Estructuras de Datos y Algoritmos
Grado en Ingeniería Informática
Universidad de Zaragoza – Curso 2023-2024

Sesión de problemas 6
Tries, colas con prioridades y montículos

Objetivos

- Implementar (en pseudocódigo) operaciones sobre *tries*.
- Implementar (en pseudocódigo) operaciones sobre colas con prioridades y montículos.

1. Ejercicios sobre tries

1.1. Definición de estructura e implementación de la operación `prefijoComún`

Implementa (en pseudocódigo) una operación `prefijoComún` que, dado un *trie* `t`, una palabra `p` y un entero `n`, escriba por pantalla todas las palabras del *trie* que empiezan por `p` y sean de longitud `n`. Incluye además la estructura de datos utilizada.

1.2. Implementación del procedimiento `eliminarPalabrasPrefijo`

Implementa (en pseudocódigo) un procedimiento `eliminarPalabrasPrefijo` que, dado un trie `t`, definido de acuerdo a la implementación estudiada en clase y con marca explícita de finalización de palabras el carácter `'$'`, elimine de `t` todas aquellas palabras que tengan como prefijo la cadena de caracteres dada en un parámetro `c` de tipo cadena recibido. Considera que el carácter `'$'` no formará parte de la cadena `c` ni de las palabras almacenadas en el *trie* `t`.

```
tipos
    trie = ↑nodo;
    nodo = registro
               dato: carácter; {como símbolo de fin de palabra se usará el '$'}
               primogenito, sigHermano: trie
           freg

procedimiento eliminarPalabrasPrefijo(e/s t: trie; ent c: cadena)
```

2. Ejercicios sobre colas de prioridades y montículos

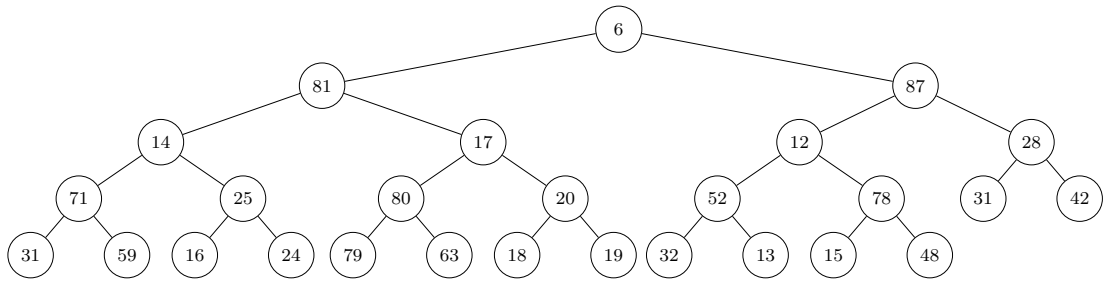
2.1. Sistema CPSI

Una *Cola con Prioridades y Servidor Imprevisible* (CPSI) es un sistema de espera al que llegan elementos con una cierta prioridad asociada para recibir servicio. Debido al extraño carácter del servidor, no siempre es servido en primer lugar el elemento de la cola que tiene mayor prioridad (como ocurre en una cola con prioridades ordinaria). Dependiendo del estado de su *ánimo*, el servidor decide servir en cada momento bien al elemento con mayor prioridad (si el servidor está contento), bien al que tiene menor prioridad (si está enfadado).

a) **Especifica el TAD que defina el tipo genérico CPSI con las operaciones de**: *añadir un elemento*, *observar el primero de la cola* (que es el de mayor prioridad si el servidor está contento o el de menor en caso contrario), *eliminar el primero de la cola* (con iguales reglas) y *observar si la cola está vacía o no*. Para poder realizar la especificación, **define además un género que identifique el estado de ánimo del servidor**.

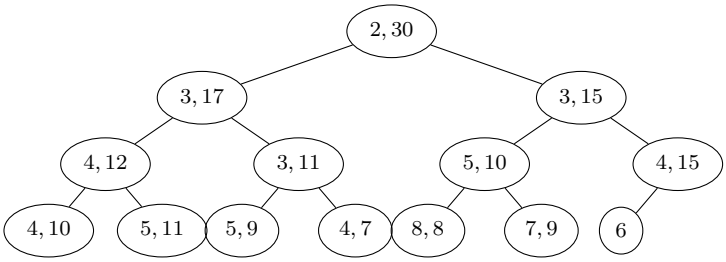
b) Diseña la interfaz de un módulo que implemente el TAD genérico CPSI. **Propón una representación estática de los datos de tipo CPSI que permita implementar todas las operaciones con un coste del orden del logaritmo del número de elementos en la cola, en el peor de los casos**. Para ello, obsérvese que una CPSI puede representarse como un montículo min-max. Un montículo min-max tiene una estructura similar a la de un montículo ordinario (árbol binario parcialmente ordenado casi-completo), pero la ordenación parcial consiste en que los elementos que se encuentran en un nivel par (0, 2, 4, ...) son menores o iguales que sus elementos descendientes, mientras que los elementos que se encuentran en nivel impar son mayores o iguales que sus descendientes (véase la figura).

c) **Implementa la operación de añadir un elemento, con un coste en el peor caso del orden del logaritmo del número de elementos en la cola**.



2.2. Montículo de intervalos

Un *montículo de intervalos* es un árbol cuasi-completo binario en el cual cada nodo P, excepto posiblemente el último, contiene dos elementos enteros, a y b, donde $a \leq b$. Se dice que el nodo P representa el intervalo cerrado $[a, b]$, donde a es el límite inferior y b es el límite superior. El intervalo $[c, d]$ está contenido en el intervalo $[a, b]$, si y sólo si $a \leq c \leq dleqb$. En un montículo de intervalos, los intervalos representados por los hijos izquierdo y derecho de cada nodo P (si existen) están contenidos en el intervalo representado por P. Cuando el último nodo contiene un solo elemento c entonces se cumple que $a \leq c \leq b$, donde $[a, b]$ es el intervalo del padre (si existe) de ese último nodo.



Las siguientes propiedades se cumplen:

1. Los límites inferiores de cada nodo de intervalos definen un montículo de mínimos. Los límites superiores de cada nodo de intervalos definen un montículo de máximos. En el caso de que el último nodo tenga un solo elemento, éste puede ser considerado tanto como un miembro del árbol de máximos como del árbol de mínimos.

2. Cuando la raíz tiene dos elementos, el límite inferior es el mínimo elemento de todo el montículo y el límite superior es el máximo elemento de todo el montículo. Cuando la raíz tiene un solo elemento el montículo contiene un solo elemento. Este elemento puede ser considerado como el mínimo y el máximo elemento.

3. Un montículo de intervalos puede ser representado sobre un array o vector, tal como se realiza para montículos ordinarios. Sin embargo, en este caso cada posición del vector debe tener espacio para dos elementos.

En este ejercicio se pide lo siguiente:

- **Define una estructura de datos en pseudocódigo para un montículo que pueda contener como máximo X números enteros** ($X > 0$).
- **Implementa (en pseudocódigo) un procedimiento para realizar la inserción de un elemento (un entero) en el montículo de intervalos enteros**, que al finalizar la inserción el montículo debe mantener las propiedades de un montículo de intervalos.

2.3. Implementación del procedimiento `de_estática_a_dinámica`

Se dispone de un montículo que almacena los productos de un carro de compra según la representación estática por todos conocida (un registro con un vector y un contador) y suponiendo que la prioridad es el número de unidades del producto en el carro (es decir, el producto con más unidades está en la raíz, etcétera). **Implementa en notación algorítmica la operación `de_estática_a_dinámica`, que copie los datos del montículo estático a un montículo generado en memoria dinámica** (respetando la estructura del árbol) en el que cada registro guarde punteros a los hijos y puntero al padre. Considera los siguientes tipos de datos ya definidos a continuación, junto con la signatura de la operación solicitada:

```
constante maxNum = ...
tipos montículoEstático = registro
                              dato: vector[1..maxNum] de producto;
                              num: 0..maxNum
                          freg;
      montículoDinámico = ↑nodo;
                   nodo = registro
                              prod: producto;
                              izq, der, padre: montículoDinámico
                          freg;

procedimiento de_estática_a_dinámica(ent me: montículoEstático; sal md: montículoDinámico)
```

