Ordenación heapsort

Ordenación heapsort

El algoritmo de ordenación heapsort es el algoritmo de ordenación rápido (O(nlogn) en el mejor caso, en el caso promedio y en el peor caso) más sencillo de implementar.

Es aconsejable conocerlo para poder utilizarlo en el examen.

Este algoritmo utiliza una cola con prioridad de la stl.

Ordenación heapsort

El algoritmo heapsort consiste en insertar todos los elementos a ordenar en una cola con prioridad, también conocida como heap (montón) o APO (árbol parcialmente ordenado). Posteriormente se extraen los elementos ya ordenados.

Cada una de estas inserciones es O(logn). Si insertamos n elementos, el orden es O(nlogn).

Cada una de las extracciones es también O(logn). Si extraemos n elementos, el orden es O(nlogn).

Por tanto, el orden de usar este algoritmo es O(nlogn).

Ordenación heapsort

Necesitaremos definir un criterio de ordenación de los elementos.

Para ello, podemos bien definir el operator< de un TDA, o definir una clase con el operator() que actúe de comparador.

Como una cola con prioridad ordena sus elementos de mayor a menor (sale primero el mayor), los operadores operator<(a,b) y operator()(a,b) devolverán true si b debería salir de la cola con prioridad antes que a.

Ordenación heapsort

Ejemplo 1 (con operator<, queremos orden decrec. por v):

```
class Nodo {
  int v;
  bool operator<(const Nodo &otro) { return otro.v>v; }
}
…
set<Nodo> conj;
set<Nodo>::const_iterator ite;
...
priority_queue<Nodo> pq;
for (ite = conj.begin();ite != conj.end();ite++)
  pq.push(*ite);
//Los elementos ya están ordenados en pq.
…
```

Ordenación heapsort

Ejemplo 2 (con operator<, queremos orden creciente por v):

```
class Nodo {
  int v;
  bool operator<(const Nodo &otro) { return otro.v<v; }
}
…
set<Nodo> conj;
set<Nodo>::const_iterator ite;
...
priority_queue<Nodo> pq;
for (ite = conj.begin();ite != conj.end();ite++)
  pq.push(*ite);
//Los elementos ya están ordenados en pq.
…
```

Ordenación heapsort

Ejemplo 3 (con operator(), queremos orden decrec. por v):

```
Class Comparador {
  public:
  bool operator()(const Nodo &a,const Nodo &b) {
    return b.getV()>a.getV();
  }
}
…
set<Nodo> conj;
set<Nodo>::const_iterator ite;
priority_queue<Nodo,vector<Nodo>,Comparador> pq;
for (ite = conj.begin();ite != conj.end();ite++)
  pq.push(*ite);
//Los elementos ya están ordenados en pq.
```

Ordenación heapsort

Ejemplo 4 (con operator(), queremos orden creciente por v):

```
Class Comparador {
  public:
  bool operator()(const Nodo &a,const Nodo &b) {
    return b.getV()<a.getV();
  }
}
…
set<Nodo> conj;
set<Nodo>::const_iterator ite;
priority_queue<Nodo,vector<Nodo>,Comparador> pq;
for (ite = conj.begin();ite != conj.end();ite++)
  pq.push(*ite);
//Los elementos ya están ordenados en pq.
```