Guía de Trabajos Prácticos número 4
Conjuntos

[1] [Teoría y Operativos]

a) ¿Cuál es el tiempo de ejecución de `find(x)` en el TAD diccionario por tablas de dispersión abiertas, en el caso promedio?

b) ¿Cual es el tiempo de ejecución para `insert(x)` en el TAD conjunto por árbol binario de búsqueda, en el peor caso?

c) ¿Cual es el tiempo de ejecución para `insert(x)` en el TAD conjunto por tabla de dispersión on cerrada, en el caso promedio?

d) *Mapeo.* La realización de conjuntos mediante vectores de bits se puede usar siempre que el *conjunto universal* se pueda traducir a los enteros de 1 a N. Describa cómo haría esa traducción (es decir las funciones `int indx(elem_t)` y `elem_t element(int)`) si el conjunto universal fuera: hecho en zinjai

1) los enteros $0, 1, ..., 99$.
2) los enteros de n a m para cualquier $n <= m$.
3) los enteros $n, n + 2, n + 4, ...n + 2k$, para cualesquier n y k.
4) los caracteres a, b, ... z.

e) *HashTDA.* Insertar los enteros `(15,10,9,19,9,29,28,17,46)` en una tabla de dispersión abierta con función de dispersión $h(x) = x \,\%B$ con $B = 10$ cubetas. hecho en carpeta

f) *HashDC.* Insertar los enteros `(14,27,24,15,34,41,57,67,55,27)` en una tabla de dispersión cerrada con $B = 10$ cubetas con función de dispersión $h(x) = x \,\%B$ y re-dispersión lineal. Mostrar cómo queda la tabla después de realizar las inserciones.

g) *HashDC2.* Suponga se están dispersando enteros en una tabla de dispersión cerrada con resolución lineal de colisiones y cinco cubetas, usando la función de dispersión $h(i) = i \,\%5$ muestre la tabla de dispersión obtenida cuando se insertan los enteros 23, 48, 35, 4, 10.

h) *SpatialHash.* Una metodología para simular fluidos en una computadora se basa en describir el mismo mediante partículas que se ejercen fuerzas entre sí. Para poder realizar esto, es necesario conocer qué partículas se encuentran cerca en el espacio. Proponga una solución que permita resolver eficientemente la búsqueda de vecinos. Ayuda: Divida al plano en celdas y utilice una función de dispersión para indexarlas.

i) ¿Qué condiciones debe satisfacer un árbol binario para ser "árbol binario de búsqueda"?

j) *llenarBB1.* Inserte los enteros `7,9,0,5,6,8,1`, en ese orden, en un árbol binario de búsqueda inicialmente vacío. Muestre el resultado de suprimir `7` y después `2` del árbol.

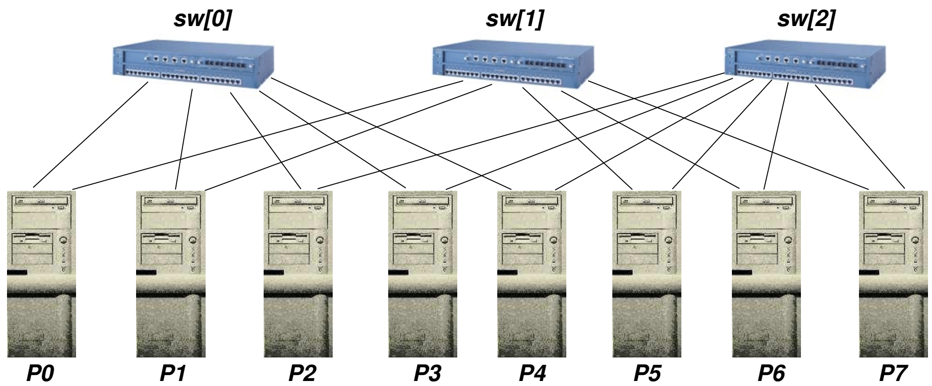
k) *llenarBB2.* Dados los enteros `16, 10, 23, 5, 6, 13, 8, 7, 4, 15` insertarlos, en ese orden, en un árbol binario de búsqueda. Mostrar las operaciones necesarias para eliminar los elementos `16, 10` y `7` en ese orden.

[2] [Programación]

a) **[Set]**

1) *Operaciones.* Escribir las funciones
   a' `void set_union(set<T> &A,set<T> &B,set<T> &C);`
   b' `void set_intersection(set<T> &A,set<T> &B,set<T> &C);`
   c' `void set_difference(set<T> &A,set<T> &B,set<T> &C);`
   en términos de los restantes métodos de la interfase de `set`.

2) *Disjuntos*. Escribir una función predicado `bool disjuntos(vector<set<T>>&v)` que verifica si todos los conjuntos A_i dentro del vector de conjuntos `v` son disjuntos de a pares, esto es: si $A_i \neq A_j$ entonces $A_i \bigcap A_j = \phi$.
3) *DiferenciaSimétrica*. Dada una lista de conjuntos de enteros `list< set<int> > l` escribir una función `void diffsym(list< set<int> > &L, set<int> &ad)` que retorna en `ad` el conjunto de los elementos que pertenecen a uno y sólo uno de los conjuntos de `L`. Por ejemplo, si `L = ({1,2,3},{2,4,5},{4,6})` entonces `ad = {1,3,5,6}`. Notar que si el número de conjuntos en `L` es 2 y los llamamos `A` y `B`, entonces debe retornar `ad = (A-B)` $\cup$`(B-A)`.
4) *ClasesDeCongruencia*. Dado un conjunto `S` implementar la función `congrClasses(set<int> &S, int m,list<set<int>> &L)` que retorna las clases de congruencia módulo `m` en la lista `list<set<int>> L`.
5) *CubreTodo*. Escribir una función predicado `cubre_todo(vector<set<T>>&v, set<T>& W)` que verifica si todos los conjuntos en el vector de conjuntos `v` están incluidos en `W`.
6) *IncluyeTodo*. Dados n conjuntos A_0, A_1, ... A_{n-1} determinar si alguno de ellos (digamos A_j) incluye a todos los otros. Es decir $A_j \subset A_k$ para todo k. En ese caso, retornar el indice j, si no retornar -1. Signatura: `int includes_all(vector< set<int> > &setv)`.
7) *subK*. Escriba una función `list<set<int>> subk(set<int> &S, int k)` que devuelva una lista conteniendo todos los subconjuntos posibles del conjunto `S` tomados de a `k`.
8) *Flat*. Se está diseñando una red interconectada por switches y se desea, para reducir lo más posible la *latencia* entre nodos, que cada par de nodos esté conectado en forma directa por al menos un switch. Sabemos que el número de nodos es `n` y tenemos un `vector< set<int> > sw` que contiene para cada switch el conjunto de los nodos conectados por ese switch, es decir `sw[j]` es un conjunto de enteros que representa el conjunto de nodos inteconectados por el switch j.
Se pide escribir una función predicado `bool flat(vector< set<int> > &sw, int n)` que retorne verdadero si cada par de enteros (j, k) con $0 \leq j, k < n$ está contenido en al menos uno de los conjuntos en `sw[]`.



En el ejemplo de la figura tenemos 8 nodos conectados via 3 switches y puede verificarse que cualquier par de nodos está conectado en forma directa a través de al menos un switch. Para este ejemplo el vector `sw` sería

$$\texttt{sw[0]} = \{0, 1, 2, 3, 4\}, \quad \texttt{sw[1]} = \{0, 1, 5, 6, 7\}, \quad \texttt{sw[2]} = \{2, 3, 4, 5, 6, 7\} \tag{1}$$

Por lo tanto `flat(sw,8)` debe retornar `true`. Por otra parte si tenemos

$$\texttt{sw[0]} = \{0, 2, 3, 4\}, \quad \texttt{sw[1]} = \{0, 1, 5, 7\}, \quad \texttt{sw[2]} = \{2, 3, 5, 6, 7\} \tag{2}$$

entonces los pares $(0, 6)$, $(1, 2)$, $(1, 3)$, $(1, 4)$, $(1, 6)$, $(4, 5)$, $(4, 6)$ y $(4, 7)$ no están conectados en forma directa y `flat(sw,8)` debe retornar `false`.

b) **[Varios]**

1) *listarABB*. Programe una función `void listar_ABB(btree<int> &BT)` que muestre en orden ascendente los elementos del conjunto implementado como ABB.
2) *insertaABB*. Programe una función `void inserta_ABB(btree<int> &BT, int v)` que inserte `v` en la posición adecuada de `BT` que permita la construcción de un árbol binario de búsqueda.
3) *eliminaABB*. Programe una función `void elimina_ABB(btree<int> &BT, int v)` que elimina, si existe, el valor `v` del árbol binario `BT` de tal manera de preservar el ordenamiento del mismo.
4) *Migración*. Para mejorar la velocidad de las operaciones, se desea reemplazar una tabla de dispersión abierta (`vector<list<T>>`) con B_1 cubetas con más de B_1 elementos por otra tabla de dispersión con B_2 cubetas. Escriba un procedimiento para construir la nueva tabla a partir de la anterior.
5) *MakeSudoku*. El grafo del sudoku se puede ver como una grilla de nodos de tamaño 9×9 dividido en 9 zonas de 3×3, donde cada nodo tiene como nodos adyacentes a todos los nodos de su misma columna, todos los nodos de su misma fila, y todos los nodos de su zona. El grafo `sgraph` está representado por un: `vector<vector<set<pair<int,int>>>>` Es decir, una matriz, donde cada elemento de la matriz es un conjunto de pares indicando los nodos adyacentes a dicho elemento por medio del par (first = fila, second = columna).
   Finalmente, en este ejercicio se solicita que se programe la función `sgraph sudoku_graph()` que ensamble y retorne el grafo del sudoku.
6) *SolveSudoku*. Dada una matriz de 9×9, donde algunos valores son 0 y otros algún número del 1 al 9 (inclusive). Reemplazar los ceros de la matriz con números del 1 al 9, de tal forma de resolver el sudoku correspondiente. Un sudoku resuelto es aquel que en una matriz de 9x9 números del 1 al 9, dividida en 9 zonas de 3×3, resulta que cada fila, cada columna, y cada zona, no tienen números repetidos, es decir, cada fila, columna y zona contiene a todos los números del 1 al 9. Ayuda: Resolver un sudoku es lo mismo que colorear un grafo. El grafo a colorear es el grafo del sudoku, se debe colorear con 9 colores donde algunos nodos ya vienen coloreados (aquellos distintos de cero). Una 9-coloración del grafo del sudoku, donde los colores los enumerados del 1 al 9, corresponde a un sudoku resuelto. `void sudoku(vector< vector<int> > &M)`.