UFV–Campus Florestal CCF211 – Algoritmos e Estruturas de Dados I – 1º Semestre de 2023
Trabalho Prático 1 – Tipos Abstratos de Dados [10 pontos]
**Professor :** Philipe de Freitas Melo
**Prazo Final para Entrega:** 01/05/2023 - 23:55

# TP2 – Cartas de Baralho

## 1 Introdução

Este trabalho tem como objetivo explorar e consolidar o conteúdo relacionado aos segundo e terceiro módulos da disciplina, que trata de Pilhas/Filas e Algoritmos de Ordenação, além também do conteúdo sobre Análise de Complexidade. O foco é compreender e manipular cada uma dessas estruturas corretamente. Para isso, será implementado um programa que simula um simples jogo de baralho e utiliza das cartas de baralho para testar algoritmos de ordenação, utilizando as estruturas aprendidas em sala.

Um baralho é um conjunto de cartas utilizado em diversos jogos de cartas. Tradicionalmente, um baralho de cartas é composto por 52 cartas, divididas em quatro naipes: copas, ouros, espadas e paus. Cada naipe contém 13 cartas, sendo elas: ás, dois, três, quatro, cinco, seis, sete, oito, nove, dez, valete, dama e rei. As cartas também possuem um valor numérico ou simbólico, que é utilizado para determinar a hierarquia das cartas durante o jogo. Além disso, os baralhos podem conter cartas coringa, que variam de acordo com a região e o jogo específico. O baralho de cartas é amplamente utilizado em jogos de cartas como pôquer, *blackjack*, paciência, truco, entre outros, e também é utilizado em truques de mágica, construção de castelos de cartas e outros tipos de entretenimento. Sua estrutura e diversidade de jogos o tornam um elemento versátil e amplamente reconhecido em diferentes culturas e contextos sociais. Um jogo conta com um "*deck´´* de 1 ou mais conjuntos de cartas de baralho. Por exemplo, no jogo de Pôquer[1], se utiliza apenas um baralho de 52 cartas, enquanto no jogo de Buraco, se utilizam 2 baralhos inteiros[2]

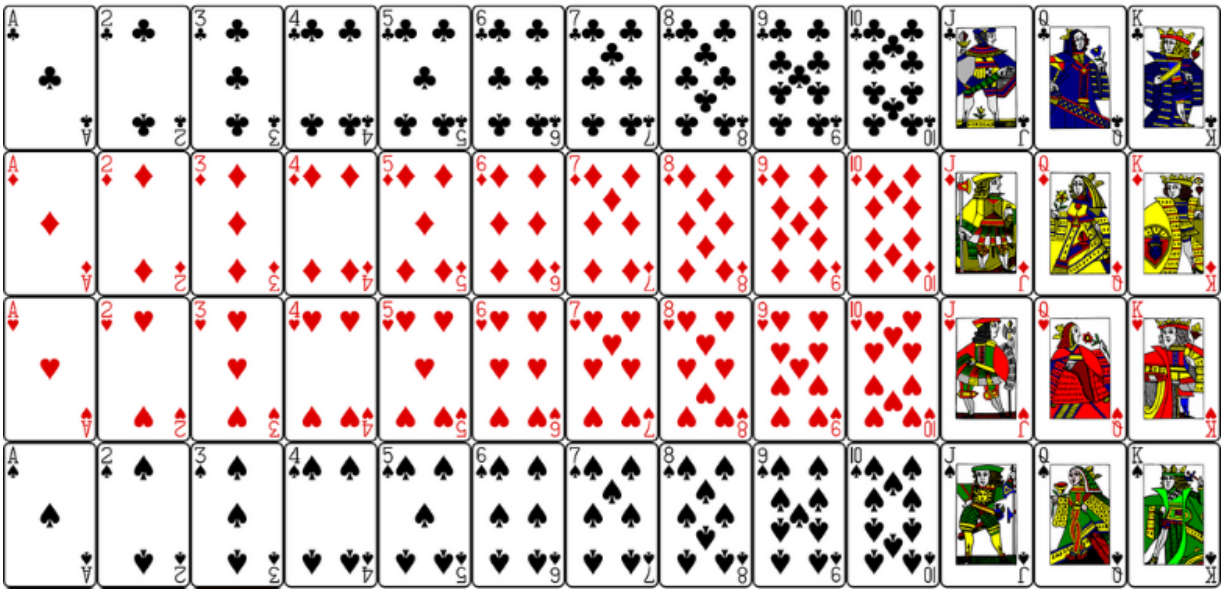
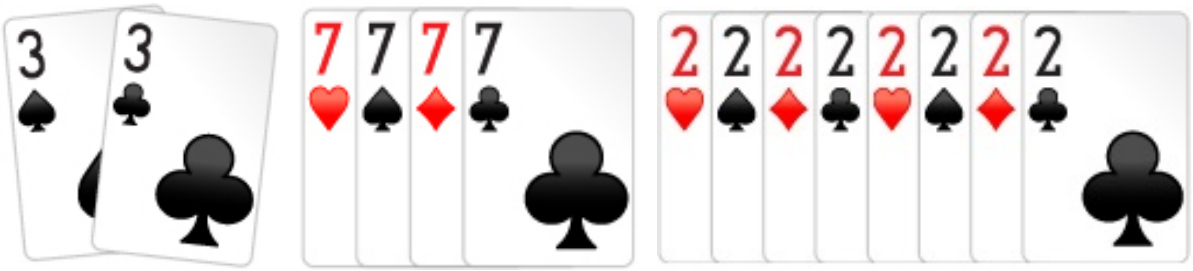


Figura 1: Lista de todas as 52 cartas de um baralho.

Este trabalho prático tem como objetivo explorar o uso de estruturas de dados e algoritmos na implementação e manipulação de um baralho de cartas. Os alunos utilizarão a estrutura de dados de pilha para representar o monte de compras e o monte de descarte do baralho num simples jogo de cartas, além de implementar três métodos de ordenação baseados no valor e naipe das cartas. Será uma oportunidade de aplicar os conhecimentos de pilhas, ordenação e análise de complexidade.

[1] https://pt.wikipedia.org/wiki/P\%C3\%B4quer
[2] https://copag.com.br/blog/detalhes/buraco



(a) Mão de 2 cartas (b) Mão de 4 cartas iguais (c) Mão de 8 cartas

Figura 2: Exemplo de condição de vitória do jogo para partidas com diferentes tamanhos de mãos dos jogadores.

# 2 Descrição das Tarefas

O objetivo deste trabalho prático é explorar o uso de estruturas de dados e algoritmos na implementação e manipulação de um baralho de cartas. Neste TP, você deverá criar um deck de baralho contendo conjuntos de múltiplos conjuntos de cartas, utilizando a estrutura de dados conhecida como pilha para representar o monte de compras e o monte de descarte do baralho. Com essa estrutura será implementado um jogo de cartas em que o jogador deve comprar cartas até ter sua "mão" completa de cartas iguais (do mesmo valor). Além disso, também será necessário implementar três métodos de ordenação diferentes, levando em consideração tanto o valor da carta quanto o seu naipe para testar e comparar diferentes ordem de complexidade de algoritmos.

A seguir será descrito individualmente cada uma das tarefas, bem como as estruturas de dados que deverão ser utilizadas para executar cada tarefa.

## 2.1 Embaralhando o monte de cartas inicial

Antes de entrar nas tarefas do jogo e de ordenação propriamente dita, primeiro é preciso entender como deve ser a criação do monte de cartas do baralho. Como explicado, cada conjunto de baralho possui 52 cartas e, diferentes jogos, utilizam de número de baralhos diferente para serem jogados e também podem ser excluídas cartas para o jogo como por exemplo, no Truco, em que as cartas 8, 9 e 10 são ignoradas nas partidas. Desta forma, nese trabalho, um número $N$ de conjuntos de baralhos utilizados será informado através de um arquivo de entrada, assim como quais cartas deverão ser excluídas do monte. Este valor $N$ dirá quantos baralhos serão usados durante a execução do programa enquanto a lista de cartas informam cartas que devem ser ignoradas pelo programa. Desta forma, para um baralho completo, sem nenhuma exclusão, temos que $N = 1$ significa um único conjunto com 52 cartas, para $N = 2$, teremos 104 cartas (duas de cada tipo), e assim por diante. Por exemplo, para uma entrada de $N = 10$ teremos 520 cartas de baralhos utilizadas.

Após saber o número de cartas, o programa deve ser capaz de "embaralhar essas cartas", ou seja, o programa deve utilizar de uma estrutura de dados para criar uma pilha com todas as cartas do baralho, empilhando todas as cartas de forma aleatória. Ao final, essa pilha conterá todas as cartas que serão utilizadas para as tarefas a serem executadas numa ordem qualquer.

## 2.2 Jogo de Pares

A primeira tarefa do trabalho consiste em implementar a mecânica de um simples jogo de cartas funciona da seguinte maneira. O jogo possui um monte de compra de cartas (que será sua pilha criada com $N$ baralhos embaralhados aleatoriamente) e uma outra pilha de descartes, na qual o jogador também só tem acesso ao elemento do topo da pilha. Cada jogador começa com $M$ cartas na mão e, para vencer, precisa completar todas as cartas da sua mão com cartas iguais, como mostrado na Figura 2 abaixo. Para isso, em cada rodada, ele deve escolher comprar uma nova carta da pilha de compras ou pegar a carta do topo da pilha de descartes. (Note que ao comprar uma carta da pilha de comprar o jogador não sabe qual carta será, enquanto ele conhece qual é a carta do topo da pilha de descarte).

Figura 3: Lista das 52 cartas de um único baralho ordenadas.

Para este trabalho este jogo será implementado então do seguinte modo para **DOIS JOGADORES**. Ao início cria-se a pilha com todas as cartas da partida embaralhas. "Compre" um número $M$ de cartas para compor a mão de cada jogador (Esse valor $M$ será informado também através do arquivo de entrada). Os turnos são alternados, ou seja, a cada rodada o primeiro jogador joga e depois o segundo faz sua vez, depois o primeiro joga novamente, depois o segundo e assim até a partida terminar.

Na vez de cada jogador o programa deve exibir na tela que é a vez deste jogador e a relação das cartas em sua mão e qual a carta que está no topo da pilha de descartes. Em seguida, deve fornecer ao jogador as duas escolhas. Então o usuário deve escolher uma dentre as duas alternativas:

1. Comprar uma nova carta do topo da pilha de compras
2. Pegar uma carta do topo da pilha de descarte

E depois, ao final do turno, ele SEMPRE deverá descartar uma carta na pilha de descarte. Para isso o usuário irá informar a posição da carta em sua mão que ele deseja descartar (Uma posição entre 0 e M-1 que representa a ordem das cartas em sua mão). E assim começa a rodada do outro jogador.

A partida termina quando um dos jogadores conseguir preencher toda a sua mão com cartas do mesmo valor, ganhando o jogo ou quando a pilha de compras acabar, terminando a partida em empate. O programa então deve ser capaz de identificar essas condições de término da partida e informar que o jogo terminou, quem venceu ou se empatou, e revelar a mão do jogador vencedor.

## 2.3 Ordenação de Cartas

A ordenação é um conceito fundamental em algoritmos e estruturas de dados. Ela consiste em organizar um conjunto de elementos de acordo com algum critério predefinido. Neste trabalho, vocês terão a tarefa de implementar três métodos de ordenação diferentes, baseados no valor e naipe das cartas. Esses métodos permitirão que as cartas do baralho sejam dispostas de forma crescente testando diferentes estratégias conforme a Figura 3 que mostra a ordenação para um único baralho completo de 52 cartas.

Para ordenar as cartas do baralho, primeira coisa é levar em consideração o valor de cada carta, começando do Ás que vale 1 até o rei (K) que vale 13. Note, porém, que no conjunto de cartas haverão várias cartas de mesmo valor, porém com naipe diferente. Cartas com exatamente o mesmo valor e mesmo naipe não possuem nenhuma ordem precedência, porém se duas cartas de mesmo valor tiverem naipes diferentes, a seguinte ordem de precedência deverá ser obedecida:

*Copas* $<$ *Ouros* $<$ *Paus* $<$ *Espadas*

Figura 4: Ordem de precedência dos naipes.

Ou seja, para cartas do mesmo valor, as cartas de ouro devem vir primeiro, depois as de ouros, logo em seguida as cartas de paus e, por último, as cartas de espadas. Note que o total de cartas a serem ordenados é definido pela entrada do programa através do valor $N$ com o total de baralhos que serão utilizados menos as cartas a serem excluídas. Então o tamanho da entrada do programa deverá obedecer e variar de acordo com estes parâmetros preestabelecidos.

Para o trabalho, três diferentes algoritmos de ordenação deverão ser implementados. Obrigatoriamente, um deles deverá ser o **QuickSort**, os outros dois métodos podem ser escolhidos dentre aqueles métodos

ensinados em sala. Todas as implementações devem ser feitas do zero, ou seja, vocês devem implementar todo o algoritmo, e não serão aceitos códigos retirados da internet. Durante a implementação, adicione impressões na tela que vá dizendo passo a passo o que o algoritmo está fazendo para ordenar o vetor. Essas mensagens na tela serão utilizadas para auxiliar na correção do trabalho e serão avaliadas como parte do código

Ao final da ordenação, o programa deverá criar um arquivo nomeado `ordenado_<metodo.txt>` para cada um dos três métodos de ordenação implementados. Neste arquivo deve ter o resultado da execução do algoritmo com todas as cartas ordenadas, uma por linha.

### 2.4 Coringa

No arquivo de entrada do programa, pode existir uma carta específica que servirá como um coringa no programa. Este coringa modificará levemente ambas as tarefas, tanto o jogo como a ordenação. O coringa é uma carta específica tanto um valor como o naipe. Por exemplo, um coringa pode ser um "*2 de Copas*", o que significa que todas instancias dessa carta específica contam como coringa. Entretanto, note que outros 2 de outros naipes ("*2 de Ouros*", "*2 de Espadas*", "*2 de Paus*") não servem como coringa. Assim como outras cartas de Copas não irão funcionar como coringa.

Para a tarefa do jogo, o coringa serve como substituto para qualquer outra carta do jogo. Dessa forma, quando tentando completar a mão de cartas iguais, mesmo que o valor seja diferente do coringa, esta carta poderá contar para completar a mão e ganhar o jogo.

Na ordenação, o coringa é a carta com maior valor, independente do naipe e do valor, e então ele sempre deverá ser a última carta na ordenação. Portanto, quando existir um coringa nos baralhos, você deverá modificar os algoritmos de ordenação de forma que o coringa venha após as outras cartas, após os reis

## 3 Estruturas de Dados

Para esse trabalho, é obrigatório o uso da estrutura da PILHAS para criar o monte de cartas embaralhados, para fazer o monte de compras e de descarte do baralho a serem utilizadas no jogo. Portanto, para implementar o jogo, você deve obrigatoriamente utilizar as funções EMPILHAR e DESEMPILHAR do TAD Pilha. O TAD Pilha pode ser implementado tanto por arranjo como por listas encadeadas como visto em aula. Você poderá escolher uma destas duas implementações que preferir para o trabalho

Para a tarefa de ordenação, utilize também a estrutura de pilha criada para inicialização do problema para criar o conjunto de cartas aleatória a ser ordenadas. Porém, não é necessário utilizar a estrutura de pilha na ordenação. A implementação pode, por exemplo, pegar a pilha de cartas inicial e transformar num vetor antes de começar a ordenação

O restante de objetos que envolvem o jogo, como os jogadores e a mão de cartas de cada jogador não precisa seguir uma estrutura específica de implementação e pode ser feita de acordo com sua preferência, porém existem ainda algumas regras que serão exigidas para o desenvolvimento do trabalho, como o desenvolvimento de TADs específicos como veremos a seguir.

## 4 Tipo Abstrato de Dados

### 4.1 Carta de Baralho

Será necessário construir para este trabalho um tipo abstrato de dados para as cartas. Cada carta de baralho deve possui um tipo próprio que é uma estrutura do tipo Carta. A pilha desenvolvida, desta maneira, será uma pilha de Cartas. Cada carta deve ter seu valor numérico que varia de 1 a 13 e seu naipe. O código também deve ser capaz de imprimir a representação das cartas com seu valor $[A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K]$ e naipes $[C, O, P, E]$. Desta forma, por exemplo, se um código for imprimir três cartas, um ás de copas, um 5 de ouros e uma dama de paus o código deverá imprimir:

**[AC] [5O] [QP]**

Portanto, ao dizer imprimir uma carta, só deve mostrar esta representação do valor e letra que simboliza a carta. Então não deverá imprimir "graficamente" nenhuma outra forma de visualização da carta. É

interessante também pensar em uma função que compare duas cartas e saiba resolver se uma carta é maior que a outra, assim ela poderá ser usada durante a ordenação para facilitar o processo

- **`ImprimeCarta(...)`**
- **`ComparaCartas(...)`**

### 4.2 Pilha

Como mencionado, neste trabalho será utilizado um TAD Pilha para o programa. Este TAD pilha pode seguir tanto a implementação usando lista encadeada com ponteiros, como também a implementação por arranjos. Cada elemento da pilha deverá conter uma carta. A pilha deverá conter obrigatoriamente as funções:

- **`Empilha(...)`**
- **`Desempilha(...)`**
- **`VerificaTopoPilha(...)`**
- **`VerificaPilhaVazia(...)`**

Essas funções deverão ser criadas pelo seu TAD e capaz de manipular a Pilha criada. Outras funções podem ser implementadas para auxiliar o desenvolvimento do programa e manipulação das cartas. O programa deverá se utilizar desses valores para realizar as operações de comprar e descartar do jogo, entre outras coisas.

### 4.3 Jogador

Embora não seja obrigatória a implementação de um tipo abstrato próprio para gerenciar a estrutura do jogador no trabalho, serão descritas aqui quais características que um jogador possui neste trabalho.

Cada jogador possui uma mão de $M$ cartas, ou seja, um conjunto fixo de cartas que não será alterado ao longo do jogo. Porém, esse valor $M$ é definido de acordo com um valor lido do arquivo de entrada do programa. A forma com que este esquema funcionará, ficará a cargo do desenvolvedor.

## 5 Entrada e Saída do Programa

O programa principal deste trabalho deverá ser escrito de forma a ler um arquivo de entrada com informações de número de baralhos, número de cartas na mão do jogador, cartas a serem excluídas do baralho e uma carta coringa para o jogo. Dada essas informações de um arquivo do arquivo de entrada, o código deve fazer duas tarefas: controlar um jogo onde o jogador tentará completar uma mão de cartas iguais e outra tarefa de ordenar as cartas de baralho seguindo as regras estabelecidas. Nesta ultima tarefa, um arquivo de saída também deverá ser criado com as cartas ordenadas.

Após a implementação do(s) TAD(s) pedidos pelo trabalho prático com as estruturas de dados a funções necessárias, o trabalho então ainda exige uma segunda parte que consiste na ordenação das cartas para testar e comparar as diferentes funções de ordenação estudadas em sala e criará um arquivo de saída para cada um desses algoritmos implementados

Nesta seção, descreveremos com detalhe como é o formato de cada um desses arquivos e como utiliza-los em seu programa

### 5.1 Arquivo de Entrada

A entrada do programa consiste em um arquivo de texto contendo exatamente 4 linhas.

1. A primeira linha do arquivo define quantos baralhos serão utilizados no programa;
2. A segunda linha é uma lista de valores que representam quais cartas não serão utilizadas nesta execução do programa. Aqui serão listados apenas os valores e todas as cartas desses valores (de todos os naipes) deverão ser **EXCLUÍDOS** do baralho inicial criado pelo programa. Caso nenhuma carta deva ser excluída a linha aparecerá com o valor 0;

3. A terceira linha define uma carta (uma carta específica com valor e naipe) que será utilizada como coringa no programa. O coringa, quando existente, altera tanto o funcionamento do jogo como a ordenação das cartas conforme explicando na descrição das tarefas. Caso não tenha coringa a ser considerado, essa linha aparecerá com valor 0;

4. A quarta e última linha define o tamanho da mão de cada jogador para a tarefa do jogo. Essa mão define quantas cartas terão na mão do jogador e assim, consequentemente, quantas cartas iguais ele deverá reunir para ganhar o jogo.

A seguir, mostraremos alguns exemplos de arquivo de entrada para auxiliar na compreensão do exercício.

<entrada_truco.txt>

```
1
8,9,10
0
3
```

No exemplo acima, temos um exemplo de como seria, por exemplo, uma entrada que simula o mesmo tipo de baralho utilizado numa partida de truco. Como visto pela primeira linha, seria utilizado um único baralho de 52 cartas. Pela segunda linha, temos que as cartas 8,9 e 10, te todos os naipes, são excluídas do baralho. A terceira linha mostra que não existe coringa para este baralho. Por último, na quarta linha temos que cada jogador começa com apenas 3 cartas na mão.

<entrada_buraco.txt>

```
2
0
2C
11
```

Nesse segundo exemplo, temos um exemplo de como simular um baralho para o jogo de buraco. Neste exemplo, começamos com dois baralhos (104 cartas no total) como dito pela primeira linha e, como observado pela segunda, não excluímos nenhuma carta para a criação do baralho inicial. Na terceira linha, temos que a carta "*2 de Copas*" é um coringa do jogo[3]. Por fim, temos que cada jogador tem 11 cartas na mão.

<entrada_generica.txt>

```
10
3
AE
6
```

De forma geral, temos aqui mais um exemplo de execução. A primeira linha mostra que serão 10 baralhos utilizados. A segunda mostra que as cartas de valor 3 (de todos os naipes) serão excluídas nesta execução. Neste exemplo, a terceira linha diz que o "*As de Espadas*" será um coringa no jogo. E na última linha, temos que cada jogador terá 6 cartas na mão

## 5.2 Arquivo de Saída

No arquivo de saída, espera-se que você descreva o resultado da ordenação de cada algoritmo implementado. O arquivo deve ter todo o baralho com todas as cartas da execução ordenadas, com uma carta por linha.

O programa deve criar três arquivos distintos, um para cada algoritmo de ordenação implementado com o nome do método correspondente. Dado um nome de entrada `<entrada.txt>` cada um dos arquivos de saída devem ter exatamente o seguinte nome: `ordenacao_<entrada>_<metodo>.txt`.

Observe então se seu programa consegue processar o arquivo de entrada e criar adequadamente os três arquivos de saída com o nome necessário

## 5.3 Compilação e Execução

O programa deverá ser possível de ser compilado nos computadores do laboratório da UFV, portanto ele não deverá conter nenhuma biblioteca que seja específica do ambiente de trabalho. **Descreva na sua documentação como compilar e executar seu código.**

[3]Em buraco, a carta 2 de todo naipe é considerado coringa, mas aqui no exemplo apenas o 2 de copas é um coringa

O programa deverá ser executado sem argumentos passados por linha de comando, mas ele deve receber um arquivo. Para uma execução padrão (sem argumentos) considere o nome do arquivo de entrada como "`entrada.txt`"e saída conforme a formatação estabelecida anteriormente. Os demais dados do programa se encontrarão normalmente no arquivo de entrada.

Caso ache necessário, a execução do seu trabalho poderá ser feita por um arquivo Makefile. Para isso descreva na sua documentação como utilizá-lo. Porém, esta etapa não é obrigatória.

# 6 Documentação

A documentação do trabalho é parte importante da sua nota! Faça uma documentação bem feita, clara e com boa formatação. O documento entregue deve estar no formato PDF e deve conter descrito os passos dados em direção a resolução do problema proposto.

Na documentação é sua oportunidade de descrever o que você pensou para resolver o problema e explicar o funcionamento do seu código. Entre outras coisas, sua documentação deve conter:

1. **Título:** Dê um título no início do seu documento e coloque seu nome e matrícula como autor do trabalho.
2. **Introdução:** descrição sucinta do problema a ser resolvido e visão geral sobre o funcionamento do programa.
3. **Modelagem e Funcionamento:** é importante discutir quais as estratégias, e quais passos você chegaram até a solução final do seu problema.
4. **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 e procedimentos utilizados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.
5. **Estudo de Complexidade:** análise da complexidade do tempo de execução dos procedimentos implementados e do programa como um todo (notação O). Observe então que cada função mais importante do código deverá ter sua análise separada e o *main* também deverá ter a complexidade de forma mais geral.
6. **Testes e Resultados:** Descreva como se deu os testes realizados durante a implementação. Também faça um teste comparativo e apresente os resultados obtidos utilizando de arquivos com diferentes tamanhos de entrada, **inclusive alguns além dos passados nos exemplos**.
7. **Algoritmos de Ordenação:** Explique a implementação de todos os algoritmos de implementação utilizados no programa e discuta sobre a ordem de complexidade de cada um. Execute o programa para testes de vários tamanhos e mostre os resultados comparativos entre os diferentes algoritmos .
8. **Conclusão:** comentários gerais sobre o trabalho e as principais dificuldades encontradas em seu desenvolvimento.
9. **Bibliografia:** Referências utilizadas para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso.

É importante dar atenção as boas práticas de programação ao organizar seu código. A documentação não deve exceder **10 páginas**. Documentações que excederem esse tamanho serão penalizadas.

# 7 O que deve ser entregue:

- Você deverá submeter no *PVA Moodle* uma pasta contendo seus arquivos compactados no formato especificadamente **.zip** A pasta de entrega do trabalho deve ser nomeada "MATRICULA_tp1", onde MATRICULA é seu número de matrícula. Dentro da pasta de entrega, o arquivo principal (o que contiver o main) deverá receber o nome de "main.c".
- Todos códigos fonte do seu programa em **C** (todos os arquivos .c e .h), bem identado e comentado.

- Arquivo executável.
- Documentação do trabalho. Documento de até no máximo 10 páginas entregue em PDF.

Não é permitido o compartilhamento de relatório ou código entre os estudantes de AEDs I. Além disso, os estudantes envolvidos em plágio (copiando ou permitindo a cópia) serão punidos. Busque empregar *suas próprias palavras* ao realizar esse trabalho.

Faça uso do fórum da disciplina para compartilhar/responder/conversar sobre dúvidas relativas a esse trabalho, tendo como guia **não fornecer soluções parciais ou completas**.

**Data de Entrega:** até 02 de Julho de 2023, às 23:55 horas, ou antes. Após essa data, haverá uma penalização por atraso: $(2^d - 1)$ pontos, em que $d$ é o número de dias de atraso. **Valor: 20 pontos.**

# 8 Desafio

Para este trabalho, existe um desafio que vale ponto extra na sua nota. O desafio do trabalho consiste em implementar uma forma do usuário jogar sozinho o jogo.

Portanto, a tarefa do desafio consiste em implementar alguma IA ou alguma estratégia em que o computador consiga jogar sozinho para o segundo jogador da partida. Assim o usuário jogador conseguira jogar contra o computador.

Dessa forma, quando um usuário estiver jogando sozinho, o programa irá pedir somente as ações feitas pelo usuário e, na vez do outro jogador, o computador deve escolher sozinho as ações por ele. Além disso, o usuário não poderá ver quais são as cartas na mão do seu oponente, mas deverá saber se o computador realizou uma ação de comprar ou de pegar a carta da pilha do descarte.

Sua IA que realizará as ações automaticamente, deverá implementar alguma estratégia para tomar decisão na sua vez de qual ação fazer (comprar uma carta nova ou pegar da pilha de descarte) e qual carta ele irá descartar no final do turno. Porém, essa estratégia não pode saber quais são as cartas do usuário (humano) que está jogando com o programa.

Sua estratégia deverá ser fundamentalmente explicada na documentação, incluindo os detalhes de implementação utilizados. Melhores ideias serão melhores avaliadas e, consequentemente, abonadas com mais pontos extras.

Bom trabalho! E comece logo a fazê-lo. Afinal, você nunca terá tanto tempo para resolvê-lo quanto agora!