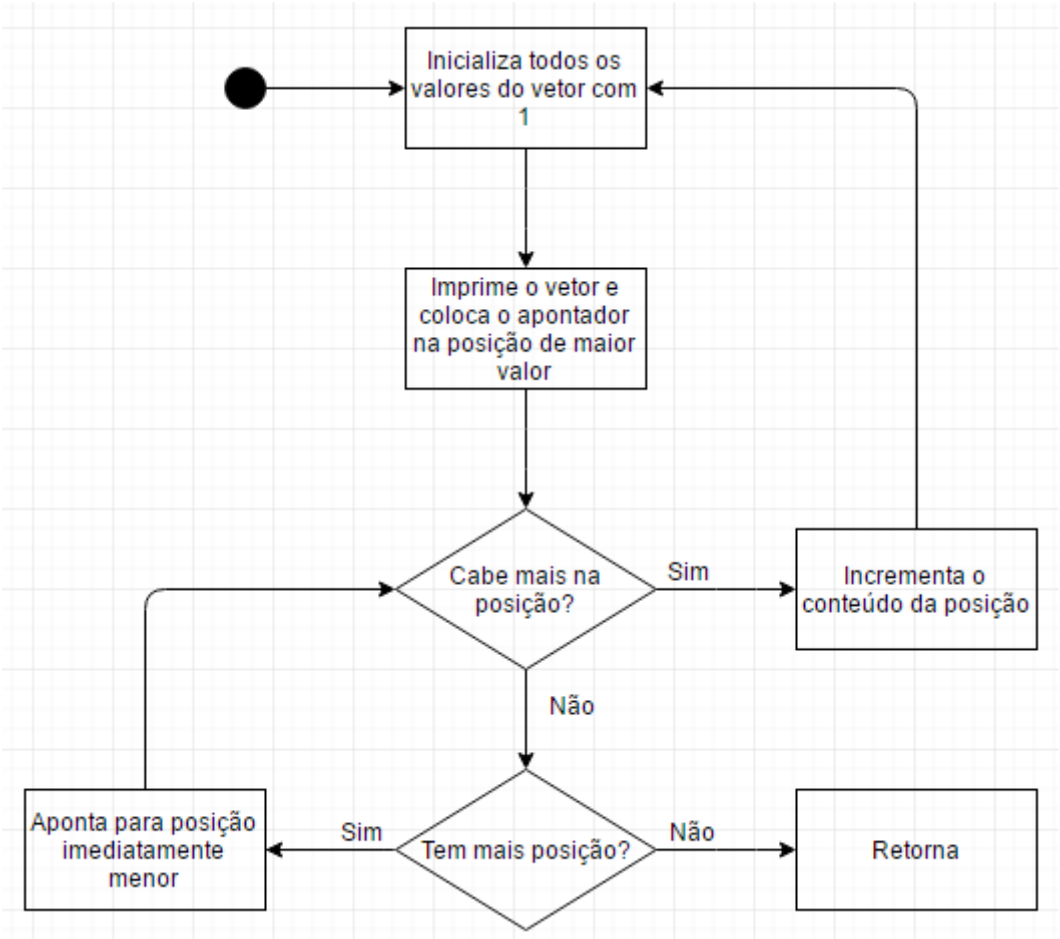
Universidade Federal de Minas Gerais
Disciplina: Matemática Discreta
Professor: Antônio Alfredo Ferreira Loureiro
Aluna: Lorena Mendes Peixoto

Documentação

- **Análise 1**

  A lógica desse programa se baseia na lógica de um relógio digital, em que são feitas comparações e as posições de memória são incrementadas com base no tamanho escolhido pelo usuário.
  No caso de um relógio, que possui o formato _ _ : _ _ : _ _ , as comparações feitas seguem um padrão no qual os valores nunca atingem 24 para as horas, 60 para os minutos ou 60 para os segundos. No caso da permutação com repetição, nenhum dos valores atinge um valor superior ao informado pelo usuário (aqui, chamamos esse valor de n), e este deve estar entre 1 e 20. O relógio tem um tamanho de 6 posições em um vetor (desconsiderando as posições necessárias para os dois pontos), e, na permutação, o tamanho é definido dinamicamente pelo usuário (o chamamos de r), podendo variar de 2 a 10.
  Antes de implementar esse programa, montei o seguinte fluxograma:



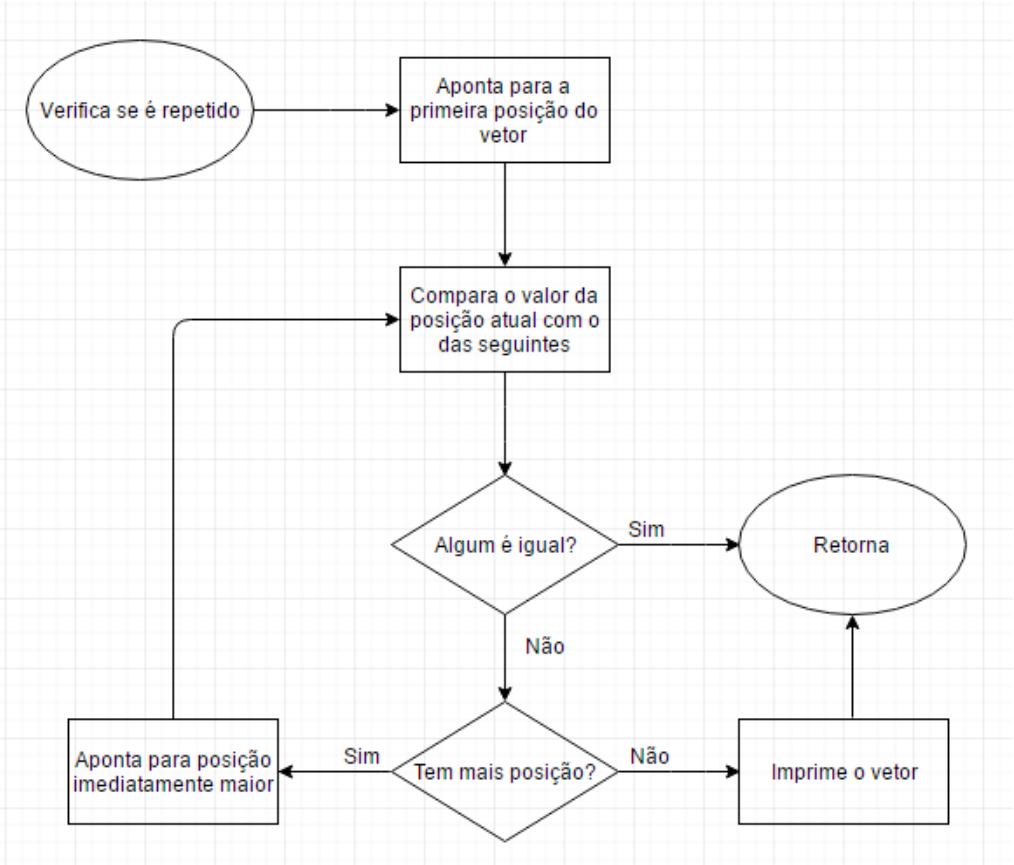
  Aqui, a ideia principal é apontar para a posição mais à direita até atingir seu máximo (r) e, assim, apontar para a posição imediatamente à esquerda e incrementá-la, resetando o valor

da última posição e refazendo esse procedimento de modo a ir desde o valor inicial (definido como 1111...1, sendo n 1's) até o final (rrrr...r, com n r's), imprimindo a cada mudança feita no vetor.
Utilizei, na implementação, um vetor, um ponteiro para percorrê-lo, alocação dinâmica de memória para definir seu tamanho e ferramentas de comparação e loop. Para limitar as entradas a valores permitidos no *range* da especificação, utilizei a estrutura *do...while*, mantendo a solicitação de entrada enquanto o usuário não digitar a correta. Utilizei funções *void* e tipos inteiros de dados. A escolha foi feita considerando a utilização de apenas valores inteiros no processo.

- **Análise 2**

A lógica desse programa se assemelha consideravelmente à do programa anterior. De forma análoga, utiliza a lógica do relógio digital e possui as mesmas especificações de entradas, bem como os mesmos tipos de dados e de funções. O que difere os programas é o fato de que este restringe as impressões de cadas sequência, não permitindo imprimir aquelas que possuem números iguais em uma mesma. Para isso, criei uma função chamada *verificaPermutacao()*, que é chamada no lugar da função *imprimePermutacao()* e, por meio de uma lógica de comparações – esta utiliza dois ponteiros para comparar todos os elementos do vetor e ver se há iguais – , a função para imprimir é chamada ou não. Antes de implementar esse programa, montei os seguintes fluxogramas:

análise2_main



análise2_verificação

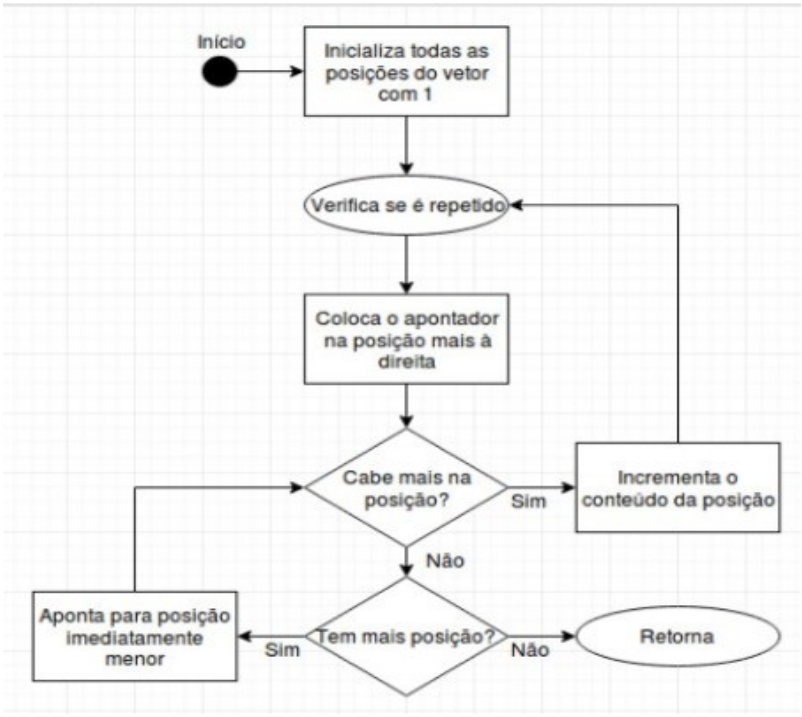
Como é possível perceber, o primeiro fluxograma (*análise2_main*) é similar ao do programa anterior, mas este possui uma rotina de verificar antes de imprimir, rotina que é mostrada no segundo fluxograma (*análise2_verificação*)..

- **Análise 3**

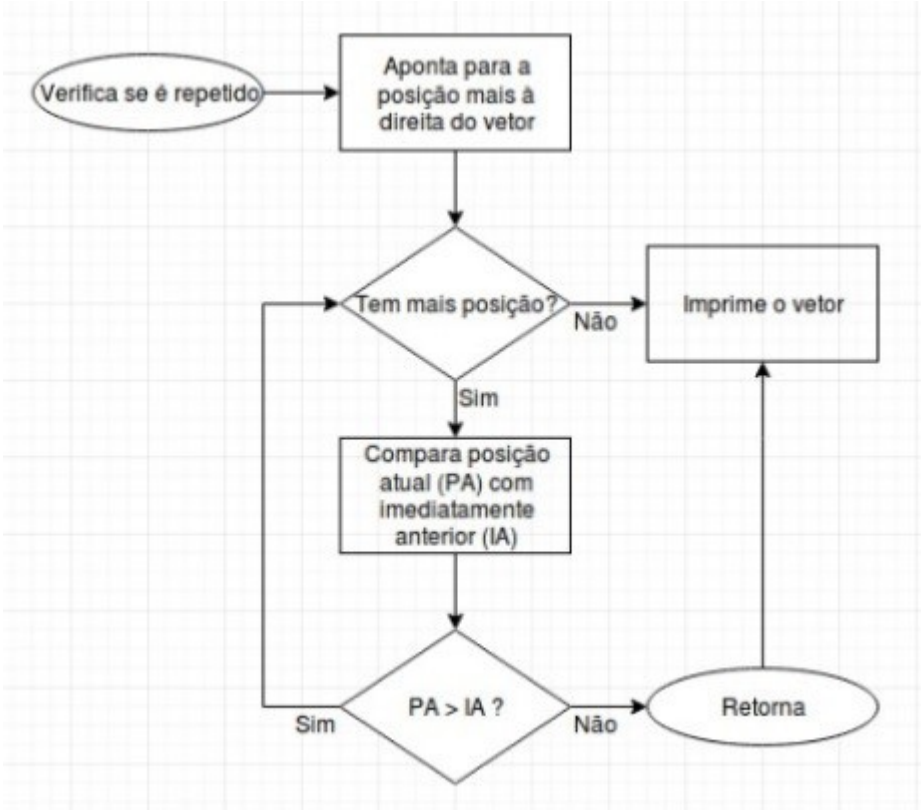
  Esse programa se assemelha muito aos anteriores, em especial ao de “Permutação sem Repetição”. Utilizei a mesma função para gerar permutações, e, delas, apenas imprimi as que formavam conjuntos com elementos não iguais. Para isso, utilizei uma sub-rotina que verifica o se cada número contido na posição de maior índice é maior ou igual ao contido em uma posição de índice imediatamente menor. Em caso positivo, o subconjunto pode ser impresso na tela; do contrário, não.

  A ideia principal que utilizei para desenvolver a lógica do programa é a de que, para todo conjunto ordenado, existe pelo menos um igual desordenado (no caso do conjunto de elementos iguais, o desordenado é ele mesmo). Da mesma forma, para todo conjunto de elementos desordenado, existe um único conjunto ordenado (com ordenado, estou considerando apenas um tipo de ordenação, crescente ou decrescente). Em meu programa, imprimo os primeiros conjuntos ordenados que aparecem em uma sequência de permutação com repetição. Os desordenados não são impressos.

  A seguir, os fluxogramas utilizados para montar o algoritmo:



análise3_main



análise3_verificação