

Ordenación por selección



Supongamos que queremos ordenar una lista de números. El algoritmo de *ordenación por selección* funciona del siguiente modo: buscamos el menor de todos ellos y lo insertamos al principio de la lista. De los restantes, volvemos a buscar el menor y lo colocamos en la segunda posición. De los restantes, volvemos a buscar el menor y lo colocamos en la tercera posición, y así sucesivamente hasta que el segmento en el que buscamos el menor se queda con un solo elemento (que ya estará colocado al final de la lista).

Por ejemplo, dada la lista [9,1,7,2], el menor de todos los elementos es 1, por lo que lo insertamos¹ al principio, quedando [1,9,7,2]. El menor de los restantes es el 2, por lo que lo colocamos después del anterior: [1,2,9,7]. Del segmento restante, el menor elemento es el 7. Al colocarlo, tenemos la lista ordenada: [1,2,7,9].

Partiendo de la implementación de listas doblemente enlazadas circulares con nodo fantasma vista en clase (ListLinkedDouble):

1. Añade los siguientes métodos privados a la clase ListLinkedDouble:

```
void detach(Node *n);  
void attach(Node *n, Node *position);  
Node * minimum(Node *start, Node *end) const;
```

El método detach recibe un nodo n y lo desengancha de la lista enlazada, sin eliminarlo del *heap*. El método attach engancha el nodo n en una lista enlazada, situándolo después del nodo apuntado por position. El método minimum devuelve el nodo con el valor más pequeño de entre todos los que pertenecen al segmento de lista comprendido entre el nodo start (incluido) y el nodo end (no incluido). Puedes suponer que start y end no apuntan ambos al mismo nodo.

2. Utilizando los métodos auxiliares anteriores, añade a la clase ListLinkedDouble el siguiente método privado:

```
void sort_and_dedup();
```

Este método ordena la lista this utilizando el algoritmo de ordenación por selección, eliminando los duplicados que se encuentren. Por ejemplo, si tenemos que xs = [10,8,7,8,8], tras la llamada xs.sort_and_dedup() se cumple que xs = [7,8,10].

Importante: En la implementación del método sort_and_dedup no pueden crearse, directa o indirectamente, nuevos nodos mediante new ni copiar valores de un nodo a otro.

3. Indica y justifica el coste de cada uno de los cuatro métodos implementados.
4. Escribe un programa que lea varias listas de entrada y llame al método sort_and_dedup sobre cada una de ellas, mostrando el resultado por pantalla. El formato se describe a continuación.

¹En realidad, el algoritmo de ordenación por selección "estándar" *intercambia* el primer elemento por el mínimo encontrado, en lugar de insertar el mínimo al principio. Esto se debe a que habitualmente se implementa este algoritmo en arrays, donde no es posible insertar eficientemente un elemento al principio. Sin embargo, nosotros sí realizaremos inserciones en lugar de intercambios, porque utilizamos listas enlazadas.

Entrada

La entrada comienza por un número que indica cuántos casos de prueba vienen a continuación. Cada caso de prueba es una línea con una secuencia de números mayores o iguales cero, que son los elementos de la lista a construir. Esta secuencia finaliza con un -1 que no forma parte de la lista. Los elementos de la lista están comprendidos entre 0 y 10^9 . Cada lista contiene, como mucho, 1 000 elementos.

Salida

Para cada caso de prueba debe imprimirse la lista ordenada y sin elementos duplicados utilizando la notación vista en clase. Puedes utilizar, para ello, la sobrecarga del operador <<.

Entrada de ejemplo

```
4
9 1 7 2 -1
10 6 3 4 5 2 4 2 -1
1 1 -1
1 2 3 4 5 -1
```

Salida de ejemplo

```
[1, 2, 7, 9]
[2, 3, 4, 5, 6, 10]
[1]
[1, 2, 3, 4, 5]
```

Créditos

Autor: Manuel Montenegro