Relatório: Experimentos com funções de hash

(1.1) (pergunta mais simples e mais geral) porque necessitamos escolher uma boa função de hashing, e quais as consequências de escolher uma função ruim?

É importante escolher uma boa função hash pois caso ela seja mal escolhida ela ficará mal distribuída, ou seja, os elementos não serão distribuídos de maneira uniforme entre os buckets. Nesse sentido, ocorreram muitas colisões o que deixara recursos como busca e remoção lentos, com mal desempenho.

(1.2) porque há diferença significativa entre considerar apenas o 1o caractere ou a soma de todos?

Há diferença pois quando se considera apenas o primeiro caractere para o valor da função de hashing há uma chance maior dos dados não se espalharem uniformemente, principalmente quando há muitas strings que começam com o mesmo caractere. Por outro lado, quando se usa a soma dos caracteres todo caractere contribui para o valor de hash, sendo mais difícil 2 elementos terem o mesmo valor e, portanto, distribuindo melhor.

(1.3) porque um dataset apresentou resultados muito piores do que os outros, quando consideramos apenas o 1o caractere?

Justamente pelo fato de que ele possuí muitos elementos com o primeiro caractere igual.

(2.1) com uma tabela de hash maior, o hash deveria ser mais fácil. Afinal temos mais posições na tabela para espalhar as strings. Usar Hash Table com tamanho 30 não deveria ser sempre melhor do que com tamanho 29? Porque não é este o resultado? (atenção: o arquivo mod30 não é o único resultado onde usar tamanho 30 é pior do que tamanho 29)

Não deveria, pois o hash 30 por ter muitos divisores (1,2,3,5,6,10,15,30) faz com que haja uma maior probabilidade de colisões em posições em certas posições da tabela para algumas sequencias de caracteres. Ou seja, há uma maior chance de diminuição da distribuição uniforme quando se compara com um hash 29, uma vez que 29 é primo e não possui divisores. Sendo assim, sempre é uma boa estratégia escolher um tamanho primo para a tabela hash.

(2.2) Uma regra comum é usar um tamanho primo (e.g. 29) e não um tamanho com vários divisores, como 30. Que tipo de problema o tamanho primo evita, e porque a diferença não é muito grande no nosso exemplo?

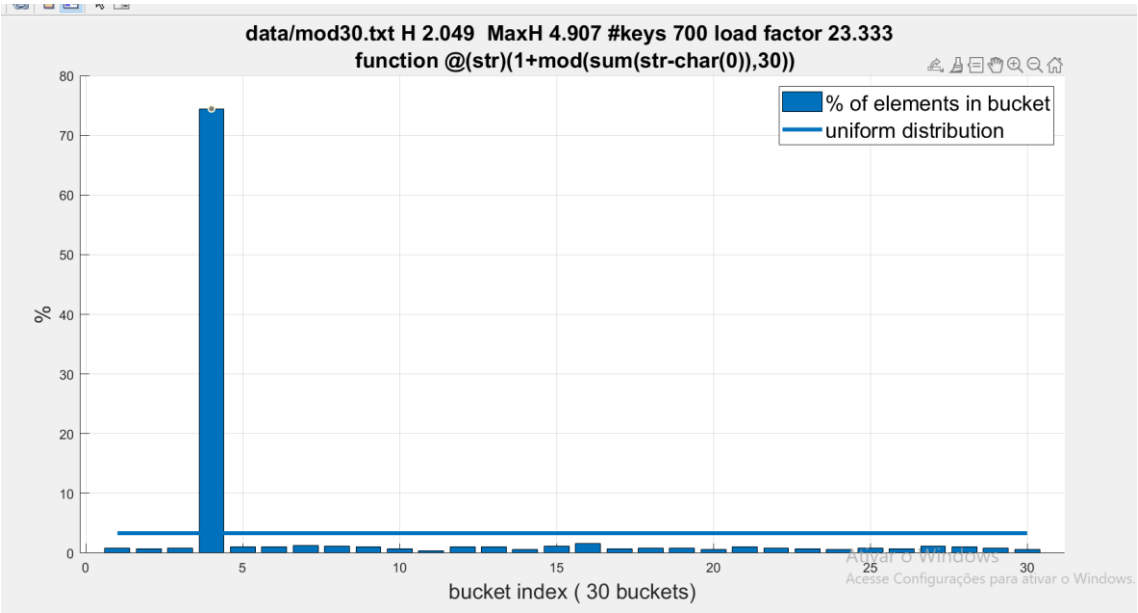
O hash 30 por ter muitos divisores (1,2,3,5,6,10,15,30) faz com que haja uma maior probabilidade de colisões em posições em certas posições da tabela para algumas sequencias de caracteres. Ou seja, há uma maior chance de diminuição da distribuição uniforme quando se compara com um hash 29, uma vez que 29 é primo e não possui divisores. Sendo assim, sempre é uma boa estratégia escolher um tamanho primo para a tabela hash.

Alguns pacotes de dados não tiveram uma diferença tão grande no desempenho pois eles não foram gerados especificamente para atacar essas posições múltiplas, mas se pegarmos dados

enviesados para esse ataque, como o pacote mod3, vemos que ocorrem múltiplas colisões em uma posição específica.

(2.3) note que o arquivo mod30 foi feito para atacar um hash por divisão de tabela de tamanho 30. Explique como esse ataque funciona: o que o atacante deve saber sobre o código de hash table a ser atacado, e como deve ser elaborado o arquivo de dados para o ataque. (dica: use plothash.h para plotar a ocupação da tabela de hash para a função correta e arquivo correto. Um exemplo de como usar o código está em em checkhashfunc)

O atacante deve saber que a função hash utiliza a operação de módulo 30 para calcular o índice da tabela, assim ele pode gerar uma sequencia de chaves que colidam em um mesmo índice da tabela como foi visto no gráfico específico:



(3.1) com tamanho 997 (primo) para a tabela de hash ao invés de 29, não deveria ser melhor? Afinal, temos 997 posições para espalhar números ao invés de 29. Porque às vezes o hash por divisão com 29 buckets apresenta uma tabela com distribuição mais próxima da uniforme do que com 997 buckets?

Mesmo aumentando o tamanho, caso a tabela não seja bem projetada, isso pode fazer com que haja um maior numero de colisões em determinadas regiões da tabela, assim, mesmo aumentando o tamanho da tabela, os valores continuarão a se concentrar, próximos uns dos outros em regiões especificas, o que gerará uma distribuição desigual de elementos. No caso de um projeto baseado na soma, strings menores irão se concentrar sempre no começo da tabela, no caso de tabelas muito grandes, diminuindo a uniformidade dos dados.

(3.2) Porque a versão com produtório (prodint) é melhor?

O método do produtorio utiliza a multiplicação dos valores SI para base do cálculo do hash ao invés da soma, isso tende a distribuir melhor os valores na tabela. Sendo também mais

resistente a ataques por colisão, pois a mudança de um caractere gera uma mudança grande no valor do hash.

(3.3) Porque este problema não apareceu quando usamos tamanho 29?

Dica: plote a tabela de hash para as funções e arquivos relevantes para entender a causa do problema - não está visível apenas olhando a entropia. Usar o arquivo length8.txt e comparar com os outros deve ajudar a entender. Isto é um problema comum com hash por divisão.

Dica: prodint.m (verifiquem o código para entender) multiplica os valores de todos os caracteres, mas sem permitir perda de precisão decorrente de valores muito altos: a cada multiplicação os valores são limitados ao número de buckets usando mod.

No caso de um projeto baseado na soma, strings menores irão se concentrar sempre no começo da tabela, no caso de tabelas muito grandes, diminuindo a uniformidade dos dados, o que já não acontece para tabelas menores como a de tamanho 29.

(4) hash por divisão é o mais comum, mas outra alternativa é hash de multiplicação (NÃO É O MESMO QUE prodint.m, verifiquem no Corben). É uma alternativa viável? porque hashing por divisão é mais comum?

O hash por divisão é mais comum pois geralmente produz bons resultados para tamanhos primos e é muito fácil de implementar. A implementação do hashing por multiplicação é bem mais difícil e nem sempre gera melhores resultados que o de divisão, sendo mais eficiente apenas em alguns casos.

(5) Qual a vantagem de Closed Hash sobre OpenHash, e quando escolheríamos Closed Hash ao invés de Open Hash? (pesquise! É suficiente um dos pontos mais importantes)

Existem algumas vantagens em se utilizar a Closed hash, uma delas é quando há necessidade de otimizar o uso de memória, com a closed hash não há necessidade de armazenar ponteiros para listas encadeadas, o que economiza espaço na memória. Quando o tamanho da tabela hash é conhecido e limitado e o acesso aleatório de elementos não é importante, a closed hash se torna mais vantajosa para gerar distribuições mais uniformes.

(6) Suponha que um atacante conhece exatamente qual é a sua função de hash (o código é aberto e o atacante tem acesso total ao código), e pretende gerar dados especificamente para atacar o seu sistema (da mesma forma que o arquivo mod30 ataca a função de hash por divisão com tamanho 30). Como podemos implementar a nossa função de hash de forma a impedir este tipo de ataque? Pesquise e explique apenas a idéia básica em poucas linhas (Dica: a estatística completa não é simples, mas a idéia básica é muito simples e se chama Universal Hash)

A ideia é ter um conjunto de funções hash aleatórias e escolher uma dessas funções de forma aleatória para cada inserção na tabela, assim, mesmo que o atacante saiba a função hash utilizada, ele não saberá a próxima, e portanto não poderá criar uma estratégia especifica de ataque.