Atividade Prática 11

Aluno: Raphael Henrique Braga Leivas

Matrícula: 2020028101

1 OBJETIVO

O objetivo da prática é visualizar os efeitos que diferentes diferentes configurações de uma estrutura de dados de conjuntos disjuntos têm no tempo de execução.

2 METODOLOGIA

Foi usado o código disponibilizado no moodle, o qual simula várias operações de make set, union e find de Conjuntos Disjuntos, sem alterações no arquivo main.cpp. Os métodos do TAD ConjuntoDisjunto foram implementados no arquivo ConjuntosDisjuntos.cpp de duas formas diferentes:

1. Com otimizações por nível (rank);
2. Sem otimizações por nível (rank);

Para isso, foi criado uma flag booleana hasRankOptimization, que faz com o método Union do TAD utilize ou não a otimização do rank.

A eficiência das operações de find e Union do TAD depende bastante da altura da árvore salva dentro do TAD, aqui representada como um vetor estático de tamanho MAX_TAM = 10000 para simplificar. Para reduzir o tamanho da árvore, usamos um array rank (nível) que indica a altura da árvore do conjunto, de modo que ao juntar dois conjuntos no método Union podemos fazer de tal maneira a reduzir o tamanho da árvore resultante, verificando qual das duas árvores é maior ou menor.

Os passos abaixo exemplificam a otimização por nível (rank). Suponha que vamos juntar dois elementos x e y através do método Union. Seja xset o representante de x, e yset o representante de y.

- Se o rank de xset é menor que o rank de yset, então o pai de xset recebe o yset;
- Se o rank de xset é maior que o rank de yset, então o pai de yset recebe o xset;
- Se o rank de xset é igual ao rank de yset, então o pai de yset recebe o xset, e aumenta o rank do xset em uma unidade.

Com essa metodologia, a altura da árvore será menor e por consequência a eficiência dos métodos find e Union será melhorada, reduzindo o tempo gasto.

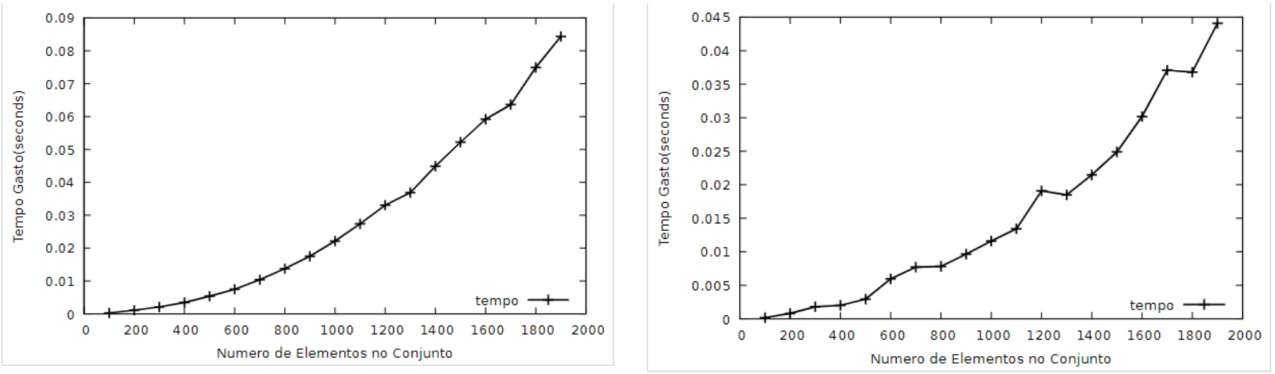
Usando o gnuplot e testando de 100 até 2000 elementos, podemos medir a diferença entre os tempos gastos com e sem a otimização do rank. A implementação feita

pode ser vista dos arquivos ConjuntoDisjunto.cpp e ConjuntoDisjunto.hpp enviados no moodle, junto com esse relatório.

3 RESULTADOS

A Figura 3.1 compara os tempos gastos com e sem a otimização dos ranks, descrita na seção anterior. Note que basta setar a flag hasRankOptimization como true ou false no construtor da classe ConjuntoDisjunto para os testes.

Figura 3.1. Tempos de execução (a) sem otimização por nível e (b) com otimização por nível, testando de 100 a 2000 elementos.

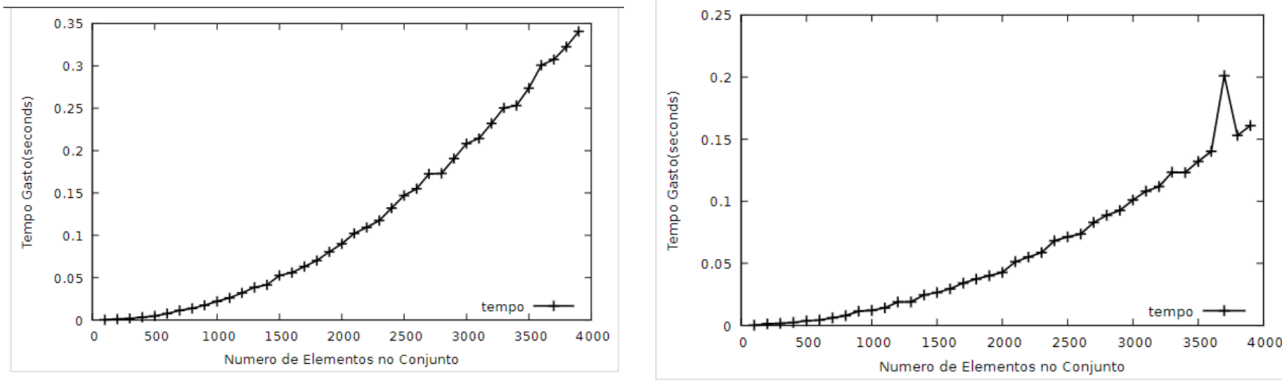


(a) (b)

Fonte: elaboração própria.

Comparando os tempos de execução das Figuras 3.1 (a) e (b), vemos que a otimização por nível causou uma redução de quase 50% no tempo de execução. A Figura 3.2 exibe o mesmo teste, agora com conjuntos de 100 a 4000 elementos.

Figura 3.2. Tempos de execução (a) sem otimização por nível e (b) com otimização por nível, testando de 100 a 4000 elementos.

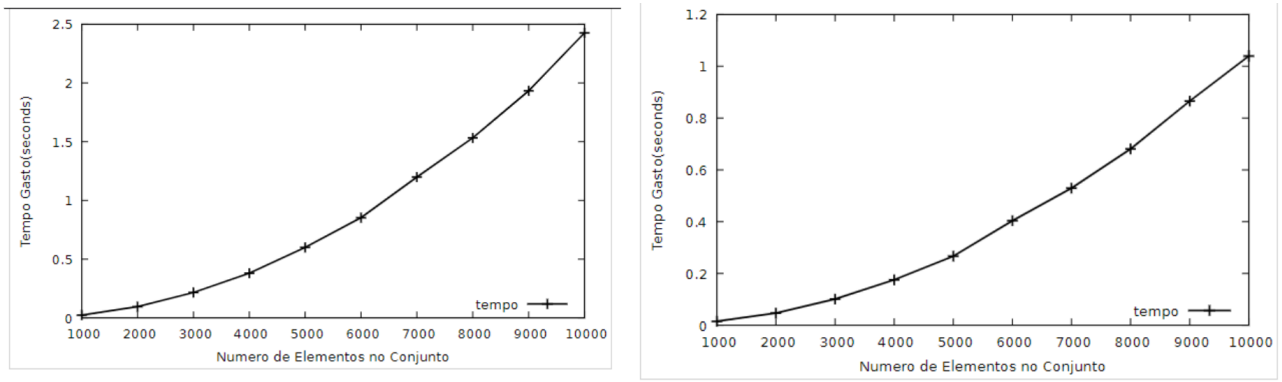


(a) (b)

Fonte: elaboração própria.

Na Figura 3.2, novamente observamos uma redução de quase 50% no tempo de execução com a otimização por nível. Por último, a Figura 3.3 exibe o mesmo teste de 1000 elementos a 10000 elementos. O mesmo comportamento é observado, indicando que a otimização por nível mantém sua redução de 50% com o aumento da entrada.

Figura 3.3. Tempos de execução (a) sem otimização por nível e (b) com otimização por nível, testando de 1000 a 10000 elementos.



Fonte: elaboração própria.

4 CONCLUSÃO

Tendo em vista o objetivo da prática, foi possível verificar o impacto de diferentes configurações e otimizações nos métodos union e find têm no desempenho do TAD Conjunto Disjuntos através da medição dos tempos de execução do código, em especial a otimização por níveis (ranks).