Lab CES12 - 2023
Yuri Mendes de Almeida

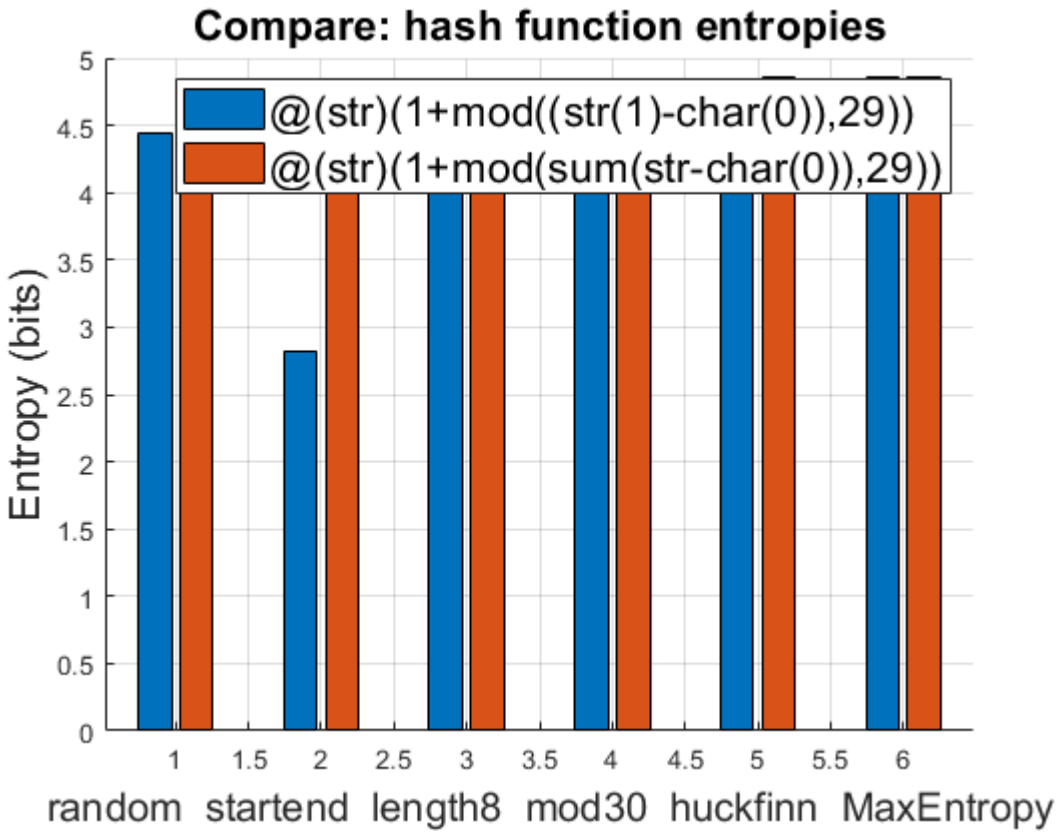
Hashing

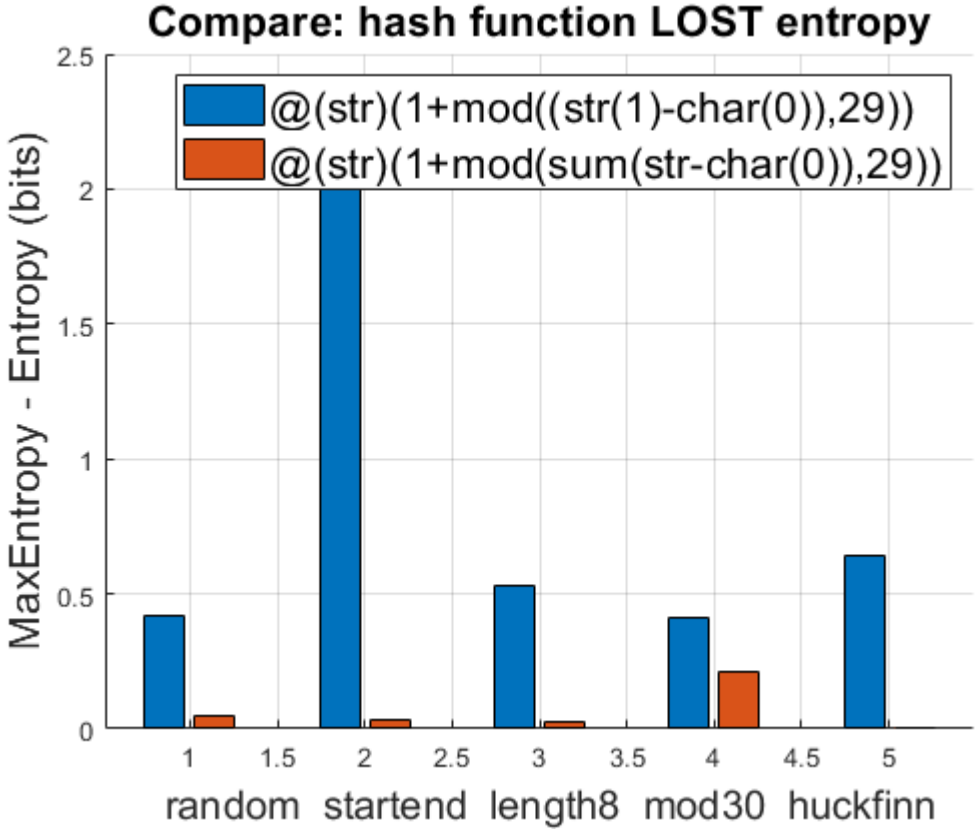
(1.1) (pergunta mais simples e mais geral) porque necessitamos escolher uma boa função de hashing, e quais as consequências de escolher uma função ruim?

Primeiro, consideramos uma boa função de hashing aquela que cumpre as premissas do hashing uniforme simples: cada chave tem igual probabilidade para qualquer bucket da tabela por uma operação de hash independente de outras chaves após o hash também. Assim teremos os dados mais uniformemente distribuídos pela tabela. Escolher uma função ruim de hashing terá como consequência algumas posições na tabela mais prováveis de serem escolhidas, tendo mais colisões e aumentando a complexidade do tempo de busca.

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

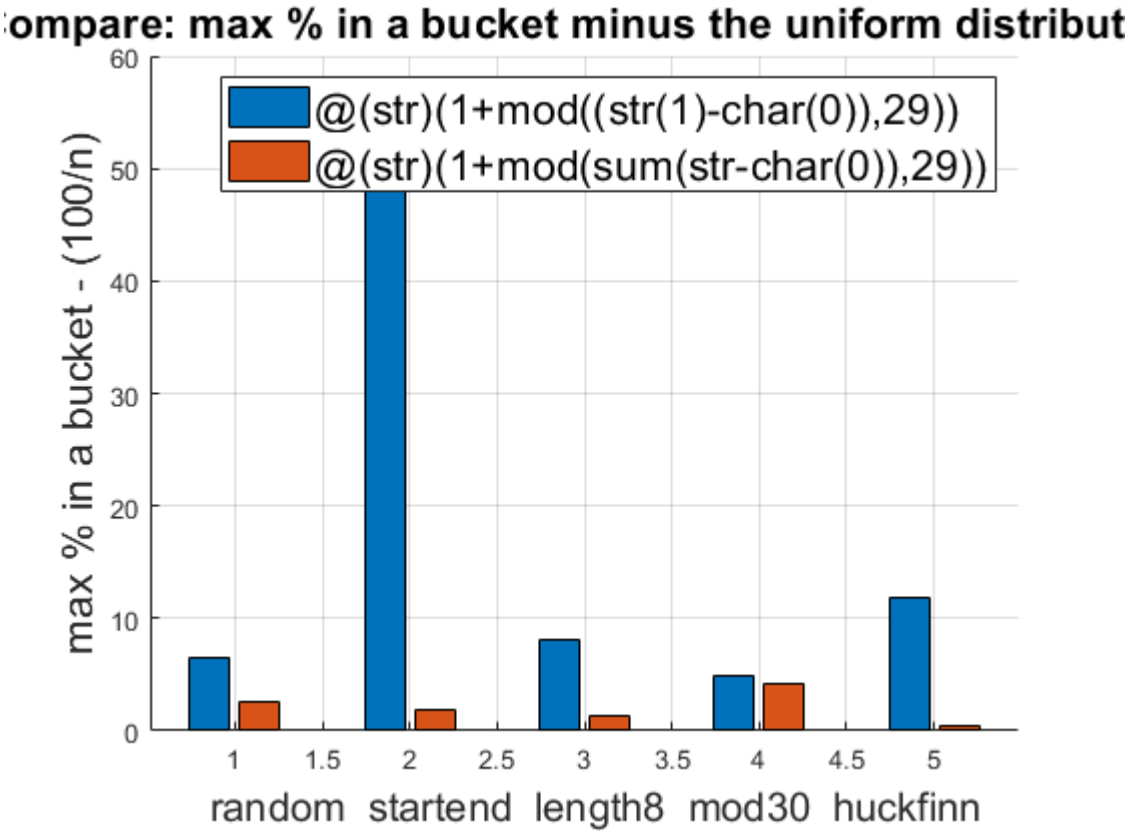
Sabemos que na situação ideal (hash function perfeita) consideramos a entropia máxima e que na prática temos um pouco de perda de entropia. Com o auxílio das figuras a seguir podemos confirmar que no caso considerando apenas o 1° caractere a perda de entropia é maior e com a soma de todos temos uma distribuição mais uniforme das probabilidades.


Compare: hash function entropies
@(str)(1+mod((str(1)-char(0)),29))
@(str)(1+mod(sum(str-char(0)),29))
Entropy (bits)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
6
random
startend
length8
mod30
huckfinn
MaxEntropy


Compare: hash function LOST entropy
@(str)(1+mod((str(1)-char(0)),29))
@(str)(1+mod(sum(str-char(0)),29))
MaxEntropy - Entropy (bits)
0
0.5
1
1.5
2
2.5
1
1.5
2
2.5
3
3.5
4
4.5
5
random
startend
length8
mod30
huckfinn

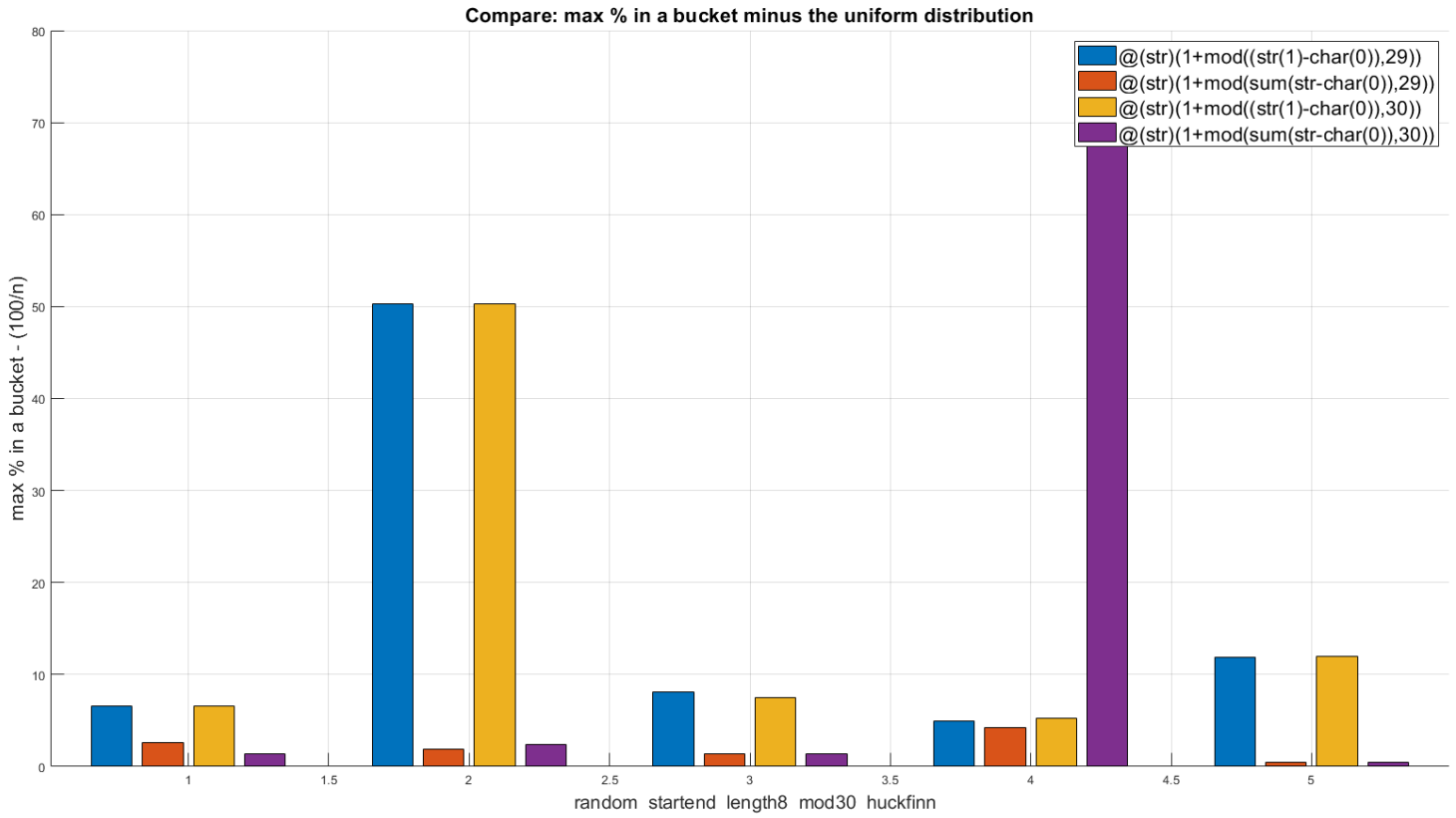
(1.3) porque um dataset apresentou resultados muito piores do que os outros, quando consideramos apenas o 1o caractere?

No “startend” temos muitas palavras começadas pela letra C, assim, sendo alocadas no mesmo bucket da tabela. Portanto, considerando apenas o 1° caractere teremos uma grande perda entrópica comparada aos demais.



(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)

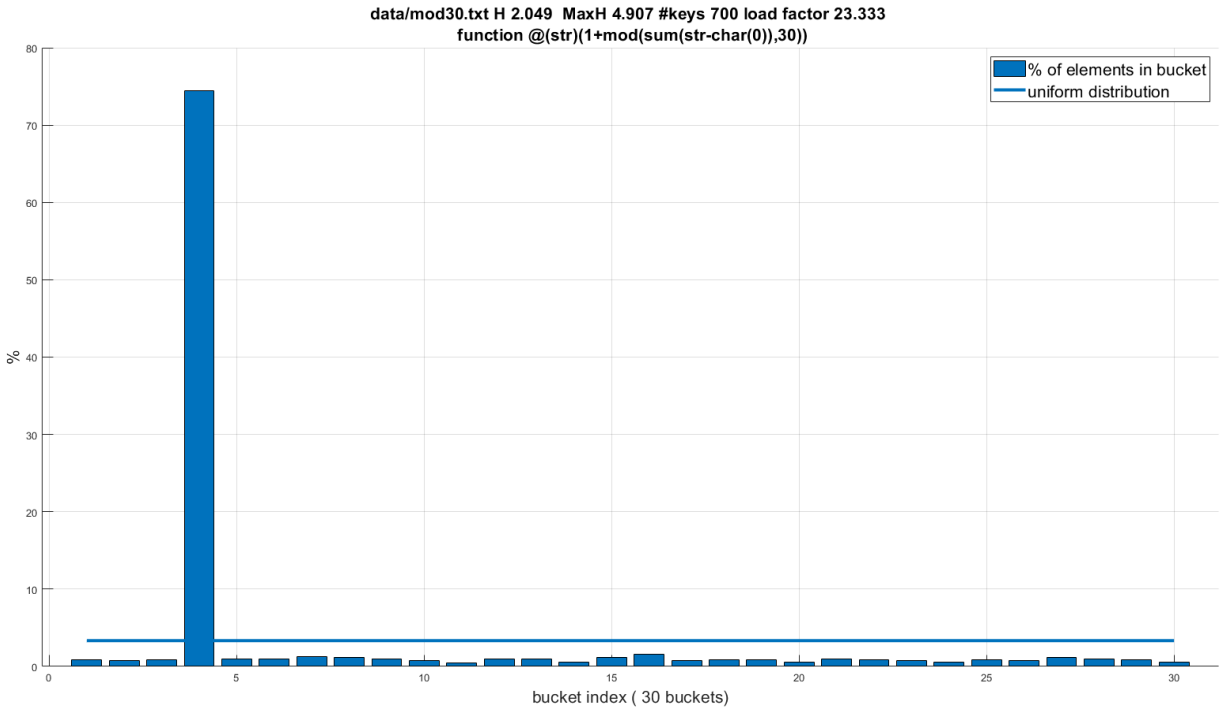
Sabemos que escolher um número primo para o tamanho tende a ser uma boa escolha no método hash por divisão, principalmente comparado a números pares com muitos divisores. Justamente pelo fato dos restos das divisões ficarem melhor distribuídos na tabela.



(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?

Como falamos no problema anterior, usar um número primo tem a tendência em distribuir melhor o resto das divisões, diferente de números não primos como por exemplo uma potência de 2 que acabam acumulando os dados em buckets específicas. Por outro lado, com os dados analisados não há muita diferença expressiva com exceção de mod30, mas facilmente poderíamos pegar outro conjunto de dados que com o número 30 que teríamos resultados ruins e pro 29 (número primo) ainda seriam razoáveis.

(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)



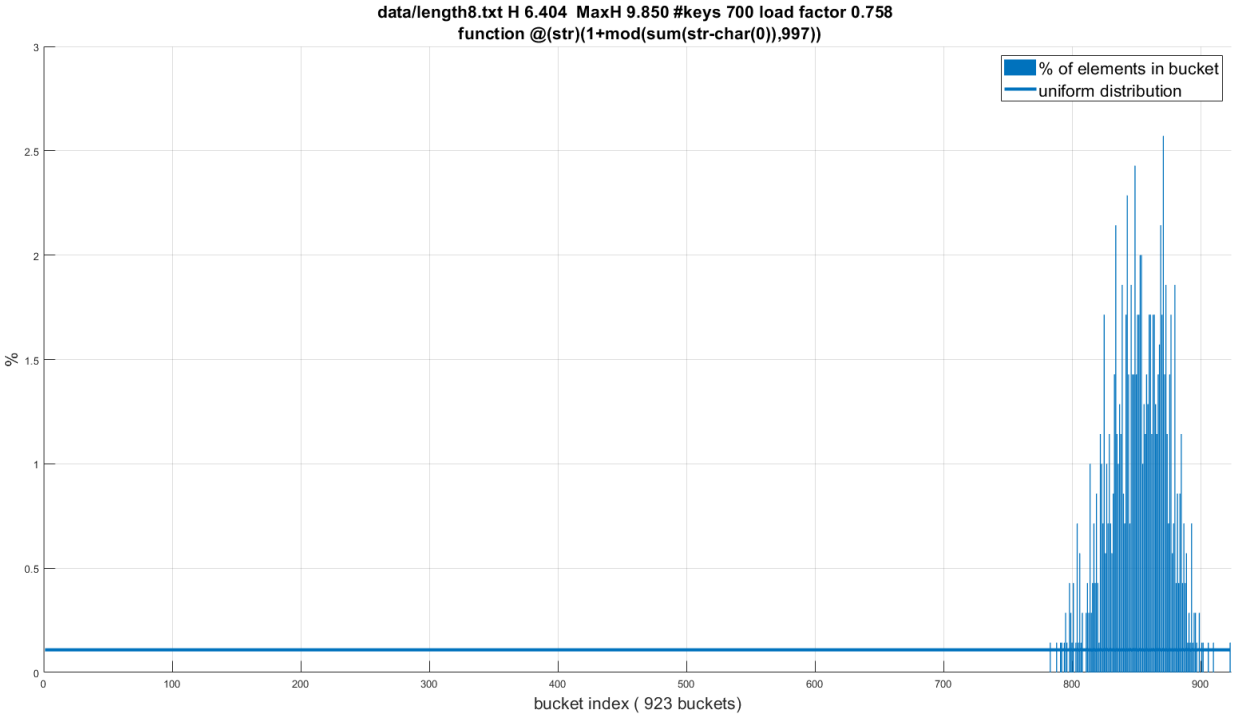
Podemos ver pela figura acima (**checkhashfunc)** que há um acúmulo bem alto no bucket 4, assim mostrando que o ataque explora o pior caso, sobrecarregando/adicionando nesse bucket específico.

(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?

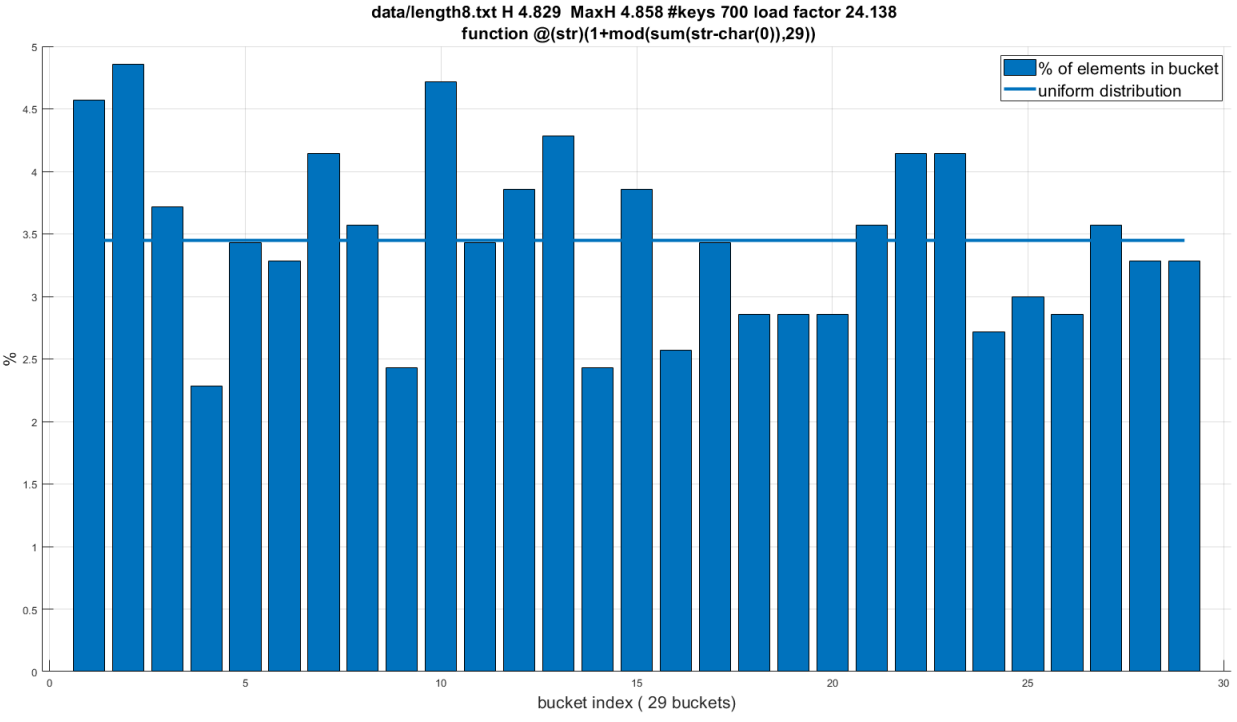
(3.3) Porque este problema não apareceu quando usamos tamanho 29?

Pode ser um pouco contra intuitivo uma tabela com tamanho 29 ter distribuição melhor que uma de 997, mas sabemos que isso depende não só do tamanho da tabela, mas também do conjunto de dados analisado (que pode interferir) e por último mas não menos importante, claro, a função hashing. Vemos que no arquivo lenght8 todas as palavras possuem o mesmo tamanho (8) e valores relativamente próximos entre si. Os gráficos abaixo ilustram bem a diferença:

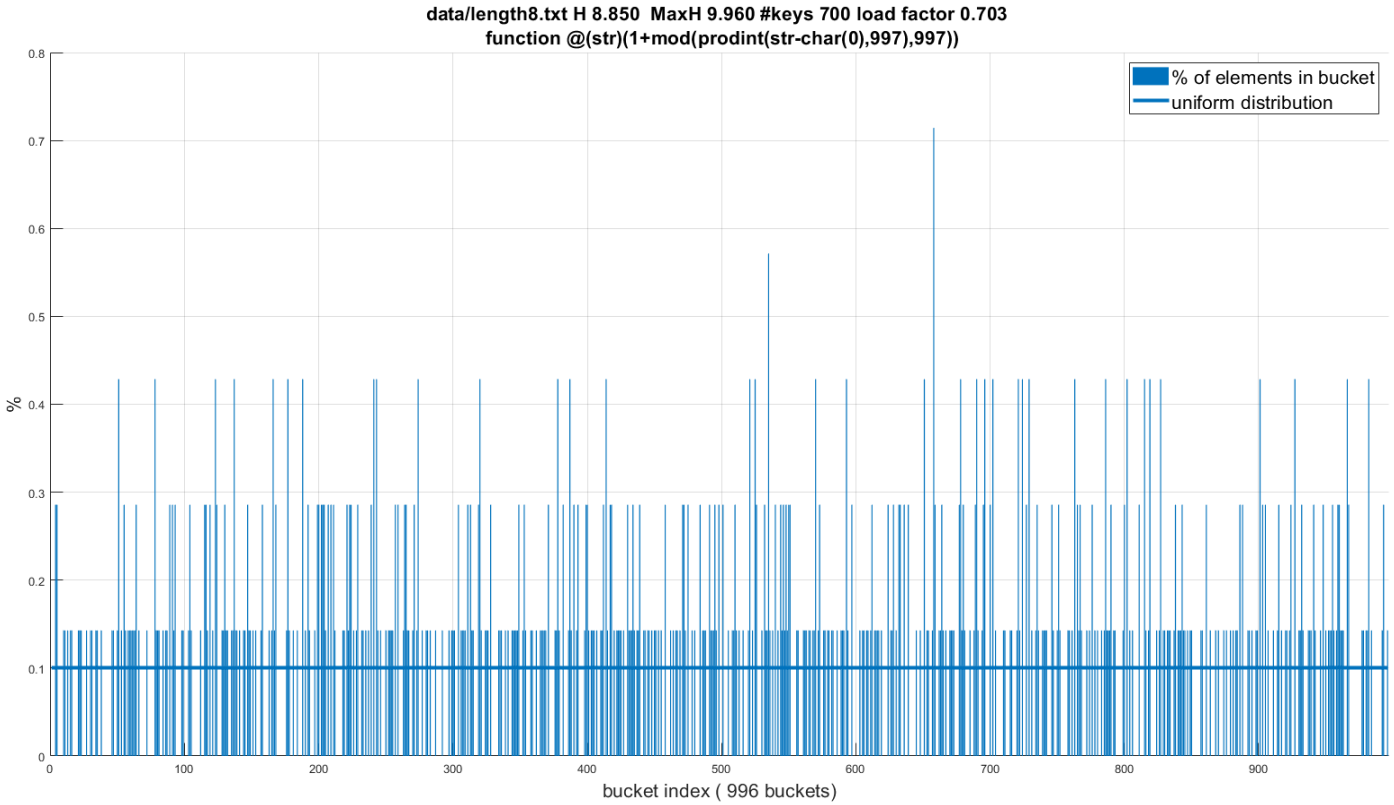
Veja o acúmulo de buckets abaixo:



Distribuição mais uniforme com o tamanho 29:



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



Pela figura acima, podemos ver que os valores dessa vez se alocam mais distribuidamente nos buckets sem gerar muito acúmulo, pois são valores numericamente maiores e assim ficam mais uniformes na tabela.

(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 método de multiplicação tem a vantagem de que a escolha do número de buckets *m* não é crítico (ao menos não tão sensível como o de divisão, que a escolha do número de buckets influencia fortemente a distribuição na tabela). Porém, é um processo realizado em mais de uma etapa, assim sendo uma operação mais custosa que o método por divisão, que é mais rápido e entrega resultados suficientemente bons e portanto acaba sendo mais comum.

(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)

A principal vantagem de Closed Hash sobre o Open Hash seria o mais fácil acesso (operações mais rápidas e ocupa menos espaços na memória) por não usar listas encadeadas, assim tendo melhor performance.
Geralmente usa-se Closed Hash em que sabemos antes o número de chaves e não é um número muito extenso, apesar de que na prática não é o que acontece na maioria das vezes, tendo que recorrer ao Open Hash.

(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 maneira mais eficaz (Hashing Universal) seria escolher a função hash de forma aleatória, de maneira que seja independente das chaves que serão alocadas nos buckets. Assim, não importando as chaves que são escolhidas pelo atacante.

Neste trabalho, usei as seguintes referências:

https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/HashIntro.html

Algoritmos: Teoria e Prática (Cormen, 3° edição)