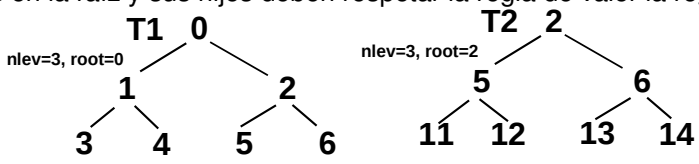
Algoritmos y Estructuras de Datos.
TPL3. Trabajo Práctico de Laboratorio. [2015-11-05]

PASSWD PARA EL ZIP: `H49G IKLJ IU6K`

Ejercicios

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

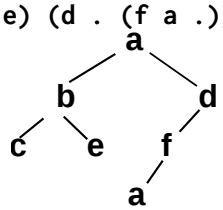
[Ej. 1] **[mkheaptree (35pt)]** Escribir una función `void mkheaptree(btree<int> &T,int nlev,int root);` que construye un árbol binario de `nlev` niveles donde los valores de los nodos corresponde a la numeración por niveles. Si `root=0` entonces la raíz contiene el valor `0`, el primer nivel `1`, y`2`, y así siguiendo. Para ello notar que si el padre es `j` entonces los hijos deben contener `2*j+1` y `2*j+2`. Si `root=0` y `nlev=3` debe retornar el árbol `T1` de la figura. Si `root>0` entonces el árbol debe contener ser construído con `root` en la raíz y sus hijos deben respetar la regla de valer la regla `2*j+1` y `2*j+2`.



Ayuda: Escribir una función recursiva
`void mkheaptree(btree<int> &T,btree<int>::iterator n,int nlev,int root);`

- Si `nlev=0` no debe hacer nada.
- Si no, debe insertar `root` en el `n`, llamar a la función sobre los hijos pasando el valor apropiado de acuerdo a la regla mencionada anterior y el nivel decrementado.

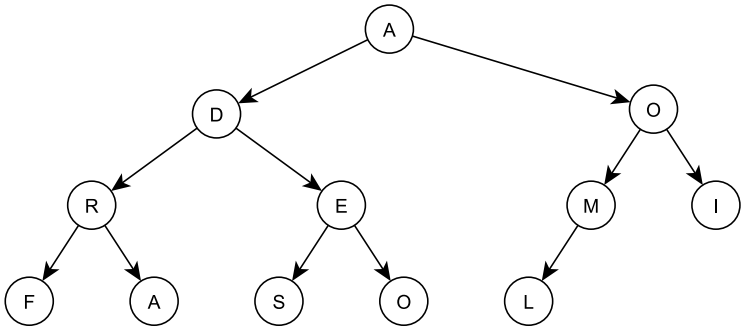
[Ej. 2] **[find-word (35pt)]** Dado un árbol de caracteres, hay que buscar una palabra dentro de lo niveles del árbol. Debe retornar el nivel de menor profundidad en el cual se encuentra la palabra, caso contrario retornar -1. Además la palabra no necesariamente debe tener el largo de nodos del nivel, sino que puede ser un substring del nivel entero.
Consigna: Escribir una función `int find_word( btree<char> &t, string s);`
Ejemplo1: Tenemos el árbol binario `(a (b c e) (d . (f a .)))`



entonces

- si queremos buscar la palabra `fa`, el agoritmo retornará -1;
- si queremos buscar la palabra `a` el algoritmo retornará 0 (si bien la letra también se encuentra como un nodo en el nivel 3, la primera ocurrencia de la misma está en el nivel 0;
- si queremos buscar la palabra `bd`, el algoritmo retornará 1;
- si queremos buscar la palabra `ef`, el algoritmo retornará 2; y así sucesivamente.

Ejemplo2:



- `find_word(T,"do") => 1`
- `find_word(T,"mi") => 2`
- `find_word(T,"la") => -1`

Ayuda:

- Escribir una función recursiva
  ```
  void fill_vector(btree<char> &t, vector<string> &v,
                   btree<char>::iterator it, int actual_level);
  ```
  que llena el `vector<string> v` con las niveles del árbol.
- Recorrer los strings y verificar si el string buscado está contenido en el mismo.
- Es muy conveniente utilizar la función `string::find()`. La expresión `w.find(s)` busca el string `s` dentro del string `w`. Si no lo encuentra retorna el valor especial `string::npos`. Por lo tanto para chequear si contiene al string deben usar la expresión
  ```
  if (w.find(s)!=string::npos) { /* w contiene a s */ }
  ```

[Ej. 3] **[filtcp (25pt)]** Hannibal quiere invitar a su novia a cenar a un restaurante elegante. A lo hora de elegir la cena (compuesta por 1 bebida + 1 plato principal, que por suerte vienen para dos personas), quiere conocer de antemano cuales son las opciones que caben dentro de su presupuesto. Para ayudarlo en su decisión, Hannibal quiere que usted le programe una función `set<pair_t> filtcp(set<int>& S1, set<int>& S2, func_t f);` que dados dos conjuntos de enteros `S1` y `S2` con los precios de cada una de las opciones en bebidas y platos principales respectivamente, retorne un conjunto de pares de enteros `set<pair_t>` con todas las opciones posibles de cena que le es posible elegir de acuerdo a una función predicado f que determina si el par (bebida,plato principal) se encuentra dentro del gasto admitido. Además, Hannibal le solicita que programe a modo de ejemplo una función predicado `bool affordable(pair_t p);` (*accesible*) la cual retorna `true` si el costo de la opción de cena (`pair_t p`) está dentro de su presupuesto `gmin<=costo<=gmax` (con `gmin=5`, `gmax=10`).
Ayuda: Hacer un doble lazo para generar los pares de valores, aplicar el predicado para cada uno de ellos y si lo satisface insertar en el conjunto de salida.
Nota: `filtcp` viene de **filter cartesian product**.

[Ej. 4] **[greatest-subset (25pt)]** Se debe implementar una función
`int greatest_subset(vector<set<char> > &vs,string s);`
que, dado un vector de conjuntos de caracteres `vs` y una palabra `s` retorna la posición del conjunto que más letras de la palabra contenga. En caso de que haya varios que tienen el mismo máximo debe retornar el primero. Si la palabra tiene letras repetidas cuentan una sóla vez.
Ejemplo: `vs= [{a,b},{c}]`, y `s="abccc"` debe retorar `0` ya que si bien `c` aparece tres veces en `s` hay que contarla una vez sóla.

Ayuda:

- Armar un **set<char>** con los caracteres de **s**.
- Recorrer los conjuntos de **vs** y usar la función de intersección de las STL.
- Recordar que el uso de **set_intersection()** es
  set_intersection(A.begin(),A.end(),B.begin(),B.end(),inserter(C.begin()));

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más abajo; 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. El paquete ya incluye el header **tree.h**.
- 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
  ```

- 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(set<int> S,string s="")**.
  - **void Eval::dump(list <int> &L,string s="")**

  - `void Eval::dump(list< list<int> > &LL,string s="")`.
  - `tree<int>::lisp_print()`: Lisp print de un árbol ordenado orientado (AOO). Nota: esta pertenece a la clase `tree`. Uso: `tree<int> T; T.lisp_print();`
  - Idem para AB: `btree<int>::lisp_print()`: Lisp print de un árbol binario (AB). Nota: esta pertenece a la clase `btree`. Uso: `btree<int> T; T.lisp_print();`

- Después del parcial deben entregar el programa fuente (sólo el `program.cpp`) renombrado con su apellido y nombre (por ejemplo `messilionel.pdf`). Primero el apellido.
- **Puntos:** Notar que la suma de los puntos es 120. La nota del parcial `min(sum(Zj),100)` es decir que si haciendo los tres primeros ejercicios se obtienen 95/100 pts. Para aprobar basta hacer 50/100 pts de manera que basta con cualquiera dos de los cuatro.
- `usercase`: Ahora las funciones `eval()` tienen dos parámetros adicionales:
  `Eval::eval(func_t func,int vrbx,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.

  - Los ejercicios 2 (`tricount`) y 3 (`min_com_subtree`) participan del **Torneo**. La tabla de posiciones para el torneo se determina a partir del tiempo que tarda en correr el programa en **50 casos muy grandes**, (por ejemplo 50 grafos aleatorios de 250.000 vértices en el caso de `tricount`).
  - Para ello la función de evaluación por **seed/hash** `evalrj()` tiene un parámetro adicional `int hardness` (dificultad) que puede tomar valores `0<=hardnes<=5`, `Eval::evalr2(tricount,seed,vrbs,hardness);`.
  - `hardness=0` (valor por defecto) corresponde a lo que se toma en el TPL para determinar el funcionamiento correcto, y por lo tanto los contenedores son pequeños.
  - Para `hardness>0` el tamaño va creciendo geométricamente y la función reporta el tiempo que tardó en realizar la tarea. Para `hardness=5` corre el tamaño máximo de los contenedores. Por ejemplo la corrida para los 5 valores de `hardness` reportan

    ```
    EJ2. Hardness 1, elapsed 0.00811529[s]
    EJ2. Hardness 2, elapsed 0.0350215[s]
    EJ2. Hardness 3, elapsed 0.239452[s]
    EJ2. Hardness 4, elapsed 1.57176[s]
    EJ2. Hardness 5, elapsed 10.8449[s] (mips 1.08925, relative 9.95625[s])
    ```

    `elapsed` es el tiempo transcurrido. De esta forma Uds pueden probar diferentes alternativas de programación y ver si reducen o no el tiempo.
  - Para definir los resultados del torneo se correrán todos los programas de los participantes en un **servidor dedicado**, por lo tanto no habrá tiempos de diferencia debido al procesador.
  - Para `hardness=5` se reportan también los `mips` de la máquina (una medida de su eficiencia) y el tiempo relativo (o sea el tiempo real dividido los mips). De esta forma se tiene una medida de la eficiencia del algoritmo independiente de la rapidez del procesador.
  - Como referencia, Para los ejercicios 2 y 3 de este TPL los valores obtenidos con un código implementado en base a la ayuda que hemos dado da los siguientes tiempos.

    ```
    EJ2. Hardness 5, elapsed 11.1808[s] (mips 1.06821, relative 10.4668[s])
    EJ3. Hardness 5, elapsed 5.57989[s] (mips 1.08289, relative 5.15276[s])
    ```