Lab 4 Report

Card Trick

by Connor McLeod, Matthew Jesus

Mathematical Experiments in Computer Science

December 10, 2023

Section 1:

For our last lab, we examined mathematics being applied in an unexpected way, this being a card trick. The exercise involved Shai and Tiziana, Tiziana would make us pick five cards out of a full deck of cards, then pick one of the cards chosen by us and hide it from Shai. Finally, she would organize the rest of the cards for Shai, who would be in charge of decoding which was the hidden card.

This lab project integrates mathematical theory with computer science applications, unraveling the connections between combinatorics, permutations, and algorithmic problem-solving. The task of encoding and decoding permutations (Programs 1 and 2) leans heavily on combinatorial theory, requiring an understanding of how to efficiently manipulate information. The utilization of ordered sets and the decoding of permutations underscore the fundamental role of combinatorial analysis within computer science, showcasing how mathematical concepts fuel computational methodologies.

Moreover, The investigation goes beyond traditional problem-solving boundaries, delving into the intricate relationship between strategies and mathematical boundaries. The search to extend the magician's card trick to larger decks unveils the limitations of combinatorial strategies. Through the lens of computer science, this becomes a puzzle of optimization, seeking efficient strategies for encoding and decoding data in an ever-expanding domain. The investigation into upper and lower bounds (Programs 3 and 6) underscores the blend of mathematical reasoning and computational exploration, shaping innovative strategies while respecting mathematical constraints.

In essence, this lab vividly illustrates the symbiotic relationship between mathematics and computer science. It's a journey that intricately weaves mathematical principles into the fabric of computational problem-solving, demonstrating how mathematical concepts, from permutations to combinatorial boundaries, form the backbone of algorithmic reasoning and the art of information manipulation.

Section 2:

Program 1:

The purpose of Program 1 is to help a magician perform a card trick to demonstrate combinatorics and discrete mathematics concepts. The program both encodes and decodes a permuation's location in a lexicographical order of n elements. When encoding, the program accepts two int inputs ‘m’ and ‘n’ and produces the m-th permutation of components (Σn!). When decoding, it returns m, where p is the m-th permutation of n elements, given n and a permutation. The idea behind the program is that even though there are only 24 ways to arrange four cards, an accomplice may make 120 different combinations by selecting which cards to reveal in specific orders to indicate what the hidden card is. This program doesn't quite do that whole process but it does however demonstrate a simple method of creating and decoding permutations.

Encoding:

```
 ----jGRASP exec: java program1
Would you like to encode or decode?(e/d)
e
Please enter the n term:
5
Please enter the m term:
3
Starting permutation: [1, 2, 3, 4, 5]
The 3th permutation is: [1, 2, 4, 3, 5]

 ----jGRASP: operation complete.
```

Decoding:

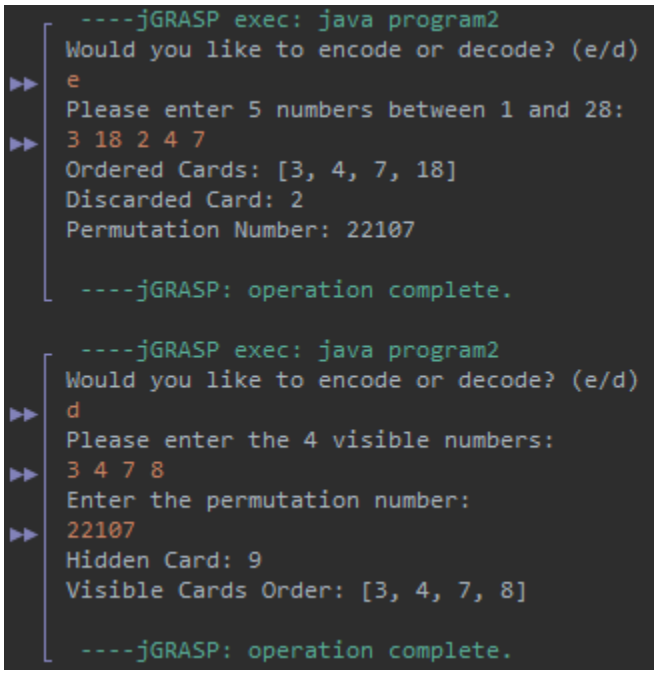
```
 ----jGRASP exec: java program1
Would you like to encode or decode?(e/d)
d
Please enter the n term:
5
Please enter the permutation:
1
2
4
3
5
The permutation is at position: 3

 ----jGRASP: operation complete.
```

Program 2:

The programming challenge involves developing a simulation for a card trick using a 28-card pack. The trick centers around the magician's assistant who must covertly signal to the magician the value of a chosen card. The assistant selects five cards, showcasing four to the magician in a particular sequence. This sequence forms a code that signifies the fifth card, which is the secret card. The program's purpose is to enable the assistant to conceal one card and to sequence the other four in a specific order to communicate the secret. On the other hand, the

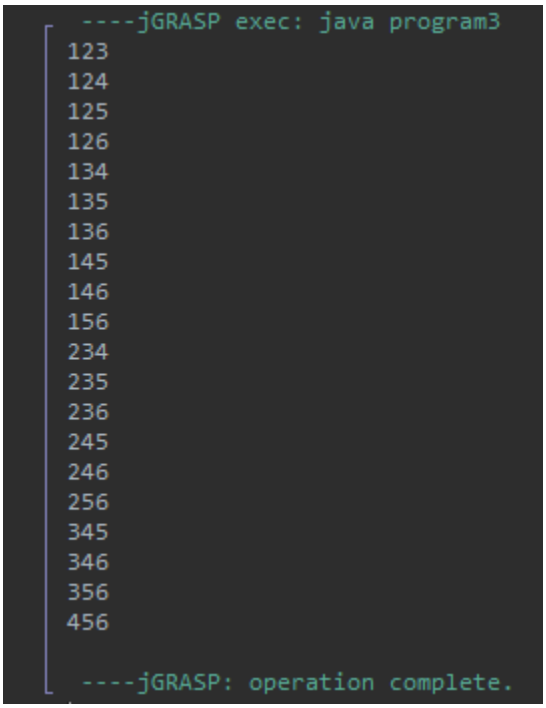
decoding part requires the magician to deduce the secret card from the sequence of the four displayed cards. For example, if the assistant's chosen cards are 2, 4, 6, 8, and 10, they might set aside the '2' and order the remaining ones to indicate that the hidden card is the second smallest in the deck. When the magician receives the cards in the sequence (20, 22, 18, 19), they need to interpret this as the 17th permutation of the possible combinations, with the secret card being the 17th in the stack of unknown cards. The program facilitates both the encoding and decoding parts of the trick, allowing the user to choose which aspect they wish to perform, thus emulating the interactive experience of a magic trick. Unfortunately, I was unable to complete the program but what you see below is the output I was able to get. There does seem to be some kind of problem in encoding which seems to have to do with the ordering of the permuted numbers.

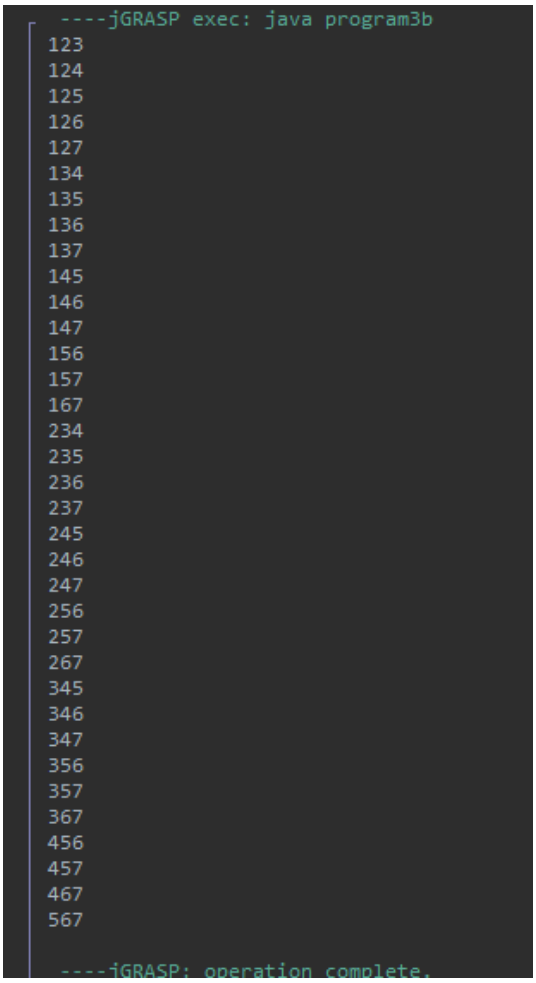


Program 3:

Program 3 involves a computational exploration of combinatorial mathematics as applied to a card trick, first with a deck of six cards and then with seven. Part a of the program successfully generates all possible combinations of choosing three numbers from a set of six (1 through 6), resulting in 20 unique combinations like 123, 124, up to 456, which are displayed in sequence. These combinations represent different card arrangements that the magician's assistant could use to signal a chosen card. Part b challenges the validity of this strategy when extended to a seven-card deck; the hint provided suggests failure after the first few combinations. The exercise concludes with the task of manually devising a successful strategy for seven cards,

implying that the straightforward approach used for six cards does not accommodate the complexity introduced by the additional card.





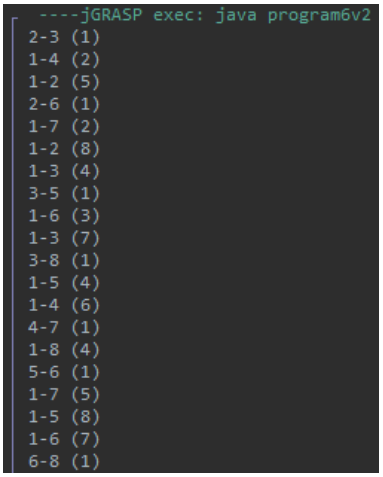
Program 4 - 5:

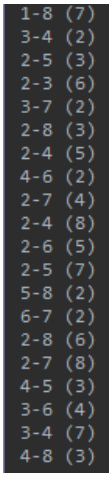
The purpose of Program 4 is to utilize a depth-first search (DFS) strategy to explore various orderings of pairs (ab, ac, bc, ba, ca, cb) for a set of three items (n=3) and an 8-card deck. The goal is to find a legal table where every ordered set appears at most once. By employing a backtracking approach within DFS, the program systematically attempts different pair orderings, filling slots (0 through 55) with these pairs until a legal table is found. However, due to the vast number of potential orderings, some strategies might backtrack extensively, leading to prolonged search times. The program uses a global counter to track backtracking instances and terminates if this count surpasses a predefined threshold to prevent excessive computation. It aims to explore various orderings, such as (ba, ab, ca, ac, cb, bc), and analyzes their success rates and backtracking tendencies to identify patterns in finding efficient solutions.

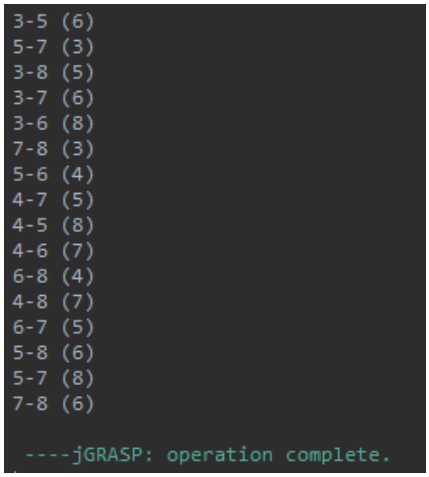
```
Attempting strategy with ordering (ba, ab, ca, ac, cb, bc):
Backtrack limit reached. Exiting.
No legal table found for the given ordering.
```

Program 6:

Program 6 is designed to implement an algorithm for a card trick using a mathematical approach for a set of eight cards, specifically aimed at creating a table with 56 entries. In this program, the deck is represented by an array of eight numbers, and the program's goal is to generate all possible combinations of three numbers (n=3) from this deck. For each combination, it determines a 'hidden' number based on the sum of the three chosen numbers modulo n (the size of the combination). This hidden number is then removed from the list, and the remaining pair is printed alongside the hidden number.







Program 7:

Program 7 is designed to facilitate a card trick with a 124-card deck, handling combinations of five cards. It provides two interactive options: first, when the user inputs five numbers, the program conceals one and orders the remaining four to encode the hidden card's index based on a permutation strategy; second, given four ordered cards, the program deduces the missing fifth card by comparing the input with all possible permutations. This Java program utilizes array manipulation, sorting algorithms, and permutation generation to accomplish its tasks without printing an extensive table, thus enabling the magician and assistant to perform the card trick seamlessly for a large audience. Unfortunately I was unable to get the program to fully work, but most of the functionality is there, I just seem to be having some logic errors.