

Algoritmos y Estructuras de Datos.

TPL3. Trabajo Práctico de Laboratorio 3. [2017-11-02]

PASSWD PARA EL ZIP: **JZQT WKBA ZLYZ**

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar.**

[Ej. 1] **[isBST]** Dado un árbol binario **T**, generar la función `bool isBST(btree<int>&T)`; que devuelve **true** si **T** es un árbol binario de búsqueda y **false** en caso de que no lo sea.

Recordar que un árbol binario es un **ABB** (o **BST** por sus siglas en inglés) si:

- Está vacío.
- Para cualquier nodo **n** del árbol, todos los valores del subárbol izquierdo del nodo **n** son menores al valor de **n**.
- Para cualquier nodo **n** del árbol, todos los valores del subárbol derecho del nodo **n** son mayores al valor de **n**.

Ejemplos:

- **T=(10 (5 (3 (7 6 9))) (18 15 (20 19 30)))** debe retornar **true**.
- **T=(10 (5 (3 (7 6 9))) (18 15 (20 19 3)))** debe retornar **false**.
- **T=(8 (5 4 7) (15 13 20))** debe retornar **true**.

[Ej. 2] **[fillBST]** Dada una lista de enteros **L** y un árbol binario **T**, generar la función `void fillBST(btree<int>&T, list<int>& L)`; que inserta los elementos de **L** en **T** formando un árbol binario de búsqueda siguiendo el orden original de **L**. Tal como en la implementación de **set**, en **T** no pueden quedar elementos repetidos (pese a que **L** los tenga).

Nota: No se puede usar **unique**.

Ejemplos:

L=[11,3,7,5,22,15,7] => debe retornar T = (11 (3 . (7 5 .)) (22 15 .))
L=[3,3,3,3] => debe retornar T = (3)
L=[45,34,23,9,89,12] => debe retornar T = (45 (34 (23 (9 . 12) .) .) 89)

[Ej. 3] **[eqsumsplit]** Escribir un predicado `bool eqsumsplit(set<int> &S)`; que retorna **true** si el conjunto **S** se puede descomponer en dos conjuntos disjuntos **S1** y **S2** tales que la suma de los elementos de **S1** es igual a la suma de **S2**.

Restricción: no usar otros contenedores que **set**.

Ejemplos:

S={1,2,3} => debe retornar true (S1={1,2},S2={3})
S={1,2,3,4} => debe retornar true (S1={1,4},S2={2,3})
S={1,2,3,4,5} => debe retornar false
S={1,2,3,4,5,6} => debe retornar false
S={1,2,3,4,5,6,7} => debe retornar true (S1={3,5,6},S2={1,2,4,7})

Ayuda:

- Se sugiere proceder recursivamente. Primero con $S1=S$. Después con todos los subconjuntos $S1$ que provienen de tomar $S1$ y extraer un elemento, y así siguiendo.
- Escribir una función auxiliar `bool eqsumsplit(set<int> &S, set<int> &W)`; que realiza la tarea consignada, con la restricción de que $S1$ debe ser un subconjunto de W .
- `eqsumsplit(S,W)` debe primero chequear si $S1=W$, $S2=S-S1$ es una solución. Si es solución debe retornar `true`.
- Si no debe formar todos los subconjuntos de $S1$ de W , que consisten de eliminar un elemento de W . Es decir, para todo x en W , formar $Wx=W-\{x\}$ y llamar recursivamente a `eqsumsplit(S,Wx)`, si en algún caso retorna `true` debe retornar `true`.
- Si para ningún Wx retorna `true`, entonces debe retornar `false`.
- El proceso se inicializa con $W=S$.
- El pseudocódigo es así:

```
bool eqsumsplit(S,W) {
    S1 = W; S2=S-S1;
    Si el par S1,S2 satisface la condicion retornar true
    para cada elemento x de W {
        Hacer W1 = W-{x}
        si eqsumsplit(S,W1) retorna verdadero, retornar verdadero
    }
    retornar falso;
}
```

- Escribir el wrapper `bool eqsumsplit(S)`;

Instrucciones generales

- El examen consiste en que escriban las funciones descritas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado `program.cpp`. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si `f` es la función a evaluar tenemos

```
ev.eval<j>(f,vrbs);
hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
```

`j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (`eval<1>` y `evalr<1>`). La primera `ev.eval<j>(f,vrbs)`; toma una serie de casos de prueba de entrada, le aplica la función del usuario `f` y compara la salida del usuario (`user`) con respecto a la esperada (`ref`). Si la verbosidad (el argumento `vrbs`) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- **ucase:** Además las funciones `eval()` tienen dos parámetros adicionales:

`Eval::eval(func_t func,int vrbs,int ucase);`

El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en `ucase=-1` que quiere "hacer todos". Por ejemplo `ev.eval4(prune_to_level,1,51)`; corre sólo el caso 51.

- **Archivo con casos tests JSON:** Los casos test que corre la función `eval<j>` están almacenados en un archivo `test1.json` o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. `datain` son los datos pasados a la función y `output` la salida producida por la función de usuario. `ucase` es el número de caso.

```
{ "datain": {
  "T1": "( 0 (1 2) (3 4 5 6) )",
  "T2": "( 0 (2 4) (6 8 10 12) )",
  "func": "doble" },
  "output": { "retval": true },
  "ucase": 0 },
```

- La segunda función `evalr<j>` es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla `seed`, se los pasa a la función del usuario `f()`. Las respuestas de la `f()` van siendo procesadas por la función interna de hash que genera un **checksum H** de las respuestas. Por ejemplo para el primer ejercicio si `seed=123` entonces el checksum es `H=523`. Una vez que el alumno termina su tarea se le pedirá que corra la función `evalr<j>()` de la clase evaluadora con un valor determinado de la semilla `seed` y se comprobará que genere el valor correcto del checksum `H`.

Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro `seed` con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de `H`.

- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
`void Eval::dump(list <int> &L,string s="")`: Imprime una lista de enteros por `stdout`. Nota: Es un método de la clase `Eval` es decir que hay que hacer `Eval::dump(VX)`; . El string `s` es un label opcional.

- `void Eval::dump(list <int> &L,string s="")`

- Después del parcial deben entregar el programa fuente (sólo el `program.cpp`) renombrado con su apellido y nombre (por ejemplo `messilione1.cpp`). Primero el apellido.

TPL3. Trabajo Práctico de Laboratorio 3. [2017-11-02]. TABLA SEED/HASH

S=123 -> H1=687 H2=067 H3=635	S=386 -> H1=411 H2=083 H3=779
S=577 -> H1=807 H2=351 H3=963	S=215 -> H1=215 H2=892 H3=883
S=393 -> H1=308 H2=709 H3=136	S=935 -> H1=678 H2=671 H3=981
S=686 -> H1=953 H2=285 H3=331	S=292 -> H1=564 H2=155 H3=770
S=349 -> H1=764 H2=951 H3=197	S=821 -> H1=392 H2=938 H3=865
S=762 -> H1=243 H2=687 H3=027	S=527 -> H1=635 H2=613 H3=498
S=690 -> H1=789 H2=747 H3=491	S=359 -> H1=366 H2=252 H3=794
S=663 -> H1=614 H2=574 H3=876	S=626 -> H1=404 H2=071 H3=305
S=340 -> H1=443 H2=467 H3=451	S=226 -> H1=566 H2=998 H3=433
S=872 -> H1=725 H2=212 H3=571	S=236 -> H1=385 H2=788 H3=410
S=711 -> H1=908 H2=008 H3=648	S=468 -> H1=726 H2=293 H3=072
S=367 -> H1=293 H2=559 H3=101	S=529 -> H1=805 H2=362 H3=774
S=882 -> H1=288 H2=696 H3=985	S=630 -> H1=533 H2=795 H3=570
S=162 -> H1=781 H2=146 H3=288	S=923 -> H1=522 H2=397 H3=504
S=767 -> H1=543 H2=146 H3=229	S=335 -> H1=412 H2=065 H3=333
S=429 -> H1=246 H2=759 H3=581	S=802 -> H1=020 H2=598 H3=603
S=622 -> H1=576 H2=013 H3=641	S=958 -> H1=217 H2=736 H3=684
S=969 -> H1=997 H2=481 H3=343	S=967 -> H1=450 H2=173 H3=450
S=893 -> H1=410 H2=941 H3=098	S=656 -> H1=124 H2=114 H3=795
S=311 -> H1=532 H2=457 H3=403	S=242 -> H1=711 H2=740 H3=363
S=529 -> H1=805 H2=362 H3=774	S=973 -> H1=166 H2=918 H3=810
S=721 -> H1=577 H2=927 H3=313	S=219 -> H1=595 H2=929 H3=252
S=384 -> H1=687 H2=867 H3=950	S=437 -> H1=524 H2=742 H3=316
S=798 -> H1=545 H2=308 H3=099	S=624 -> H1=832 H2=496 H3=046
S=615 -> H1=004 H2=140 H3=663	S=670 -> H1=937 H2=614 H3=986