Técnicas Avançadas de Programação – 2019/1
Curso: Bacharelado em Sistemas de Informação – Campus Serra
Turma: 20191.BSI.6

1ª Atividade: Implementação do TAD dicionário baseado em técnica de hashing.

Leia o relato abaixo, modelos os dados e construa uma implementação pedida para o sistema :

Faça a leitura, análise e implementação do diagrama de projeto abaixo (Figura 1). Siga as orientações e restrições impostas ao projeto e descritas nos tópicos de I a V e abordadas em aula.

I. Diagrama de Projeto

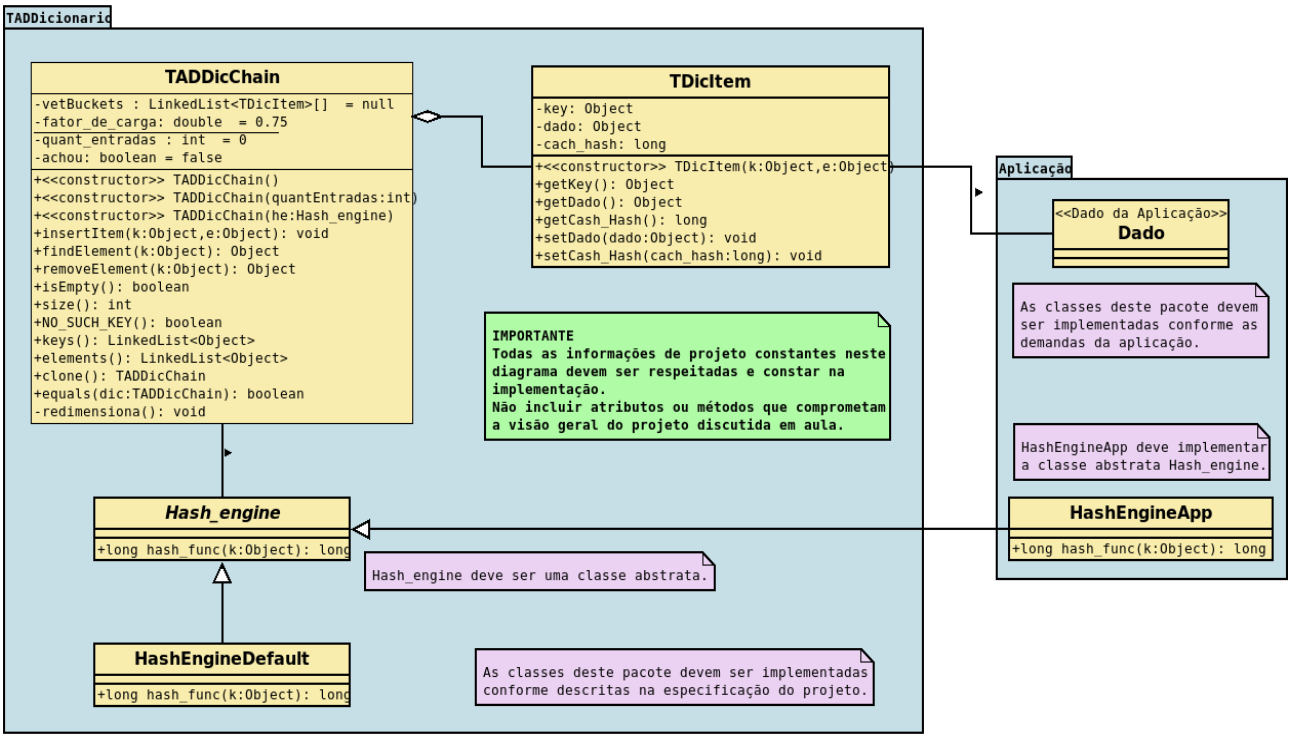

Figura 1: diagrama de classes de projeto

Utilizar a linguagem Java com as seguintes restrições (discutidas e adotadas em aula): não utilizar tratamento de exceções (apenas as necessárias às soluções discutidas e adotadas em aula), utilizar no máximo as seguintes classes nativas *LinkedList*, *ArrayList*, *String*, *Stack*, *Queue*, *Char*, *Integer*, *Float* e/ou qualquer outra construção usada somente em aula.

Utilizar as soluções e direções fornecidas pelo livro texto, adaptadas, discutidas e implementadas em sala de aula.

Leia com atenção os tópicos a seguir. Este é um projeto de implementação. A implementação é correta quando atende a todos os requisitos constantes no projeto.

Especial atenção no tópico V, Entrega. Construa o produto da atividade no formato solicitado e conforme descrito ao longo deste documento.

Dentre os materiais fornecidos consta o arquivo *DiagramaTADDicChain.png* com o original da figura 1. Consulte este arquivo caso a reprodução na figura 1 fornecer alguma dificuldade na leitura dos detalhes das classes do projeto.

II. Descrição das Principais Classes do Projeto

TADDicChain

Principal classe do projeto. Deve ser implementada de acordo com as especificações das figuras 1 (diagrama de classes) e 2 (diagrama conceitual) e recomendações dadas em aula. Os seguintes métodos fazem parte da interface considerada mínima e necessária:

Public TadDicionario(HashEngine)*:* constrói uma instância do dicionário que utiliza um objeto HashEngine para ser o responsável pelas operações sobre o código de hashing (geração e comparação de chaves). O tamanho do vetor de dados da tabela de hashing subjacente ao dicionário é pré-definido internamente e seu valor é 1024 (seguindo padrão do livro texto, padrão Java);

Public TadDicionario(Tam, HashEngine): semelhante ao método anterior, com a diferença que o tamanho do vetor de posições da tabela de hashing é determinado pela aplicação que cria o dicionário.

Public void insertItem(k, e): cadastra uma entrada do dicionário associado um conteúdo e a uma chave k. O método deve saber tratar tanto a inclusão de uma nova entrada quanto a alteração de uma entrada já existente (associar um novo conteúdo a uma chave/entrada já cadastrada). O dicionário deve permitir tanto a inclusão e dados de valor igual a null (tratar o valor null como um conteúdo válido).

Public object removeElement(k): remove a entrada do dicionário associada à chave k e retorna o conteúdo associado à chave. Após a chamada desse método, a invocação de NO_SUCH_KEY() deve retornar *false* se a remoção ocorreu com sucesso, e retornar *true* caso a chave fornecida na remoção não exista no dicionário.

Public object findElement(k): retorna o conteúdo do dicionário associado à chave k. Após a chamada desse método, a invocação NO_SUCH_KEY() deve retornar *false* se a remoção ocorreu com sucesso, e retornar *true* caso a chave solicitada na remoção não exista no dicionário.

Public boolean NO_SUCH_KEY(): deve retornar *false* caso a última operação de pesquisa de uma chave no dicionário tenha encontrado a chave procurada, e deve retornar *true* caso contrário. Os métodos que disparam operações de pesquisa são *findElement(k)* e *removeElement(k)*.

Public int size(): retorna a quantidade de entradas cadastradas no dicionário.

Public boolean isEmpty(): retorna *true* se a quantidade de entradas do dicionário for zero, retorna *false* caso contrário.

Public LinkedList getItens()*:* retorna uma lista de objetos elementos de dados (dados da aplicação) cadastrados e ativos no dicionário;

Public TADDicChain clone(): cria e retorna um dicionário clone do dicionário corrente (this). O clone é definido como um dicionário que possui a mesma quantidade de entradas que o dicionário original (diferentes TDicItem, mesmas referências para os dados da plicação, mesmos valores de chaves).

Private void redimensiona(): redimensiona o vetor de buckets para 1,5 o tamanho do vetor original caso a média dos tamanhos das listas encadeadas chegue a 30% do tamanho atual do vetor. O redimensionamento TEM de fazer uso do cache do código de hashing armazenado no TDicItem.

Public LinkedList<Object> keys(): retorna uma lista contendo as chaves de todas as entradas do dicionário.

Public LinkedList<Object> elements(): retorna uma lista contendo conteúdos de todas as entradas do dicionário.

Public boolean equals(TADDicChain d): retorna *true* se o dicionário atual (this) for igual ao dicionário *d* passado como argumento. Retorna *false* caso contrário. Dois dicionários são iguais se possuírem a mesma quantidade de entradas, as mesmas chaves e os mesmo dados da aplicação como conteúdos das entradas.

Public int[] getVetColisoes(): retorna um vetor de inteiros contendo em cada posição as quantidade de colisões das posições correspondentes no vetor de buckets (tamanho das listas encadeadas). A ideia é usar esse vetor para construir a lógica, externa à classe, de confecção dos gráficos de colisão para uma dada função de hashing.

Hash_engine

Classe abstrata contendo a lógica para o cálculo do código de hashing. A função *public long hash_func(Object k)* deve ser implementada pelos descendentes desta classe. Os descendentes desta classe deverão ser construídos por cada aplicação. O corpo da função *hash_func(Object k)* conterá o algoritmo de hashing adotado pela aplicação.

Public long hash_func(Object k): calcula e retorna o código de hashing associado à chave passada como argumento (*Object k*).

TDicItem

Classe que representa o contêiner base das listas associadas a cada posição do vetor de buckets. Ou seja, a entrada do dicionário (chave + conteúdo, dado da aplicação). Ver as figuras 1 e 2 para contextualizar a classe dentro da implementação do *TADDicChain*. A classe também é responsável pelo cache do código de hashing calculado pela função *hash_func(Object k) ,* classe *Hash_engine*. Para maiores esclarecimentos recorra às anotações de aula, ao livro texto da disciplina ou ao professor.

Public TDicItem(Object k, Objecte): instancia uma entrada do dicionário a partir dos pares chave (*Object k*) e conteúdo (*Object e*, dado da aplicação).

Public Object getKey(): retorna a chave da entrada do dicionário.

Public Object getDado(): retorna o conteúdo da entrada do dicionário.

Public void setDado(Object dado): altera o conteúdo da entrada do dicionário. Não verifica a consistência com a chave da entrada.

Public long getCash_Hash(): retorna o código de hashing obtido a partir da chave da entrada e da função *hash_func(Object k)*, classe Hash_engine.

Public void setCash_Hash(long cach_hash): armazena o código de hashing obtido a partir da chave da entrada e da função *hash_func(Object k)*, classe *Hash_engine*.

III. Implementação

A implementação do TADDicChain deve seguir à risca as orientações do livro texto no que diz respeito ao esquema das estruturas de dados envolvidas. O TADDicChain deve possuir como contêiner de dados um vetor de listas encadeadas de objetos TDicItem. A figura 2 resume o esquema por trás do diagrama de classes da figura 1.

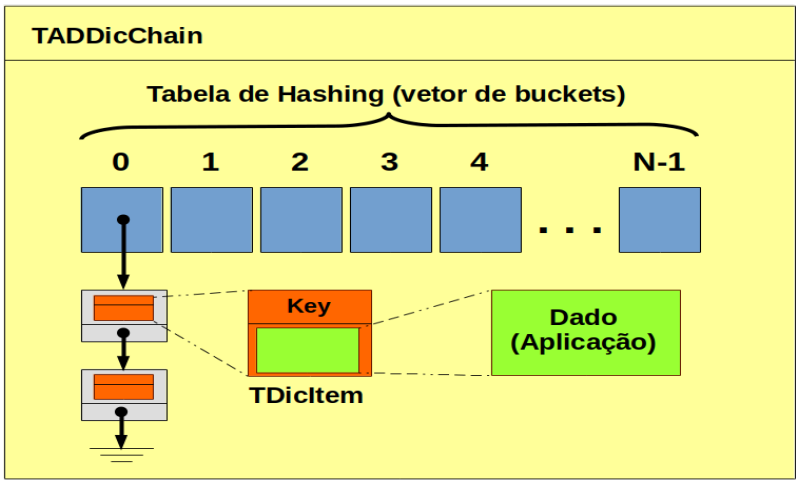

Figura 2: esquema de implementação do TADDicChain.

Na implementação é permitido acrescentar métodos privados que visam contribuir para construção da interface pública (descrita pelo diagrama de classes da figura 1 e pela seção II). Contudo tais métodos não podem descaracterizar nenhuma assinatura e nenhum dos comportamentos descritos na seção II.

IV. Lista de Verificação

A lista de a seguir visa auxiliar a verificação de pontos chaves da implementação, se esta está de acordo com pontos essenciais da especificação. A lista é formada por perguntas chaves.

A resposta afirmativa a todas as perguntas atesta que a implementação está de acordo com o que foi solicitado.

a) A implementação está de acordo com as figuras 1 e 2?
b) A implementação disponibiliza a interface descrita na seção II?
c) A implementação conseguiu ser utilizada pelo benchmark sem nenhuma alteração dos itens descritos nas figuras 1 e 2 e na seção II?
d) A implementação aceita como chaves e conteúdos qualquer classe de objetos?
e) O dicionário implementado trata corretamente o redimensionamento do vetor de buckets?
f) O dicionário implementado trata corretamente Null como uma chave válida?

g) O dicionário implementado trata corretamente inclusões e alterações de chaves já existentes ?
h) A implementação e uso da função *NO_SUCH_ITEM()* está de acordo a especificação e as discussões em aula?

V. Aplicação

Construa uma aplicação usuária que usa o TADDicChain como um provedor de funcionalidades. A especificação segue abaixo:

V.1) TAD Matriz

Construa o arquivo tadmatriz.py a ser utilizado como um módulo por outros programas. O arquivo implementa um tipo abstrato de dados matriz (TAD) que segue a especificação a seguir. O trabalho será considerado correto somente se a expecificação abaixo for seguida.

Especificação TAD Matriz (classe TADMatriz):

a) Requisito: a matriz não deve armazenar elementos de valor zero.

b) Modelo/Estrutura de dados: O TADMatriz deve ser implementado como um classe usuária do TADDicChain. A modelagem eematriz/dicionário faz parte da atividade.

Sugestão de modelagem usando como referência uma estrutura de dados em Python.

{'lins': <int>, 'cols': <int>. 'dados': {<chave tupla lin, col: dado elem <float>> }}

Onde *lins* e *cols* são, respectivamente, quantidade de linhas e quantidade de colunas.

c) Interface do TADMatriz

c.1) *public TADMatriz(int linhas, int colunas)*: cria um objeto matriz do tipo tad matriz representando uma matriz de dimensões <int linhas> x <int colunas>.

c.2) *public Float getElem(int i, int j)*: retorna o elemento número real armazenado na posição *i* (linha), *j* (coluna) da matriz. Retorna *Null* se o elemento não existir (*i* e *j* forem valores que extrapolam as dimensões da matriz).

c.3) *public Float setElem(int i, int j, Float valor)*: armazena o elemento valor na posição i (linha), j (coluna) da matriz. Retorna *Null* se a posição não existir (i e j forem valores que extrapolam as dimensões da matriz). Retorna valor, caso contrário.

c.4) ***public TADMatriz soma(TADMatriz m)***: soma a matriz corrente, *this*, com a matriz *m* e retorna uma terceira matriz com o resultado da soma. O método deve usar o conceito matemático de soma de matrizes. Retorna *Null* se as matrizes não puderem ser somadas.

c.5) ***public void vezesK(float k)***: altera a matriz corrente, *this*, multiplicando os seus elementos por *k*.

c.6) ***public TADMatriz multi(TADMatriz m)***: multiplica a matriz corrente, *this*, pela matriz *m* de entrada e retorna uma terceira matriz com o resultado da multiplicação. O método deve usar o conceito matemático de multiplicação de matrizes. Retorna *Null* se as matrizes não puderem ser multiplicadas.

c.7) ***public int quantLinhas()***: retorna a quantidade de linhas da matriz de entrada.

c.8) ***public int quantColunas()***: retorna a quantidade de colunas da matriz de entrada.

c.9) ***public TADMatriz transposta()***: retorna uma nova matriz com a transposta da matriz corrente, *this*.

c.10) ***public static TADMatriz carrega(String nome_arq)***: carrega uma matriz a partir de um arquivo texto de nome *nome_arq*. Retorna uma matriz do tipo TADMatriz preenchida com o conteúdo arquivo. No arquivo, a matriz está representada da seguinte forma: elementos de cada linha da matriz separados por espaço, cada linha do arquivo texto é uma linha da matriz.

c.11) ***public String salva(String nome_arq)***: salva uma matriz em um arquivo texto de nome *nome_arq*. Retorna o nome do arquivo usado para salvar a matriz. No arquivo, a matriz deve estar representada da seguinte forma: elementos de cada linha da matriz separados por espaço, cada linha de texto é uma linha da matriz.

V.2) Aplicação Matriz

Construa a classe aplicação Apmattpa. A aplicação deve, obrigatoriamente, fazer uso da classe TADMatriz. A aplicação deve também ler e processar LINHA A LINHA o arquivo **bdaritmat.csv**. Um exemplo do conteúdo e formato do arquivo está disponível na figura 3. No arquivo, cada letra maiúscula representa o nome de uma matriz. A letra minúscula ***t*** representa a operação transposta de uma matriz *m* qualquer. Por último, constantes inteiras são apenas valores multiplicadores. Usando a figura 3 como exemplo, vemos que a primeira linha do arquivo contém apenas o nome de uma matriz. Isto significa que os dados dessa matriz devem ser carregados na memória e transformados em um tad matriz. A partir daí, encontramos em cada linha uma dupla separada por vírgula contendo uma operação aritmética e um nome de matriz (exceção da operação de transposição, apenas a letra *t* minúscula).

A interpretação da tarefa pode ser obtida diretamente da observação da figura 3 e das explicações dada pelo professor em sala de aula.

As matrizes que aparecem em cada linha de **bdaritmat.csv** tem seus dados armazenados nos arquivos de mesmo nome <nome da matriz>.txt (Ex. A.txt, B.txt, etc.). Os arquivos <nome da matriz>.txt estão dentro da pasta **bdmatrizes**.

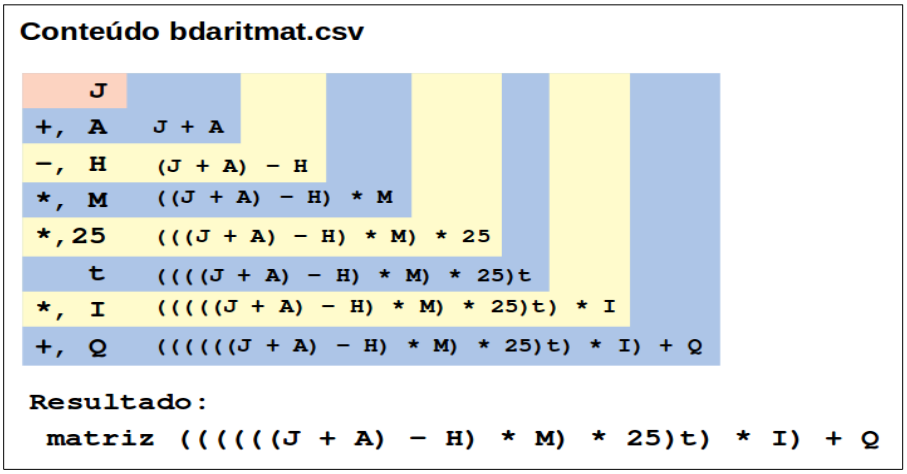

Figura 3: conteúdo exemplo do arquivo bdaritmat.csv.

Ao final do processamento, a função main da aplicação deverá: i) salvar a matriz resultado no arquivo resposta.txt; ii) imprimir na tela, de forma organizada, os elementos das diagonais principal e secundárias da matriz resultado.

V. Entrega

A atividade deve ser entregue da seguinte forma: um arquivo com o nome do aluno no formato zip, contendo:

1. uma pasta de nome taddic contendo todos os fontes relativos ao TADDicChain.
2. Uma pasta chamada aplicacao contendo fontes e dados envolvidos na aplicação.

Veja o novo prazo no link da atividade no moodle.

Fim!