Heap\_anh\_tung

class Heap {

protected:

T\* elements;

int capacity;

int count;

public:

Heap()

{

this->capacity = 10;

this->count = 0;

this->elements = new T[capacity];

}

~Heap()

{

delete[]elements;

}

void push(T item);

int getItem(T item);

void remove(T item);

void clear();

void printHeap()

{

cout << "Max Heap [ ";

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

cout << elements[i] << " ";

cout << "]\n";

}

private:

void ensureCapacity(int minCapacity);

void reheapUp(int position);

void reheapDown(int position);

};

void reheapDown(int maxHeap[], int numberOfElements, int index)

{

if(index<0 || index>=numberOfElements)

{

return;

}

int i=index,child1=2\*i+1,child2=2\*i+2;

while(i<numberOfElements)

{

if(child2<numberOfElements &&maxHeap[i]<maxHeap[child2])

{

int temp=maxHeap[child2] ;

maxHeap[child2] =maxHeap[i];

maxHeap[i]=temp;

i=child2;

child1=2\*i+1;

child2=2\*i+2;

if(child1>=numberOfElements)

break;

}

else

if(child1<numberOfElements && maxHeap[i]<maxHeap[child1])

{

int temp=maxHeap[child1] ;

maxHeap[child1] =maxHeap[i];

maxHeap[i]=temp;

i=child1;

child1=2\*i+1;

child2=2\*i+2;

if(child1>=numberOfElements)

break;

}

else

{

break;

}

}

}

void reheapUp(int maxHeap[], int numberOfElements, int index)

{

if(index<0 || index>=numberOfElements)

{

return;

}

int i=index,parent=(i-1)/2;

while (i != 0 && maxHeap[parent] < maxHeap[i])

{

int temp=maxHeap[parent] ;

maxHeap[parent] =maxHeap[i];

maxHeap[i]=temp;

i = parent;

parent=(i-1)/2;

}

}

int minWaitingTime(int n, int arrvalTime[], int completeTime[]) {

vector<pair<int, int>> v(n);

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

v[i].first=arrvalTime[i];

v[i].second=completeTime[i];

}

sort(v.begin(), v.end()); //sap xep theo tg toi

int sum = 0;

vector<pair<int,int>> q;

int t = v[0].first;

int it = 0;

while (it < n || q.size()) {

while (it < n && v[it].first <= t) {

pair<int,int> element;

element.first=v[it].second;

element.second=it;

q.push\_back(element);

++it;

}

if (q.empty()) {

t = v[it].first;

} else {

make\_heap(q.begin(),q.end(),std::greater<>{});

int i = q.begin()->second;

q.erase(q.begin());

t += v[i].second;

sum += t-v[i].first;

}

}

return sum;

}

static void heapify(T arr[], int n, int i)

{

int largest = i; // Initialize largest as root

int l = 2 \* i + 1; // left = 2\*i + 1

int r = 2 \* i + 2; // right = 2\*i + 2

// If left child is larger than root

if (l < n && arr[l] > arr[largest])

largest = l;

// If right child is larger than largest so far

if (r < n && arr[r] > arr[largest])

largest = r;

// If largest is not root

if (largest != i) {

swap(arr[i], arr[largest]);

// Recursively heapify the affected sub-tree

heapify(arr, n, largest);

}

}

static void heapSort(T\* start, T\* end){

int size = end - start;

// Build heap (rearrange array)

for (int i = size / 2 - 1; i >= 0; i--)

heapify(start, size, i);

// One by one extract an element from heap

for (int i = size - 1; i > 0; i--) {

// Move current root to end

swap(start[0], start[i]);

// call max heapify on the reduced heap

heapify(start, i, 0);

}

Sorting<T>::printArray(start,end);

}