Projeto Prático - Sistema de Cadastro em Hashing Extensível

1. Observações Iniciais

O objetivo deste trabalho é implementar um sistema de cadastro (com recursos para busca e alteração de dados) em Hashing Extensível. O projeto vale 20 pontos e deverá ser realizado em equipe com dois a três alunos. A avaliação do projeto não será feita apenas sobre o código entregue mas também com a entrevista realizada com os membros de cada grupo. O projeto está dividido em três etapas, cada uma com avaliação independente. As etapas e suas respectivas pontuações são as seguintes:

- Sistema de cadastro com índice (diretório) em tabela hash: 8 pontos;
- Árvore Trie de dados binários: 4 pontos;
- Sistema de cadastro em Hashing Extensível : 8 pontos.

Os temas disponíveis para cadastro são apresentados na Seção 3 e, na sequência, tem-se a descrição de cada etapa.

2. Instruções gerais para desenvolvimento e entrega do projeto prático:

Seguem as instruções e recomendações gerais para o desenvolvimento do projeto prático:

- Os exercícios devem ser implementados em C++, usando técnicas de orientação a objetos (ou seja, os elementos principais devem ser implementados como classes).

- Nada de programação não-estruturada, não queremos ver um único "`goto`" no trabalho. E "`break`" só é aceito em "`switch`" (qualquer outro uso deve ser justificado antes com os professores). Também é recomendável não usar "`continue`". Variáveis globais que poderiam ser evitadas também reduzem grandemente o valor de seu trabalho.

- Portabilidade é questão chave: você deve programar lembrando-se que sua implementação será corrigida em linux. Portanto: nada de "`conio.h`" ou funções cheias de frescuras que não são portáveis (e que, geralmente, nada contribuem para o que realmente importa na resolução). Caso você precise realmente usar uma função que não é portável, verifique anteriormente com o professor o que pode ser feito neste caso. Não é exigido interface gráfica, mas se você quiser implementá-la, faça-o de forma que o sistema final fique portável.

- Boa organização e indentação do arquivo contam pontos na nota final. Má organização e indentação retiram pontos. Vocês tem liberdade para escolherem o estilo de indentação que julgarem mais adequado, mas respeitem o estilo escolhido. Recomendamos o uso de espaços para indentação, mas você pode usar tabulações, desde que não misture os dois tipos.

- Os exercícios deverão ser implementados com compilação separada, com classes principais em arquivos separados da aplicação. O uso de `Makefile` também é necessário.

- -> Cada etapa deve ser entregue em arquivo compactado (preferência por zip, tar.gz ou tar.bz), sendo que o nome do arquivo deve conter os nomes dos membros da dupla e etapa do projeto, por exemplo: `joao_maria_etapa2.tar.gz`.

- Todo código-fonte deve ter um cabeçalho que permita identificar a equipe e a utilidade do arquivo, por exemplo:

```
/*
  Trabalho de Introdução à Estripotologia
    (nome da disciplina)
  Método de Zumbi-Xupacabra
     (aqui vai o nome do seu trabalho)
  Copyright 2016 by Sicrano Beltrano Fulano
     (aqui vai o seu nome)
  Arquivo que não faz nada
     (caso seu trabalho tenha vários arquivos,
      informe o que cada arquivo faz, qual a sua função)
*/
```

- Uma implementação que cumpre a obrigação recebe nota 80. Se você quer mais que 80, precisa avançar do que foi pedido, por exemplo:
  - fazendo uma boa interface, que seja prática (usando passagem de parâmetro no terminal por exemplo, caso a questão permita);
  - mantendo boa organização do código, com uso adequado de comentários;
  - utilizando recursos de programação não solicitados, como controle de exceção, uso de templates, etc.
  - entrega antecipada;
  - implementação de recursos não solicitados, que mostrem esforço da equipe;
  - etc.

Em cada etapa, além do código fonte, **deverá ser entregue um relatório e um arquivo de dados de exemplo**. O relatório deverá obrigatoriamente conter:

- Uma descrição de todas as estruturas de dados utilizadas;
- Uma descrição **em alto nível** (isto é, sem código) explicando a lógica do programa.

O arquivo de dados de exemplo deverá conter ao menos 40 itens cadastrados.

O trabalho deverá ser implementado utilizando C++ e ser passível de compilação e execução em um Linux genérico qualquer.

3. Temas Disponíveis

Os temas disponíveis para implementação dos projetos são os seguintes:

Tema	Exemplo de estrutura armazenada

1 - Deuses Gregos	`struct objeto {`
	`  int id;`
	`  char nome[50];`
	`  char dominio[50]; // ex: deus da chuva`
	`  char biografia[200];`
	`  //Outros campos`
	`};`

2 - Heróis da Marvel	`struct objeto {`
	`  int id;`
	`  char nome[50];`
	`  char poderes[50]; // ex: imortalidade e força`
	`  char biografia[200];`
	`  //Outros campos`
	`};`

3 - Pókemons	`struct objeto {`
	`  int id;`
	`  char nome[50];`
	`  char tipo1[10]; // ex: water`
	`  char tipo2[10]; // para pókemon de dois tipos`
	`  char descricao[200];`
	`  //Outros campos`
	`};`

4 - Musicais	`struct objeto {`
	`  int id;`
	`  char nome[50];`
	`  char atorPrincipal[50];`
	`  char atrizPrincipal[50];`
	`  char enredo[200]; // breve síntese da história`
	`  //Outros campos`
	`};`

Continuação dos temas…

Tema	Exemplo de estrutura armazenada

5 - Instrumentos Musicais	`struct objeto {`
	`  int id;`
	`  char nome[50];`
	`  char tipo[10]; // ex: corda`
	`  int anoCriacao; // ano de surgimento`
	`  char descricao[200];`
	`  //Outros campos`
	`};`

6 - Deuses Egípcios e Nórdicos	`struct objeto {`
	`  int id;`
	`  char nome[50];`
	`  char dominio[50]; // ex: deus do trovão`
	`  char biografia[200];`
	`  //Outros campos`
	`};`

7 - Corpos Celestes	`struct objeto {`
	`  int id;`
	`  char nome[50];`
	`  char tipo[20]; // ex: estrela, planeta`
	`  double distancia; // distância aproximada`
	`  // da terra em anos-luz`
	`  char localizacao[20]; // ex: constelação`
	`  // de Virgo`
	`  char descricao[200];`
	`  //Outros campos`
	`};`

Continuação dos temas…

Tema	Exemplo de estrutura armazenada

8 - Saga Geek	`struct objeto {`
	`  int id;`
	`  char titulo[50]; // ex: Harry Potter,`
	`  // Star Wars, etc.`
	`  char personagensPrincipais[20][4];`
	`  // personagens da saga (2 a 4)`
	`  // ex: Harry Potter, Hermione,`
	`  // Ron e Voldemort`
	`  int anoLancamento;`
	`  int totalEpisodios; // quantidade de`
	`  // episódios da saga`
	`  // ex: Star Wars`
	`  // tem 8 episódios`
	`  char sinopse[200];`
	`  //Outros campos`
	`};`

9 - Chocolates em Barra	`struct objeto {`
	`  int id;`
	`  char nome[50];`
	`  char marca[20]; // ex: Garoto, Lacta`
	`  int teor; // teor de cacau`
	`  // (-1 se desconhecido)`
	`  char ingSecundario[20]; // segundo`
	`  // ingrediente,`
	`  // caso exista`
	`  // (ex. café,`
	`  // leite, menta)`
	`  char descricao[200];`
	`  //Outros campos`
	`};`

Continuação dos temas…

Tema	Exemplo de estrutura armazenada

10 - Cafés	`struct objeto {`
	`  int id;`
	`  char nome[50];`
	`  char marca[20]; // ex: Nescafé, Cafesal`
	`  float preco; // valor do quilo`
	`  // (-1 se desconhecido)`
	`  char tipo[20]; // arábica, solúvel, etc.`
	`  char descricao[200];`
	`  //Outros campos`
	`}`

11 - Aves Brasileiras	`struct objeto {`
	`  int id;`
	`  char nome[50];`
	`  char regiao[20]; // ex: Cerrado, Amazônia`
	`  int populacao; // tamanho da espécie`
	`  // (-1 se desconhecido)`
	`  char nomeCientifico[30];`
	`  char descricao[200];`
	`  //Outros campos`
	`};`

12 - Temperos	`struct objeto {`
	`  int id;`
	`  char nome[50];`
	`  char origem[20]; // ex: Índia, Itália`
	`  int ano; // ano de primeiro registro`
	`  // (-1 se desconhecido)`
	`  char nomeAlternativo[50];`
	`  char descricao[200];`
	`  //Outros campos`
	`};`

Continuação dos temas…

Tema	Exemplo de estrutura armazenada

13 - Plantas do Cerrado	`struct objeto {`
	`  int id;`
	`  char nome[50];`
	`  char tipo[20]; // ex: frutífera, flor`
	`  int populacao; // tamanho da espécie`
	`  // (-1 se desconhecido)`
	`  char estado[30]; // Estado (ex: MG, GO)`
	`  // com maior predomínio`
	`  // da planta`
	`  char descricao[200];`
	`  //Outros campos`
	`};`

14 - Hinos de Países	`struct objeto {`
	`  int id;`
	`  char nome[50];`
	`  char pais[20];`
	`  int ano; // ano de composição`
	`  char compositor[30];`
	`  char historico[200]; // breve histórico`
	`  // do hino`
	`  //Outros campos`
	`};`

15 - Monstros da Cultura Popular	`struct objeto {`
	`  int id;`
	`  char nome[50];`
	`  char pais[20];`
	`  char nome_alternativo[50];`
	`  char descricao[200];`
	`  //Outros campos`
	`};`

Os temas só se repetirão entre as equipes, após todos já terem sido escolhidos, sendo que a ordem para escolha do tema será sorteada em aula. Em caso de repetição, será permitida uma única vez.

4. Etapa 1

Ao ser aberto, o programa abrirá o arquivo e permitirá as seguintes operações:

1. Inserir um novo objeto no arquivo.

2. Remover um objeto do arquivo. Fica a critério do grupo não apagar o objeto diretamente, mas marcar o espaço para reutilização. Nesse caso, a inserção deverá obrigatoriamente reutilizar espaços disponibilizados por remoção.

3. Consultar a um objeto no arquivo, usando busca binária ou sequencial.

4. Imprimir o arquivo, com todo seu conteúdo, na ordem de armazenamento.

5. Imprimir os registros de um dado bloco de modo ordenado, caso o armazenamento não seja feito de forma ordenada.

Os dados deverão ser armazenados em um arquivo de blocos indexado por uma tabela hash. Cada bloco irá armazenar até TAM_BLOCO registros, sendo que TAM_BLOCO é um valor constante definido no código. Por sua vez, o arquivo de dados será capaz de armazenar até CAPACIDADE de blocos. Ou seja, ao todo, o arquivo terá capacidade para CAPACIDADE × TAM_BLOCO registros. Para este projeto prático, considere que cada bloco armazena **quatro registros** e que o arquivo tem capacidade total para **dezesseis blocos**.

Para efeito de simplificação, recomenda-se que o diretório dos dados (a tabela hash) seja armazenada em um arquivo auxiliar e seja carregado integralmente para a memória primária. O arquivo de dados nunca é lido integralmente, mas um bloco por vez, com todos os registros contidos neste bloco. Ao procurar um dado registro, por exemplo, pode ser necessário percorrer o bloco inteiro, caso os dados não sejam armazenados de forma ordenada em cada bloco. Em qualquer momento da execução do sistema, no máximo três blocos poderão estar alocados na memória primária.

O diretório de dados armazena, para cada chave, o endereço relativo do bloco no arquivo. O hash será calculado a partir do ID do objeto sendo armazenado, gerando uma codificação em **número binário com quatros dígitos**. A partir dessa chave, será associada uma posição relativa (índice do bloco no vetor de blocos) no disco contendo o bloco de dados. Obviamente, o tamanho do diretório é de CAPACIDADE entradas.

Como cada bloco só permite TAM_BLOCO registros, isso implica que há um tratamento de colisão usando vetor, ou seja, um número fixo de colisões é permitido. Acima desse limite, o aplicativo deve emitir uma mensagem de erro e não permitir a gravação de dados com aquele ID.

Data de Entrega da Etapa 1: 27 de maio de 2018

5. Etapa 2 - DESCONSIDERAR, AINDA NÃO PRONTA!

Para preparar para a próxima etapa, vocês deverão implementar uma TRIE, uma árvore de prefixos.
Os objetos da etapa anterior agora deverão ser carregados do disco e armazenados ordenadamente em um vetor expansível. Considere que cada pedaço (bucket) do vetor expansível só possui capacidade para no máximo cinco elementos. São esperadas as seguintes propriedades do vetor expansível:

1. Cada pedaço encontra-se ordenado e elementos de um pedaço são menores que o próximo.
2. Os pedaços são encadeados (a decisão por encadeamento simples ou duplo fica a critério do grupo).
3. Quando um pedaço cheio receber um elemento adicional, ele deverá primeiro tentar repassar um de seus elementos para o pedaço anterior ou próximo. Caso isso não seja possível, o pedaço será dividido em dois novos pedaços.
4. É interessante, mas não exigido que nenhum pedaço seja mantido com menos que a metade do número máximo de elementos (ou seja: pedaços devem ter ao menos dois elementos). Nesse caso, a solução é por empréstimo dos pedaços vizinhos ou fusão de dois pedaços em um único pedaço. Note que isso não se aplica a um único pedaço.

Data de Entrega da Etapa 2: 2 de Dezembro de 2017

5. Etapa 3

Os objetos da etapa anterior agora deverão ser ordenados de acordo com ***outra(s) chave(s)*** (por exemplo o nome), utilizando-se de ***método de ordenação externa baseado em intercalação polifásica.*** Note que já se espera que o vetor expansível esteja ordenada com uma chave inicial, o que se pede aqui é a possibilidade de **ordenação por outra(s) chave(s)**.

A ordenação será realizada de forma a ***gerar um novo arquivo ordenado, sem alterar o arquivo original***. ***O arquivo gerado deve ser um arquivo binário de registros***. Obviamente, deverá ser providenciado um mecanismo para visualizar o novo arquivo ordenado.

Esta etapa pode ser implementada tanto como uma opção adicional na aplicação original, ou como uma aplicação adicional para a tarefa. Por exemplo, supondo que seu sequence set esteja com dados no arquivo `dados.bin`, você poderia implementar uma aplicação que funcionasse da seguinte forma:

```
./ordena_por_nome dados.bin dados_ordenados_por_nome.bin
./vizualiza dados_ordenados_por_nome.bin
```

Data de Entrega da Etapa 3: 3 de fevereiro de 2018

6. Etapa 4

Os objetos das etapas anteriores deverão agora ser armazenados em uma árvore B em disco. Nós da árvore serão representados por uma estrutura similar à seguinte:

```
struct noh  {
      int blocos[2*ORDEM]; // valores que separam os blocos
      bool folha; // se é folha ou não da árvore-B
      struct noh *filhos[2*ORDEM+1]; // ponteiros para os filhos
      int numChaves; //número de chaves no nó
};
```

Nesta estrutura, `ORDEM` representa a ordem da árvore B. Você pode utilizar outras definições de ordem da árvore B, a seu critério. Também a seu critério, elementos apagados podem apenas ser marcados para posterior reutilização. Obviamente, nesse caso, é esperado que a reutilização ocorra realmente.

Funcionamento do programa:

Ao ser aberto, o programa carregará a árvore B e permitirá as seguintes operações:

1. Inserir um novo objeto. Insere um novo objeto no sequence set (arquivo). Se houver uma divisão, insere o índice e o apontador para o novo bloco na árvore.

2. Remover um objeto. Opcionalmente, elementos apagados podem apenas ser marcados para posterior reutilização. Obviamente, nesse caso, é esperado que a reutilização ocorra realmente.

3. Buscar um objeto. Busca um objeto utilizando a árvore até encontrar o endereço do bloco correspondente no disco.

4. Imprime Ordenado. Imprime os registros de modo ordenado, em mais de uma chave.

Data de Entrega da Etapa 4: 4 de fevereiro de 2018