINGUAGEM DE PROGRAMAÇÃO 4](#_Toc20606886)

[3. ESTRUTURAS DE PROJETO E ALGORITMOS 4](#_Toc20606887)

[3.1. DIRETÓRIO DO PROJETO 4](#_Toc20606888)

[3.2. ESTRUTURA DE DADOS 5](#_Toc20606889)

[3.3. ALGORITMOS 5](#_Toc20606890)

[4. ANÁLISE DE COMPLEXIDADE 5](#_Toc20606891)

[4.1. TEMPO 6](#_Toc20606892)

[4.2. ESPAÇO 7](#_Toc20606893)

[5. AVALIAÇÃO EXPERIMENTAL 7](#_Toc20606894)

[5.1. Q1 7](#_Toc20606895)

[5.2. Q2 8](#_Toc20606896)

[5.3. Q3 8](#_Toc20606897)

[Bibliografia 9](#_Toc20606898)

# 1. DESCRIÇÃO DO PROBLEMA

Esta documentação é referente ao trabalho prático 1 da disciplina de Algoritmos 1 do curso de Sistemas de Informação da UFMG.

A motivação do trabalho foi baseada em um jogo denominado Blackjack: um jogo de azar praticado com cartas em casinos e que pode ser jogado com 1 a 8 baralhos de 52 cartas, em que o objetivo é ter mais pontos do que o adversário, mas sem ultrapassar os 21 (caso em que se perde) [1].

A proposta do trabalho apresentava um cenário hipotético no qual havia um grupo composto por alunos de Matemática, Engenharia e cursos de Computação da UFMG. O grupo se organizava para realizar um estudo aprofundado do livro [Beat the Dealer](https://pt.scribd.com/doc/92066804/Beat-the-Dealer-Edward-O-Thorp), de Edward Oakley Thorp, e de técnicas do jogo. O objetivo é de realizar uma viagem para Las Vegas para ganhar muito dinheiro nos casinos sem serem pegos em flagrante, utilizando as técnicas de Blackjack: maximizar as chances de ganhos sem serem suspeitos ou flagrados pelos cassinos utilizando tais técnicas.

Nesse cenário, o grupo foi organizado em subgrupos por áreas de conhecimento: (1) alunos de Matemática propuseram a formação de subequipes de tal forma que haveria um representante por cada equipe; (2) os de Engenharia sugeriram que a cada certo período, a estrutura hierárquica da equipe fosse remodelada, favorecendo rotatividade de responsabilidades entre os membros da equipe e, por fim, (3) aos estudantes de Computação foi designado desenvolver um programa para realizar três tipos de instruções.

As instruções, as quais serão detalhadas mais adiante, e a respectiva implementação delas que são a parte central do trabalho. As seções seguintes estão assim estruturadas:

* descrições técnicas sobre hardwares, softwares e linguagem de programação utilizados ao longo do desenvolvimento do trabalho;
* explicações das estruturas de dados e dos algoritmos implementados como parte da solução do problema;
* apresentação das análises de complexidade, espaço e tempo, da solução;
* resultados dos experimentos a partir de testes utilizando os datasets disponibilizados pelos monitores da disciplina.

# 2. DESCRIÇÕES TÉCNICAS

## 2.1. HARDWARE

O desenvolvimento e testes do programa foram em um dispositivo cujas configurações são: (1) Intel® Core™ i5-7200U CPU 2.50GHz 2.71GHz, (2) 8GB de memória RAM, (3) Sistema Operacional Windows Pro 10 x64.

## 2.2. SOFTWARE

Os programas utilizados foram: (1) Visual Studio Code, editor de código-fonte desenvolvido pela Microsoft. Para criar a estrutura do projeto do programa, utilizou-se a extensão do VS (2) Easy C++ projects.

## 2.3. LINGUAGEM DE PROGRAMAÇÃO

Programa desenvolvido em linguagem C++, com utilização somente das bibliotecas padrões disponíveis até a versão C++17 (2014). Não são necessárias bibliotecas adicionais, de terceiros e/ou que exijam a instalação para compilação e execução do programa.

# 3. ESTRUTURAS DE PROJETO E ALGORITMOS

## 3.1. DIRETÓRIO DO PROJETO

A estrutura dos projetos provê:

* *dataset:* estão os arquivos de entrada (subpasta input) disponibilizados no ambiente Moodle da disciplina e os respectivos arquivos de saída (subpasta ouput);
* *docs*: constam os arquivos do tipo documento (.pdf, .doc etc.) disponibilizado no ambiente Moodle da disciplina e produzidos ao longo do trabalho como, por exemplo, a documentação do trabalho;
* *include*: constam todos os arquivos de header (.h) que possuem as classes, atributos e assinatura dos métodos implementados;
* *src*: constam todos os arquivos de implementação/corpo (.cpp) que possuem o código dos métodos assinados em .h, bem como o arquivo main.cpp
* *pasta raiz*: contém os outros arquivos como, por exemplo, *makefile* e arquivos do GitHub (.gitattributes, gitignore e README.md)

## 3.2. ESTRUTURA DE DADOS

Para a solução do problema, foram utilizadas as seguintes classes e respectivas bibliotecas:

|  |  |  |  |
| --- | --- | --- | --- |
| **Classe** | **Objeto** | **Biblioteca** | **Descrição** |
| **<vector>** | vector | STL | Container que encapsula arrays de tamanho dinâmico. Gerencia alocação e realocação de memória para os elementos, que são armazenados contiguamente. [2] |
| **<Grafo.h>** | grafo | Implementada | Implementa o tipo abstrato de dados referente à grafo. |
| **<Blackjack.h>** | blackjack | Implementada | Implementa os métodos necessários para as operação de swap, commander e meeting, verificação de ciclo e impressão da saída. |

A seguir, são apresentados os conceitos de estruturas de dados utilizados para;

|  |  |  |  |
| --- | --- | --- | --- |
| **Classe** | **Conceito** | **Objeto** | **Descrição** |
| **<Grafo.h>** | Vértices | struct | Os vértices foram definidos como uma estrutura de três atributos: (1) int valor, (2) string nome e (3) vector<int> listaAdj. Segue o conceito de lista de adjacência. |
| **<Grafo.h>** | Lista de Adjacência | vector | Para cada vértice existente no grafo, implementa um vector (array dinâmico) com a lista de todos os vértices que ele conecta |

## 3.3. ALGORITMOS

Como parte da solução deste problema, recorreu-se ao entendimento e implementação do(s) seguinte(s) algoritmo(s):

* Busca em profundidade (depth-first search – DFS): esse algoritmo é essencial para determinar algumas propriedades de um grafo dirigido. A ordem de busca em profundidade é útil para encontrar ciclos e componentes fortes de maneira eficiente 🡪 implementado na classe Blackjack, no método verificaCiclo (int origem, int vértice, bool inicio).

# 4. ANÁLISE DE COMPLEXIDADE

Quanto à complexidade das classes, abaixo um quadro-resumo com a quantidade de métodos implementados:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Classe** | **Nome** | **Tipo** | **Atributos** | **Métodos** | | |
| **public** | **protected** | **private** |
| <Grafo.h> | Vertice | struct | 3 | - | - |  |
| <Grafo.h> | Grafo | class | - | 13 | 7 | 1 |
| <Blackjack.h> | Blackjack | class | - | 6 | 5 | 0 |

A classe Blackjack herda atributos e métodos da classe Grafo. O quadro, então, apresenta os valores em números absolutos, sem consideração das relações de herança.

## 4.1. TEMPO

Vamos apresentar intuitivamente a análise de tempo para as principais funções da solução apresentada cujo custo tende a variar conforme o tamanho da entrada para o programa:

|  |  |  |  |
| --- | --- | --- | --- |
| **Classe** | **Método** | **Tempo** | **Descrição** |
| **<Grafo.h>** | addVertice | O(n) | O método percorre um índice para verificar se o vértice v já existe e, se não existir, insere ele ao final do vetor de vértices. |
| **<Grafo.h>** | rmVertice | O(n) | O método deleta a lista adjacência do vértice v e, em seguida, remove o vértice. |
| **<Grafo.h>** | addAresta | O(n) | O método percorre a lista de adjacência do vértice v verificando se já existe a aresta (v,w). Caso não exista, insere a aresta. |
| **<Grafo.h>** | rmAresta | O(n) | O método percorre a lista de adjacência do vértice v verificando se já existe a aresta (v,w). Caso exista, deleta a aresta. |
| **<Grafo.h>** | imprimeVertices | O(n) | Imprime os valores dos n vértices do grafo. |
| **<Grafo.h>** | ImprimeGrafo | O(V+A) | Imprime os “v” vértices e “a” arestas do grafo. |
| **<Blackjack.h>** | swap | O(V+A) | Segue da complexidade do método verificaCiclo. |
| **<Blackjack.h>** | commander | O(log V+ log A) | O método percorre uma subárvore a qual o vértice v pertence. |
| **<Blackjack.h>** | meeting | O(V+A) | O método percorre todo o grafo |
| **<Blackjack.h>** | verificaCiclo | O(V+A) | Recursivamente, o método percorre os vértices e o número de arestas do grafo até encontrar um ciclo no grafo e, caso identifique um ciclo, desfaz a operação swap. |

## 4.2. ESPAÇO

Quanto ao espaço utilizado pelas principais estruturas do programa proposto como solução, destacam-se:

|  |  |  |  |
| --- | --- | --- | --- |
| **Classe** | **Objeto** | **Espaço** | **Descrição** |
| **<Grafo.h>** | Index de Vértices | O(|V|) | Os v vértices do grafo de entrada |
| **<Grafo.h>** | Lista de Adjacência | O(|V| + |A|) | Os “v” vértices mais “a” arestas do grafo de entrada. |

# 5. AVALIAÇÃO EXPERIMENTAL

A avaliação experimental foi executada, em primeiro momento, com os arquivos de testes (datasets) disponibilizados no ambiente Moodle da disciplina. Entretanto, estes arquivos apresentam entradas pequenas, não permitindo uma mensuração mais precisa de tempo de execução do programa do algoritmo e estruturas implementadas.

Ainda, não foi trivial, apesar de algumas tentativas, de criar outros arquivos com entradas maiores. O problema é não dispor de outras ferramentas que gerassem um arquivo com entradas grandes e corretas, principalmente quanto à operação. E criar novos arquivos manualmente demonstrou significativamente improdutivo. Recorreu-se, também, a outros alunos da disciplina para verificar se possuíam outros arquivos, além dos que foram fornecidos, porém sem sucesso.

Como observação e sugestão, tais como em outras disciplinas onde testes são importantes para verificar a qualidade do código implementado, em futuros trabalhos práticos priorizar em fornecer alguns datasets que possam ter tamanhos de entradas significativamente diferentes.

## 5.1. Q1

Por que o grafo tem que ser dirigido? O problema apresentado possui conceitos de hierarquia, os quais são utilizados na composição e organização das equipes de Blackjack da UFMG.

Seja um grafo G dirigido com V vértices e A arestas. Para cada par de vértices (v,w) conectado no grafo G, existe somente uma aresta “a” que indica essa conexão. A ponta final de uma aresta é diferente da ponta inicial. Diz-se que uma aresta “v – w” sai de v e entra em w. O tamanho de um grafo G é a soma de V + A (soma de vértices e arestas).

Considere agora as operações implementadas no problema – swap, commander e meeting. Essas operações criam, alteram (quando possível) e indicam as relações de dependência dentro da hierarquia das equipes. Assim, as relações existentes nas equipes possuem as propriedades de um grafo, mais especificamente dirigido, com seus subconjuntos V (vértices) e A (arestas) e algumas restrições: (1) não existem self-loops e (2) entre dois vértices (v,w) existe somente uma aresta a, ou seja, ou a conexão (v,w) ou a conexão (v,w), mas nunca as duas simultaneamente.

## 5.2. Q2

O grafo pode ter ciclos? Não. Em teoria dos grafos, um grafo ciclo ou grafo circular é um grafo que consiste de um único ciclo, tal que é possível encontrar um caminho que passa por n vértices n1, n2,..., nn e n1 = nn

As características do problema apresentado neste trabalho não permitem ciclos, pois os conceitos de hierarquia não seriam válidos por conta da existência de algum ciclo. Por exemplo, na operação de swap, ela é só processada se a operação não cria um ciclo. Caso contrário, nada é executado. Para isso, é implementado um algoritmo de Busca em profundidade, uma vez que a ordem de busca em profundidade é útil para encontrar ciclos e componentes fortes eficientemente.

## 5.3. Q3

O grafo pode ser uma árvore? O grafo necessariamente é uma árvore?

O grafo é necessariamente uma árvore. Os conceitos de hierarquia que fazem parte da característica do problema é explorado ao se utilizar busca em profundidade, o qual realiza uma busca ou travessia numa árvore, estrutura de árvore ou grafo. Intuitivamente, o algoritmo começa no vértice (ou nó) raiz (ou selecionando um vértice como raiz no caso de um grafo) e explora tanto quanto possível cada um dos seus ramos, antes de executar o backtracking.

# Bibliografia

*[2] INF1010 - Estruturas de dados avançadas Introdução a C++*. (2 de 05 de 2018). Acesso em 28 de 09 de 2019, disponível em PUC Rio: http://webserver2.tecgraf.puc-rio.br/eda/EDA\_09\_aula-stl.pdf

*[1] Blackjack*. (23 de 07 de 2019). Acesso em 26 de 09 de 2019, disponível em Wikipedia: https://pt.wikipedia.org/wiki/Blackjack