**CSC 323 Project 3.1 (C++)**

Tyler Gaugler

Due Date: 10/20/2016

Algorithm Steps for this project:

Step 0: inFile <-- open the input file  
        Top <-- create a new stack  
  
Step 1: data <-- read a data from the input file  
        newNode <-- create a new listNode for the data  
push (Top, newNode) <-- push newNode onto the top of the stack  
longestStringLength <-- you need to keep tract of string length of the data here  
  
step 2: repeat step 1 until file is empty  
  
step 3: print the stack  
  
step 4: create hashTable[2][tableSize] // make sure initialize all 256 queues in each hashTable!  
  
Step 5: currentDigit <-- 0 // the first digit/position from the right of the string.  
currentTable <-- 0  
  
Step 6: // move each listNode from stack to hashTable[0]  
- node <-- pop from the stack  
- padString (node) // pad the data in the node to be as long as longestStringLength  
- val <-- getVal (node, currentDigit)  
// get the character from the currentDigit of the data in the node  
  
- hashIndex <-- hashIndex (val) // use atoi(val) or cast as (int) val  to get the  index  
- addTail (hashTable[currentTable][ hashIndex]) // add the node at the tail of the queue at hashTable[currentTable][ hashIndex]  
  
Step 7: repeat step 6 until stack is empty  
Step 8:  printTable (HashTable[currentTable] // print only none empty queue in the table.  
            // See printing format given above.  
  
Step 9:  
- currentDigit++  
- currentTable <-- 1  
- previousTable <-- 0  
- currentQueue <-- 0  
  
Step 10: // moving nodes from previous table to current table  
  
node <-- deleteHead from the currentQueue in  
                        hashTable[previousTable][currentQueue]  
  
 val <-- getVal (node, currentDigit)  
// get the character from the currentDigit of the data in the node  
  
hashIndex <-- hashIndex (val) // use atoi(val) or cast as (int) val to get the index  
  
addTail (hashTable[currentTable][ hashIndex])  
// add the node at the tail of the queue at hashTable[currentTable][ hashIndex]  
  
step 11:  repeat steps 10 until the currentQueue is empty  
  
Step 12: currentQueue ++ // process the next queue in the previous hashTable  
  
 Step 13: repeat step 10 to step 12 until currentQueue >= tableSize - 1  
  
step 14: temp <-- currentTable  
currentTable <-- previousTable  
previousTable <-- temp  
currentQueue <-- 0  
  
Step 15: repeat step 9 to 14 until currentDigit >= longestStringLength.  
  
Step 16: print currentTable // see printing format below

**Source Code:**

//main.cpp

#include <iostream>

#include <fstream>

#include <string>

#include <stdlib.h>

using namespace std;

ofstream outputFile;

class listNode{

public:

string data;

listNode\* next;

listNode(void);

listNode(string data);

~listNode(void);

};

listNode::listNode(){

next=NULL;

data="";

}

listNode::listNode(string d){

next=NULL;

data=d;

}

listNode::~listNode(void){

}

class linkedListStack{

public:

listNode\* top;

linkedListStack(void);

void push(listNode\* newNode);

listNode\* pop();

int isEmpty();

void printStack(string fileName1);

};

linkedListStack::linkedListStack(){

top= new listNode();

}

void linkedListStack::push(listNode\* newNode){

newNode->next= top->next;

top->next= newNode;

}

listNode\* linkedListStack::pop(){

listNode\* nodeToPop= top->next;

top->next= nodeToPop->next;

return nodeToPop;

}

int linkedListStack::isEmpty(){

if(top->next == NULL) return 1;

else return 0;

}

void linkedListStack::printStack(string fileName1){

// ofstream outputFile1;

// outputFile1.open(fileName1, std::ofstream::out | std::ofstream::app);

listNode\* current= top->next;

while(current!=NULL ){

outputFile<<current->data<<endl;

current=current->next;

}

outputFile<<endl<<endl;

}

class linkedListQueue{

public:

listNode\* tail;

listNode\* head;

linkedListQueue(void);

void addTail(listNode\* newNode);

listNode\* deleteHead();

int isEmpty();

void printQueue(int tableIndex, int queueIndex,string fileName1);

};

linkedListQueue::linkedListQueue(){

head= new listNode();

head->next = new listNode("dummy");

tail = new listNode();

tail->next = head->next;

}

void linkedListQueue::addTail(listNode\* newNode){

listNode\* tailNode = tail->next;

newNode->next= tailNode;

tail->next= newNode;

}

listNode\* linkedListQueue::deleteHead(){

listNode\* headNode = tail->next;

while(headNode->next){

headNode=headNode->next;

}

listNode\* current = tail;

listNode\* newHeadNode;

// cout<<"\n current: "<<current->data<<" head: "<<head->next->data<<"\n\n";

// cout<< headNode->data;

// if(headNode==NULL)cout<<"yeet";

while(current->next){

// cout<<current->next->data<<endl<<headNode->data<<endl<<endl;

if(current->next->data == headNode->data){

// cout<<"yeeet";

newHeadNode=current;

}

current=current->next;

}

newHeadNode->next=NULL;

head->next=newHeadNode;

return headNode;

}

int linkedListQueue::isEmpty(){

if(tail->next == NULL || tail->next->data == "dummy") return 1;

else return 0;

}

void linkedListQueue::printQueue(int tableIndex, int queueIndex,string fileName1){

listNode\* current= tail->next;

outputFile<<"Table["<<tableIndex<<"]["<<queueIndex<<"]: ";

while(current!= NULL){

outputFile<<current->data<<", ";

current = current->next;

}

outputFile<<endl;

}

class radixSort{

public:

int tableSize;

linkedListQueue\*\* hashTable;

int currentTable;

int tableIndex;

radixSort(void);

void padString(listNode\* node, int longestStringLength);

char getVal(listNode\* node, int currentDigit, int longestStringLength);

int hashIndex(char value);

void printTable(int currentTableIndex,string filename);

};

radixSort::radixSort(){

tableSize=256;

hashTable= new linkedListQueue\*[2];

for (int i = 0; i < 2; ++i) hashTable[i] = new linkedListQueue[tableSize];

}

void radixSort::padString(listNode\* node, int longestStringLength){

while(node->data.length() < longestStringLength){

node->data= " "+node->data;

}

}

char radixSort::getVal(listNode\* node, int currentDigit, int longestStringLength){

return node->data[longestStringLength-currentDigit];

}

int radixSort::hashIndex(char value){

return (int)value;

}

void radixSort::printTable(int currentTableIndex,string filename){

for(int queue=0;queue<tableSize;queue++){

if(hashTable[currentTableIndex][queue].isEmpty() != 1){

hashTable[currentTableIndex][queue].printQueue(currentTableIndex,queue,filename);

}

}

}

int main(int argc, char\* argv[]){

outputFile.open(argv[2]);

linkedListStack\* stack= new linkedListStack();

int longestStringLength=0;

ifstream inputFile;

inputFile.open(argv[1]);

if(inputFile.is\_open()){

string data;

while(inputFile >> data){

listNode\* node= new listNode(data);

stack->push(node);

if(data.length() > longestStringLength) longestStringLength= data.length();

}

}

stack->printStack(argv[2]);

cout<<longestStringLength;

inputFile.close();

radixSort\* radix= new radixSort();

int currentDigit=0;

int currentTable=0;

while(stack->isEmpty() != 1){

listNode\* node= stack->pop();

radix->padString(node,longestStringLength);

char val= radix->getVal(node, currentDigit, longestStringLength-1);

int hashIndex= radix->hashIndex(val);

radix->hashTable[currentTable][hashIndex].addTail(node);

}

radix->printTable(currentTable,argv[2]);

currentDigit++;

currentTable=1;

int previousTable=0;

int currentQueue=0;

cout<<endl;

while(currentDigit < longestStringLength){

while(currentQueue < radix->tableSize){

while(radix->hashTable[previousTable][currentQueue].isEmpty() != 1){

listNode\* node= radix->hashTable[previousTable][currentQueue].deleteHead();

char val= radix->getVal(node,currentDigit,longestStringLength-1);

int hashIndex= radix->hashIndex(val);

radix->hashTable[currentTable][hashIndex].addTail(node);

}

currentQueue++;

}

int temp= currentTable;

currentTable=previousTable;

previousTable=temp;

currentQueue=0;

currentDigit++;

}

outputFile<<endl<<endl;

radix->printTable(previousTable,argv[2]);

outputFile.close();

}

**Output**

//output1.txt

xiao

teng

edward

kun

kar

joe

changjie

anthony

lawrence

stefan

philip

ryan

angelo

maaz

ahmar

oliver

zuoping

yishi

casablanca

yat

adolfas

manuel

stacy

huan

jiaxin

sebastain

rafael

zxc

aabb

a

bbaa

ccaabb

Table[0][97]: bbaa, a, casablanca, dummy,

Table[0][98]: ccaabb, aabb, dummy,

Table[0][99]: zxc, dummy,

Table[0][100]: edward, dummy,

Table[0][101]: lawrence, changjie, joe, dummy,

Table[0][103]: zuoping, teng, dummy,

Table[0][105]: yishi, dummy,

Table[0][108]: rafael, manuel, dummy,

Table[0][110]: sebastain, jiaxin, huan, ryan, stefan, kun, dummy,

Table[0][111]: angelo, xiao, dummy,

Table[0][112]: philip, dummy,

Table[0][114]: oliver, ahmar, kar, dummy,

Table[0][115]: adolfas, dummy,

Table[0][116]: yat, dummy,

Table[0][121]: stacy, anthony, dummy,

Table[0][122]: maaz, dummy,

Table[1][32]: sebastain, lawrence, changjie, zuoping, anthony, adolfas, stefan, rafael, philip, oliver, manuel, jiaxin, edward, ccaabb, angelo, yishi, stacy, ahmar, xiao, teng, ryan, maaz, huan, bbaa, aabb, zxc, yat, kun, kar, joe, a,

Table[1][99]: casablanca,

**Input**

//input.txt

ccaabb bbaa a aabb zxc

rafael sebastain

jiaxin huan

stacy manuel adolfas

yat casablanca

yishi

zuoping oliver ahmar maaz

angelo ryan philip stefan lawrence

anthony

changjie joe kar kun

edward teng

xiao