Projeto - Algoritmos e Estruturas de Dados II

Poker Hand Evaluator

Camila Ferreira Rodrigues	11082810
Fabricio Ribeiro Bueno Yamamoto	11036814
Victor Yoshidi Sanches Natumi	11053310

O que *é* o projeto?

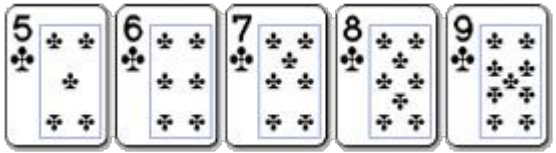
O que é o projeto?

Comparação entre diferentes abordagens de estruturas de dados aplicado ao problema de avaliar a força relativa de uma mão de pôquer.

Apresentaremos três diferentes abordagens de, dadas duas mãos de pôquer com cinco cartas, quantificar a força dessas mãos seguindo as regras do jogo, e determinar a vencedora.

Classificação das mãos de poker

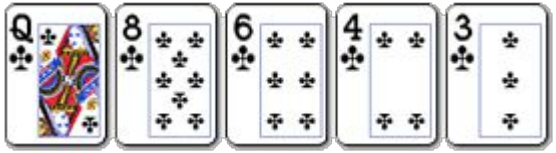
1. Straight Flush : Cinco cartas na sequência, do mesmo naipe. Em caso de uma mão similar, o valor da carta mais alta da sequência vence.



2. Quadra : Quatro cartas de mesmo valor, e uma outra carta como 'kicker'. Em caso de empate, o jogador com a maior carta restante ('kicker') vence.

3. Full House : Três cartas do mesmo valor, e duas outras cartas diferentes de mesmo valor. Em caso de empate, as três cartas de mesmo valor mais altas vencem.

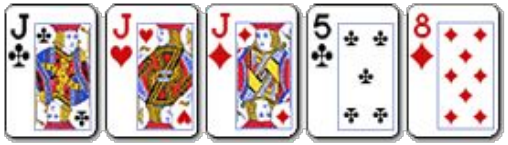
4. Flush : Cinco cartas do mesmo naipe, não em sequência. Em caso de empate, o jogador com a maior carta de maior valor vence.



Classificação das mãos de poker

5. Sequência : Cinco cartas de naipes diferentes em sequência.

6. Trinca : Três cartas do mesmo valor, e duas outras cartas não relacionadas.



7. Dois Pares : Duas cartas de mesmo valor e mais duas cartas diferentes de mesmo valor, além do kicker.

8. Par : Duas cartas do mesmo valor, e três outras cartas não relacionadas.

9. Carta Alta : Qualquer mão que não esteja nas categorias acima. Em caso de empate, a carta mais alta vence.

1ª Abordagem : Naive Evaluator

1ª Abordagem : Naive Evaluator

Algoritmo com estratégia mais “intuitiva”:

1. Leitura. Armazenamos os valores de rank e naipe, checando por cartas inexistentes ou duplicadas.
2. Classificação. Determina-se qual a classificação da mão de acordo com testes específicos para cada grupo.
3. Ordenação. Ordenamos a mão.
4. Comparação. Quando duas mãos são comparadas, primeiro compare os grupos, e então se forem da mesma categoria, compare cada carta restante individualmente para determinar qual das duas mãos é mais forte.

1ª Abordagem : Naive Evaluator

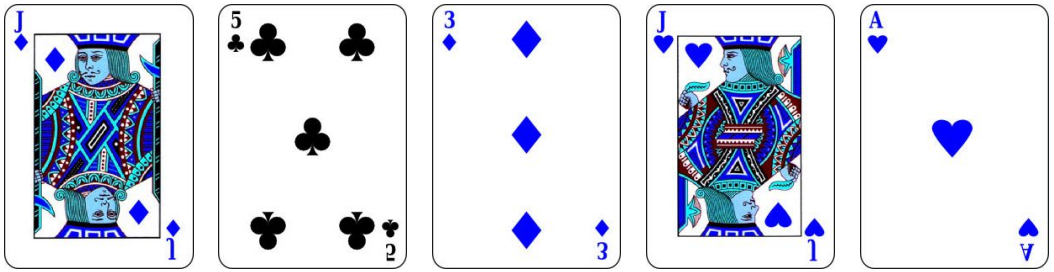
Estruturas de dados utilizados:

- Array
- Quick Sort

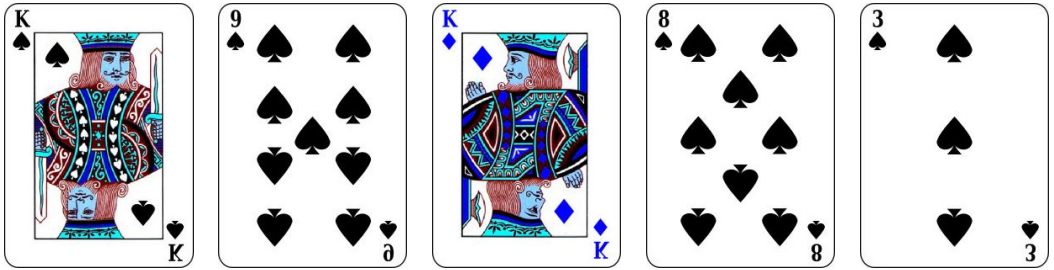
1ª Abordagem : Naive Evaluator

Caso de teste: 1.Leitura

Mão 1



Mão 2

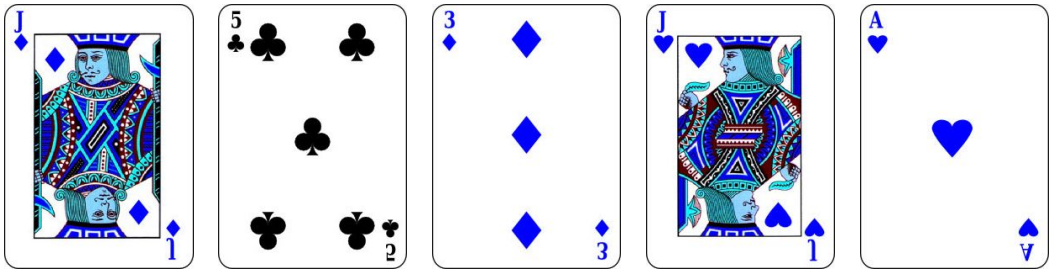


1ª Abordagem : Naive Evaluator

Caso de teste: 2. Classificação

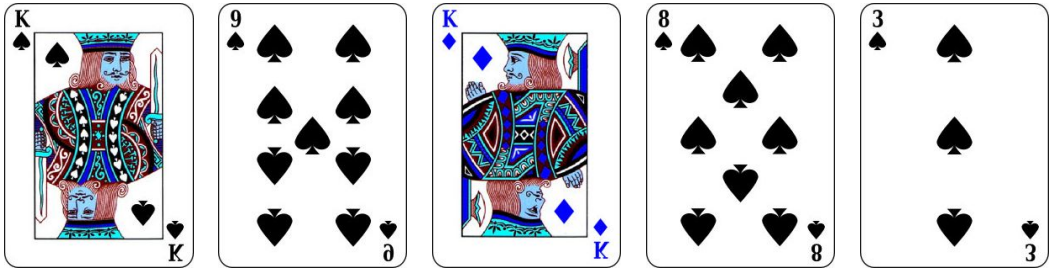
Mão 1

1 Par



Mão 2

1 Par

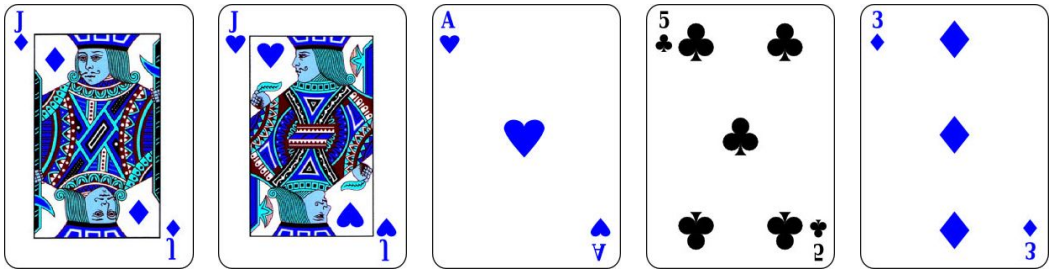


1ª Abordagem : Naive Evaluator

Caso de teste: 3. Ordenação

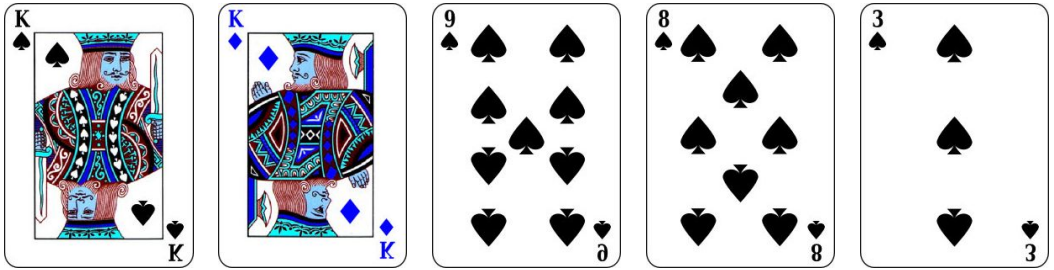
Mão 1

1 Par



Mão 2

1 Par



1ª Abordagem : Naive Evaluator

Caso de teste: 4. Comparação

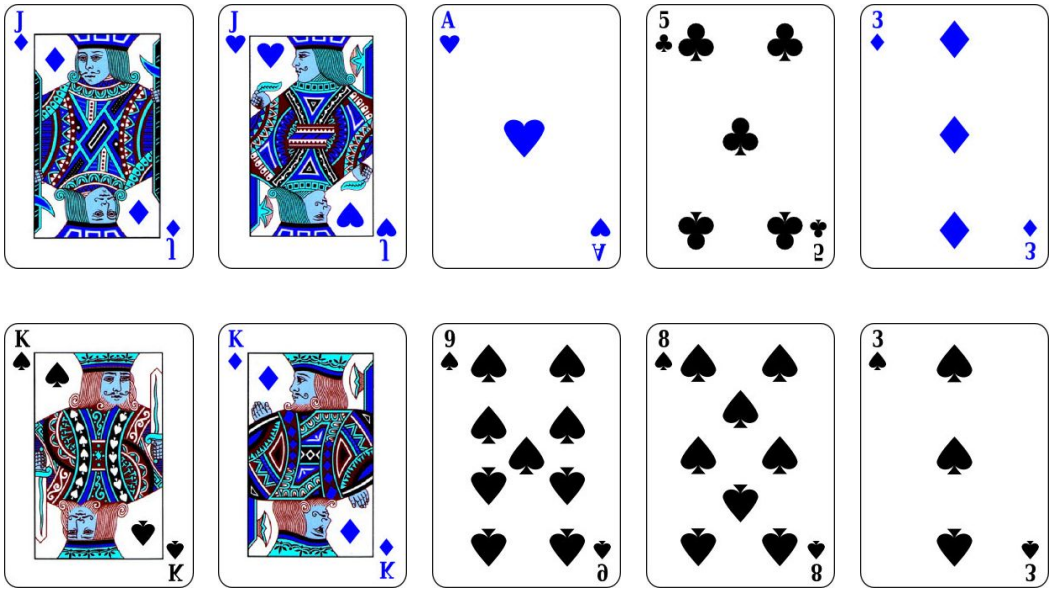
1 Par de J

<

1 Par de K

∴

Mão 2 vence!



1ª Abordagem : Naive Evaluator

Análise de complexidade:

- A complexidade é $\Theta(n \log n)$ por conta da ordenação.
- Seu melhor caso seria quando, na comparação de duas mãos, elas fossem de categorias diferentes.
- Já o pior seria se as categorias fossem iguais e com cartas muito parecidas.

1ª Abordagem : Naive Evaluator

Análise de complexidade:

- Se alterássemos o algoritmo para classificar e comparar todas as mãos possíveis, levaria tempo $\Theta(n^2)$,
  onde n > 2 milhões.

2ª Abordagem : Cactus Kev

2ª Abordagem : Cactus Kev

Introdução:

- Um baralho comum de 52 cartas tem C(52,5) = 2.598.960 mãos únicas de pôquer.
- De todas essas mãos, há apenas 7462 valores diferentes para a força das mãos.

2ª Abordagem : Cactus Kev

Introdução:

Grupo da mão	Mãos Únicas	Mãos Distintas
Straight Flush	40	10
Quadra	624	156
Full House	3744	156
Flush	5108	1277
Sequência	10200	10
Trinca	54912	858
Dois Pares	123552	858
Um Par	1098240	2860
Carta Alta	1302540	1277
Total	2598960	7462

2ª Abordagem : Cactus Kev

Representação das cartas:

- Uma maneira de conseguir evitar a necessidade de ordenação das cartas é a utilização de números primos.

Carta	Primo	Binário
2	2	0000
3	3	0001
4	5	0010
5	7	0011
6	11	0100
7	13	0101
8	17	0110
9	19	0111
10	23	1000
J	29	1001
Q	31	1010
K	37	1011
A	41	1100

2ª Abordagem : Cactus Kev

Representação das cartas:

- Cada carta ocupa 4 bytes (integer), organizados da seguinte forma:

xxxAKQJT	98765432	CDHSrrrr	xxPPPPPP
xxxbbbbb	bbbbbbbb	cdhsrrrr	xxpppppp

2ª Abordagem : Cactus Kev

Algoritmo:

1. Leitura. Converter as cartas para a representação em bits
2. Determinar se é flush, se sim, então verificar em lookup table valor da mão
3. Caso não seja flush, mas contenha cinco cartas de valores distintos, sendo sequência ou carta alta, verificar em lookup table valor da mão
4. Caso contrário, determinar valor da multiplicação com números primos e realizar busca binária em tabela para determinar valor da mão
5. Comparação. Comparar o valor de duas mãos, o menor valor vence.

2ª Abordagem : Cactus Kev

Estruturas de dados utilizados:

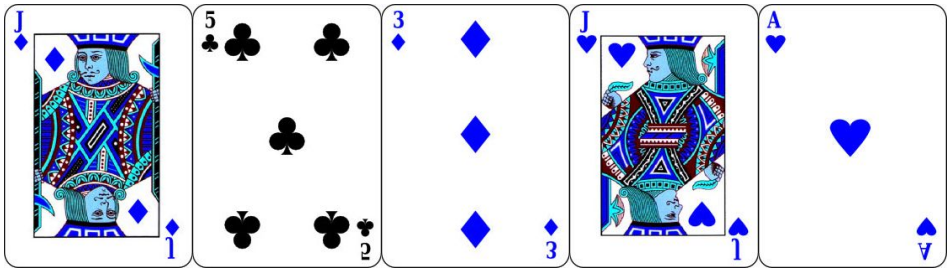
- Array
- Bitwise Operation
- Lookup Table
- Binary Search

2ª Abordagem : Cactus Kev

Caso de teste: 1.Leitura

Mão 1:

	xxxAKQJT	98765432	CDHSrrrr	xxPPPPPP
JD	00000010	00000000	01001001	00011101
5C	00000000	00001000	10000011	00000111
3D	00000000	00000010	01000001	00000011
JH	00000010	00000000	00101001	00011101
AH	00010000	00000000	00101100	00101001

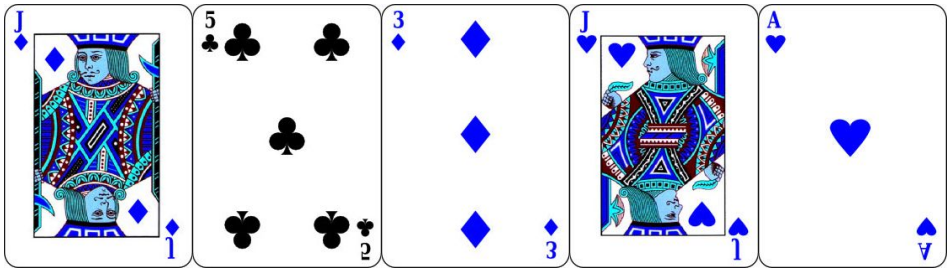


2ª Abordagem : Cactus Kev

Caso de teste: 2. Determinar se é Flush

(c1 AND c2 AND c3 AND c4 AND c5 AND 0xF000)

JD	00000010	00000000	01001001	00011101
5C	00000000	00001000	10000011	00000111
3D	00000000	00000010	01000001	00000011
JH	00000010	00000000	00101001	00011101
AH	00010000	00000000	00101100	00101001
0xF000	00000000	00000000	11110000	00000000
&&	00000000	00000000	00000000	00000000



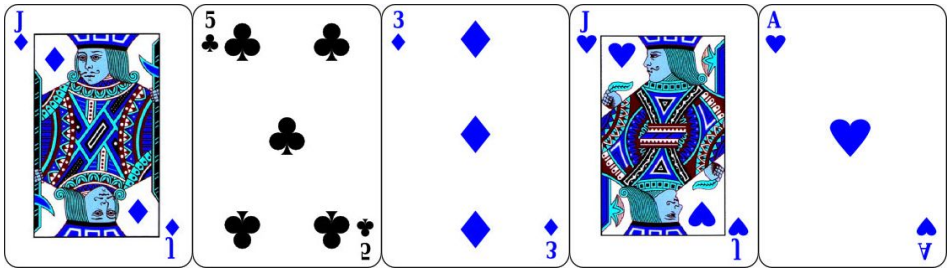
⇨ = 0 ∴ não é flush!
não retornar *flushes[q]*

2ª Abordagem : Cactus Kev

Caso de teste: 3. Determinar se é Sequência ou Carta alta

q = (c1|c2|c3|c4|c5) >> 16

JD	00000010	00000000	01001001	00011101
5C	00000000	00001000	10000011	00000111
3D	00000000	00000010	01000001	00000011
JH	00000010	00000000	00101001	00011101
AH	00010000	00000000	00101100	00101001
||	00010010	00001010	----------	----------



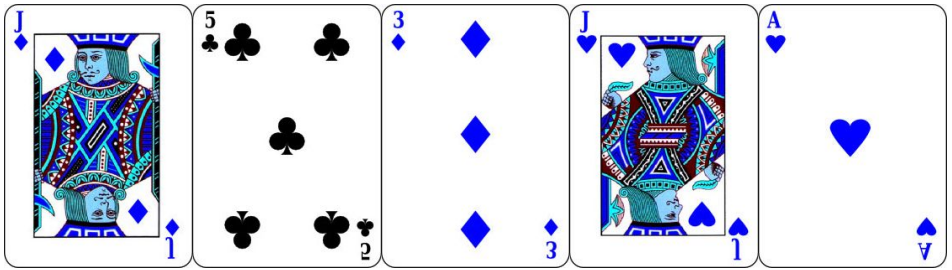
⇨ **≠ 5 bits acesos ∴ não possui cartas únicas (sequência ou high card)**
***unique5[q]* retorna 0**

2ª Abordagem : Cactus Kev

Caso de teste: 4. Multiplicação dos primos

q = (c1&0xFF) * (c2&0xFF) * (c3&0xFF) * (c4&0xFF) * (c5&0xFF)

JD && 0xFF	00011101	29
5C && 0xFF	00000111	7
3D && 0xFF	00000011	3
JH && 0xFF	00011101	29
AH && 0xFF	00101001	41
Mão 1		724101



⇨ **∴ q = 724101**

Binary Search: q = find(q)

2ª Abordagem : Cactus Kev

Caso de teste: 4. Binary Search

É executado em uma array *products[4888]*, que possui o valor das multiplicações dos números primos. Ao ser encontrado, retorna o índice *i*, este que corresponde ao mesmo em outra array *values[4888]* .

values[q] = 4036 (valor final da mão)

2ª Abordagem : Cactus Kev

Caso de teste: 5. Comparação

Ao chamar a função do avaliador com o valor da mão como parâmetro, é retornado um inteiro de 1 a 7462.

Ao fazer isso com mãos diferentes, comparamos o valor de retorno, quanto menor, melhor.

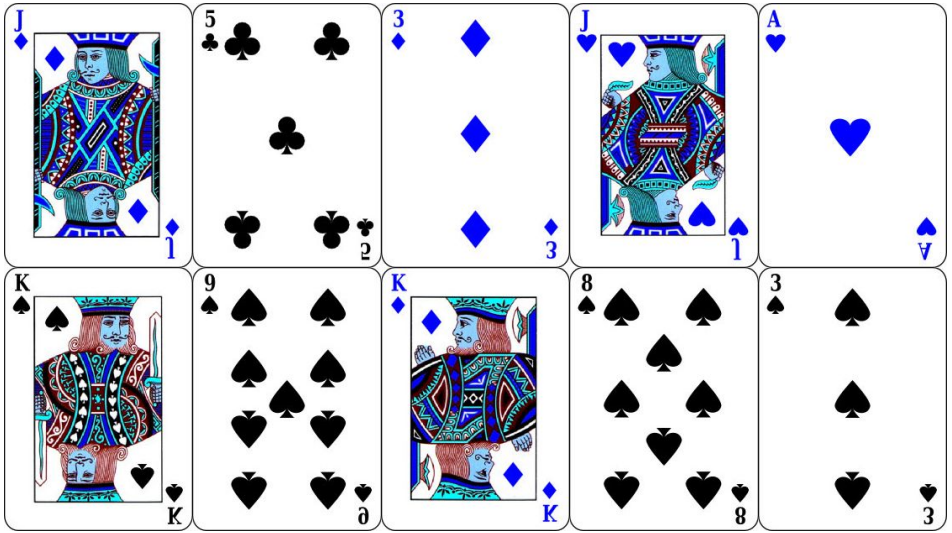
2ª Abordagem : Cactus Kev

Caso de teste: 5. Comparação

Para categoria “1 Par” valor varia entre [6185...3325]

Mão 1 = 4036

Mão 2 = 3714



Neste caso mão 2 < mão 1 ∴ mão 2 vence!

2ª Abordagem : Cactus Kev

Análise de complexidade:

- Melhor caso: Ω(1)
- Caso médio: Θ(log n)
- Pior caso: O(log n)

3ª Abordagem : Cactus Kev com Hashing Perfeito

3ª Abordagem : Cactus Kev com Hashing Perfeito

Hashing Perfeito

Sabendo de antemão o conjunto de chaves do problema podemos criar uma tabela com tempo de busca O(1) para o pior caso, usando uma função de hash que não gere colisões.

Uma função de Hashing Perfeito para um conjunto S é uma função que mapeia elementos distintos de S para um conjunto de inteiros, sem colisões.

$$g(x) = (kx \bmod p) \bmod n.$$

3ª Abordagem : Cactus Kev com Hashing Perfeito

Algoritmo:

1. Leitura. Converter as cartas para a representação em bits
2. Determinar se é flush, se sim, então verificar em lookup table valor da mão
3. Caso não seja flush, mas contenha cinco cartas de valores distintos, sendo sequência ou carta alta, verificar em lookup table valor da mão
4. Caso contrário, determinar valor da multiplicação com números primos e utilizar função de hash para determinar valor da mão em tabela
5. Comparação. Comparar o valor de duas mãos, o menor valor vence.

3ª Abordagem : Cactus Kev com Hashing Perfeito

Estruturas de dados utilizados:

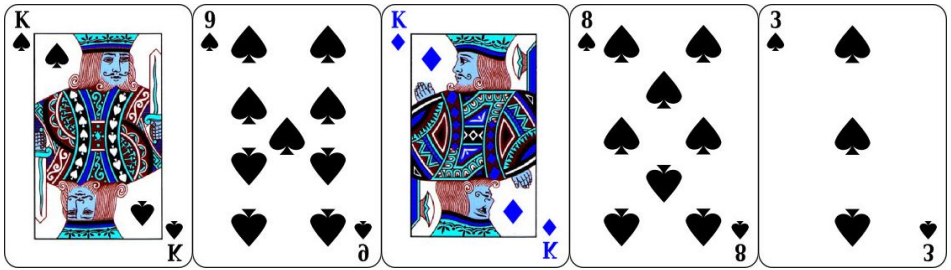
- Array
- Bitwise Operation
- Lookup Table
- Perfect Hashing

3ª Abordagem : Cactus Kev com Hashing Perfeito

Caso de teste: 1.Leitura

Mão 2:

	xxxAKQJT	98765432	CDHSrrrr	xxPPPPPP
KS	00001000	00000000	00011011	00100101
9S	00000000	10000000	00010111	00010011
KD	00001000	00000000	01001011	00100101
8S	00000000	01000000	00010110	00010001
3S	00000000	00000010	00010001	00000011

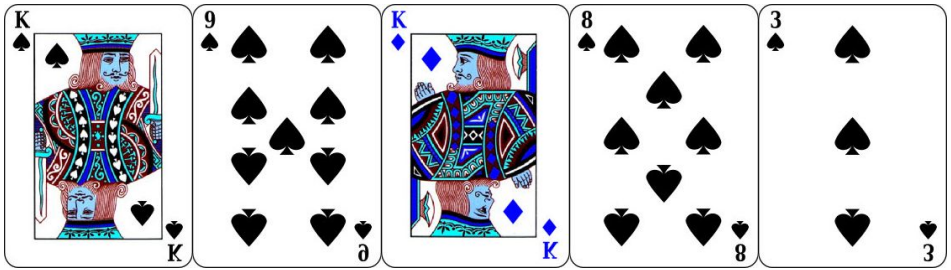


3ª Abordagem : Cactus Kev com Hashing Perfeito

Caso de teste: 4. Multiplicação dos primos

q = (c1&0xFF) * (c2&0xFF) * (c3&0xFF) * (c4&0xFF) * (c5&0xFF)

KS && 0xFF	00100101	37
9S && 0xFF	00010011	19
KD && 0xFF	00100101	37
8S && 0xFF	00010001	17
3S && 0xFF	00000011	3
Mão 2		1326561



⇒ ∴ q = 1326561

retorna hash[find_fast(q)]

3ª Abordagem : Cactus Kev com Hashing Perfeito

Caso de teste: 4. Valores de índice em tabela de hash

Mão 1	724101
Mão 2	1326561

Função de Hash

x1
x2

3ª Abordagem : Cactus Kev com Hashing Perfeito

Caso de teste: 5. Comparação

Ao chamar a função do avaliador com o valor da mão como parâmetro, é retornado um inteiro de 1 a 7462.

Ao fazer isso com mãos diferentes, comparamos o valor de retorno, quanto menor, melhor.

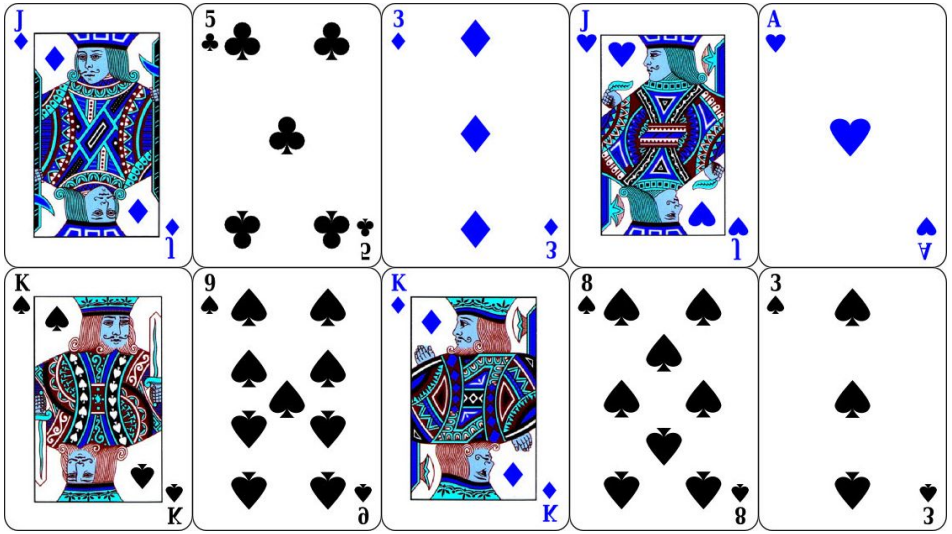
3ª Abordagem : Cactus Kev com Hashing Perfeito

Caso de teste: 5. Comparação

Para categoria “1 Par” valor varia entre [6185...3325]

(Par de J) = Hash[x1] = 4036

(Par de K) = Hash[x2] = 3714



Neste caso (Par de K) < (Par de J) ∴ mão 2 vence!

3ª Abordagem : Cactus Kev com Hashing Perfeito

Análise de complexidade:

- Melhor caso: $\Omega(1)$
- Caso médio: $\Theta(1)$
- Pior caso: $O(1)$

Obrigado(a)!