Algoritmos y Estructuras de Datos

Parcial 3

1. (2.5 puntos) Implemente una versión simplificada del juego 2048. Este juego genera valores potencia de 2, y busca agregarlos hasta llegar al valor 2048. El juego que deben desarrollar contará con 3 pilas y una cola. Un generador de potencias de 2 aleatorias (2, 4, 8, 16 y 32) incluirá 10 valores en la cola. Luego, se asignarán los valores a las pilas de manera secuencial. Cuando una pila recibe un valor e inmediatamente en el tope se encuentra un valor similar, estos dos se suman y este valor reemplaza al valor en el tope. Esta operación de verificación debe hacerse recursivamente hasta que no se puedan agregar más valores.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 32 | 4 | 8 | 2 | 32 | 16 | 2 | 4 | 2 |

Se añaden los valores del 8 al 4

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 8 | 32 | 4 |

Se añade el 8 en la primera pila, se agrega con el 8 existente

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 16 | 32 | 4 |

Se añade los valores del 2 al 32

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  | 2 | 32 |
| 16 | 32 | 4 |

Se añade el 16 en la primera pila, se agrega con el 16 existente

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  | 2 | 32 |
| 32 | 32 | 4 |

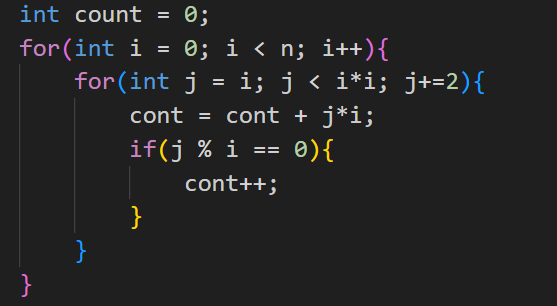
Se añade el 2 en la segunda pila, se agrega con el 2 existente

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  | 4 | 32 |
| 32 | 032 | 4 |

Se añade el valor 2

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  | 4 |
| 2 | 4 | 32 |
| 32 | 32 | 4 |

1. (1.8 puntos) Implemente una estructura tipo HashMap que cumpla con las siguientes características:
   1. Almacene cadenas de caracteres de longitud variable
   2. El map no recibe una tupla sino solo la cadena. La estructura cuenta con una función hash interna (que usted también debe implementar – no usar la vista en clase) que genere la llave que será usada en la tupla. Cuando se añade una cadena, se muestra por pantalla la llave.
   3. La estructura le permitirá determinar si un elemento está en la estructura, con base en la llave. No se permite la búsqueda por el contenido de la cadena.
   4. La estructura permite ver por pantalla el conjunto completo de llaves de las tuplas almacenadas.
   5. Esta estructura permite la función Unión con otro objeto HashMap.
2. (0.7 puntos) Calcule el T(n) y el O(n) del siguiente bloque de código:



Plantillas

1.

#include<iostream>

#include<math.h>

using namespace std;

template<typename T>

class Nodo{

private:

T dato;

Nodo\* next;

public:

Nodo(T d){

dato = d;

next = NULL;

}

~Nodo(){

}

T get\_dato(){

return dato;

}

void set\_dato(T d){

dato = d;

}

Nodo\* get\_next(){

return next;

}

void set\_next(Nodo\* n){

next = n;

}

};

template<typename T>

class Lista{

private:

Nodo<T>\* ptr;

int size;

public:

Lista(){

ptr = NULL;

size = 0;

}

~Lista(){

Nodo<T>\* temp = ptr;

if(ptr != NULL){

Nodo<T>\* temp\_next = ptr->get\_next();

while(temp\_next != NULL){

delete temp;

temp = temp\_next;

temp\_next = temp->get\_next();

}

delete temp; //Borrar el último nodo

}

}

void add(T d){

Nodo<T>\* nodo = new Nodo<T>(d);

if(ptr == NULL){//La lista está vacía

ptr = nodo;

}else{//La lista no está vacía

Nodo<T>\* temp = ptr;

while(temp->get\_next() != NULL){

temp = temp->get\_next();

}

temp->set\_next(nodo);

}

size++;

}

void add\_front(T d){

/\*

Inserte su código aquí

\*/

}

T pop(){

/\*

Inserte su código aquí

\*/

}

T peek(){

/\*

Inserte su código aquí

\*/

}

void insert(T d, int i){

if(i<=size && i>=0 && ptr != NULL){

Nodo<T>\* nodo = new Nodo<T>(d);

if(i == 0){

nodo->set\_next(ptr);

ptr = nodo;

}else{

int j = 0;

Nodo<T>\* temp = ptr;

while(j<i-1){

temp = temp->get\_next();

j++;

}

nodo->set\_next(temp->get\_next());

temp->set\_next(nodo);

}

size++;

}else{//Si el índice es incorrecto o la lista está vacía, se añade al final

add(d);

}

}

void print(){

if(ptr == NULL){//La lista está vacía

cout<<"La lista está vacía"<<endl;

}else{//La lista no está vacía

Nodo<T>\* temp = ptr;

while(temp != NULL){

//cout<<temp->get\_dato()<<","<<temp->get\_next()<<"\t";

cout<<temp->get\_dato()<<"\t";

temp = temp->get\_next();

}

cout<<endl;

}

}

Nodo<T>\* get(int i){

if(i>=0 && i<size){

int j=0;

Nodo<T>\* temp = ptr;

while(j<i){

temp = temp->get\_next();

j++;

}

return temp;

}

return NULL;

}

};

template<typename T>

class Pila{

private:

Lista<T> pila;

public:

Pila(){

}

~Pila(){

}

void push(T d){

/\*

Inserte su código aquí

\*/

}

T pop(){

return pila.pop();

}

T peek(){

return pila.peek();

}

void print(){

pila.print();

}

};

template<typename T>

class Cola{

private:

Lista<T> cola;

public:

Cola(){

}

~Cola(){

}

void push(T d){

/\*

Inserte su código aquí

\*/

}

T pop(){

return cola.pop();

}

T peek(){

return cola.peek();

}

void print(){

cola.print();

}

};

class Pila2048{

private:

Pila<int> pila;

public:

Pila2048(){

}

~Pila2048(){

}

void push(int d){

/\*

Inserte su código aquí. Recuerde que esta función debe ser recursiva.

\*/

}

void print(){

pila.print();

}

};

int genpot2(int i){

return pow(2,i);

}

int main()

{

srand(5678);

int num\_pilas = 3;

int num\_valores = 10;

/\*

Inserte su código aquí para instanciar un vector que contenga 3 objetos Pila2048<int>.

\*/

Cola<int> cola;

int temp;

cout<<"Los valores generados son:"<<endl;

/\*

Inserte su código aquí para agregar valores a la cola. Puede generar los valores con esta función: temp = genpot2(1+rand()%5); para generar potencias de 2.

\*/

/\*

Inserte su código aquí para agregar valores a las pilas. Puede usar esta expresión i%num\_pilas para asignar valores a las pilas de manera consecutiva, a partir del índice de los valores

\*/

cout<<endl;

cout<<endl;

cout<<"Las pilas, en su estado final, son:"<<endl;

for(int i=0; i<num\_pilas;i++){

pilas[i].print();

cout<<endl;

}

delete[] pilas;

return 0;

}

Salida del programa:

Los valores generados son:

2 32 16 2 2 32 32 16 16 4

Las pilas, en su estado final, son:

4 32 4

16 2 32

16 32 16

\*\* Process exited - Return Code: 0 \*\*

2.

#include<iostream>

using namespace std;

class Key{

string llave;

public:

Key(string k){

llave = k;

}

int compare\_to(string k){

return llave.compare(k);

}

string get\_key(){

return llave;

}

};

template <typename T2>

class Tupla{

Key\* key;

T2 dato;

public:

Tupla(){

}

~Tupla(){

}

Tupla(string k, T2 d){

key = new Key(k);

dato = d;

}

T2 getDato(){

return dato;

}

Key get\_key(){

return \*key;

}

bool compare\_to(Tupla t){

return key->compare\_to(t.get\_key());

}

std::string to\_string() {

return "("+key->get\_key() + ":" + dato+")";

}

/\*friend std::ostream& operator<<(std::ostream& os, Tupla& b) {

return os << b.to\_string();

}\*/

friend std::ostream& operator<<(std::ostream& os, Tupla b) {

return os << b.to\_string();

}

friend std::ostream& operator<<(std::ostream& os, Tupla\* b) {

return os << b->to\_string();

}

Tupla& operator=(const Tupla& f) {

key = f.key;

dato = f.dato;

return \*this;

}

};

template<typename T>

class Nodo{

private:

T dato;

Nodo\* next;

public:

Nodo(T d){

dato = d;

next = NULL;

}

~Nodo(){

}

T get\_dato(){

return dato;

}

void set\_dato(T d){

dato = d;

}

Nodo\* get\_next(){

return next;

}

void set\_next(Nodo\* n){

next = n;

}

};

template<typename T>

class Lista{

private:

Nodo<T>\* ptr;

int size;

public:

Lista(){

ptr = NULL;

size = 0;

}

~Lista(){

Nodo<T>\* temp = ptr;

if(ptr != NULL){

Nodo<T>\* temp\_next = ptr->get\_next();

while(temp\_next != NULL){

delete temp;

temp = temp\_next;

temp\_next = temp->get\_next();

}

delete temp; //Borrar el último nodo

}

}

void add(T d){

Nodo<T>\* nodo = new Nodo<T>(d);

if(ptr == NULL){//La lista está vacía

ptr = nodo;

}else{//La lista no está vacía

Nodo<T>\* temp = ptr;

while(temp->get\_next() != NULL){

temp = temp->get\_next();

}

temp->set\_next(nodo);

}

size++;

}

void insert(T d, int i){

if(i<=size && i>=0 && ptr != NULL){

Nodo<T>\* nodo = new Nodo<T>(d);

if(i == 0){

nodo->set\_next(ptr);

ptr = nodo;

}else{

int j = 0;

Nodo<T>\* temp = ptr;

while(j<i-1){

temp = temp->get\_next();

j++;

}

nodo->set\_next(temp->get\_next());

temp->set\_next(nodo);

}

size++;

}else{//Si el índice es incorrecto o la lista está vacía, se añade al final

add(d);

}

}

void print(){

if(ptr == NULL){//La lista está vacía

cout<<"La lista está vacía"<<endl;

}else{//La lista no está vacía

Nodo<T>\* temp = ptr;

while(temp != NULL){

//cout<<temp->get\_dato()<<","<<temp->get\_next()<<"\t";

cout<<temp->get\_dato()<<"\t";

temp = temp->get\_next();

}

cout<<endl;

}

}

int get\_size(){

return size;

}

Nodo<T>\* get(int i){

if(i>=0 && i<size){

int j=0;

Nodo<T>\* temp = ptr;

while(j<i){

temp = temp->get\_next();

j++;

}

return temp;

}

return NULL;

}

};

template <typename T2>

class Map{

private:

Lista<Tupla<T2>> tuplas;

public:

Map(){ }

~Map(){ }

void add(Tupla<T2> t){

tuplas.add(t);

}

bool has(Key k){

/\*

Ingrese su código aquí

\*/

}

Tupla<T2> get(Key k){

/\*

Ingrese su código aquí

\*/

return Tupla<T2>("",""); //Retorne una tupla vacía, en caso de no encontrar el elemento buscado

}

void print(){

tuplas.print();

}

};

template <typename T2>

class Hashmap{

private:

Map<T2> mapa;

public:

Hashmap(){ }

~Hashmap(){ }

string gen\_hash(T2 s){

long code;

/\*

Ingrese su código aquí

\*/

return to\_string(code); //Garantize que el hash se almacena como string

}

void add(T2 t){

/\*

Ingrese su código aquí

\*/

}

bool has(T2 k){

/\*

Ingrese su código aquí

\*/

}

Tupla<T2> get(T2 k){

/\*

Ingrese su código aquí

\*/

}

void print(){

mapa.print();

}

};

int main()

{

Hashmap<string> m;

for(int i=0; i<10; i++){

m.add("Hola"+to\_string(i));

}

m.print();

if(m.has("Hola9"))

cout<<"si está1"<<endl;

Tupla<string> temp = m.get("Hola3");

if(temp.get\_key().get\_key() != "")

cout<<"si está: "<<temp<<endl;

return 0;

}

Salida del programa:

(761216:Hola0) (485408:Hola1) (209600:Hola2) (933792:Hola3) (657984:Hola4) (382176:Hola5) (106368:Hola6) (830560:Hola7) (554752:Hola8) (278944:Hola9)

si está1

si está: (933792:Hola3)

Nota: el valor de las llaves puede diferir dependiendo del método de hash que diseñe.