UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
ESCOLA DE INFORMÁTICA APLICADA

# [Relatório do Algoritmo de Clusterização ]

[João Felipe Souza da Silva e Vitor Costa Duarte]

RIO DE JANEIRO, RJ – BRASIL
[AGOSTO] DE [2024]

# 1 - Classes

- Arvbin
- HeapBináriaMínima
- Ponto
- Cluster
- Distancia
- ClusterPrioridadeNaive
- ClusterPrioridade
- Main

## 1.1 - Arvbin

Classe comum vista em sala de aula para a implementação de árvore binária, não foi feita alteração no código desta classe, seu uso teve a função de armazenar as combinações de cluster feitas.

## 1.2 - HeapBinariaMinima

Classe vista em sala de aula, levemente alterada para alocar um vetor de distâncias.
Foi necessário alterar o construtor, adicionando uma segunda entrada de tamanho, para poder realizar a remoção de itens sem problemas.

## 1.3 - Ponto

Classe referente aos objetos ponto utilizados no trabalho, implementa somente a criação do objeto, tendo um construtor com valores de X e Y randômicos para os pontos iniciais e um construtor com valores X e Y recebidos como parâmetro.

## 1.4 - Cluster

Classe referente aos objetos do tipo Cluster, que são junção de dois ou mais pontos, cada cluster possui um vetor armazenando quais pontos estão contidos nele, um ponto “centróide” que é a média dos pontos contidos, o tamanho que equivale a quantidade de pontos contidos, e uma árvore binária para armazená-los para exibição.
Contém um construtor que transforma os pontos gerados em clusters de somente um ponto, e um construtor que cria um novo cluster baseado em 2 outros passados como parâmetro.
Implementa também um método para calcular o centróide de cada cluster, obtendo as coordenadas X e Y dos pontos contidos nele e fazendo sua média.
Além disso implementa também um método compareTo para comparar os centróides dos clusters.

## 1.5 - Distancia

Classe responsável pela distância entre dois clusters, contendo um construtor que recebe 2 clusters e chama um método para calcular a distância entre eles por meio da biblioteca Math, utilizando a fórmula fornecida.
Também contém um método para comparar as distâncias entre si.

## 1.6 - ClusterPrioridadeNaive

Classe responsável pela tarefa de escolher qual cluster está mais próximo e combiná-lo, utilizando as outras classes; Contém um arrayList de clusters e uma HEAP mínima com as distâncias entre os clusters.
Possui um “for” com o método “geraCluster” que recebe o número de entradas (quantidade de pontos a serem utilizados) criando pontos e em sequência clusters, adicionando-os ao array.
Em seguida chama o método “calcDist” da classe, que recebe a lista de clusters e obtém a quantidade de distâncias a serem calculadas, então calcula a distância do primeiro para todos, após o segundo para todos, após o terceiro para todos e assim por diante, adicionando a heap tendo assim a menor distância.

Por fim a classe cria um novo cluster com os pontos mais próximos, remove os utilizados e adiciona o novo a lista, repetindo este processo até que reste somente 1 cluster.

### 1.7 - ClusterPrioridade

Possui os mesmos métodos e funcionamento da classe ClusterPrioridadeNaive,com a adição de um novo método chamado “adicionaDist”, que tem a função de remover as distâncias utilizadas na criação do novo cluster mas não as outras, adicionando o novo cluster gerado, assim não há a necessidade de sempre calcular a distância entre eles, bastando esta conta ser feita somente uma vez.
Este método recebe como parâmetro a Heap de distância dos clusters e por meio de um loop transfere as distâncias existentes mais a nova, sem adicionar as utilizadas na criação do novo clusters, retornando uma heap atualizada com a distância entre cada clusters sem a necessidade de recalcula-la.

### 1.8 - Main

Este método, por meio de um loop, chama a função ClusterPrioridade ou ClusterPrioridadeNaive passando a quantidade de pontos a serem utilizados, e executa o algoritmo 10 vezes, marcando o tempo de cada execução e criando um arquivo de texto com este tempo.

## 2- Observação:

Os testes a partir de 14.000 pontos não foram possíveis devido ao erro “java.lang.OutOfMemoryError: Java heap space”

## 3 - Complexidade:

A complexidade do algoritmo naive é de O (n^2), e do algoritmo otimizado é de O(n^2 log n)

## 4 - Gráficos:

Figura 1: Execução de 10 pontos a 200 pontos, em Milissegundos

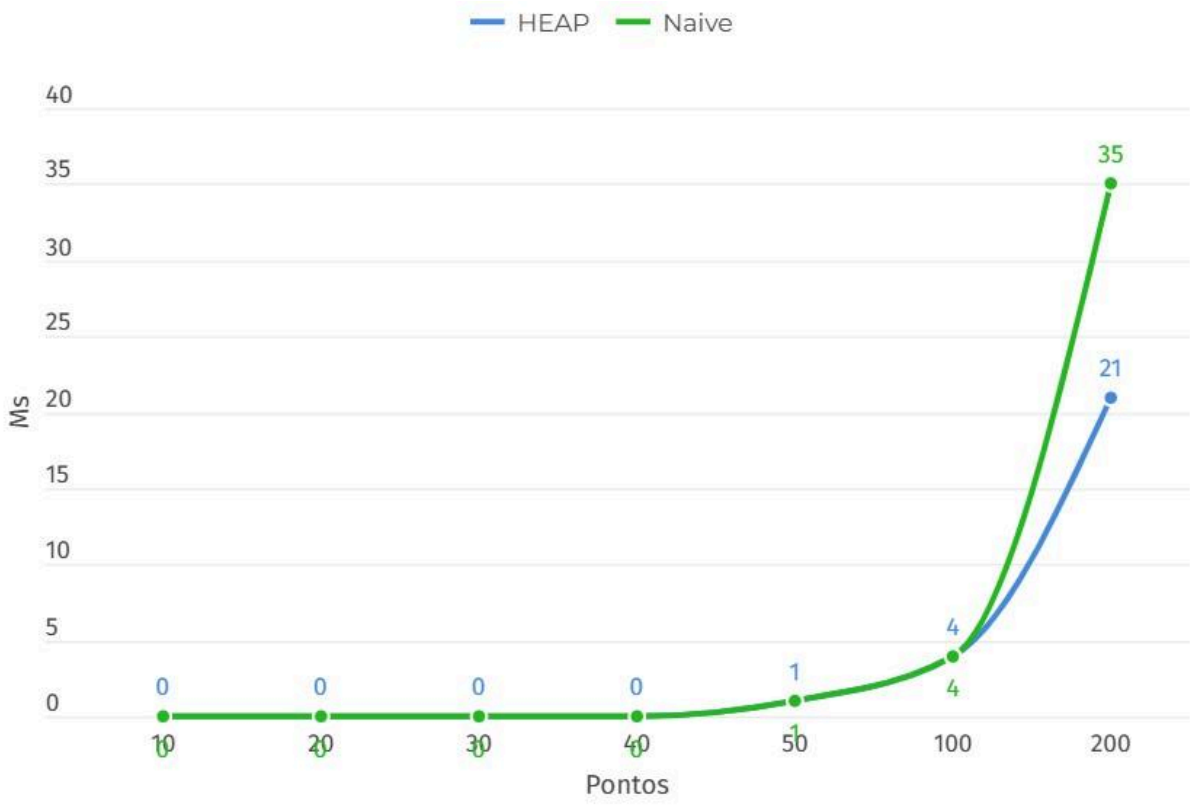



Figura 2: Execução de 500 pontos a 10.000 pontos, em Minutos

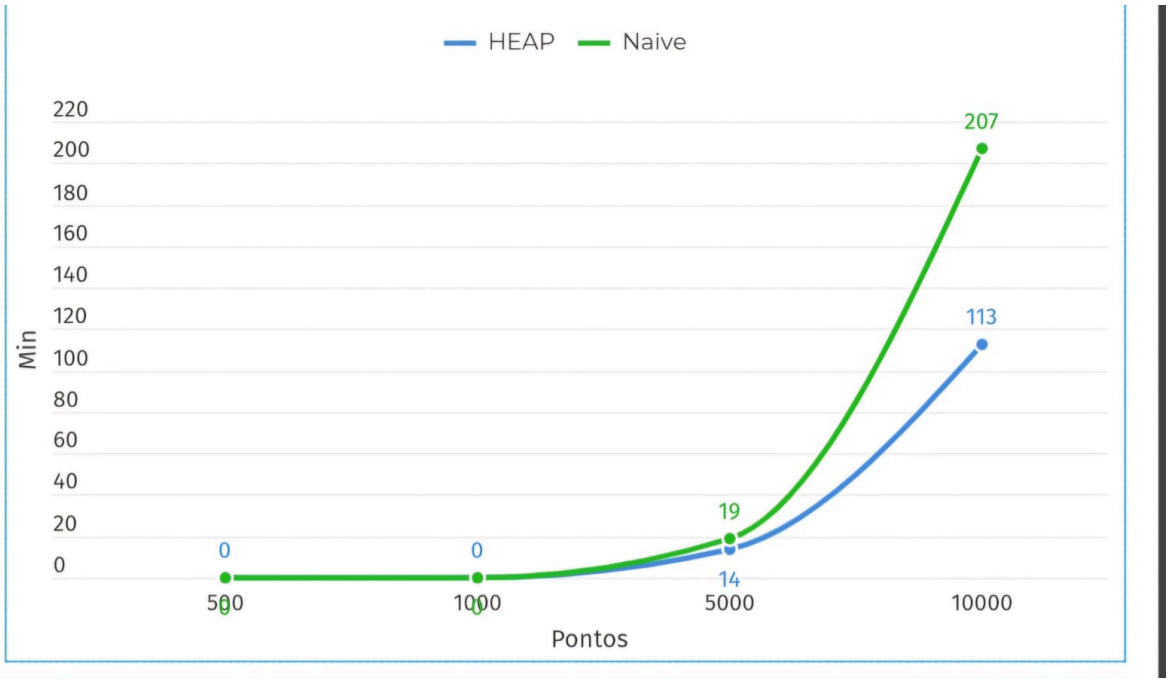
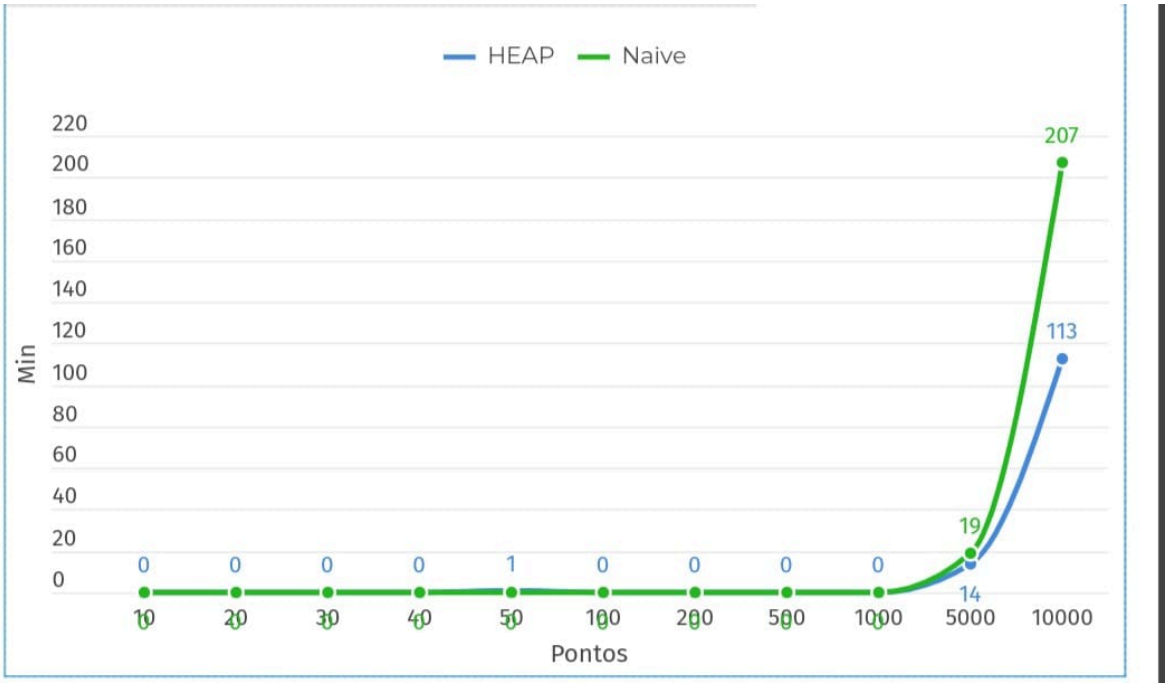


Figura 3: Execução de todos os pontos, 10 a 10.000, em Minutos




Todos os testes foram realizados na mesma máquina, com as configurações descritas abaixo:

-SO: Windows 10 Home

-Ram: 16Gb (8 x 2) 2666mhz

-Processador: i5 10400, 6 núcleos