Apellidos:
Nombre:
Convocatoria:
DNI:

Examen TAD/PED julio 2004
Modalidad 0

Normas:
- La entrega del test no corre convocatoria.
- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
- **El test vale un 40% de la nota de teoría.**
- En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
La operación crear_pila() es constructora modificadora.	❑	☑	F	1.
En C++, una forma correcta de copiar una cadena es la siguiente: *char a[50] = "Tipos Abstractos de Datos";* *char *b;* *b = new char[strlen(a)];* *strcpy(b, a);*	❑	☑	F	2.
El caso peor de la búsqueda es más eficiente en una lista ordenada que en una lista cuyos elementos no están ordenados.	❑	☑	F	3.
En un árbol binario cada elemento puede tener como máximo dos predecesores.	❑	☑	F	4.
Si se implementa el algoritmo de ordenación de un vector "heapsort" utilizando un heap máximo los elementos quedan ordenados en el vector de forma descendente.	❑	☑	F	5.
Dado un recorrido en preorden (RID) de un árbol AVL es posible reconstruir un único árbol AVL.	☑	❑	V	6.
El número máximo de elementos que se puede almacenar en un árbol 2-3 de altura h es 3^h-1	☑	❑	V	7.
Al borrar un elemento en un árbol 2-3-4 se puede realizar una operación de DIVIDERAIZ.	❑	☑	F	8.
En un árbol B m-camino de búsqueda con m=16: en cualquier nodo excepto la raíz hay 8 ítems como mínimo.	❑	❑	F	9.
La especificación algebraica de la siguiente operación permite la inserción de claves repetidas (C: ConjuntoConClavesRepetidas; x, y: Ítem): Insertar(Insertar(C, x), y) ⇔ Insertar(Insertar(C,y), x)	☑	❑	V	10.
En el TAD Diccionario con dispersión cerrada, con función de redispersión "h_i(x)=(H(x) + k(x)*i) MOD B", con B=6 se puede dar la situación de que en una búsqueda no se acceda a todas las posiciones de la tabla.	☑	❑	V	11.
El siguiente vector representa un montículo máximo: *10* | *5* | *3* | *1* | *2*	☑	❑	V	12.
Para todo nodo de un árbol Leftist, se cumple que la altura de su hijo izquierdo es menor que la de su hijo derecho.	❑	❑	F	13.
La complejidad temporal de la búsqueda en un trie es lineal respecto al número de palabras almacenadas	❑	❑	F	14.
Un bosque extendido en profundidad de un grafo no dirigido es un grafo acíclico.	☑	❑	V	15.

Examen TAD/PED julio 2004

Normas:
- Tiempo para efectuar el ejercicio: **2 horas**
- En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: ***Apellidos, Nombre***. Cada pregunta se escribirá en folios diferentes.
- Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
- Las soluciones al examen se dejarán en el campus virtual.
- Se puede escribir el examen con lápiz, siempre que sea legible
- **Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.
- **Publicación de notas de exámenes y prácticas**: 6 de julio. **ÚNICA fecha de revisión de exámenes y prácticas:** 9 de julio. El lugar y la hora se publicará en el campus virtual.
- **Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas**

1. Sean 2 leftist mínimos representados como árboles binarios (se asume que no están vacíos y que no hay elementos repetidos) y cuyas etiquetas son números naturales. Utilizando exclusivamente las operaciones constructoras generadoras de los árboles binarios definir la sintaxis y la semántica de la primera parte de la operación *combinar* en la que se crea un árbol resultado de la combinación de ambos, sin comprobar la condición de leftist y actualización de caminos mínimos.

2. El siguiente fragmento de código define una clase en C++ que representa una lista doblemente enlazada de punteros a objetos TPoro. Se pide el código del destructor, el constructor de copia y la sobrecarga del operador de asignación (en éstos hay que duplicar la memoria de modo que si se hace una asignación y se destruye una lista la otra no se destruya).

```
class TNodo {
  TPoro *dato;
  TNodo *sig, *ant;
};
class TLista {
  public:
    TLista();
    TLista(TPoro *);
    ...
  private:
    TNodo *primero, *ultimo;
};
```

3. Sea un árbol B de orden m=6 inicialmente vacío:

a) Insertar los siguientes elementos 10, 5, 7, 3, 12, 6, 20, 30, 35, 40, 45, 41.
b) Del árbol B resultante, borrar los elementos 12, 45.
c) Del árbol B resultante, insertar los elementos 2, 4, 8, 9, 21, 22, 23, 25, 26, 27, 28, 29, 33, 34, y después borrar 8 y 9.
d) El árbol B resultante cumple también las propiedades para ser un árbol 234. Enumera cuáles son estas propiedades e inserta el ítem 31 utilizando el algoritmo de inserción del árbol 234.

Reglas a tener en cuenta:

1. En la inserción cuando hagamos la división de nodos dejaremos la menor cantidad de elementos posibles en el nodo de la izquierda.
2. Al realizar un borrado de un ítem interior (no hoja), lo sustituiremos por el mayor de la izquierda.
3. Al realizar un borrado de un ítem, elegiremos como nodo adyacente el nodo de la izquierda. En el caso de que el nodo objeto de la transformación sea el nodo más a la izquierda, consideraremos el nodo derecha de éste como nodo adyacente.

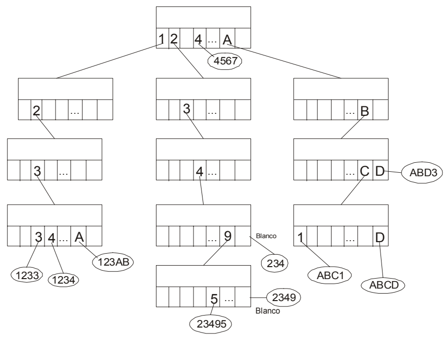
4. **a)** Sobre un *trie* con nodos terminales como el del ejemplo, calcular las cotas superiores de complejidad temporal en su caso peor y mejor, para las siguientes operaciones:

Insertar(palabra),
Insertar(conjuntoDePalabras) (por ejemplo conjuntoDePalabras={1,123,AB})
ListarPalabrasConPrefijo(prefijo) (del trie ejemplo, con *prefijo=234,* esta función devolvería {234,2349,23495}),
NumeroDeElementos() (del trie ejemplo esta función devolvería el valor 10).

Para ello utilizar los siguientes parámetros: *n (longitud de "palabra"), m (longitud de "prefijo"), k (número de letras del alfabeto o tamaño del vector), L (longitud de la palabra más larga posible), T (tamaño de "conjuntoDePalabras"), p (número de palabras del diccionario).*

b) Sea un grafo no dirigido de *n* vértices numerados de *1* a *n*, representado con una lista de adyacencia, en la que sólo se almacenan las aristas *(i, j)* tal que *i<j*, con *i* y *j* de tipo "vértice". Calcular las cotas superiores de complejidad temporal en su caso peor y mejor, para las siguientes operaciones: *Insertar(i, j), Adyacencia(i), NumeroDeAristas()*.

NOTA: Hay que explicar cada complejidad con un máximo de 3 líneas. Si la explicación no es correcta, no se valorará positivamente.



Examen TAD/PED diciembre 2003. Soluciones

1)

combinar: arbin, arbin → arbin

Var i,l,d,r:arbin; x,y:natural;

combinar(crear_arbin(),enraizar(i,x,d)) = enraizar(i,x,d)
si x<y
entonces combinar(enraizar(i,x,d), enraizar(l,y,r)) = enraizar(i,x,combinar(d,enraizar(l,y,r))
sino combinar(enraizar(i,x,d), enraizar(l,y,r))=enraizar(l,y,combinar(r,enraizar(i,x,d))

2)

a) Destructor

```
TLista::~TLista()
{
  TNodo *aux;

  while(primero != NULL)
  {
    aux = primero;
    primero = primero->sig;
    delete aux->dato;
    delete aux;
  }
  primero = ultimo = NULL;
}
```

b) Constructor de copia

```
TLista::TLista(TLista &l)
{
  TNodo *a;
  TNodo *aux = l.primero;

  if(aux != NULL)
  {
    a = new TNodo;
    a->dato = new TPoro(*(aux->dato));
    a->sig = NULL;
    a->ant = NULL;
    primero = ultimo = a;

    aux = aux->sig;
    while(aux != NULL)
    {
      a = new TNodo;
      a->dato = new TPoro(*(aux->dato));
      a->sig = NULL;
      a->ant = ultimo;
      ultimo = a;
      aux = aux->sig;
    }
  }
  else
    primero = ultimo = NULL;
}
```

Otra alternativa:

```
TLista::TLista(TLista &l)
{
  primero = ultimo = NULL;
  *this = l;
```

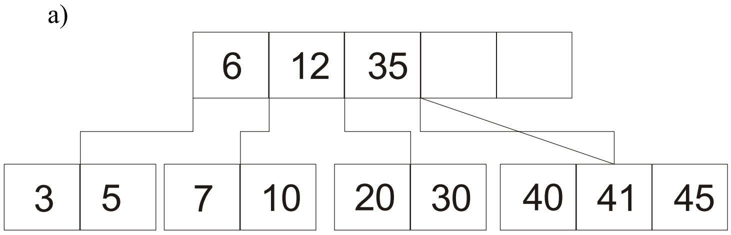
```
}
```

c) Sobrecarga del operador asignación

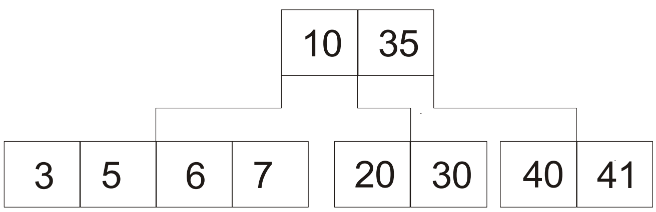
```
TLista&
TLista::operator=(TLista &l)
{
  TNodo *a;
  TNodo *aux = l.primero;

  if(this == &l)
    return *this;

  this->~TLista();
  if(aux != NULL)
  {
    a = new TNodo;
    a->dato = new TPoro(*(aux->dato));
    a->sig = NULL;
    a->ant = NULL;
    primero = ultimo = a;

    aux = aux->sig;
    while(aux != NULL)
    {
      a = new TNodo;
      a->dato = new TPoro(*(aux->dato));
      a->sig = NULL;
      a->ant = ultimo;
      ultimo = a;
      aux = aux->sig;
    }
  }
  else
    primero = ultimo = NULL;

  return *this;
}
```

3)

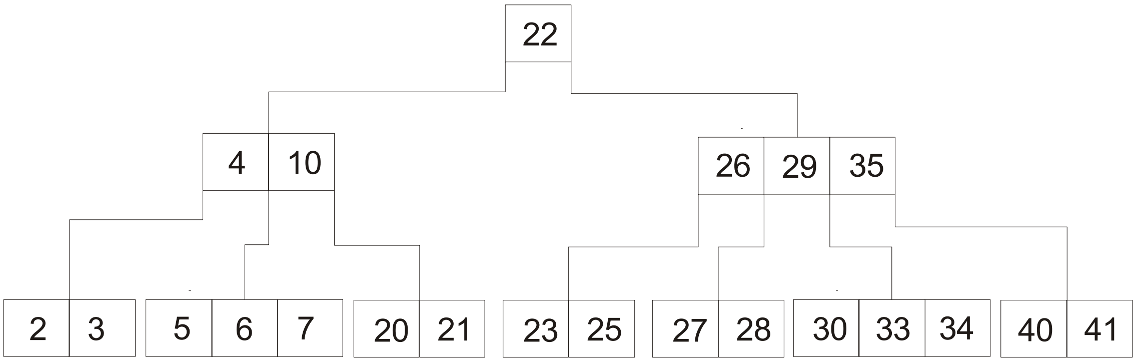
a)



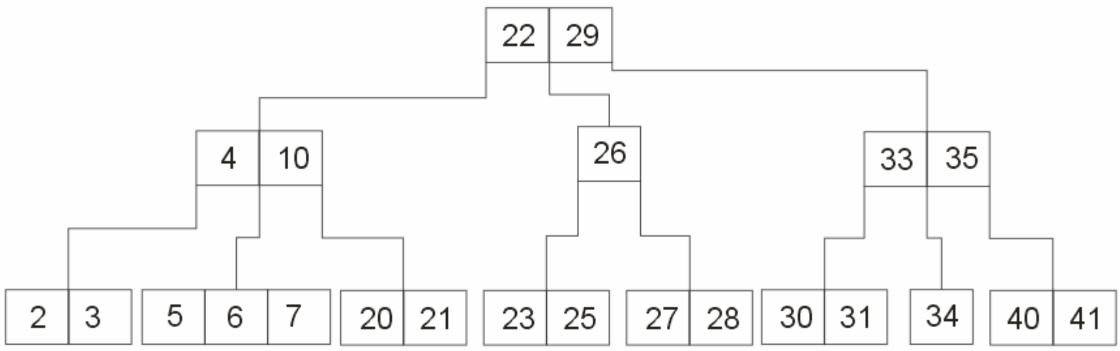
b)



c)



d)



4) a)

	Peor Caso	Mejor Caso
Insertar(palabra)	O(n) hay que crear un nodo no terminal por cada letra de "palabra"	Ω(1) cuando sólo hay que añadir un nodo terminal, por ejemplo para *insertar(BCD)*
Insertar(conjuntoDePalabras)	O(L*T) misma explicación que el caso anterior para las T palabras de *conjuntoDePalabras*	Ω(T) misma explicación que el caso anterior para las T palabras de *conjuntoDePalabras*
ListarPalabrasConPrefijo(prefijo)	O(m+k*(L-m)) cada letra del prefijo aparecerá como nodo no terminal, y habrá que recorrer cada posición del vector del nodo terminal de su última letra, y como peor caso cada palabra tendrá longitud L, con cada letra representada con nodos no terminales.	Ω(1) caso que sólo hay una palabra con ese prefijo en todo el diccionario, y ninguna otra palabra que comparta ninguna otra letra del prefijo. ó Ω(m+k) caso que haya que recorrer cada letra del prefijo como nodo no terminal y recorrer el vector de ese nodo, en el que cada palabra que tenga ese prefijo esté representada por un único nodo terminal
NumeroDeElementos()	O(L*p) cada una de las "p" palabras del diccionario, todas sus letras estarán representadas con nodos no terminales, con una longitud máxima de palabra "L"	Ω(p+k) cada palabra estará representada por un único nodo no terminal

b)

	Peor Caso	Mejor Caso
Insertar(i, j), *suponiendo que i<j*	O(n) hay que recorrer la lista del vértice "i", y el caso peor es que esa lista tenga "n-1" vértices	Ω(1) cuando la lista del vértice "i" está vacía
Adyacencia (i)	O($\sum_{j=1}^{i}(n-j)$) = O(n^2), cada lista "j" del vector tendrá "n-j" vértices como máximo (caso peor), y habrá que recorrer todas las listas de la posición "1" del vector hasta "i"	Ω(i) cada lista del vector desde "1" hasta "i" estará vacía

NumeroDeAristas()	O($\frac{n*(n-1)}{2}$) = O(n^2) máximo número de aristas en un grafo no dirigido	Ω(n) grafo vacío, en el que habrá que recorrer secuencialmente cada casilla del vector