Instituto Tecnológico de Aeronáutica
Divisão de Ciência da Computação

CTC-12

Laboratório 01

Hashing

Aluno:

Daniel Araujo Cavassani

Professor:

Luiz Gustavo Bizarro Mirisola

QUESTÕES

(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?

Uma boa função é necessária para evitar colisões excessivas, isto é, para que haja uma boa distribuição das chaves ao longo da tabela de hash (alta entropia de bit). Caso a função seja ruim, haverá muitas chaves mapeadas para a mesma posição, o que pode levar a uma performance ruim nas operações de busca, adição e remoção, principalmente devido à formação de clusters de chaves em algumas posições da tabela.

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

O uso do primeiro caractere pode levar a colisões mais frequentes, já que muitas strings possuem o primeiro caractere igual (principalmente a depender do exemplo). Já a soma de todos os caracteres pode gerar uma distribuição mais uniforme das chaves, o que pode reduzir as colisões e, portanto, é mais benéfico.

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

Imagine, por exemplo, que consideremos um dataset do nome das pessoas que aparecem na primeira página de um resultado de vestibular. Acontece que este vestibular ordena as pessoas de forma alfabética, e não por notas. Dessa forma, teríamos diversas pessoas com a letra 'A' inicial, de forma que TODAS elas teriam a mesma chave utilizando tal função de Hash, isto é, a entropia deste sistema é 0 (pior caso possível). Dessa forma, usar o primeiro caractere é mais suscetível a colisões

(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)

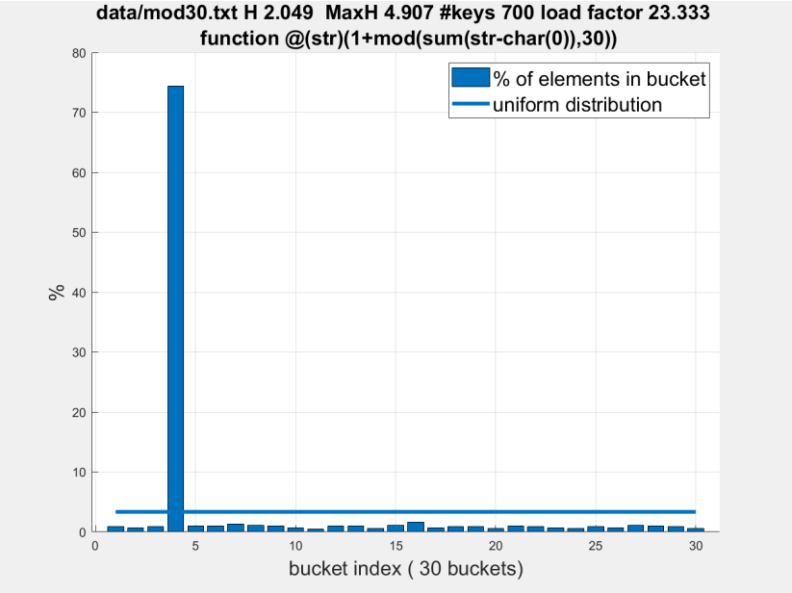
A ideia de aumentar o tamanho da tabela de hash é de permitir que as chaves sejam distribuídas de maneira mais uniforme e com mais opções. No entanto, nem sempre isso ocorre, visto que a função de hash não é a função ideal, podendo conter falhas que se sobreponham ao efeito de simplesmente aumentar a tabela, o que acarretaria num aumento de número de colisões (e não diminuição). Na verdade, para alguns conjuntos de dados e funções de hashing, uma tabela menor pode ser mais eficiente.

(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 um tamanho primo ajuda a evitar colisões em alguns casos em que o tamanho da tabela é um fator no padrão de distribuição das chaves. Por ser primo, o número que damos ao tamanho da tabela evita que a distribuição das chaves coincida com qualquer ciclo de divisão no espaço de hash (Por exemplo, 2, 12, 22, ... , são todos números que deixam resto 2 se divididos por 10, o que seria um exemplo de um padrão a ser explorado, mas não há este tipo de padrão se escolhermos 11 em vez de

10 para o tamanho, pois 11 é primo). No nosso exemplo, a diferença não é muito grande pois a função de hashing não é sensível a esse tipo de alteração em conjunto com os dados que foram fornecidos.

(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 ataque por divisão funciona tentando mapear um conjunto de chaves para a mesma posição na tabela de hash. Para tanto, o atacante deve conhecer a função de hash utilizada, de forma que seja possível gerar um conjunto de chaves que causem colisões em posições específicas da tabela. O arquivo mod30 foi elavorado para atacar uma tabela de hash de tamanho 30, gerando chaves que causariam colisões em várias posições específicas da tabela. O exemplo citado na questão 2.2 é perfeito para elucidar o tipo de padrão que poderia surgir numa tabela de tamanho 30, facilitando o Stack de diversos valores numa mesma chave por meio deste ataque por divisão.

(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 tamanho da tabela de hash pode influenciar na distribuição das chaves, mas a qualidade da função de hashing também é importante, assim como explicado na questão 2.2. às vezes, a tabela de hashing com 29 posições pode apresentar melhor desempenho entrópico do que a tabela com 997 posições, tudo depende de como aa função de hashing funciona.

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

Essa versão é melhor pois produz um resultado mais uniforme e disperso, reduzindo as chances de colisões e melhorando a eficiência do algoritmo de hashing, visto que

essa abordagem permite que o algoritmo seja menos suscetível a padrões nos dados de entrada e, portanto, é uma abordagem mais robusta.

(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.

O problema que surgiu quando utilizamos o tamanho 997 na tabela de hash não ocorreu quando utilizamos tamanho 29 pois o tamanho menor da tabela torna mais difícil para o atacante encontrar padrões nos dados que permitam gerar colisões de forma consistente.

(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 hashing por multiplicação é uma alternativa viável, mas é menos comum por ser mais complexo e menos intuitivo do que o hashing por divisão. A vantagem desse método é que ele funciona bem em uma variedade de tamanhos de tabela de hash e produz uma distribuição uniforme das chaves. No entanto, tal hashing pode ser menos eficiente em alguns casos, principalmente quando há restrições de hardware no sistema utilizado.

(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 do Closed Hash em relação do Open Hash é que ele evita a necessidade de armazenar ponteiros extras para listas vinculadas, tornando o algoritmo mais eficiente em termos de memória e tempo de acesso. A escolha de um ou outro depende da natureza do problema. Em geral, o Open Hash é melhor para lidar com conjuntos de dados dinâmicos, enquanto o Closed Hash é mais eficiente para conjuntos de dados estáticos.

(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 básica da Universal Hash é escolher a função de hash de forma aleatória de um conjunto pré-definido de funções, de forma que o atacante não seja capaz de prever qual função de hash será usada. Essa técnica pode tornar o ataque de colisão muito mais difícil, pois o atacante teria que prever e contornar todas as funções possíveis, 3em vez de apenas uma.