INFO7011 - Desempenho de SGBD - PPGInf - UFPR
Marisa Sel Franco
Orientação: prof. Dr. Eduardo Almeida

Relatório trabalho 1 - Cuckoo Hashing

Este trabalho apresenta uma implementação simplificada da Cuckoo Hashing (PAGH e RODLES, 2001)[1], técnica que utiliza um dicionário simples, com tempo de consulta constante no pior caso. O algoritmo lida com colisões para garantir que um dado seja recuperado com, no máximo, dois acessos à memória no pior caso. Conforme apontam os autores (PAGH e RODLES, 2001), o algoritmo também é bastante competitivo no caso médio. No Cuckoo Hashing, são utilizadas duas tabelas hash (T1 e T2), de tamanho M, indexadas por funções de espalhamento distintas (h1 e h2). Cada chave do dicionário é armazenada em apenas uma das tabelas e toda nova inserção é realizada na primeira tabela (T1). Em caso de colisão no momento da inserção, a chave anteriormente armazenada em T1 é movida para a segunda tabela (T2). Se houver uma nova colisão em T2, o dado é movido para uma nova posição em T1. O processo continua até o fim das colisões ou até se atingir o limite pré-definido de iterações. Nesses casos, é feito um rehashing com novas funções de espalhamento. O nome do algoritmo deriva do comportamento de algumas espécies de cuco, nas quais o filhote empurra outros ovos ou filhotes para fora do ninho em uma variação do comportamento conhecido como parasitismo de ninhada.

Conforme explicado anteriormente, inserir uma nova chave em uma tabela de Cuckoo Hashing pode empurrar uma chave mais antiga para um local diferente nas tabelas. Pagh e Rodles (2001) salientam ainda que a exclusão é simples de executar em tempo constante e que há a possibilidade de encolher as tabelas se estiverem se tornando muito esparsas A presente implementação simula o algoritmo Cuckoo Hashing, porém de forma simplificada, sem realizar as operações de rehashing. Conforme a especificação solicitada, as chaves duplicadas também foram ignoradas na implementação, que mantém o armazenamento da primeira chave inserida e cujo detalhamento será apresentado a seguir

Compilação, execução, entrada e saída do programa

Para compilar, execute no terminal:

```
$ make
```

Para executar, digite no terminal:

```
$ ./myht < teste.in > teste.out
```

No arquivo de entrada (`teste.in`), feita pela entrada padrão, cada linha representa uma operação e uma chave inteira. A inclusão é sinalizada com (i) e remoção com (r). Já o arquivo de saída (`teste.out`) traz três informações por linha, separadas por vírgula: chave, tabela (T1 ou T2), posição de armazenamento. A ordenação, em ordem crescente, acontece pela chave. Caso não existam chaves armazenadas, nenhum valor será gravado na saída padrão.

Organização

O programa principal está no arquivo `myht.c`. Foram implementadas ainda duas bibliotecas: `hash_table.c`, que traz as funções necessárias à implementação da versão simplificada do algoritmo Cuckoo Hashing, e `auxiliar.c`, que traz as funções necessárias à geração da saída.

---

[1] PAGH, Rasmus; RODLER, Flemming Friche. Cuckoo hashing. In: **European Symposium on Algorithms**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. p. 121-133.

INFO7011 - Desempenho de SGBD - PPGInf - UFPR
Marisa Sel Franco
Orientação: prof. Dr. Eduardo Almeida

Estruturas de dados

- `hash_t` - Utilizada para a criação das tabelas hash, é composta por dois inteiros: `int k` - armazena o valor da chave; `int status` - armazena o status da posição da tabela. Na implementação, -1 representa que a chave foi excluída da posição, 0 representa nulo (posição nunca ocupada) e 1 representa que a posição está ocupada;
- `saida_t` - Utilizada para a criação da saída, é composta por dois inteiros e um vetor de char: `int k` - armazena o valor da chave; `char tabela[3]` - armazena a tabela: “T1” ou “T2”; `int posicao` - armazena a posição no qual o valor está armazenado na tabela;

Funções principais

- `void inicializa_tabelas(hash_t *t1, hash_t *t2)` - recebe duas tabelas hash e inicializa tanto a chave quanto o status com 0, indicando valores nulos e que as posições nunca foram ocupadas;
- `int h1(int k)` - recebe uma chave (k) e retorna sua posição aplicando a função hash h1(k) = k mod m;
- `int h2(int k)` - recebe uma chave (k) e retorna sua posição aplicando a função hash h2(k) = ⌊m * (k * 0.9 − ⌊k * 0.9⌋)⌋;
- `int busca_chave(int k, hash_t *t1, hash_t *t2)` - recebe uma chave (k) e as duas tabelas hash. Devido ao tratamento de colisão na inclusão e na exclusão implementado, se a posição da chave k calculada poer h1(k) estiver vazia, indica que a chave não foi encontrada. Caso contrário, retorna a posição da chave em T1 ou em T2;
- `void insere_chave(int k, hash_t *t1, hash_t *t2)` - recebe uma chave e as duas tabelas hash. Caso não seja uma chave duplicada/já inserida, a função verifica se a posição em T1 está vazia, insere a chave caso esteja e atualiza o status da posição para 1. Se houver colisão em T1, a chave originalmente grava em T1 é movida para T2 e a nova chave é armazenada na T1. Os status das posições são atualizados;
- `void exclui_chave(int k, hash_t *t1, hash_t *t2)` - recebe uma chave e as duas tabelas hash. Caso a busca encontre a chave e ela esteja na T1, a chave é excluída (k recebe 0) e o status da posição na tabela é atualizado para indicar a exclusão (status recebe -1). Caso esteja na T2, esses parâmetros são atualizados em T2.

Há comentários complementares ao longo do código que explicam de forma mais esmiuçada as implementações dessas funções e das funções auxiliares.