Aula 05: 07/03/2019

Tópico

- Filas priorizadas (PQ = *Priority Queues*)
- Filas priorizadas com itens mutáveis

Filas priorizadas

Leitura: Filas priorizadas, Paulo feofiloff, Priority queues, S&W

Vídeo: Priority queues, Sedgewick

Uma **fila priorizada** (ou **fila com prioridades**) é um ADT (*abstract data type*) que generaliza tanto a fila quanto a pilha.

PQ de máximo

Uma fila priorizada decrescente ou **PQ de máximo** é um ADT que manipula um conjunto de itens por meio de duas operações fundamentais:

- inserção de um novo item no conjunto e
- remoção de um item máximo.

Isso significa que uma fila priorizada manipula itens comparáveis.

API

`public class MaxPQ<Item extends Comparable<Item>>`

`public class`	`MaxPQ`	
	`MaxPQ()`	cria uma PQ de máximo
	`MaxPQ(int cap)`	cria uma PQ de máximo com capacidade `cap`
	`MaxPQ(Item[] a)`	cria uma PQ de máximo com os itens que estão em `a[]`
`void`	`insert(Item v)`	insere o item `v` nesta PQ
`Item`	`max()`	devolve um item máximo deste PQ
`Item`	`delMax()`	remove e devolve um item máximo desta PQ
`boolean`	`isEmpty()`	esta PQ está vazia?
`int`	`size()`	número de itens desta PQ

PQ com itens mutáveis

Não sei se *PQ com itens mutáveis* é um bom nome para o que S&W chamam de *index priority queues*.

Em algumas aplicações é razoável permitirmos que o cliente altere a prioridade de um item que já esta na fila.

Uma maneira de lidar com isso é associar um único índice a cada item. Já comentamos essa estratégia quando

tratamos de *union-find*.

API

`public class IndexMinPQ<Item extends Comparable<Item>>`

public class	IndexMinPQ	
	`IndexMinPQ(int maxN)`	
`void`	`insert(int k, Item item)`	insere um item associado a `k`
`void`	`change(int k, Item item)`	muda o item associado com k para item
`boolean`	`contains(int k)`	k está associado a algum item
`void`	`delete(int k)`	remove k e o item associado
`Item`	`min()`	retorna o menor item
`int`	`minIndex()`	retorna o índice do menor item
`int`	`delMin()`	retorna o menor item e retorna o seu índice
`boolean`	`isEmpty()`	a fila está vazia?
`int`	`size()`	número de itens

Cliente

```
/******************************************************************************
 *  Compilation:  javac Multiway.java
 *  Execution:    java Multiway input1.txt input2.txt input3.txt ...
 *  Dependencies: IndexMinPQ.java In.java StdOut.java
 *  Data files:   http://algs4.cs.princeton.edu/24pq/m1.txt
 *                http://algs4.cs.princeton.edu/24pq/m2.txt
 *                http://algs4.cs.princeton.edu/24pq/m3.txt
 *
 *  Merges together the sorted input stream given as command-line arguments
 *  into a single sorted output stream on standard output.
 *
 *  % more m1.txt
 *  A B C F G I I Z
 *
 *  % more m2.txt
 *  B D H P Q Q
 *
 *  % more m3.txt
 *  A B E F J N
 *
 *  % java Multiway m1.txt m2.txt m3.txt
 *  A A B B B C D E F F G H I I J N P Q Q Z
 *
 ******************************************************************************/

import edu.princeton.cs.algs4.IndexMinPQ;
import edu.princeton.cs.algs4.StdOut;
```

```
import edu.princeton.cs.algs4.In;

/**
 *  The {@code Multiway} class provides a client for reading in several
 *  sorted text files and merging them together into a single sorted
 *  text stream.
 *  This implementation uses a {@link IndexMinPQ} to perform the multiway
 *  merge.
 *
 *  For additional documentation, see <a href="http://algs4.cs.princeton.edu/24pq">Section 2.4
 *  of <i>Algorithms, 4th Edition</i> by Robert Sedgewick and Kevin Wayne.
 *
 *  @author Robert Sedgewick
 *  @author Kevin Wayne
 */

public class Multiway {

    // This class should not be instantiated.
    private Multiway() { }

    // merge together the sorted input streams and write the sorted result to standard output
    private static void merge(In[] streams) {
        int n = streams.length;
        IndexMinPQ<String> pq = new IndexMinPQ<String>(n);
        for (int i = 0; i < n; i++)
            if (!streams[i].isEmpty())
                pq.insert(i, streams[i].readString());

        // Extract and print min and read next from its stream.
        while (!pq.isEmpty()) {
            StdOut.print(pq.minKey() + " ");
            int i = pq.delMin();
            if (!streams[i].isEmpty())
                pq.insert(i, streams[i].readString());
        }
        StdOut.println();
    }


    /**
     *  Reads sorted text files specified as command-line arguments;
     *  merges them together into a sorted output; and writes
     *  the results to standard output.
     *  Note: this client does not check that the input files are sorted.
     *
     * @param args the command-line arguments
     */
    public static void main(String[] args) {
        int n = args.length;
```

```
        In[] streams = new In[n];
        for (int i = 0; i < n; i++)
            streams[i] = new In(args[i]);
        merge(streams);
    }
}
```

Implementação

Exrecício 2.4.33 e 2.4.34 do S&W.

Alterar de `MaxPQ.java` para `IndexMinPQ.java`. Note que `swim()` e `sink()` não foram alterados, mas `exch()` e `less()` foram :-).

```
public class IndexMinPQ<Item extends Comparable<Item>>{
    private int n = 0; // heap fica em pq[1..n]

    private int[] pq; // heap binário
    private int[] qp; // inversa de pq: qp[pq[i]] = pq[qp[i]] = i
    private Item[] itens;

    public IndexMinPQ(int maxN) { // construtor
        itens = (Item[]) new Comparable[maxN+1];
        pq   = new int[maxN+1];
        qp   = new int[maxN+1];
        for (int i = 0; i <= maxN; i++) {
            qp[i] = -1;
        }
    }

    public boolean isEmpty() {
        return n == 0;
    }

    public boolean contains(int k) {
        return qp[k] != -1;
    }

    public int size() {
        return n;
    }

    public void insert(int k, Item item) {
        n++;
        pq[n] = k;
        itens[k] = item;
        qp[k] = n;
        swim(n);
    }
```

```
public void change(int k, Item item) {
    itens[k] = item;
    swim(qp[k]);
    sink(qp[k]);
}

public Item minKey() {
    return itens[pq[1]];
}

public int minIndex() {
    return pq[1];
}

public int delMin() {
    int indexOfMin = pq[1];
    exch(1, n--);
    sink(1);
    itens[pq[n+1]] = null; // avoid loitering
    qp[pq[n+1]] = -1;
    return indexOfMin;
}

public void delete(int k) {
    int j = pq[n];
    exch(qp[k],n--);
    // heapfy
    sink(qp[j]);
    swim(qp[j]);
    // destroi o rastros de k
    itens[k] = null;
    qp[k] = -1;
}

// não altera :-)
private void swim(int k) {
    while (k > 1 && less(k/2, k)) {
        exch(k/2, k);
        k = k/2;
    }
}

// não altera :-)
private void sink(int k) {
    while (2*k <= n) {
        int j = 2*k;
        if (j < n && less(j, j+1)) j++;
        if (!less(k, j)) break;
        exch(k, j);
        k = j;
```

```
        }
    }

    // altera
    private boolean less(int i, int j) {
        return itens[pq[i]].compareTo(itens[pq[j]]) > 0;
    }

    // altera
    private void exch(int i, int j) {
        int t = pq[i];
        pq[i] = pq[j];
        pq[j] = t;
        // para consistência
        qp[pq[i]] = i;
        qp[pq[j]] = j;

    }
}
```