Lab CES12 - 2023
Hashing

Objetivo:

Implementar uma hash table
Entender hash functions, e as propriedades necessárias.
Instalar e se familiarizar com as ferramentas de teste e compilação

Teoria: Hash Tables

Os seguintes links apresentam simulações iterativas de hash table.

https://www.cs.usfca.edu/~galles/visualization/OpenHash.html
https://www.cs.usfca.edu/~galles/visualization/ClosedHashBucket.html
https://www.cs.usfca.edu/~galles/visualization/ClosedHash.html

Open Hash, ou Espalhamento Aberto, é um nome para a variante onde cada bucket contém uma lista ligada. Assim, potencialmente a hash table com k buckets pode conter um número de elementos $n > k$, embora a aproximação O(1) seja cada vez menos apropriada à medida que o *load factor* aumenta (*load factor* é o número médio de itens em cada bucket, ou seja, n / k).

Podemos pensar em várias pequenas variações para Open Hash:
Aceitamos elementos repetidos, ou recusamos inserir elementos repetidos?
Ou, se cada nó da lista ligada contém além da chave, um contador de repetições, podemos utilizar este contador para indicar quantas repetições do mesmo elemento existem.
Reflita em como as operações de Busca, Inserção, e Deleção são afetadas pelas variações acima: elas podem ser verdadeiramente O(1), ou O(n/k). Note que O(n/k) pode ser aproximada para O(1) apenas se o load factor for suficientemente pequeno.

Closed Hash é o nome da variante onde cada bucket não é uma lista ligada mas contém apenas espaço alocado para um (ou alguns poucos) elementos. Obviamente, só faz sentido com load factors bem menores do que o tamanho do bucket b - caso contrário, a aproximação O(1) para cada busca também se torna menos realista.

Mais ainda, como em cada bucket cabem b elementos, se, por exemplo, $n / k = b \Rightarrow n = k * b$, isso significa que não há espaço livre na hash table e cada busca resulta em O(n) colisões!

Então, qual a vantagem de Closed Hash sobre Open Hash?

Implementações C++:

O aluno apenas precisa implementar a própria classe Hash Table de forma a passar nos testes.
O gtest linka o meu e o seu código com o `gtest_main`, então o aluno nem sequer precisa escrever uma função main()

A função hash a ser usada é o chamado hash por divisão, ou seja, somar os valores dos caracteres da string e fazer mod da soma.

Implementações matlab fornecidas

Foi fornecida em matlab, datasets de strings, e funções matlab para simular funções de hash.

comparehashfunctions.m

Dados um vetor de funções de hash e um vetor de nomes de arquivos com datasets, esta função matlab simula as hash functions para comparar as funções dadas com os datasets dados. Gera 3 gráficos-barra:

1. entropia da distribuição. Idealmente os valores deveriam ser sempre iguais à entropia máxima se a hash function fosse perfeita.
2. Perda de entropia: é apenas uma subtração, realizada para facilitar a visualização dos valores. Como sabemos a entropia máxima, i.e., a entropia da distribuição uniforme com o mesmo número de outcomes (valores possíveis), simplesmente fazemos Perda de entropia = Entropia máxima - Entropia medida.
    a. Por exemplo: com 8 buckets, a máxima entropia seria 3 bits (distribuição uniforme com ⅛ para cada possibilidade). Então, se a entropia da distribuição dos dados na tabela de hash for 2.5 bit, significa que perdemos 3 - 2.5 = 0.5 bit. Idealmente este gráfico deveria estar em zero se a hash function fosse perfeita.
3. Percentagem do bucket mais cheio menos a percentagem da distribuição uniforme. Este é *diretamente o que queremos medir*, ou seja, o **pior caso da busca na hash table comparado com pior caso de uma função hash ideal**, ou seja, subtraindo o custo da busca supondo uma distribuição uniforme, que devemos pagar de qualquer forma. Exemplo ainda com 8 buckets: No caso ideal, teríamos ⅛ dos elementos em cada bucket. Se, o bucket com a fila mais longa tiver 5/16 dos elementos, então: 5/16 - ⅛ = 3/16. Isso significa que o custo é 3/16 maior do que o ideal - idealmente este valor seria zero se a hash function fosse perfeita.

runcomparehashfunctions.m

Simplesmente chama comparehashfunctions() com as funções hash e os arquivos corretos para responder as perguntas. Também é exemplo de como usar comparehashfuntions()

plothash.m

Dada uma função hash e um arquivo de dados, lê o arquivo, simula a hash function e plota quantos elementos seriam colocados em cada bucket da hashtable.

checkhashfunc.m

Exemplo de como chamar plothash.m, com vários nomes de arquivo e funções hash comentados para conveniência.

H.m

A fórmula da entropia de uma distribuição. Entropia H = $\sum_{i=1}^{n} p_i \log p_i$ para uma distribuição de probabilidade discreta com n outcomes.

defhashfuncs.m

Define vários ponteiros de funções, definindo assim as funções de hash necessárias.

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?

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

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

(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)

(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?

(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)

(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?

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

(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.

*(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?*

(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)

*(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)*

Sugestão de leitura:

Muitas referências cobrem o básico de hashing. O livro do Corben é mais completo e discute os problemas tratados aqui.

Cálculo da Nota:

Testes em “AuxLab1Hash” não valem nada (apenas garante que o valor da função de hash está dentro da faixa correta)

Cada teste em “Lab1Hash” vale 1 ponto. A soma máxima possível de pontos vale nota 5 na implementação.
Cada questão bem respondida, vale 1 ponto. A soma máxima possível de pontos vale nota 5 nas questões.

Somando-se as duas partes temos a nota do lab, no máximo 10.

Libs auxiliares

Suponho que todos passaram em CES11, e já sabem implementar listas ligadas, árvores, etc.. Assim, a STL está liberada, e inclusive já forneço uma sugestão com as classes corretas de vetor e lista simplesmente ligada. Só não vale usar soluções prontas para resolver diretamente o propósito do exercício, ou seja, não pode usar classe com hash table ou map pronta. Em caso de dúvida, pergunte.

Entrega e lib do professor

Há 3 diretórios: tests, data, e src
Apenas entregue zip *do seu diretório src*.
Compilarei o seu diretório src com a minha cópia do código em tests e dados em data.
Assim, se mudar algum código em tests ou data, **isso não será transferido para mim**.

Vale utilizar, se inspirar ou copiar trechos do código fornecido, **mas se quiser mudar algo em lib e data, deve criar uma versão sua em src, mudando o nome dos identificadores para evitar conflitos de compilação**. Havendo dúvida pergunte.

Não submeta código com função main(). O seu código (solução) e o meu código (lib e testes) serão linkados com gtest_main, que já contém a função main(). Não compila se houver outra função main. Se preferir usar uma função main própria durante o seu processo de desenvolvimento, comente-a antes de submeter.

Aliás, recomendo que tentem entender o código fornecido, seria um bom aprendizado de OO e testes.

Quanto ao relatório, sugiro que copiem a seção “Relatório: Experimentos com funções de hash” em um documento novo e preencham as respostas. Não é necessário incluir ou reproduzir novamente outras partes deste documento. Quando estiver pronto, gerem um pdf para submeter.

Dicas:

Cmake x make: chame cmake ao criar novos arquivos fonte!

Cmake gera os makefiles que o make utiliza para compilar. Cmake precisa saber quais os arquivos a serem compilados.
Assim, se apenas mudar o conteúdo de .cpp, não precisa chamar cmake de novo. Mas, ao criar um novo arquivo .cpp, os makefiles não sabem que ele existe. É preciso chamar cmake novamente.

FAQ

Preciso usar a STL? Não posso implementar tudo na mão?

Pode implementar um vetor de listas ligadas e ignorar a STL.
Só precisa obedecer a mesma interface (cabeçalhos dos métodos) para passar nos testes, não interessa o que ocorre dentro do objeto desde que esteja implementada uma open hash table da forma estabelecida (vetor de listas ligadas).
MAS
Vector e forward_list são algumas das classes mais simples da STL, e este lab é muito simples. Até já indiquei as classes corretas (a STL tem árvore, listas duplamente ligadas, hash, heap, etc) Aproveitar esta oportunidade de aprender a STL vale a pena por si mesmo, e **facilitará enormemente implementações maiores nos próximos labs**.
A documentação da STL é confusa, mas o usual é *procurar por exemplos que complementem a documentação*. Assim a documentação resolve os aspectos formais (exatamente o que retorna esse método?), e os exemplos mostram como usar. Com o tempo, ganha-se experiência para interpretar a documentação e as mensagens de erro.

Que é essa %&#$@*% de entropia?

É ***a*** medida da **incerteza** de uma distribuição de probabilidade. O mais importante que vocês precisam saber:

Entropia é usualmente representada pela letra H. A “entrada” deve ser uma distribuição de probabilidade, e a “saída” é um número em bits. Assim, não faz sentido H(0.5), pois {0.5} não é uma distribuição de probabilidade - os elementos não somam 1.
Apenas faz sentido H({ 0.5, 0,5 }) que seria a entropia de um cara-coroa justo, que é exatamente 1 bit.

Porque esta unidade “bit”? Porque a entropia é um *limite inferior para o comprimento médio ponderado das palavras de um código que representa a distribuição*. Nesse caso, 0 e 1 são duas palavras com comprimento 1 bit, formando um código binário que pode representar “cara e coroa”, por exemplo 0 = cara e 1 = coroa.

A entropia da distribuição uniforme é máxima para um mesmo número de possibilidades (outcomes).

Exemplo: um sorteio justo com 4 possibilidades
H ({ ¼ ,¼ ,¼ ,¼ }) = 2 bits, deve ser claro que as palavras 00, 01, 10, 11 codificam esta distribuição com comprimento de palavra de 2 bits. Nenhuma outra distribuição de probabilidades com 4 outcomes (possibilidades) tem entropia maior.

Compare com um sorteio ainda com 4 outcomes mas não completamente justo:
H ({ ½ , ¼, ⅛, ⅛ }) = 1.75 bit
Codificando com o código 0, 10, 110, 111, teremos comprimento médio de 1 * ½ + 2 * ¼ + 3 * ⅛ + 3 * ⅛ = 1.75 bit, e como o comprimento médio é igual à entropia, o código é ótimo! (ótimo no sentido em que sabemos que não pode existir código com comprimento médio menor)

Outro exemplo extremo é a incerteza zero:
H ({ 1, 0 , 0 , 0 }) = 0 bit
ou seja, o primeiro sempre ganha, os outros 3 nunca ganham. Como já sabemos que o primeiro outcome sempre ganha, não precisamos transmitir nenhuma mensagem. O comprimento médio é zero.

Há muitos resultados sobre entropia para códigos, mas o importante aqui é que uma *distribuição de probabilidades menos uniforme implica em menor entropia*. Não é uma medida linear, mas é uma medida objetiva e matematicamente relevante.

No caso da hash table, o caso *ideal* seria que os dados fossem espalhados uniformemente por todos os buckets, portanto idealmente teríamos sempre a entropia máxima! (Considerando a contagem de elementos em cada bucket como uma aproximação da distribuição de probabilidade)

E além disso, **se sempre** o caso ideal fosse alcançado, aumentando o número de buckets sempre aumentaria a entropia. Ou seja, com 30 buckets sempre seria melhor espalhado do que com 29 buckets no caso ideal.

O construtor da classe Hash recebe um ponteiro para uma função que tem como parâmetro apenas uma string. Contudo, a função hashdiv que vamos implementar recebe uma string e um int. Então, supondo que eu crie uma main para chamar o construtor, eu teria que fazer uma função hashdiv com uma hashtable de tamanho pré-determinado e passar como parâmetro do construtor?

A função hashdiv não tem uma tabela, a tabela está dentro da classe Hash (atributo _table). A função hashdiv apenas entra string e sai um inteiro. Mas se você se refere a saber qual o tamanho da tabela para fazer a operação de mod (%), a função hashdiv é genérica. Para criar uma função para um tamanho especifico, eu criei uma função de conveniência hashdiv29 que esta em hashfunctions.cpp

Pretendia pedir mais algumas coisas nesse lab, que no final foram cobertas pela parte matlab, e por isso está mais genérico do que precisava.

2. O método hash da classe Hash apenas retorna o índice em que a string está ou também insere a string na hashtable?

É uma boa pergunta porque a classe está muito mais transparente do que o usual. A operação de hash normalmente nem precisa ser pública. Hash::hash() normalmente é private, apenas chama a função de hash que está armazenada, que foi passada no construtor. E o método Hash::add() é o que o usuário da classe precisa ver e usar.

Então: Hash::add() chama internamente Hash::hash() para saber onde inserir a nova string

3. o método worst_case é só para retornar o tamanho da maior lista ligada contida no hashtable, certo?

Sim, olhe o comentario em hash.h

Na função hashdiv, o que é o int m, ele é o tamanho da hashtable?

Sim.

4. Ainda na função hashdiv, o que é para ser retornado nessa função? Apenas 0, como está no código? Ou seria o índice que a string fica na hashtable?

“return 0” é só para ter algum código que compile, execute, e falhe nos testes. Você tem que implementar a função de hash, que retorna o índice onde a string fica na hashtable. O mesmo vale para todos os métodos que só tem “return 0” como implementação, são cabeçalhos vazios para vocês completarem: apague a linha ‘return 0’ e implemente o retorno do valor correto.

5. Está comentado no código que repetições não são permitidas. Nesse caso, duas strings "abc" e "acb", que têm o mesmo valor de soma de caracteres, são consideradas repetições ou não, dado que são strings diferentes?

Talvez tenha faltado contexto, mas em estrutura de dados, “repetições” significa dois elementos com valores iguais. “abc” e “acb” não são strings com valores iguais.

O que não pode é aparecer “abc” mais de uma vez. O que por sua vez significa que é preciso fazer uma busca antes de inserir.
Aliás, se não aceitasse 2 strings diferentes com o mesmo valor de hash, então cada lista ligada só poderia ter um elemento!

A função hash deve somar os valores dos caracteres da string como ints da tabela ascii ou deve realizar as operações com bits igual na simulação existente no site do demo animado mostrado na aula?

É uma boa pergunta pois ha varias funções hash possíveis, mas precisamos implementar exatamente a mesma para passar nos testes. O demo implementou uma solução diferente do que fiz no matlab - eu implementei da mesma forma que implementei no matlab.
Ou seja, somando os caracteres como inteiros e fazendo mod da soma.
MAS, **se** fosse para fazer XOR, existe um operador em C para fazer XOR bit a bit, não funciona com string direto mas funciona com int, long ou char.

Se mudar o cabeçalho dos métodos em "hash.h" ocorrerá problemas na compilação? Gostaria de adicionar um argumento em "contains", de modo que fique da seguinte forma:

int Hash::contains(std::string str, int &collisions, **int *h=NULL**);

Esse argumento "h" a mais permitiria não precisar chamar novamente a função "hash" na implementação da função "add", pois o código de add usaria o índice na hash table encontrado por contains.

resposta 1:
O teste chama diretamente a função contains, portanto você não pode mudar o cabeçalho. No entanto, o argumento opcional não quebra o teste, seria ok.
resposta 2:
É verdade que a classe está mais publica, mas aberta e visível do que o necessário, já expliquei que é para facilitar. Mas um método "contains" deveria existir em qualquer interface pública. Como poderia um programador usar uma classe hash sem poder checar se um valor está presente no hash?
e seria deselegante haver um terceiro argumento extra para o programador-usuário com um valor que só faz sentido internamente a classe. O argumento collisions já é desnecessário, adicionar mais uma tralha no cabeçalho, fica deselegante, pois o programador-usuário tem que ver o código do header, e existiria um argumento que ele não tem como usar - e precisa gastar tempo para entender isso.

E não é necessário. chame a função hash, ela serve para isso. chamar 2x uma função simples não fará grande diferença.

e mesmo assim, se quiser mesmo armazenar o inteiro retornado pela função hash para evitar uma segunda chamada a função de hash, vc pode não modificar o contains, mas criar um outro contains com 3 argumentos (não pode ser argumento opcional) que só é chamado internamente, e pode ser private. Lembra que é possível existir methods com o mesmo nome, mas número de argumentos diferente. Assim fica elegante pois o programador-usuário não precisa entender o que é o terceiro argumento.

Óbvio que esta classe é simples demais para se preocupar muito com detalhes de OO, mas, aproveitando para dar um bom exemplo: Para evitar copy-paste de código, o método com menos argumentos pode simplesmente chamar o método com mais argumentos e ignorar a funcionalidade extra:

```
int Hash::contains(std::string str, int &collisions) { // publico
   int aux;
   return contains(str,collisions,aux); // metodo interno privado
}
```

O que evita ter que implementar o metodo contains 2x. Basta ignorar o 3o argumento e a interface pública fica mais simples que a privada.

O teste não passa só porque inseri no fim da lista ligada ao invés do início?

Dado que a ordem dos elementos em cada lista não importa, reflita se inserir no fim da lista ligada é mesmo correto. Mesmo que houvesse um método para inserir no fim não significaria que é de graça - std::forward_list não é lista duplamente ligada (std::list é). Além disso, o teste impõe algumas convenções de qualquer forma - e.g., a forma de contar colisões - e faz parte do exercício seguir todas as convenções.
