Estructures De Dades

Pràctica 4.Heaps i hashing

Nombre	NIUB
Ling Zhu	18088210
Jonny Muñez Cano	18088081

Ejercicio1

Problema planteado: Implementar el TAD MinHeap correspondiente con una representación en vector. Cada nodo del heap está representada por un apellido(que es la key) y un contacto(que son los valores).

Decisiones tomadas

1) Hemos decidido utilizar Contacte y NodeTree de la práctica 3. En este ejercicio hemos cambiado la clase NodeTree de la práctica anterior: eliminar los atributos privados que tenemos definidos(punteros que apunta al siguiente nodo o el anterior) ya que en el heap utilizamos un vector por lo tanto no es necesario utilizar punteros.

2) En la clase MinHeap, hemos declarado y implementado los atributos y las operaciones básicas:

   Atributos privados:

   ➔ vector <NodeTree<E>*> heap: el vector que guarda los nodos de un heap.

   Operaciones:

   - Constructor de MinHeap: MinHeap(). Construye el heap vació.
   - Destructor de MinHeap: ~MinHeap(); destructor de los nodos del heap.

   Métodos consultores:

   - int size(): devuelve el número de los nodos que hay en el heap.
   - bool empty(): devuelve true si el heap esta vació y false si el heap no esta vació.
   - const string min():devuelve la clave mínima del heap
   - int profunditat(): devuelve la profundidad del heap
   - const E& minValues(): devuelve los valores de la clave mínima del heap
   - void printHeap(): recorre todo el heap y imprime por la pantalla todo el heap
   - Contacte getContacte(const string cognom): obtener el contacto por el apellido pasado por el parámetro.
   - int getLeftChild(int parent): este método sirve para obtener el hijo izquierdo
   - int getRightChild(int parent):este método sirve para obtener el hijo derecho
   - int getParent(int child):este método sirve para obtener el padre del nodo child

Métodos modificadores:

- void insert(string key, Contacte c):Añade un nuevo elemento al heap. esta función recibe “clau” y “contacte”. Primero se compruebe si el heap está vació. Si está vació hacemos un push_back para añadir un contacto con la clave. Y si el heap hacemos también el push_back del contacto y llamamos a la función unHeap() para re-ordenarlo.

- Void removeMin(): Elimina el contacto con la clave mínima del heap. Primero se compruebe si el heap está vació con un if-else. Si está vació el heap(if), lanza una excepción. Y por el contrario(else), se crea un atributo nuevo y lo asignamos a heap.size()-1 y luego llamamos a la frunció swap(child,0) para intercambiar los dos contactos. Hemos creado un while dentro de la else para reestructurar el heap en función de la clave de cada contacto.

- Void swap(int child, int parent): este método intercambia el contenido de los dos nodos. Se crea una variable auxiliar de tipo NodeTree.

- Void upHeap(): este método sirve como soporte para una vez añadido un elemento reestructure el heap según los criterios. Hemos creado un while con tres condiciones: si la clave del hijo es más pequeño que su padre y child >= 0 && parent >= 0. En este caso, hacemos un swap para intercambiar los dos nodos.

Coste computacional teórico de las funciones del TAD BinarySearchTree

- MinHeap coste O(1) – la instrucción se ejecuta una vez y no depende del tamaño.
- ~MinHeap coste O(n) – recorre todo el heap para destruir todos los nodos-.
- size() coste O(n) – se tiene que recorrer todo el heap
- empty() coste O(1) – la instrucción se ejecuta una vez y no depende del tamaño.
- RemoveMin() coste O(log n) : en el peor de los casos intercambia padre-hijo hasta el último nivel.
- PrintHeap() coste O(n): imprime todos los valores del heap
- min(): coste O(1): la instrucción se ejecuta una vez y no depende del tamaño
- insert():O(logn)
- minValues():coste O(1): la instrucción se ejecuta una vez y no depende del tamaño
- swap():costeO(1)
- upHeap():O(logn)
- getLeftChild(int parent)/ getRightChild(int parent)/ getParent(int child): O(1)

Ejercicio2

Problema planteado: Implementar un buscador HeapContactFinder

Decisiones tomadas:

1) Hemos podido reutilizar las clase NodeTree, Contacte y MinHeap del ejercicio 1.

2) Hemos añadido en la clase MinHeap un método que devuelve un booleano: bool getContacteT(const string cognom). Devuelve true si lo ha encontrado el contacto según el apellido pasado por el parámetro.

3) No hemos hecho HeapContactFinder con template ya que este ejercicio nos pide cosas relacionadas con los contactos. En la clase HeapContactFinder hemos declarado y implementado el atributo y los métodos siguientes:

   Atributos privados:

   - MinHeap<Contacte> heap: un minHeap de tipo contactos

   Métodos: estos métodos son iguales en el ejercicio 2 de la práctica 3.

   - HeapContactFinder(): constructor de HeapContactFinder
   - void appendContactList(const string filename): Aquest metode rep el nom d'un fitxer i emmagatzema el seu contingut dins de l'arbre.
   - void insertContact(const string nom, const string cognom, const string adreca, const string ciutat, string telefon, const string email);
   - string showContact(string cognom);
   - Contacte findContact(string cognom);
   - bool findContactBool(string cognom);
   - void viewIndex();
   - int height();

Ejercicio3 y ejercicio4

Problema planteado: Implementar el TAD HashMap (una tabla abierta) y el HashMapContactFinder (buscador de contactos).

1)No hemos cambiado nada la clase Contacte.

2)En la clase LinkedHashEntry, se declaran y se implementan los atributos y operaciones básicas:

-Atributos privados:

- string key;
- Contacte values;
- LinkedHashEntry* next;

-Métodos:

- LinkedHashEntry(string key,E values): Constructor de LinkedHashEntry. Inicializamos los atributos key y values. Y igualamos next a nullptr.
- const string getKey()const: getter de Key
- const E getValues() const: getter de Values
- void setValues(const E& newValues): setter de Values
- LinkedHashEntry* getNext(): getter de Next
- void setNext(LinkedHashEntry* next): setter del atributo next

3) En la clase HashMap, se declaran y se implementan los atributos y operaciones básicas:

-Atributos privados:

- static constexpr int MIDA_MAX = 233 : tamaño del array
- LinkedHashEntry<E>* hash[MIDA_MAX]: tabla hash
- int numColisions: numero de colisiones
- int numElementsMaxCelda: numero de elementos que hay en la celda
- int celdasOcupades: celdas ocupadas

-Métodos:

- HashMap();/~HashMap(); constructor y destructor de HashMap
- int getHashCode(string key): dado un clave devuelve el indice correspondiente
- void put(string key, Contacte c): añade el contacto a la tabla de hash
- bool get(string key): compruebe si existe una clave
- Contacte getContacte(): devuelce un contacto
- int getNumColisions():devuelve el numero de colisiones que ha producido
- int getNumCeldas(): tamaño de la tabla

- int getNumElementsMaxCelda(): devuelve el máximo de elementos de todas las celdas
- int getCeldasOcupades(): devuelve un entero de celdas ocupadas
- void printHashMap(): imprime por la pantalla la tabla hash

4) En la clase HashMapContactFinder, se declaran y se implementan los atributos y operaciones básicas:

-Atributos privados:

- HashMap<Contacte> tablaHash; la tabla de hash

-Métodos:

- HashMapContactFinder() /~HashMapContactFinder(): constructor y destructor de HashMapContactFinder
- void appendContactList(const string filename):guarda los contactos en la tabla hash
- void insertContact(const string nom, const string cognom, const string adreca, const string ciutat, string telefon, const string email): añade un contacto
- string showContact(): mostrar contacto
- Contacte findContact(string cognom): buscar contacto
- bool findContactBool(string cognom);
- void viewIndex();
- int height();
- int getNumColisions():devuelve el numero de colisiones que ha producido
- int getNumCeldas(): tamaño de la tabla
- int getNumElementsMaxCelda(): devuelve el máximo de elementos de todas las celdas
- int getCeldasOcupades(): devuelve un entero de celdas ocupadas

<u>Coste computacional de las operaciones de HashMap</u>

-constructor y destructor de HashMap coste O(n)- recorre toda la tabla
-getHash: depende del tamaño del string
-put():mejor caso O(1)-celda vaciá-, si hay colisiones depende del tamaño de la lista asociada
-get(): mejor caso O(1) -celda vaciá-, si hay colisiones depende del tamaño de la lista asociada
-Funciones get O(1): la instrucción se ejecuta una vez y no depende del tamaño

Problemas obtenidas durante la realización de los dos ejercicios

- Hemos tenido en hora de encontrar el tamaño más apropiado de la tabla de hash. Ya que cuando ponemos una tabla de menos tamaño para que el porcentaje de celda vaciás sean menos, el numero de las colisiones se aumenta.

- Para el ejercicio 3 y 4 solo hemos entregado un proyecto ya que los dos ejercicios están relacionados por lo tanto nos conviene hacer solo un proyecto.

- También hemos tenido problema con el contador de contactos leídos de un fichero.

Ejercicio5

Problema planteado: Haced una evaluación del rendimiento experimentado de las dos implementaciones anteriores(heap y hash), y razonad las diferencias: contad los tiempo de acceso para dos textos de diferentes tamaño; contad el tiempo de generación de la estructura para dos textos de diferente tamaño.

	Acción	Texto pequeño	Texto grande
heap	Inserción	1.2564ms	4.3667ms
hash	Inserción	1.7345ms	5.8483ms
heap	Búsqueda	0.5943ms	1.2758ms
hash	Búsqueda	0.2012	0.2608ms

	Inserción	Búsqueda
heap	O(logn)	O(logn)
hash	O(1)	O(1)