Documentação relacionada ao meu trabalho do RA3

Gabriel Vidal Andreoli

[1]Escola politécnica – Pontificia Universidade Católica do Paraná (PUCPR)

gabrielvidalandreoli@gmail.com

Abstract. *Doing this project only because i want to pass in this subject, even when i dont know why am i still studing to this colege.*

Resumo. *Esse arquivo possui toda a documentação da minha avaliação 3 da matéria de resolução de problemas estruturados em computação.*

1. Inicio

O código que se encontra no meu github possui 3 pastas, cada uma delas possui uma classe registro que será utilizado na tabela hash e uma classe teste onde foram realizados os testes e benchmarks que apareceram nas tabelas mais abaixo.

O pacote 1 possui a classe TabelaPorResto, nessa tabela eu estou fazendo o alocamento de cada registro utilizando a função de resto de divisão, isso também é usado na busca.

O pacote 2 possui a classe TabelaPorMultiplicação, onde para realizar o alocamento é utilizado um hash de multiplicação, esse hash consiste em multiplicar a chave do registro por uma constante entre 0 e 1 e pegar sua parte inteira modulando por um e então multiplicando pelo tamanho da minha tabela. A constante escolhida foi 0.618033988749895, pois é a parte não inteira do número que representa a sequência de Fibonacci.

O pacote 3 possui a classe TabelaPorDobramento, nesse caso é aplicado o hashing de dobramento que consiste em pegar um número inteiro e somar todos os seus algarismos, então se pega o valor resultante e se tira o resto da divisão pelo tamanho da minha tabela.

Importante ressaltar que eu escolhi essas 3 funções de hash pois elas já estavam no anunciado e me pareceram fáceis de se fazer, nessa mesma lógica eu também escolhi o tamanho dos vetores utilizados e também a quantidade de elementos inseridos em cada vetor, exceto pela inserção de 5 milhões de elementos, pois tentar inserir isso tudo em um vetor de 10 posições iria levar mais de 1 hora, então o máximo de inserções feitas foi de 1,5 milhão por ser mais rápido.

Para a realização dos meus testes eu utilizei a seed 735192042, ela foi escolhida de forma aleatória e todas as chaves geradas para os registros possuíam 9 dígitos.

Como já explicado anteriormente, realizei os testes em vetores de 10, 100, 1000, 10000 e 100000 posições, realizando inserções de 20000, 100000, 500000, 1000000 e 1500000 nas minhas tabelas com diferentes tipos de inserção e busca.

2. Tabela com resto de divisão

Veja os resultados da tabela com inserção por resto da divisão a seguir (inserção, número de colisões e tempo de busca):

resto							
	incerção						
		Tamanho Do vetor					
	nº de elementos		20 k	100 k	500 k	1 m	1,5 m
		10	71ms	3,5 s	2 min	8 min	19 min
		100	25 ms	566 ms	12 s	38 s	1,5 min
		1000	37 ms	137 ms	2 s	5 s	12 s
		10000	58 ms	180 ms	995 ms	2 s	4 s
		100000	238 ms	1 s	4 s	9 s	14 s

colisões	20 k	100 k	500 k	1 m	1,5 m
10	19990	99990	499990	999990	1499990
100	19900	99900	499900	999900	1499900
1000	19000	99000	499000	999000	1499000
10000	11335	90000	490000	990000	1490000
100000	1965	36998	400691	900004	1400000

busca						
	Tamanho Do vetor	958592912	980708018	999420024	905985948	995523228
nº de elementos		20 k	100 k	500 k	1 m	1,5 m
	10	2 ms	2 ms	6 ms	7 ms	8 ms
	100	1 ms	2 ms	2 ms	4 ms	2 s
	1000	1 ms	2 ms	1 ms	1 ms	1 ms
	10000	1 ms	1 ms	1 ms	1 ms	1 ms
	100000	3 ms	4 ms	1 ms	4 ms	4 ms

3. Tabela com multiplicação

Veja os resultados da tabela com inserção por multiplicação a seguir (inserção, número de colisões e tempo de busca):

Multiplicação							
	inserção						
		Tamanho Do vetor					
	nº de elementos		20 k	100 k	500 k	1 m	1,5 m
		10	71 ms	3 s	2,5 min	7,6 min	18 min
		100	30 ms	623 ms	15 s	40 s	1,9 min
		1000	38 ms	163 ms	2 s	5 s	12 s
		10000	56 ms	174 ms	971 ms	2 s	4 s
		100000	225 ms	1 s	4 s	9 s	14 s

colisões	20 k	100 k	500 k	1 m	1,5 m
10	19990	99990	499990	999990	1499990
100	19900	99900	499900	999900	1499900
1000	19000	99000	499000	999000	1499000
10000	11354	90001	490000	990000	1490000
100000	1865	36858	400690	900001	1400000

busca						
	Tamanho Do vetor	958592912	980708018	999420024	905985948	995523228
nº de elementos		20 k	100 k	500 k	1 m	1,5 m
	10	1 ms	2 ms	8 ms	9 ms	12 ms
	100	2 ms	1 ms	2 ms	3 ms	3 ms
	1000	2 ms	1 ms	1 ms	2 ms	2 ms
	10000	2 ms	1 ms	1 ms	2 ms	2 ms
	100000	4 ms	4 ms	4 ms	4 ms	5 ms

4. Tabela com dobramento

Veja os resultados da tabela com inserção por dobramento a seguir (inserção, número de colisões e tempo de busca):

Dobramento						
	inserção					
		Tamanho Do vetor				
	nº de elementos		20 k	100 k	500 k	1 m	1,5 m
		10	68 ms	3 s	2 min	9 min	23 min
		100	62 ms	1 s	53 s	3,8 min	9,6 min
		1000	58 ms	1 s	54 s	3,7 min	9,4 min
		10000	64 ms	1 s	52 s	3,4 min	9,4 min
		100000	62 ms	1 s	50 s	3,5 min	9,4 min

colisões	20 k	100 k	500 k	1 m	1,5 m
10	19990	99990	499990	999990	1499990
100	19940	99931	499929	999929	1499946
1000	19940	99931	499929	999929	1499946
10000	19940	99931	499929	999929	1499946
100000	19940	99931	499929	999929	1499946

busca						
	Tamanho Do vetor	958592912	980708018	999420024	905985948	995523228
nº de elementos		20 k	100 k	500 k	1 m	1,5 m
	10	1 ms	2 ms	4 ms	8 ms	10 ms
	100	1 ms	3 ms	2 ms	4 ms	6 ms
	1000	1 ms	2 ms	4 ms	4 ms	7 ms
	10000	1 ms	1 ms	2 ms	4 ms	4 ms
	100000	3 ms	3 ms	4 ms	8 ms	8 ms

(obs: o valor acima da quantidade de elementos é o valor que estava sendo procurado em cada quantidade de registros na tabela, todos os testes procuraram exatamente os mesmos números)

5. Conclusões

Executando os testes e analisando os resultados eu pude tirar as seguintes conclusões:

Tabelas que possuem um vetor de registros muito pequeno como os de 10 não são recomendados por serem muito mais lerdos que tabelas com vetores grandes, estou assumindo que isso se deva ao fato de que por possuir poucas possibilidades de onde alocar os registros eles ficam muito afunilados, e como a cada inserção na tabela o registro é comparado com os outros da mesma coluna, ocorre uma perda enorme de performance quando há a inserção de muitos registros, por exemplo no caso da inserção de 1,5 milhão de registros, onde era necessário realizar milhares de comparações para inserir um único elemento dentro das 10 colunas.

Ao mesmo tempo que trabalhar com vetores pequenos não é o indicado, trabalhar com vetores muito grandes como o de 100000 posições também não é muito benéfico, acredito que ele perde um pouco de sua performance pois é necessário percorrer muitos índices do vetor para finalmente encontrar o lugar onde o registro vai ser alocado.

Em geral os melhores tamanhos para se trabalhar com a inserção são os de 10000 e de 1000 pois com os testes se provaram ter a melhor performance, com exceção da inserção por dobramento, o que irá ser explicado mais adiante. Tanto a tabela que possui a inserção por hashing de resto da divisão quanto por multiplicação possuem uma performance muito boa e muito parecida na inserção, já na busca a de divisão tem uma performance um pouquinho melhor, mas nada muito perceptível.

A tabela com inserção e busca por dobramento possuiu o pior desempenho das 3 tanto em inserção quanto em busca, isso pode se dever ao fato de que, desconsiderando o vetor de 10 posições, todos os tamanhos de vetores possuíam a mesma quantidade de colisões, isso indica que a distribuição não está sendo eficiente e todos os registros estão sendo alocados nas mesmas colunas ao invés de serem dispersados entre as várias possíveis. Isso também explicaria o motivo da busca ser mais lenta que nas outras tabelas.