Relatório: Experimentos com funções de hash

Para entender os resultados dos experimentos e responder as perguntas, supõe-se pesquisar em bibliografia os conceitos e verificar o conteúdo de cada arquivo de dados (os nomes dos arquivos de dados são boas dicas).

(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?

A escolha de uma boa função de hashing impacta diretamente na quantidade de colisões que o algoritmo da hash table apresentará durante seu funcionamento. Uma função ruim criaria uma hash table com uma entropia alta, reduzindo a eficiência do algoritmo.

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

Porque é muito mais comum duas entradas iniciarem com o mesmo caractere do que ambas possuírem o mesmo valor de soma dos caracteres. Isso aumenta o número de colisões.

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

Provavelmente esse dataset apresenta várias strings iniciadas pelo mesmo caracter, em comparação com os outros datasets.

(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 <u>sempre</u> 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)

Isso não é verdade, a tabela com tamanho 30 não necessariamente é melhor que a com tamanho 29. Isso ocorre pois o número 30 não é primo, de forma que ao utilizarmos mod30, podem haver colisões devido a divisores comuns entre o 30 e o valor da soma.

(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 uso de primos evita com que ocorram colisões devido a divisores comuns entre o valor do tamanho e o valor da função no elemento a ser adicionado. Isso faz com que os elementos fiquem mais bem distribuídos por toda a hash table.

(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)

Para atacar um certo algoritmo de hash table, é necessário que o atacante conheça o tamanho da hash table e qual é a função de hashing. Com essa informações, ele consegue montar exemplos de elementos que colidam valores quando se aplica mod no tamanho específico, aplicado naquela função.

(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?

Como a maioria das strings são pequenas, o valor da soma delas é menor que 997, de forma que a maioria dos valores de hashing encontrados estão distribuídos num range de onde as somas dos valores dos caracteres chega em média. Portanto, como 29 é menor que esse valor, existe maior distribuição para quando o tamanho da hash table é 29.

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

Porque um mesmo número pode ser escrito de muito mais formas como uma soma do que como um produto. Assim com o produtório acontece muito menos casos de strings com tamanhos parecidos possuírem o mesmo valor de hashing.

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

Porque 29 é bem menor que o valor médio das somas dos caracteres nos valores da entrada (até pelo número do código ASCII das letras) e portanto há uma maior distribuição dentro da table.

*(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 de multiplicação é uma alternativa viável ao hash por divisão, analisando os resultados obtidos ao comparar a diferença de entropia de ambas as funções de hash. Isso se deve ao fato de que os resultados de ambos métodos apresentam resultados muito semelhantes. Como o desempenho dos dois métodos é muito semelhante, geralmente se utiliza o hashing por divisão, por ser mais simples de se implementar e não utilizar ponto flutuante em seu cálculo. Além disso o hash de multiplicação depende da escolha de um

bom fator de multiplicação irracional, enquanto o hash de divisão depende apenas de uma escolha de tamanho primo para a hash table para funcionar otimamente.

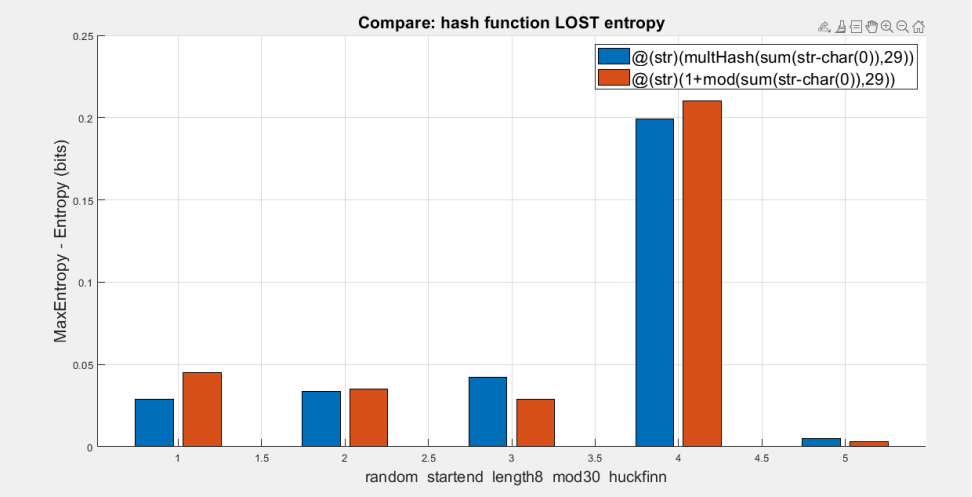

Gráfico 1: Entropia Perdida nos hastings por divisão e multiplicação com tamanho 29

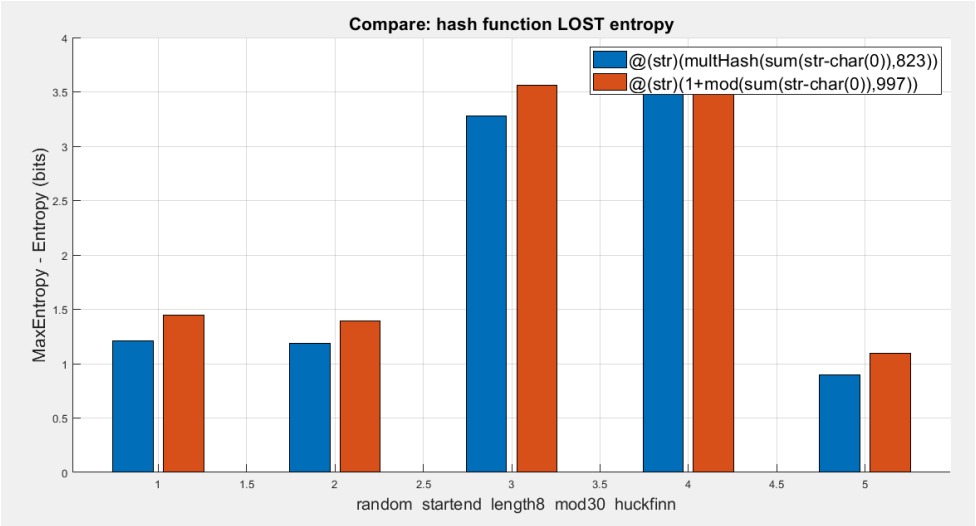

Gráfico 2: Entropia Perdida nos hastings por divisão e multiplicação com tamanho 1000

(4.1) para fundamentar a resposta do item (4), realizem testes próprios comparando hashing por multiplicação contra hashing por multiplicação, com os dados e funções fornecidos e incluam pelo menos um gráfico ou tabela. Julgue os seus resultados.

(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)

O closed hash é melhor em situações em que há poucas colisões e a tabela não fica muito cheia, pois ele ocupa menos memória RAM e não trabalha com ponteiros e alocação dinâmica. Escolhemos o closed invés do open quando temos certeza de que a table não estará sobrecarregada e em situações em que o uso de memória é uma preocupação importante.

(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 para evitar ataque na construção de uma hash table é usar o Universal Hash, que consiste em um conjunto de funções de hashing, do qual a função de hashing escolhida para montar a hash table é aleatorizada na construção da estrutura de dados. Essa aleatoriedade pode vir da seleção entre várias funções de hashing com leis diferentes ou apenas a aleatoriedade dos parâmetros de uma mesma lei da função. Isso evita com que o atacante conheça completamente a função do hash, pois ela será sempre aleatorizada.