Algoritmos y Estructuras de Datos.
TPL2. Trabajo Práctico de Laboratorio 2. [2017-10-12]

PASSWD PARA EL ZIP: MPBR PZXR 5QPP

Ejercicios

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

[Ej. 1] [prom-nivel]
Dado un árbol `tree<int> T`, generar mediante la función
`void prom_nivel(tree<int> &T, list<float> &P);` una lista de reales `P`, donde el primer elemento de la lista sea el promedio de los nodos del árbol de nivel 0, el segundo sea el promedio de los de nivel 1, el tercero el promedio de los de nivel 2, y así sucesivamente. Es decir, que si el árbol tiene profundidad `N`, la lista tendrá `N+1` elementos de tipo `float`.
Ejemplos:

- `T= (5 (10 8 9) (7 3))  ->  P=(5, 8.5, 6.66666666)`
- `T= (23 78 52 (69 50) 28 (79 13)) -> P=(23, 61.2000007629395, 31.5)`

Nota: recuerde que en C++ la división entre enteros siempre arroja un resultado entero (por ejemplo, 7/3 da 2). Para lograr un resultado real puede castear uno o ambos operandos a float: float(7)/3 da 2.33333.
Ayuda: utilizar una o dos estructuras auxiliares donde guardar las sumatorias y los conteos de nodos por nivel (por ejemplo dos mapas, uno de nivel a sumatoria, otro de nivel a cantidad de nodos).

[Ej. 2] **[es-circuito]** Dado un grafo G representado por un map de nodos a lista de vecinos (`map<int,list<int>>`), y una lista de nodos `list<int> L` , escriba una función `bool esCircuito(map<int,list<int>>&G,list<int>&L)` que determine si la secuencia de nodos `L` es un camino dentro del grafo `G`.
Ejemplos:

- `G=(1->{2,4},2->{1,3},3->{2,4},4->{1,3}), L={1,2,3,4} -> true`.
- `G=(1->{2},2->{1,3},3->{2,4},4->{3}), L={1,2,3,4} -> false`.

Ayuda: para cada par de elementos consecutivos en la lista, y también para el par formado por el último y el primero, verificar que exista en el grafo una arista que una a ambos.

[Ej. 3] **[map-graph]** Dado un grafo

```
typedef map<int,vector<int>> graph_t;
graph_t Gin;
```

y una permutación `map<int,int> P`, encontrar el grafo `Gout` que resulta de permutar los vértices de `Gin` por la pertmutación `P`, es decir si la arista `(j,k)` está en `Gin`, entonces la arista `(P[j],P[k])` debe estar en `Gout`.
Consigna: Escribir una función
`void map_graph(graph_t &Gin,map<int,int> &P,graph_t &Gout);`
Ejemplos:

- `Gin= (0->{1}, 1->{0,2,3},2->{1,4},3->{1,4},4->{2,3}),`
  `P=(0->1,1->2,2->3,3->4,4->0)`
  Debe quedar:
  `Gout= (0->{3,4},1->{2},2->{1,3,4},3->{2,0},4->{2,0}),`

- `Gin= (0->{1}, 1->{0,2,3},2->{1,4},3->{1,4},4->{2,3}),`
  `P=(0->1,1->0,2->2,3->4,4->3)`
  Debe quedar:
  `Gout= (0->{1,2,4},1->{0},2->{0,3},3->{2,4},4->{0,3}),`

Nota: En `Gout` los vecinos del grafo deben quedar ordenados de menor a mayor.
Ayuda: Con un doble lazo recorrer las aristas de `Gin`. Para cada arista `(i,j)` en `Gin` agregar la arista `(P[i],P[j])` en `Gout`.

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.

mstorti@galileo/aed-3.1-84-g1d9b1d7b/Thu Oct 12 09:06:40 2017 -0300