Documentatie

Name: Ursan Victor

Group: 917

DSAProject

Map

Interface:

class Map {

virtual void addValueAndKey(Key key, Value value) = 0;

virtual void updateValueForKey(Key key, Value value) = 0;

virtual Value valueForKey(Key key) = 0;

virtual Key keyForValue(Value value) = 0;

virtual void removeKey(Key key) = 0;

virtual void removeValue(Value value) = 0;

virtual bool doesKeyExist(Key key) = 0;

virtual bool doesValueExist(Value value) = 0;

virtual DynamicVector<Key> getKeys() = 0;

virtual DynamicVector<Value> getValues() = 0;

}

First representation:

-Dynamic Vector

class DynamicVector {

private:

TElement \*elements;

int size = 0;

int capacity = 10;

void resize();

public:

DynamicVector();

DynamicVector(TElement t) {};

~DynamicVector();

void add(TElement e);

void removeAtIndex(int index);

void updateElement(int index, TElement element);

TElement elementAtIndex(int index);

int indexForElement(TElement e);

int getSize();

bool isEmpty();

bool doesContainElement(TElement e);

}

template<typename Key, typename Value>

class MapDVector: Map<Key, Value> {

private:

DynamicVector<Key> keys;

DynamicVector<Value> values;

}

Second representation:

-hashtable, collision resolution through chaining; use sorted singly linked list

template <typename Key, typename Value>

class Pair {

Key key;

Value value;

public:

Pair(Key key, Value value) {this->key = key; this->value = value;};

Key getKey() {return key;};

Value getValue() {return value;};

void setKey(Key key) {this->key = key;};

void setValue(Value value) {this->value = value;};

};

template <typename Key, typename Value>

class Node {

Pair<Key, Value> \*pair;

Node<Key, Value> \*next;

public:

Node() {};

void setData(Pair<Key, Value> \* aData) { pair = aData; };

void setNext(Node<Key, Value> \* aNext) { next = aNext; };

Pair<Key, Value> \* getData() { return pair; };

Node<Key, Value> \* getNext() { return next; };

};

template <typename Key, typename Value>

class SortedList {

Node<Key, Value> \*head;

int length;

public:

SortedList() {head = NULL; length = 0;};

void removePair(Pair<Key, Value> \*data);

void addPair(Pair<Key, Value> \*data);

Value findValueForKey(Key key);

Key findKeyForValue(Value value);

void updateKeyForValue(Value value, Key key);

void updateValueForKey(Key key, Value value);

bool doesKeyExist(Key key);

bool doesValueExist(Value value);

};

template <typename Key, typename Value>

class MapHash: Map<Key, Value> {

SortedList<Key, Value> lst[134217728];

DynamicVector<Key> keys;

}

Requirements:

Choose one of the problems from the list (see section problem) and solve it by using the given ADT and two DS to implement it.

Use a list of at least 5000 words; consider only words longer than 3 characters long.

1. Find all the anagrams of a given word

An anagram of a word is the result of rearranging the letters of a word to produce a new word, using all the original letters, and having the same letter frequencies.