Equipe: Rodrigo Machado - 383877
Yago Cruz – 383879
Curso: Ciência da Computação
Disciplina: Estrutura de Dados Avançadas
Turno: Matutino

Análise computacional das implementações para o algoritmo da Árvore Geradora Mínima do Kruskal

Tabela com os valores mostrados no console, do algoritmo da Árvore Geradora Mínima com as implementações de find_set() e link() em que se aproveita o melhor do desempenho (forma inteligente):

Arquivo	Soma das Arestas	Tempo(ms)
tb8ch60_0.txt	2900	51
tb8ch60_1.txt	2793	6
tb8ch60_2.txt	2763	4
tb8ch60_3.txt	2895	3
tb8ch100_0.txt	3876	25
tb8ch100_1.txt	3611	13
tb8ch100_2.txt	3566	14
tb8ch100_3.txt	3649	15
tb8ch200_0.txt	5225	79
tb8ch200_1.txt	4956	73
tb8ch200_2.txt	4945	51
tb8ch200_3.txt	5201	44
tb8ch300_0.txt	6232	91
tb8ch300_1.txt	6179	90
tb8ch300_2.txt	6040	80
tb8ch300_3.txt	6247	80
tb8ch400_0.txt	7187	196
tb8ch400_1.txt	7152	210
tb8ch400_2.txt	7036	211
tb8ch400_3.txt	7069	283
tb8ch500_0.txt	7942	439
tb8ch500_1.txt	7977	453
tb8ch500_2.txt	7808	508
tb8ch500_3.txt	7716	451
tb8ch600_0.txt	8458	949
tb8ch600_1.txt	8790	876
tb8ch600_2.txt	8557	923
tb8ch600_3.txt	8452	1129
tb8ch700_0.txt	9155	2052
tb8ch700_1.txt	9397	1932

tb8ch700_2.txt	9171	2030
tb8ch700_3.txt	9114	2174
tb8ch800_0.txt	9818	3902
tb8ch800_1.txt	10057	3630
tb8ch800_2.txt	9857	3704
tb8ch800_3.txt	9672	3406
tb8ch900_0.txt	10459	6055
tb8ch900_1.txt	10521	5731
tb8ch900_2.txt	10473	5576
tb8ch900_3.txt	10138	5900
Média dos tempos:		5343,9

Tabela com os valores mostrados no console, do algoritmo da Árvore Geradora Mínima com as implementações de find_set() e link() de forma que não se aproveita o melhor do desempenho (forma burra):

Arquivo	Soma das Arestas	Tempo(ms)
tb8ch60_0.txt	2900	34
tb8ch60_1.txt	2793	10
tb8ch60_2.txt	2763	16
tb8ch60_3.txt	2895	17
tb8ch100_0.txt	3876	28
tb8ch100_1.txt	3611	14
tb8ch100_2.txt	3566	17
tb8ch100_3.txt	3649	17
tb8ch200_0.txt	5225	91
tb8ch200_1.txt	4956	92
tb8ch200_2.txt	4945	82
tb8ch200_3.txt	5201	68
tb8ch300_0.txt	6232	156
tb8ch300_1.txt	6179	141
tb8ch300_2.txt	6040	118
tb8ch300_3.txt	6247	105
tb8ch400_0.txt	7187	290
tb8ch400_1.txt	7152	321
tb8ch400_2.txt	7036	350
tb8ch400_3.txt	7069	419
tb8ch500_0.txt	7942	802
tb8ch500_1.txt	7977	659
tb8ch500_2.txt	7808	893
tb8ch500_3.txt	7716	1032
tb8ch600_0.txt	8458	1813

tb8ch600_1.txt	8790	1591
tb8ch600_2.txt	8557	1225
tb8ch600_3.txt	8452	1557
tb8ch700_0.txt	9155	2577
tb8ch700_1.txt	9397	2563
tb8ch700_2.txt	9171	3253
tb8ch700_3.txt	9114	3546
tb8ch800_0.txt	9818	4387
tb8ch800_1.txt	10057	3665
tb8ch800_2.txt	9857	4125
tb8ch800_3.txt	9672	4295
tb8ch900_0.txt	10459	6597
tb8ch900_1.txt	10521	6908
tb8ch900_2.txt	10473	6393
tb8ch900_3.txt	10138	6117
Média dos tempos:		6638,4

Analisando as duas tabelas acima, podemos chegar a conclusão clara de que a implementação que utiliza de métodos que gera um maior desempenho do algoritmo de Árvore Geradora Mínima do Kruskal. Se analisarmos melhor os dados, veremos que houve um aumento de, aproximadamente, 20% do tempo de execução da forma “inteligente” para a “burra”. Ou seja, apesar de mais simples para programar, a forma que não gera um melhor desempenho para a máquina pode acarretar numa perda de desempenho. O que significa que, dependendo do tipo de aplicação que utilizasse essa implementação, talvez fosse preciso perder um pouco mais de tempo na parte de desenvolvimento do programa, para que se ganhe mais desempenho.

Mas, qual dos dois métodos utilizados na forma “burra” que comprometem mais o desempenho da máquina? Para descobrir isso, vamos realizar um processo de execução do find_set() “inteligente” com o link() “burro” e depois o processo ao contrário.

Tabela com os valores mostrados no console, do algoritmo da Árvore Geradora Mínima com as implementações de find_set() “inteligente” com o link() “burro”:

Arquivo	Soma das Arestas	Tempo(ms)
tb8ch60_0.txt	2900	138
tb8ch60_1.txt	2793	36
tb8ch60_2.txt	2763	16
tb8ch60_3.txt	2895	21
tb8ch100_0.txt	3876	67
tb8ch100_1.txt	3611	34
tb8ch100_2.txt	3566	49
tb8ch100_3.txt	3649	28
tb8ch200_0.txt	5225	142
tb8ch200_1.txt	4956	161

tb8ch200_2.txt	4945	67
tb8ch200_3.txt	5201	34
tb8ch300_0.txt	6232	113
tb8ch300_1.txt	6179	110
tb8ch300_2.txt	6040	122
tb8ch300_3.txt	6247	107
tb8ch400_0.txt	7187	320
tb8ch400_1.txt	7152	339
tb8ch400_2.txt	7036	273
tb8ch400_3.txt	7069	432
tb8ch500_0.txt	7942	615
tb8ch500_1.txt	7977	578
tb8ch500_2.txt	7808	656
tb8ch500_3.txt	7716	621
tb8ch600_0.txt	8458	1226
tb8ch600_1.txt	8790	1283
tb8ch600_2.txt	8557	1336
tb8ch600_3.txt	8452	1247
tb8ch700_0.txt	9155	2229
tb8ch700_1.txt	9397	2306
tb8ch700_2.txt	9171	2314
tb8ch700_3.txt	9114	2204
tb8ch800_0.txt	9818	3864
tb8ch800_1.txt	10057	3841
tb8ch800_2.txt	9857	3839
tb8ch800_3.txt	9672	3700
tb8ch900_0.txt	10459	6596
tb8ch900_1.txt	10521	6346
tb8ch900_2.txt	10473	6150
tb8ch900_3.txt	10138	9622
Média dos tempos:		6318,2

Tabela com os valores mostrados no console, do algoritmo da Árvore Geradora Mínima com as implementações de find_set() “burro” com o link() “inteligente”:

Arquivo	Soma das Arestas	Tempo(ms)
tb8ch60_0.txt	2900	77
tb8ch60_1.txt	2793	10
tb8ch60_2.txt	2763	3
tb8ch60_3.txt	2895	12
tb8ch100_0.txt	3876	42
tb8ch100_1.txt	3611	8

tb8ch100_2.txt	3566	6
tb8ch100_3.txt	3649	22
tb8ch200_0.txt	5225	95
tb8ch200_1.txt	4956	47
tb8ch200_2.txt	4945	22
tb8ch200_3.txt	5201	34
tb8ch300_0.txt	6232	98
tb8ch300_1.txt	6179	79
tb8ch300_2.txt	6040	86
tb8ch300_3.txt	6247	74
tb8ch400_0.txt	7187	239
tb8ch400_1.txt	7152	240
tb8ch400_2.txt	7036	293
tb8ch400_3.txt	7069	408
tb8ch500_0.txt	7942	579
tb8ch500_1.txt	7977	654
tb8ch500_2.txt	7808	596
tb8ch500_3.txt	7716	712
tb8ch600_0.txt	8458	1192
tb8ch600_1.txt	8790	1198
tb8ch600_2.txt	8557	1251
tb8ch600_3.txt	8452	1145
tb8ch700_0.txt	9155	2215
tb8ch700_1.txt	9397	2062
tb8ch700_2.txt	9171	2369
tb8ch700_3.txt	9114	2117
tb8ch800_0.txt	9818	3853
tb8ch800_1.txt	10057	3657
tb8ch800_2.txt	9857	3833
tb8ch800_3.txt	9672	3891
tb8ch900_0.txt	10459	5874
tb8ch900_1.txt	10521	6302
tb8ch900_2.txt	10473	6519
tb8ch900_3.txt	10138	5919
Média dos tempos:		5783,3

Como vimos nas tabelas acima, podemos perceber que o método link() possui uma maior relação com o tempo de execução e com o desempenho de processamento da máquina. Chegamos nessa conclusão com base nos tempos médios de cada execução de teste.

Anteriormente, já tínhamos analisado que com a forma inteligente de implementar os métodos, o desempenho do programa melhoraria. Mas restava saber qual dos dois métodos interferiria mais, se ele tivesse sido implementado na forma “burra”.

Usando a implementação do find_set() não-otimizado com o link() otimizado, vemos pelos cálculos que houve um aumento de, aproximadamente, 8% do tempo de execução sobre a implementação dos dois métodos da forma otimizada. Já se usarmos a implementação do find_set() otimizado com o link() não otimizado, podemos perceber que há um aumento de, aproximadamente, 16% sobre o tempo de execução. Isso significa justamente que o método link() otimizado renderia mais capacidade de desempenho ao algoritmo de Kruskal.