Lab CTC212 - Hashing

16 de março de 2025

1 Introdução

Para entender os resultados dos experimentos e responder às 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).

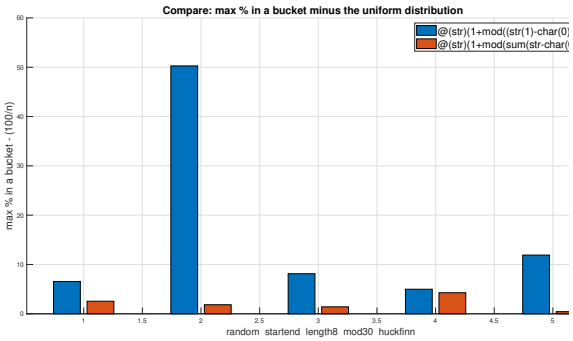
2 Questões e Respostas

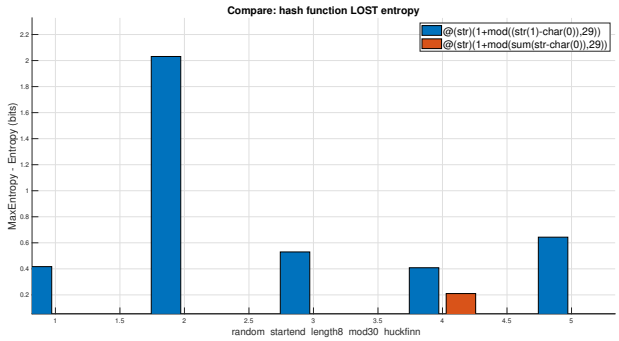
2.1 (1.1) Por que necessitamos escolher uma boa função de hashing, e quais as consequências de escolher uma função ruim?

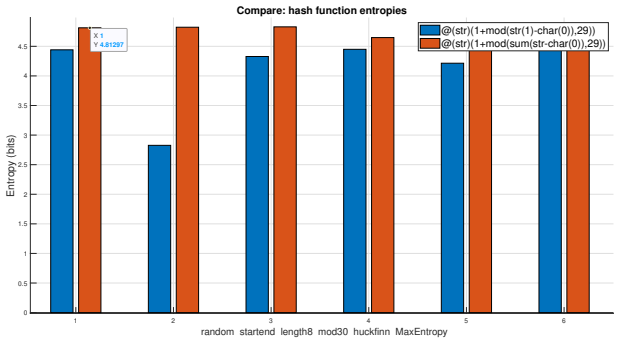
Uma boa função de hash tenta fazer com que cada chave tenha a mesma probabilidade de ocupar qualquer posição na hash table. Caso escolhamos uma função de hash ruim, haverá um viés para que as chaves ocupem posições específicas, gerando várias colisões e dificultando a busca.

2.2 (1.2) Por que há diferença significativa entre considerar apenas o primeiro caractere ou a soma de todos?

Os gráficos obtidos foram:







A distribuição de probabilidade da função de hashing é menos enviesada ao considerar a soma de todos os caracteres, pois leva em conta mais informações. Por exemplo, ao considerar

apenas o primeiro caractere, as chaves `"abcd"`, `"afgw"`, `"areg"` cairiam na mesma posição, o que não ocorreria ao considerar a soma de todos os caracteres.

2.3 (1.3) Por que um dataset apresentou resultados muito piores do que os outros, quando consideramos apenas o primeiro caractere?

Pela distribuição de elementos em cada bucket para o conjunto de dados "startend.txt":

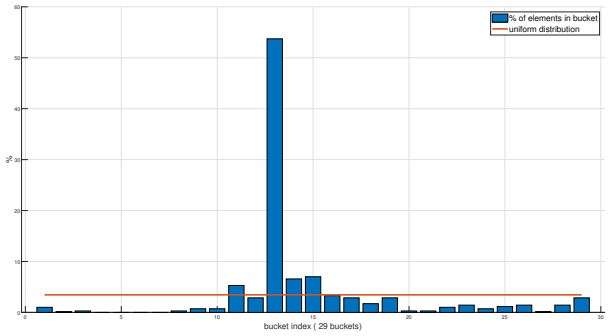
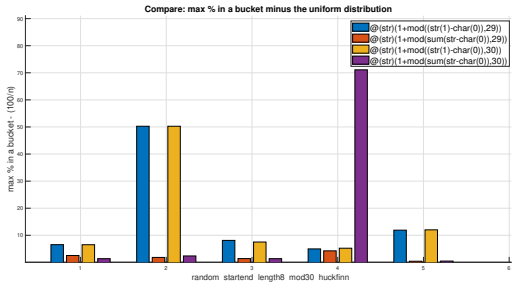


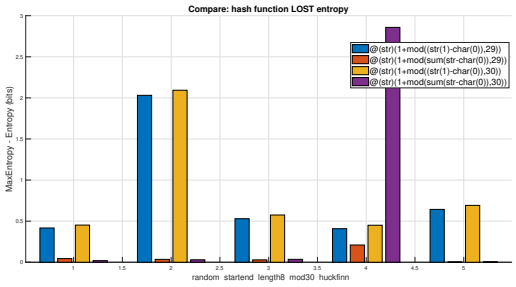
Figura 1: Caption

Percebe-se que há uma concentração alta de chaves em apenas um bucket, fazendo com que se distancie de uma hash table ideal em que a chance de encontrar um chave em cada célula é aproximadamente uniforme.

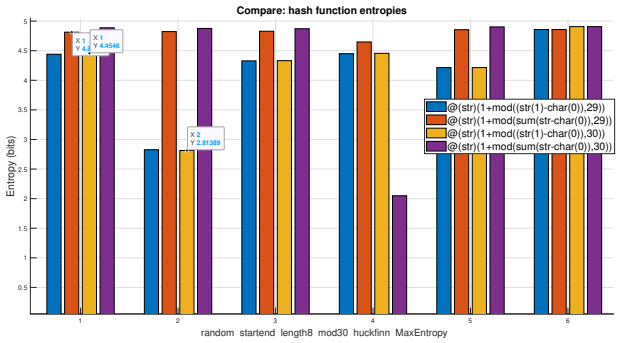
2.4 (2.1) Com uma tabela de hash maior, o hash deveria ser mais eficiente. Usar uma tabela de tamanho 30 não deveria ser sempre melhor do que 29? Por que isso não ocorre?

Resultado obtido:





A expectativa é que uma hash table maior diminuísse o número de colisões, já que há um aumento da entropia da função, conforme é mostrado gráfico de entropia para as funções.Isso



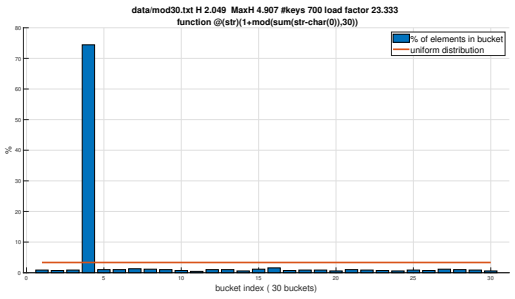
aconteceria no caso ideal, em que os dados são distribuídos uniformemente. No entanto, os dados reais podem apresentar vieses. No caso da função hash de divisão, esse viés poderia aparecer caso esses dados divisores em comum com o valor do tamanho da tabela. Como o número 30 tem mais divisores do que o 29, há um maior viés para o 30, fazendo com que a tabela hash se afaste mais do ideal esperado.

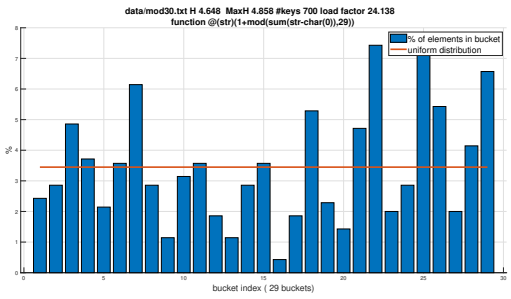
2.5 (2.2) Qual o benefício de usar um tamanho primo (e.g., 29) em vez de um número com múltiplos divisores, como 30?

Conforme explicado anteriormente, o tamanho primo evita padrões repetitivos na distribuição dos valores na tabela, reduzindo colisões.

2.6 (2.3) Como o arquivo `mod30` foi feito para atacar uma tabela de hash de tamanho 30?

Para atacar uma tabela de hash, o atacante deve conhecer a função de hash e elaborar um conjunto de dados que gere colisões excessivas. O arquivo `mod30` foi projetado para criar muitas colisões ao dividir por 30.

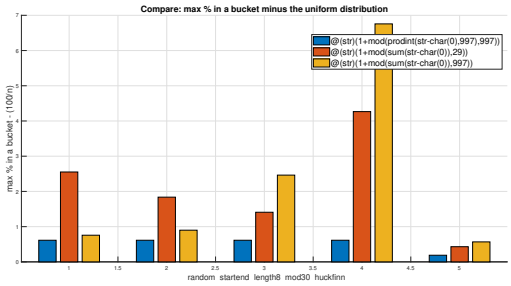


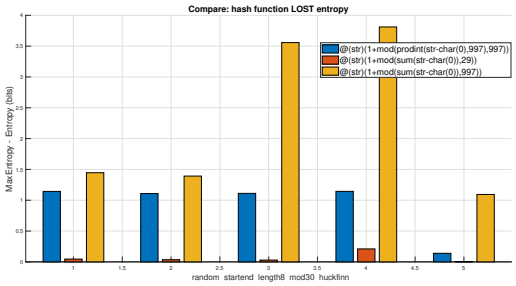


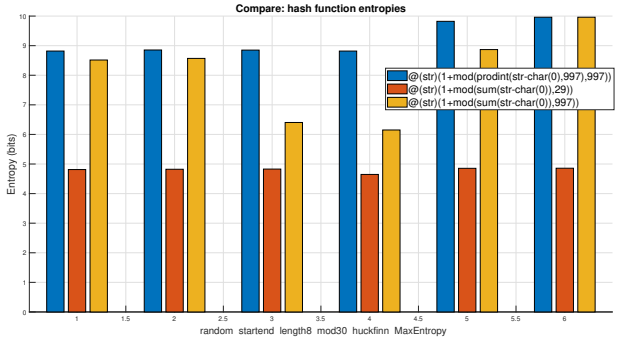
Ao analisar a distribuição de elementos por bucket, percebe-se que a maior parte deles deixam resto 3 na divisão por 30 (na tabela, o endereço do bucket é dado por 1 +mod30). Dessa forma, o número é do tipo: n = 30q+3 = 3(10q+1). Ou seja, a maior parte das chaves são múltiplas de 3. Para o 29, não há esse tipo de vício, já que 29 é um número primo e não apresenta divisores.

2.7 (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?

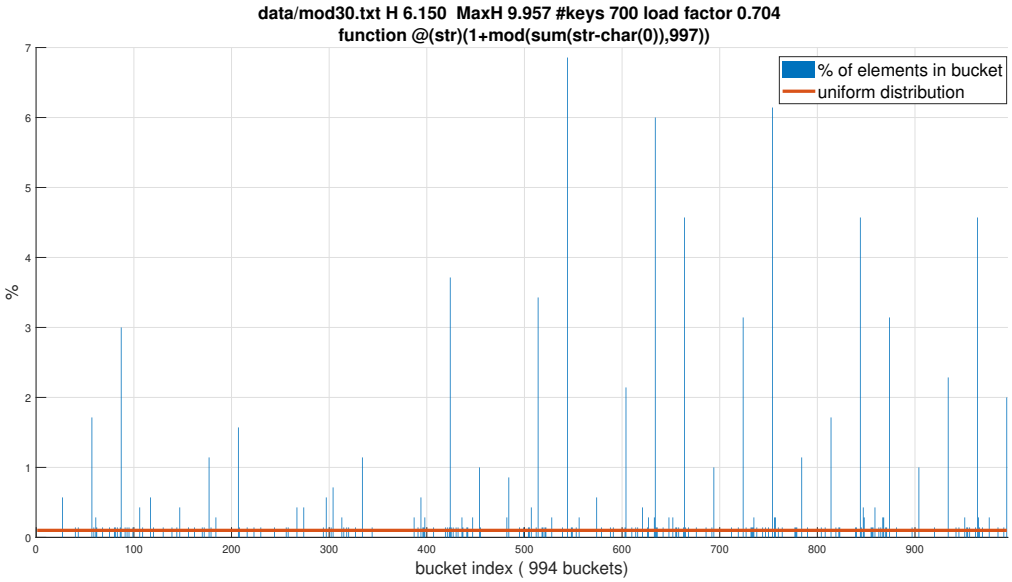
O resultado obtido foi:







Teoricamente, é esperado que a divisão com 997 buckets seja melhor, uma vez que há mais posições para se ocupar na tabela hash. Essa expectativa é mostrada ao termos a entropia da função com mod997 maior que com mod29. Ao observarmos o conjunto de dados "mod30.txt"que apresentou um desempenho signicativamente pior na ocupação da hash table pela função mod997:

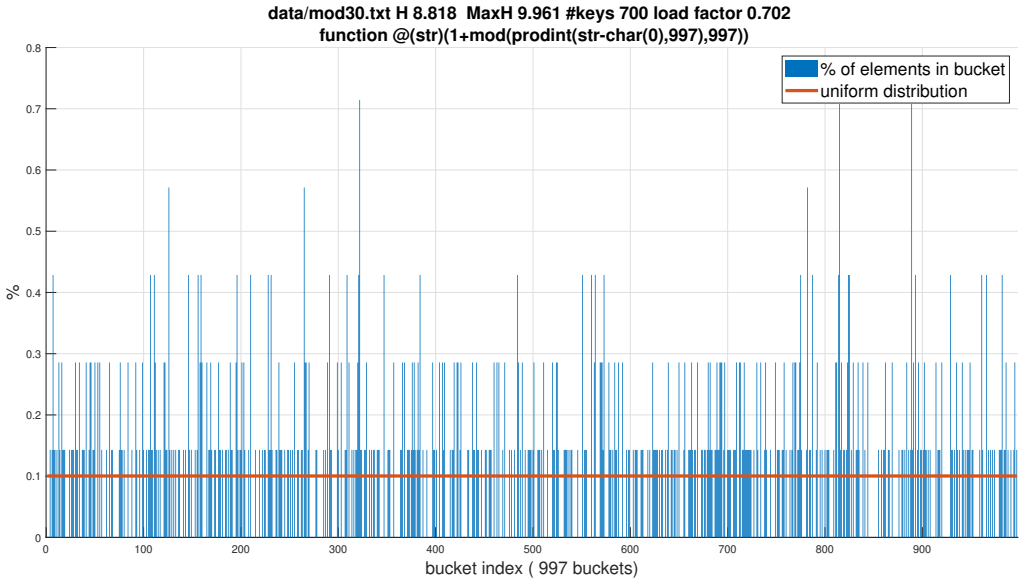


É fácil perceber que os dados estão concentrados em alguns buckets espaçados de forma padronizada. Conforme visto anteriormente, isso ocorre porque os dados, na maior parte, são múltiplos de um número X. Ao fazermos mod997, como 997 é um valor alto em comparação com o padrão que se dão os múltiplos do número X, esse padrão aparece na tabela hash fazendo com que os elementos se concentrem nos múltiplos desse número.

Para o mod29, isso não ocorre por ser um número menor, o que permite que ele "quebre"esse padrão percebido devido aos dados serem múltiplos de um número X.

2.8 (3.2) Por que a versão com `prodint` é melhor?

A função `prodint` substitui cada elemento do conjunto de dados "v"por $resultado = mod(resultado * v(i$
A distribuição obtida é dada por:



Percebe-se que há uma distribuição mais uniforme do que antes. Isso acontece pois ao multiplacarmos pelo fator que depende de m^2*ederesultadoscumulativosanteriores*, *adicionamosentropiaaosis*

2.9 (3.3) Por que esse problema não apareceu quando usamos tamanho 29?

Conforme dito, para o mod29, isso não ocorre por ser um número menor, o que permite que ele "quebre"esse padrão percebido em intervalos específicos.

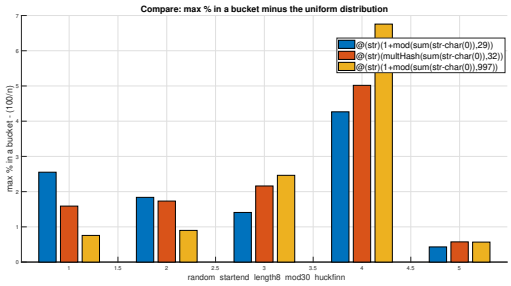
2.10 (4) Hashing por divisão é o mais comum, mas e o hashing por multiplicação?

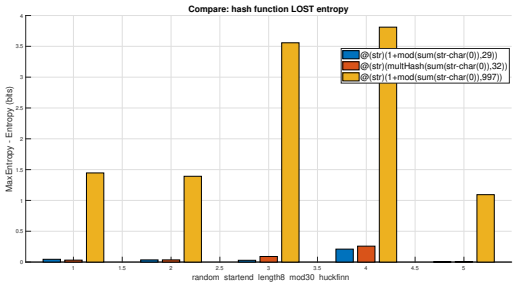
O função hash por multiplicação é dada da seguinte forma:

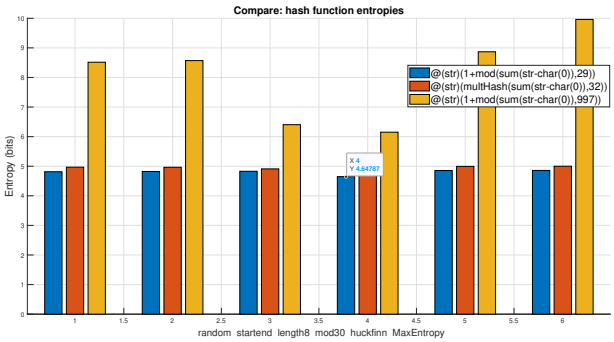
$$h(k) = \lfloor m \cdot (kA - \lfloor kA \rfloor) \rfloor,$$

Em que o valor de m é o tamanho da hash table e geralmente é escolhido de forma a ser uma potência de 2. Restringe-se o A para ser uma fração da forma $\frac{s}{2^w}$, em que s é um inteiro na faixa $0 < \text{x} < 2^w$

Os resultados obtidos comparando hash por divisão (mod29 e mod997) e hash por multiplicação (m = 32)







Pode-se perceber que hash por divisão e multiplicação apresentação uma espalhamento aproximado, considerando o caso ideal, ao olharmos a entropia das funções. Para o hash por multiplicação, a vantagem é que temos maior flexibilidade para o m, no entanto, ele tem um fator a mais de sensibilidade que é o A. Além disso, percebe-se que não há uma regra geral de quando o hash por divisão ou por multiplicação é melhor. Conforme pode ser observado, para conjuntos de dados diferentes, um pode performar melhor que o outro.

Geralmente o hash por divisão é mais usado por ser mais fácil a sua implementação e ter boa dispersão quando m é primo.

2.11 (5) Qual a vantagem de Closed Hash sobre Open Hash?

Para o Closed Hash, todos os elementos ficam na própria hash table. Cada entrada da tabela é um elemento ou está vazio. Diferentemento da Open Hash, não há nenhuma lista e nenhum elemento armazenado fora da tabela.

A maior vantagem é que o Closed Hash economiza espaço ao armazenar as chaves diretamente na tabela, enquanto Open Hash usa listas encadeadas para tratar colisões.

2.12 (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?

A melhor maneira de impedir este tipo de ataque é escolher, no início da execução, a função de hash aleatoriamente dentro de um conjunto de funções projetadas, de modo que seja independente das chaves que serão armazenadas. Essa abordagem se chama hashing universal e se mostra eficaz na média.