Universidade Federal do ABC

Projeto de conjuntos disjunos

Jurandir de Mattos Jardim Neto
Davidson Oliveira Rodrigues
Rafael Zambianco

Índice

Introdução.....2
Metodologia.....3
Análise de gráficos.....4
Conclusão.....8

Introdução

Em computação, uma estrutura de dados conjunto-disjunto é uma estrutura de dados que considera um conjunto de elementos particionados em vários subconjuntos disjuntos.

Um algoritmo union-find é um algoritmo que realiza duas operações úteis nesta estrutura de dados:

Find: Determina a qual conjunto um determinado elemento pertence. Também útil para determinar se dois elementos estão no mesmo conjunto.

Union: Combina ou agrupa dois conjuntos em um único conjunto.

Por suportar estas duas operações, uma estrutura de dados conjunto-disjunto é conhecida como uma estrutura de dados union-find ou conjunto merge-find. Uma outra operação importante, MakeSet, que faz um conjunto conter apenas um dado elemento (um singleton), é geralmente trivial.

Com estas três operações, muitos problemas práticos de particionamento podem ser resolvidos.

Metodologia

O projeto basicamente une elementos através de um array de tamanho pré-definido, utilizando uma abordagem de lista encadeada ou floresta, como foi visto nas aulas de AED2.

Basicamente todas as operações necessárias se encontram na classe UnionFind, a qual implementa as funções de union, find, make entre outras auxiliares.

Os elementos que compõem as estruturas são divididos em 2:

LES(lista encadeada simples)

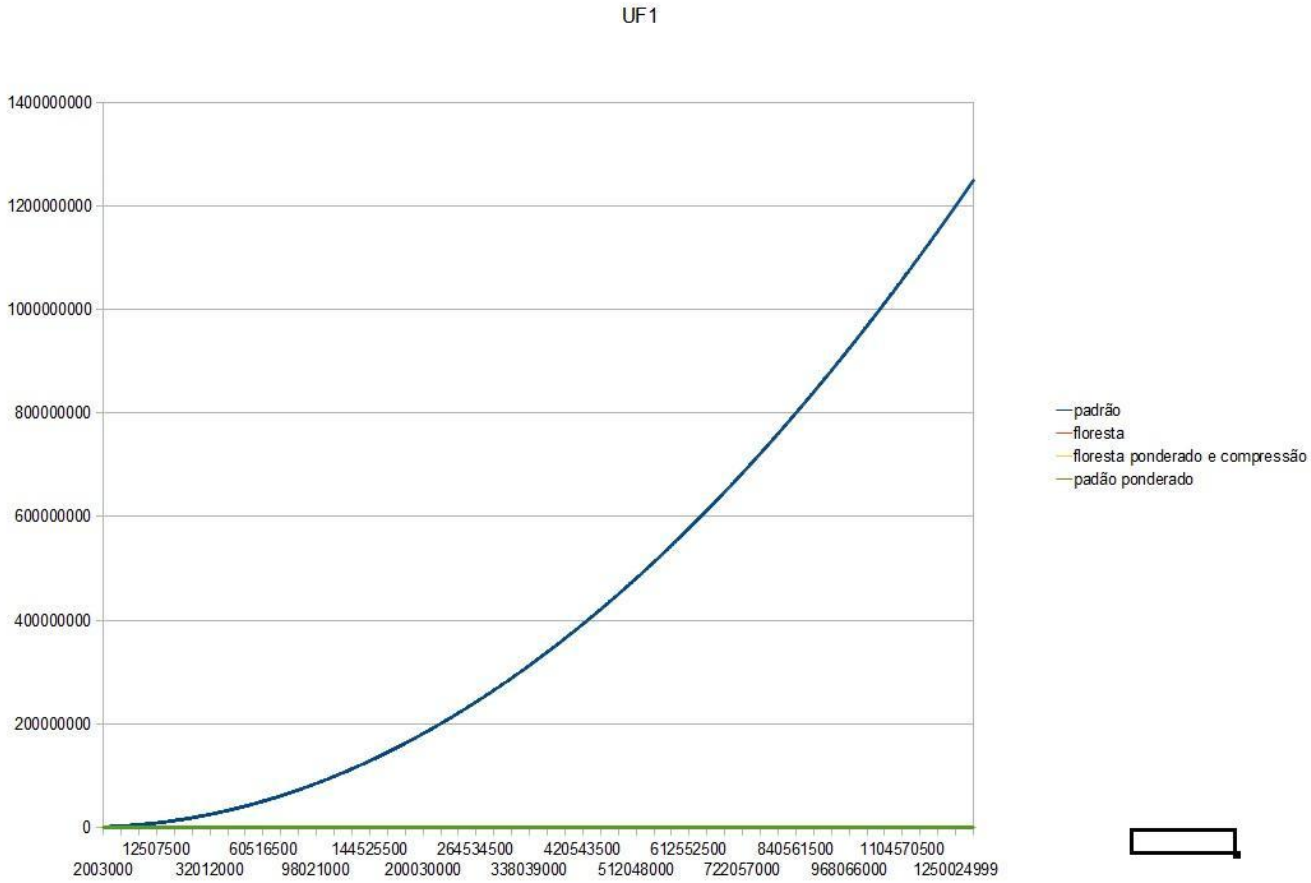
Elemento_Floresta:
É utilizado na implementação de floresta para conjunto disjunto, diferencia-se de LES por apenas possuir um ponteiro que aponta para o elemento pai.

Na implementação dos elementos que compõem o conjunto, foi utilizada a característica de POO herança, pois os elementos tinha alguns elementos em comum (valor e quantidade de filhos e ou elementos no conjunto). Dessa maneira a classe Base, concentra as caracteristicas que seriam encontradas nas duas classes de elementos (LES e Elemento_Floresta).

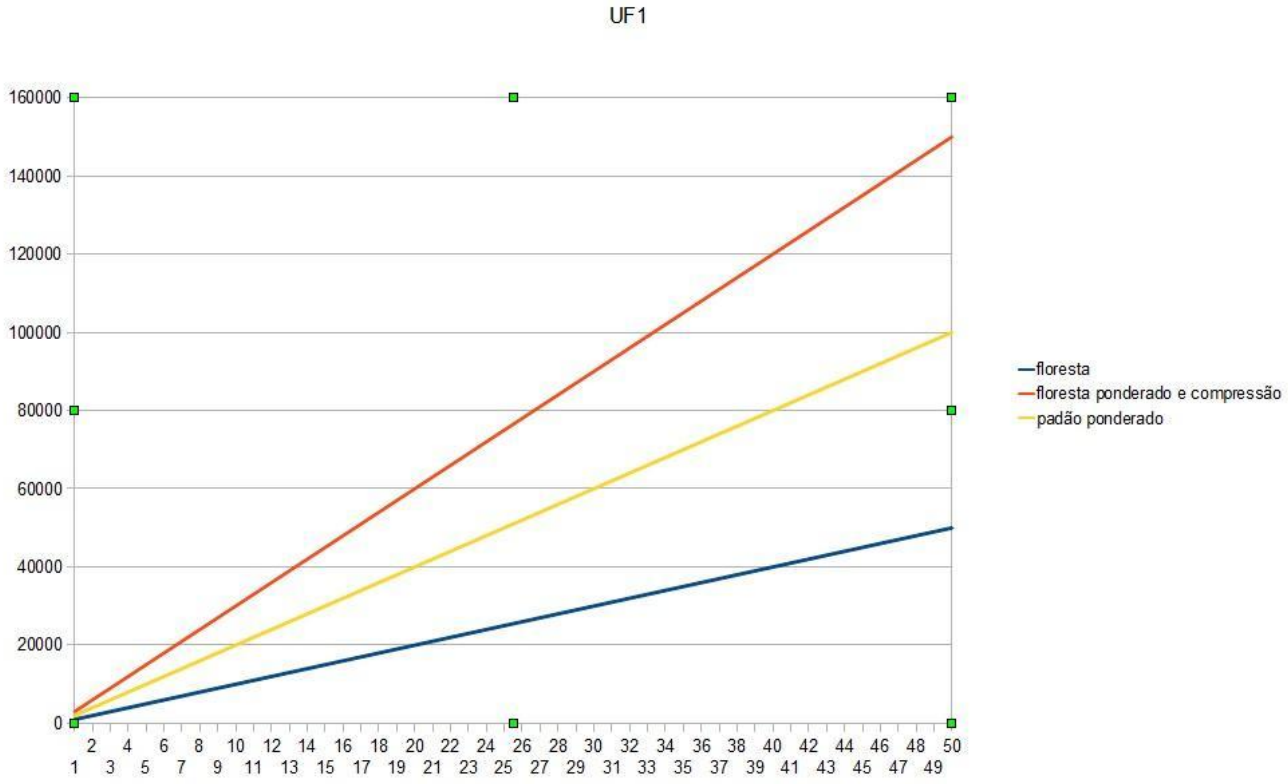
Análise de gráficos

Para os dois arquivos, alguma curva ficou muito acima das outras, o que resultou em um gráfico em que não se pode distinguir muito bem todas as curvas contidas no mesmo (algo semelhante ocorreu no LAB1).

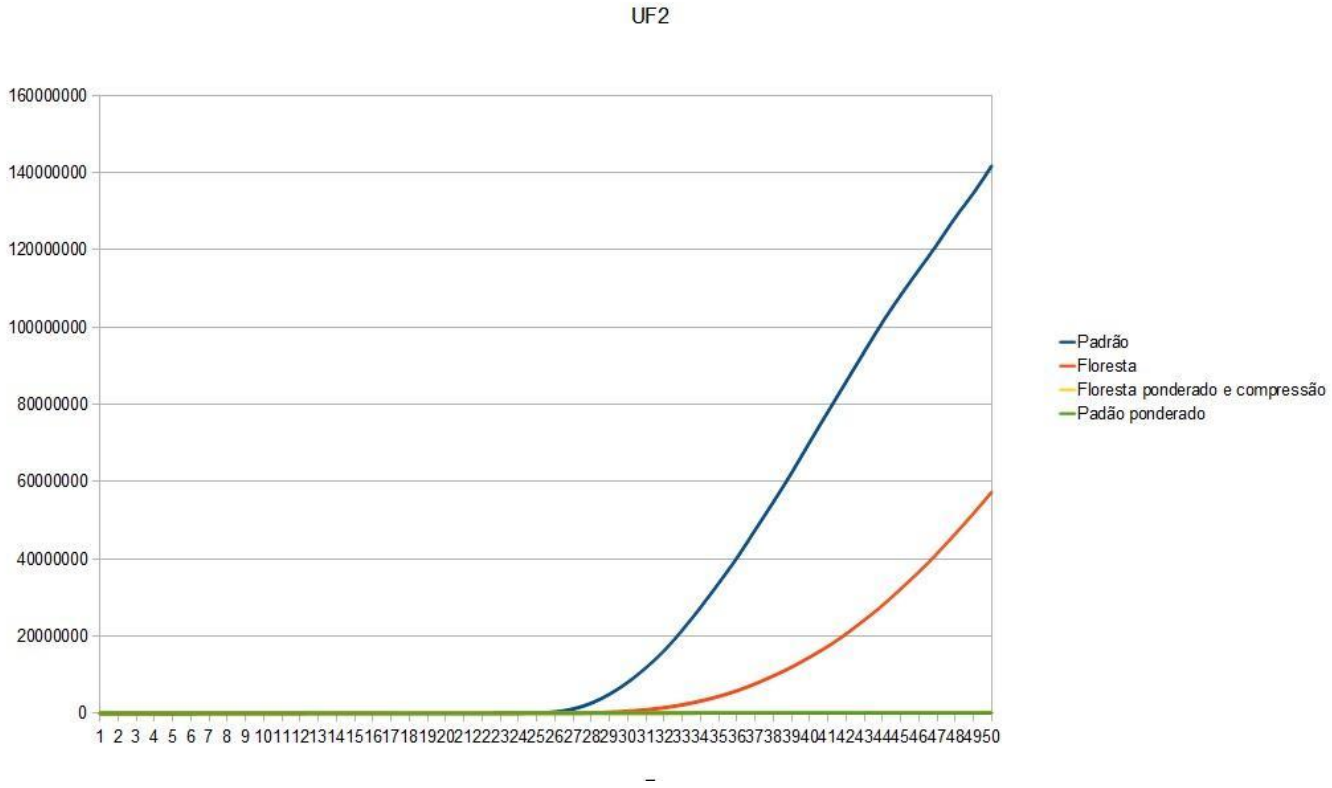
Por isso para uma melhor análise, para cada arquivo foi gerado dois gráficos, um contendo todas as curvas existentes e em outro, contendo apenas as curvas menores.



Par a curva azul (lista encadeada sem heuristica) resultou em uma enorme curva, o que era esperado, já que sem heuristica nenhuma ocorria sempre a unição do grupo da esquerda com o da direita, sendo o primeiro sempre maior.



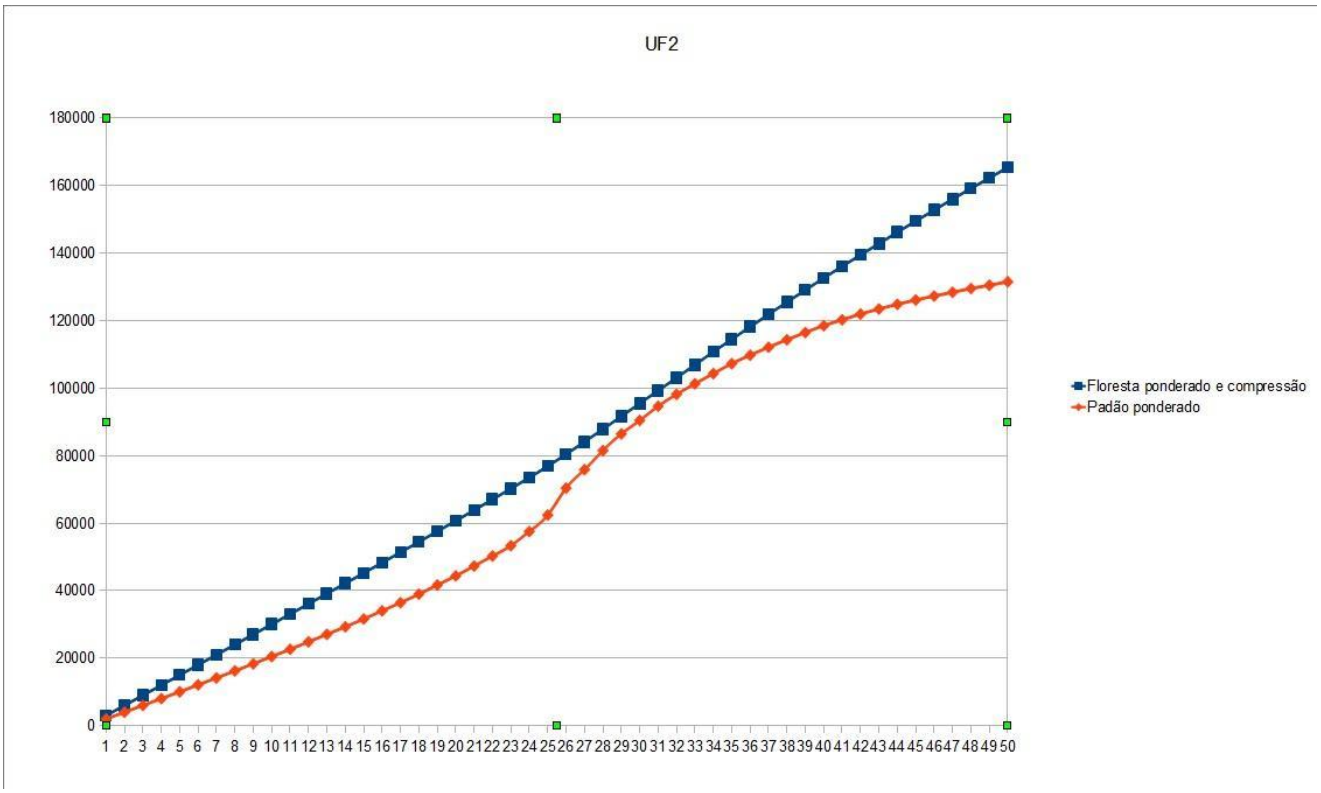
Para o primeiro arquivo nota-se uma quantidade grande de operações no caso de floresta utilizando heuristicas, isso era esperado, pois para a heuristica de compressão de caminhos, operações com ponteiros são muito utilizadas para se encontrar a raiz da arvore para assim poder fazer a compressão, diminuindo o tamanho da arvore.



No caso do segundo arquivo, que se tratava de um arquivo diferente, contendo valores não ordenados como havia no primeiro, pode-se visualizar uma aplicação real da performance das abordagens aplicadas, no caso da floresta sem heuristicas, é possível notar uma quantidade de operações muito maior do que no arquivo uf1, isso se deve ao problema da falta de heuristica. A falta da heuristica (compressao e ponderação) resultou em arvores de tamanhos grandes, esses tamanhos influenciaram drasticamente na performance dos algoritmos como pode-se notar nas curvas do gráfico.

Assim como no caso da árvore a lista encadeada sofre do mesmo mau, tendo que realizar diversas operações para setar o representante em casa elemento do conjunto.

Mais abaixo é possível notar que para o arquivo uf2 as duas abordagens utilizando heuristica, escaparam do grafico de grande curvas significando que suas perfomances foram melhores do que as sem heuristicas, com uma diferença relativamente grande.



Para as curvas que utilizaram heuristicas, pode-se notar no primeiro gráfico a diferença gritante do uso de heuristicas nos algoritmos, as mesmas curvas se encontram no primeiro gráfico, porem ficaram sem valor relevante em comparação com o numero enorme das abordagens sem heuristicas, não chegando a aparecer por completo no primeiro gráfico.

Conclusão

Para ambas abordagens de implementação ficou claro que as heuristicas proporcionam um ganho de performance bem considerável se compararmos com os testes sem heurísticas.

O uso de estruturas union-find pode ser encontrado em grafos, ou em computação gráfica, resolvendo problemas de nivel de detalhamento (LOD).

A estrutura union find foi apresentada com duas heuristicas, (compressão de caminhos e ponderação) sendo que as duas quando usadas melhoram em muito a performance tanto em busca quanto em manipulação da estrutura.