實作

#include"PQ.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void createPQ(PQ\_t \*pq, H\_class pqClass, int elementSize, int maxSize, int (\*compare)(void\* elementA, void \*elementB))

{

pq->heap.numElements = 0;

pq->heap.elements = (void \*)malloc(elementSize\*maxSize);

pq->elementSize = elementSize;

pq->maxSize = maxSize;

pq->compare = compare;

pq->pqClass = pqClass;

}//在此時初始化結構中的各個值, 也把compare傳入, 因為elements是Void \*,所以要在此時給他足夠量的記憶體空間來做之後的運算, 在這邊我是把maxSize當成元素個數來做

void Enqueue(PQ\_t \*pq, void \* elementA/\*, void (\*printindex)(void\* element), void (\*replace)(void\* element, void\* elemenA)\*/)

{

if(IsFull(pq))

{

puts("Heap is full");

}

else

{

pq->heap.numElements++;

void \*tmp = pq->heap.elements+(pq->heap.numElements-1)\*pq->elementSize;

memcpy(tmp,elementA,pq->elementSize);

ReheapUp(pq, 0, pq->heap.numElements-1);

}

}//如果滿了就不給插入, 在做儲存elements因為它是void \*的形式, 所以要直接從記憶體來進行配置

int IsEmpty(PQ\_t \*pq)

{

if(pq->heap.numElements == 0)

return 1;

return 0;

}//沒元素return 1

int IsFull(PQ\_t \*pq)

{

if(pq->heap.numElements == pq->maxSize)

return 1;

return 0;

}//滿了return 1

void \* Dequeue(PQ\_t \*pq)

{

if(IsEmpty(pq))

{

puts("No data");

}

else

{

swap(pq->heap.elements, pq->heap.elements+(pq->heap.numElements-1)\*pq->elementSize, pq->elementSize);

pq->heap.numElements--;

ReheapDown(pq, 0, pq->heap.numElements-1);

}

}//同Enqueue

static void ReheapDown(PQ\_t \* pq, int root, int bottom)

{

int child, left, right;

left = root\*2+1;

right = root\*2+2;

if(left<=bottom)

{

int result;

if(left == bottom)

child = left;

else

{

result = pq->compare(pq->heap.elements+right\*pq->elementSize, pq->heap.elements+left\*pq->elementSize);

if(pq->pqClass == MINHEAP)

{

result\*=-1;

}

if(result == 1)

child = right;

else

child = left;

}

result = pq->compare(pq->heap.elements+child\*pq->elementSize, pq->heap.elements+root\*pq->elementSize);

if(pq->pqClass == MINHEAP)

{

result\*=-1;

}

if(result == 1)

{

swap(pq->heap.elements+child\*pq->elementSize, pq->heap.elements+root\*pq->elementSize, pq->elementSize);

ReheapDown(pq, child, bottom);

}

}

}//基本跟老師投影片一樣, 除了老師在做陣列的儲存時要改用記憶體來配置, 另外MINHEAP和MAXHEAP的差別在一個比大的, 另一個比小的, 所以當原本a比b大回傳a時, 改成回傳b, 所以只要把result乘以-1就好

static void ReheapUp(PQ\_t\* pq, int root, int bottom)

{

int parent;

if(bottom > root)

{

parent = (bottom-1)/2;

if(pq->compare(pq->heap.elements+bottom\*(pq->elementSize), pq->heap.elements+(parent\*pq->elementSize)) == 1)

{

swap(pq->heap.elements+bottom\*pq->elementSize, pq->heap.elements+parent\*pq->elementSize, pq->elementSize);

ReheapUp(pq, root, parent);

}

}

}//基本跟老師投影片一樣, 除了老師在做陣列的儲存時要改用記憶體來配置, 另外MINHEAP和MAXHEAP的差別在一個比大的, 另一個比小的, 所以當原本a比b大回傳a時, 改成回傳b, 所以只要把result乘以-1就好

static void swap(void \*elementA, void\* elementB, int elementSize)//memcpy swap

{

void \*tmp = (void \*)malloc(sizeof(elementSize));

memcpy(tmp, elementA, elementSize);

memcpy(elementA, elementB, elementSize);

memcpy(elementB, tmp, elementSize);

free(tmp);

}//建立第三個變數來達到AB彼此互換的效果

MAIN

#include <stdio.h>

#include <stdlib.h>

#include"PQ.h"

#include <string.h>

typedef struct myElement {

char ID[10];

int math;

int eng;

} student\_t;

int compareMath(void \*elementA, void \*elementB) {

int mathA = ((student\_t \*)elementA)->math;

int mathB = ((student\_t \*)elementB)->math;

if(mathA>mathB) {

return 1;

}else if(mathA<mathB){

return -1;

}

return 0;

}

void printindex(void \*index)

{

student\_t \*temp = (student\_t \*)index;

printf("ID=%s,math=%d, eng=%d\n", temp->ID, temp->math, temp->eng);

}

void print(PQ\_t \*pq) {

student\_t \*temp;

for (int i=0; i<pq->heap.numElements;i++){

temp = (student\_t \*)(pq->heap.elements+i\*sizeof(student\_t));

//printf("-------------%d--------------\n",pq->heap.elements+i\*sizeof(student\_t));

printf("index=%d, ID=%s,math=%d, eng=%d\n",i,temp->ID, temp->math, temp->eng);

}

}

int main() {

student\_t node[6]={{"C10308001", 70, 100},{"B20406001", 60, 90},{"D10306001", 80, 95},{"A20407001", 65, 90},{"D20506001", 10, 70},{"A10406001", 90, 90}};

PQ\_t maxPQ;

createPQ(&maxPQ, MAXHEAP, sizeof(student\_t), 6, compareMath);

for(int i=0;i<6;i++)

Enqueue(&maxPQ, &node[i]/\*, printindex,replace\*/);

print(&maxPQ);

Dequeue(&maxPQ);

print(&maxPQ);

}