Algoritmos y Estructuras de Datos.
Recuperatorio de Trabajos Prácticos de Laboratorio. [2016-11-10]

PASSWD PARA EL ZIP: `N28G 02VV FQFD`

Ejercicios

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

[Ej. 1] **[max_sublist_m (tpl1 - 25pts)]** Programar una función `int max_sublist_m(list<int> &L, int m)` que reciba una lista de enteros `L` y un entero `m` tal que `0<m<=L.size()`, y encuentre y retorne el valor de la mayor suma de entre todas las posibles sumas de sublistas de longitud m.
Por ejemplo: Si `L: {1,2,-5,4,-3,2}`

- `m=1` ⇒ `4` (por `{4}`)
- `m=2` ⇒ `3` (por `{1,2}`)
- `m=3` ⇒ `3` (por `{4,-3,2}`)
- `m=4` ⇒ `2` (por `{1,2,-5,4}`)
- `m=5` ⇒ `0` (por `{2,-5,4,-3,2}`)
- `m=6` ⇒ `1` (por `{1,2,-5,4,-3,2}`)

[Ej. 2] **[remove_max_sibling (tpl2 - 25pts)]** Basado en la historia bíblica de las 7 plagas de Egipto. En la última plaga Dios ordenó matar todo primogénito Egipcio. Suponga ahora que recibe un árbol `T` con las edades de una familia egipcia, dado un grupo de hermanos, elimine al de mayor edad. Al finalizar el algoritmo, en `T` debe quedar las edades de los sobrevivientes de la familia. Implemente la función `void remove_max_sibling(tree<int>&T)` que dado un AOO `T`, por cada grupo de hermanos elimine aquel de etiqueta mayor que no tenga descendencia. La eliminación debe realizarse en post-orden.
Ayuda:

- Dado un nodo `it`
  - Buscar la etiqueta máxima entre sus hijos sin descendencia (si los hay)
  - Si había hijos sin descendencia, eliminar el de la etiqueta máxima
  - Aplicar recursivamente sobre los descendientes de los hijos no eliminados

Ejemplos:

- `(50 (25 7 4) 35 20)` ⇒ `(50 (25 4) 20)`
- `(50 25 (35 7 4) 20)` ⇒ `(50 (35 4) 20)`

[Ej. 3] **[max_siblings_sum (tpl1/tpl2 - 50pts)]** Programar una función `void max_siblings_sum(tree<int>&T,list<int>&L)` que reciba un AOO y retorne la lista de nodos hermanos (ordenados desde el más izquierdo) que obtenga la mayor suma entre todas sus etiquetas. Si hay varias listas que den la misma suma, se debe retornar la primera en preorden.
Por ejemplo:

- `T = (2 (5 7 2 3)(-1 9))` ⇒ `L = {7,2,3}`
- `T = (2 (5 -7 2 3)(-1 9))` ⇒ `L = {9}`
- `T = (2 (5 -7 2 3)(-1 -9))` ⇒ `L = {5,-1}`
- `T = (2 (5 7 2 3)(-1 12))` ⇒ `L = {7,2,3}`

[Ej. 4] **[max_valid_path (tpl2/tpl3 - 50pts)]** Implementar la función `int max_valid_path(map<int,set<int>>& G, bool (*pred)(int))` que recibe un grafo no dirigido `G` y retorna la longitud del camino más largo (sin repetir vértices) tal que cada uno de los nodos que lo compone satisface el predicado `pred`. Recordar que la longitud de un camino es el número de aristas que lo compone. Si el grafo no tiene ningún nodo que cumpla con el predicado, la función debe retornar `-1`. Por ejemplo, si `G` es:

```
1 -> {2, 4}
2 -> {1, 3, 4}
3 -> {2, 4}
4 -> {1, 2, 3}
```

entonces:

- Predicado `es_impar` ⇒ `0` (desde un nodo impar no hay camino hacia otro impar)
- Predicado `es_par` ⇒ `1` (el camino 2 - 4 o viceversa)
- Predicado `es_menor_a_5` ⇒ `3` (por ejemplo el camino 1 - 2 - 3 - 4)

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas 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 **messilionel.cpp**). Primero el apellido.