INSTITUTO TECNOLÓGICO DE AERONÁUTICA

HASH TABLE

Aluno: Kalil Georges Balech

(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?

Necessitamos escolher uma boa função de hashing, pois, dependendo do número de elementos que o vetor hashTable tiver e dependendo da ordem de grandeza do número de elementos que queremos inserir na hashTable, o tempo de procura pelo elemento pode crescer, de forma que, no pior caso, o tempo de procura pelo seu elemento se torna Θ(n). As consequências de escolher uma função ruim é um baixo grau de uniformidade da distribuição dos elementos pelos buckets da hashTable, ou seja, mínima entropia e maior tempo de procura pelos elementos.

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

Há diferença significativa, pois ao considerar apenas o primeiro caractere, caracteriza-se pouco a string a ser inserida na hashTable, de forma que se aumenta o número de strings inseridas numa mesma forward_list do vetor _table. Dessa forma, a porcentagem de elementos inseridos em cada bucket, subtraído da quantidade relativa à distribuição uniforme, informa que a entropia do sistema em que se considera apenas o primeiro caractere possui entropia significantemente menos, e, portanto, uniformidade de distribuição menor, aumentando o tempo de procura dos elementos pela hashTable.

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

O data set startend apresentou resultados muito piores, pois nele, temos muitas palavras com iniciais ‘c’, ‘d’, ‘e’, ‘h’ e ‘i’. Dessa forma, como estamos considerando apenas o primeiro caractere, a função hash dessas palavras são iguais, gerando uma distribuição pouco uniforme pela hashTable, uma baixa entropia e um alto tempo de procura pelos elementos.

(**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?**

Esse resultado não ocorre, pois ao termos uma hash table com tamanho primo, temos uma distribuição mais uniforme das strings. Isso ocorre, pois strings cuja soma dos caracteres tiver algum fator em comum com o tamanho da hash table vão sempre ser alocadas nos mesmos buckets. Dessa forma, caso o tamanho da hash table for primo, isso ocorre com uma frequencia quaser nula, pois temos um único fator na decomposição em fatores primos do tamanho da hash table, que é o próprio tamanho da hash table.

(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 tipo de problema que o primo evita é de alocar os números com as mesmas características nos mesmos buckets da hash table. Dessa forma, o grau de uniformidade da distribuição se torna maior. Por exemplo, num caso em que o tamanho da hash table é 10, temos que todas as strings cuja soma é par serão alocadas em buckets de ordem par. Ao mesmo tempo, caso o tamanho da hash table for 11, as string cuja soma é par serão alocadas tanto em buckets de ordem par quanto em buckets de ordem ímpar, de forma a melhor distribuir as strings na 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.

O ataque funciona de forma que o atacante saiba qual é o tamanho da hash table a ser utilizada pelo arquivo de dados que ele produzirá. Dessa forma, ele produz um arquivo de dados que concentrará a maior parte dos dados em um único bucket da hash table e o restante dos dados (parte minoritária) aproximadamente igualmente distribuído entre o restante dos buckets. Assim sendo, o tempo de procura por algum elemento na hash table tem seu tempo de procura na ordem de $\Theta(n)$, pois a chance do elemento a ser procurado estar no bucket que concentra a maior parte dos dados é de grande relevância.

(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?

Isso ocorre, pois no geral, os valores atingidos por um arquivo de dados (soma dos caracteres de uma string, por exemplo) estão em uma faixa de valores mais provável (de 1 a 150, por exempo). Dessa forma, ao utilizar uma hash table de alto número de buckets, os buckets de ordem maior que o teto da faixa de valores atingida pelos dados vão ficar praticamente vazios, enquanto que para uma hash table de menor número de buckets, teríamos os dados distribuídos de maneira mais próxima da uniforme.

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

A versão com produtório é melhor, justamente porque no caso de considerar o produto dos caracteres, a faixa de valores atingida pelo produtório é exponenciamente maior quando maior o número de letras da palavra. Dessa forma, de forma visivelmente mais fácil o resultado do produtório ultrapassa o número 997. Dessa forma, calcular o mod desse resultado por 997 faz com que os dados fiquem bem melhor distribuídos na hash table.

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

Esse problema não apareceu usando tamanho 29, pois, novamente, multiplicando os caracteres de uma string e calculando o mod após cada multiplicação faz com que o número gerado ao final seja aleatório e de magnitude bem maior que o tamanho da hash table. Dessa forma, o resultado gerado para cada string se distribuirá entre os buckets da hash table, gerando mais uniformidade e, assim, não caracterizando tal situação com o problema da situação anterior.

(4) hash por divisão é o mais comum, mas outra alternativa é hash de multiplicação. É uma alternativa viável? Porque hashing por divisão é mais comum?

O método de hash de multiplicação é uma alternativa viável, dado que em grande parte dos casos, dependendo do fator de escala, ele pode distribuir os dados de maneira bastante próxima da uniforme pela tabela hash.

O hashing por divisão é mais comum, pois é um método mais simples de se entender, de se implementar e de se calcular. Isso faz com que a performance do computador para implementá-lo seja melhor do que comparado com o hashing por multiplicação.

(5) Qual a vantagem de Closed Hash sobre OpenHash, e quando escolheríamos Closed Hash ao invés de Open Hash?

A vantagem do closed hash sobre o open hash é que em casos em que temos um pequeno volume de dados a ser alocado, comparativamente com o tamanho da tabela hash, o closed hash facilita o armazenamento dos dados, ocupando menos espaço de memória. Essa técnica é bem utilizada quando o número de elementos a serem armazenados não é grande e é caracterizado por um baixo número de colisões, sendo esses casos os melhores para se escolher o closed 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?

Ao invés de termos uma função hash fixa, teremos um conjunto de funções hash, em que toda vez que uma hash table for gerada, escolheremos aleatoriamente alguma das funções hash do conjunto, de forma que a pessoa que tem acesso ao código não consiga prever qual das funções hash será utilizada.