Priority Queue

#include <stdio.h>

#include <stdlib.h>

typedef struct

{

  int key;

  int value;

} Element;

typedef struct

{

  Element \*array;

  int capacity;

  int size;

} MinPriorityQueue;

MinPriorityQueue \*createMinPriorityQueue(int capacity)

{

  MinPriorityQueue \*queue = (MinPriorityQueue \*)malloc(sizeof(MinPriorityQueue));

  queue->array = (Element \*)malloc(sizeof(Element) \* capacity);

  queue->capacity = capacity;

  queue->size = 0;

  return queue;

}

void swap(Element \*a, Element \*b)

{

  Element temp = \*a;

  \*a = \*b;

  \*b = temp;

}

void heapifyUp(MinPriorityQueue \*queue, int index)

{

  while (index > 0)

  {

    int parent = (index - 1) / 2;

    if (queue->array[index].key < queue->array[parent].key)

    {

      swap(&queue->array[index], &queue->array[parent]);

      index = parent;

    }

    else

    {

      break;

    }

  }

}

void insert(MinPriorityQueue \*queue, int key, int value)

{

  if (queue->size < queue->capacity)

  {

    Element element = {key, value};

    queue->array[queue->size] = element;

    heapifyUp(queue, queue->size);

    queue->size++;

  }

  else

  {

    printf("Priority queue is full!\n");

  }

}

void heapifyDown(MinPriorityQueue \*queue, int index)

{

  int leftChild = 2 \* index + 1;

  int rightChild = 2 \* index + 2;

  int smallest = index;

  if (leftChild < queue->size && queue->array[leftChild].key < queue->array[smallest].key)

  {

    smallest = leftChild;

  }

  if (rightChild < queue->size && queue->array[rightChild].key < queue->array[smallest].key)

  {

    smallest = rightChild;

  }

  if (smallest != index)

  {

    swap(&queue->array[index], &queue->array[smallest]);

    heapifyDown(queue, smallest);

  }

}

Element extractMin(MinPriorityQueue \*queue)

{

  if (queue->size > 0)

  {

    Element min = queue->array[0];

    queue->array[0] = queue->array[queue->size - 1];

    queue->size--;

    heapifyDown(queue, 0);

    return min;

  }

  else

  {

    printf("Priority queue is empty!\n");

    Element dummy = {-1, -1}; *// Return a dummy element*

    return dummy;

  }

}

void decreaseKey(MinPriorityQueue \*queue, int value, int newKey)

{

  for (int i = 0; i < queue->size; i++)

  {

    if (queue->array[i].value == value)

    {

      queue->array[i].key = newKey;

      heapifyUp(queue, i);

      break;

    }

  }

}

void printPriorityQueue(MinPriorityQueue \*queue)

{

  printf("Priority Queue: ");

  for (int i = 0; i < queue->size; i++)

  {

    printf("(%d, %d) ", queue->array[i].key, queue->array[i].value);

  }

  printf("\n");

}

void destroyMinPriorityQueue(MinPriorityQueue \*queue)

{

  free(queue->array);

  free(queue);

}

int main()

{

  MinPriorityQueue \*queue = createMinPriorityQueue(10);

  insert(queue, 4, 100);

  insert(queue, 2, 200);

  insert(queue, 7, 300);

  insert(queue, 1, 400);

  printPriorityQueue(queue);

  Element min = extractMin(queue);

  printf("Min element: (%d, %d)\n", min.key, min.value);

  decreaseKey(queue, 300, 3);

  printPriorityQueue(queue);

  destroyMinPriorityQueue(queue);

  return 0;

}

Heap Sort

#include <stdio.h>

void swap(int \*a, int \*b)

{

  int temp = \*a;

  \*a = \*b;

  \*b = temp;

}

void heapify(int arr*[]*, int n, int i)

{

  int largest = i;

  int left = 2 \* i + 1;

  int right = 2 \* i + 2;

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

    largest = left;

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

    largest = right;

  if (largest != i)

  {

    swap(&arr[i], &arr[largest]);

    heapify(arr, n, largest);

  }

}

void heapSort(int arr*[]*, int n)

{

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

    heapify(arr, n, i);

  for (int i = n - 1; i > 0; i--)

  {

    swap(&arr[0], &arr[i]);

    heapify(arr, i, 0);

  }

}

void printArray(int arr*[]*, int n)

{

  for (int i = 0; i < n; i++)

    printf("%d ", arr[i]);

  printf("\n");

}

int main()

{

  int arr*[]* = {12, 11, 13, 5, 6, 7};

  int n = sizeof(arr) / sizeof(arr[0]);

  printf("Original array: ");

  printArray(arr, n);

  heapSort(arr, n);

  printf("Sorted array: ");

  printArray(arr, n);

  return 0;

}