Algoritmos y Estructuras de Datos.
TPL3. Trabajo Práctico de Laboratorio. [2016-11-03]

PASSWD PARA EL ZIP: 6N4V CLD8 2E43

Ejercicios

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

[Ej. 1] **[puede-simplificar (33pt)]** Se utiliza un árbol binario `btree<string> T` para representar una expresión matemática, donde los nodos interiores representan operadores binarios, y los nodos hoja operandos (variables o constantes).
Por ejemplo, la expresión `3*q+3*(f-1)` se representa con el árbol `(+ (* 3 q) (* 3 (- f 1)))`".
Escriba una función `bool puede_simplificar(btree<string> T, string fc);` para determinar si la expresión que representa el árbol puede simplificarse extrayendo `fc` como factor común.
Notar que: `fc` es un factor común válido si: o bien a) el nodo raíz del árbol es un producto, y uno de sus dos hijos es el valor `fc`, o bien b) la etiqueta del nodo raíz es `+` o `-` y `fc` es un factor común válido para los dos subárboles que cuelgan a derecha e izquierda del mismo.
Algoritmo sugerido:
Dado un árbol `T`, un iterador `it`, y un factor `fc`:

- Si `it` apunta a una hoja,
  - Si su etiqueta es `fc` retornar verdadero
  - Sino retornar falso
- Si la etiqueta de `it` es `*`
  - Invocar la función recursivamente para ambos hijos, y
  - Si alguna de las llamadas recursivas retorna verdadero $\Longrightarrow$ retornar verdadero
- Si la etiqueta de `it` es `+` o `-`
  - Invocar la función recursivamente para ambos hijos, y
  - si ambas llamadas recursivas retornan verdadero $\Longrightarrow$ retornar verdadero
- Retornar falso

[Ej. 2] **[prune-and-return (33pt)]**
Implemente la función `void prune_and_return(btree<int>&T, int v, list<int>& L);` que dado un árbol binario `T` busque el nodo cuya etiqueta sea `v` y:

- Retorne en `L` la lista en preorden de todos los nodos del subárbol que lo tiene a él como raíz.
- Elimine el nodo del árbol.

Notar que si existen varios nodos con la misma etiqueta, el proceso debe hacerse sobre el primero de estos nodos al listar el árbol en orden prefijo. Si el nodo no existe se retorna la lista L vacía.
Hints:

- Utilice una función auxiliar `void preorden(btree<int>&T, btree<int>::iterator it, list<int>& L);` que almacene la etiqueta de los nodos del subárbol de `T` con raíz en `it` en la lista `L`.
  El algoritmo recursivo es el siguiente:
  Si el subárbol no es vacío:
  - Si la lista a completar aún está vacía

    - Si la etiqueta es la buscada
        - Listar el subárbol en preorden
        - Eliminar el subarbol
  - Sino
    - Verificar el subárbol izquierdo
    - Verificar el subárbol derecho

Ejemplos:

- Si recibe `T= (8 (10 4 2) (5 . (7 9 10)))` con `v=5`, retorna `T = (8 (10 4 2) .), L = (5 7 9 10)`
- Si recibe `T= (8 (10 4 2) (5 . (7 9 10)))` con `v=15`, retorna `T = (8 (10 4 2) (5 . (7 9 10))), L = ()`
- Si recibe `T= (8 (10 4 2) (5 . (7 9 10)))` con `v=8`, retorna `T = (), L = (8 10 4 2 5 7 9 10)`

[Ej. 3] [gather-sets (33pt)]

Dado un vector de conjuntos `vector<set<int>> VS;` y un predicado `bool pred(int);` retornar la unión `set<int> S` de todos los conjuntos que contienen al menos un elemento que satisface el predicado.
`void gather_sets(vector<set<int>> &VS,bool (*pred)(int),set<int> &S);`
Ayuda:

- Limpiar `S`
- Para conjunto `R` de `VS`, verificar si `R` contiene algún elemento que satisface el predicado.
- Si contiene tal elemento hacer `S = S` $\cup$ `R` (Acordarse de que se en este caso se debe usar un temporario en las operaciones binarias).

Ejemplo:
Si `VS=[{1,3}{1,2,3},{2},{2,3,4},{2,4,6,8},{3,5,7}]` entonces

- `gather_sets(VS,even,S) => S={1,2,3,4,6,8}`
- `gather_sets(VS,odd,S) => S={1,2,3,4,5,7}`

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-3.0-58-g66c7314/Wed Nov 2 14:14:06 2016 -0300

TPL3. Trabajo Práctico de Laboratorio. [2016-11-03]. TABLA SEED/HASH

```
S=123 -> H1=961 H2=891 H3=140
S=577 -> H1=802 H2=161 H3=073
S=393 -> H1=493 H2=850 H3=726
S=686 -> H1=677 H2=068 H3=527
S=349 -> H1=764 H2=586 H3=094
S=762 -> H1=662 H2=724 H3=605
S=690 -> H1=262 H2=686 H3=198
S=663 -> H1=560 H2=392 H3=178
S=340 -> H1=741 H2=765 H3=439
S=872 -> H1=512 H2=785 H3=651
S=711 -> H1=672 H2=043 H3=912
S=367 -> H1=501 H2=497 H3=498
S=882 -> H1=505 H2=198 H3=602
S=162 -> H1=524 H2=815 H3=734
S=767 -> H1=357 H2=766 H3=725
S=429 -> H1=727 H2=734 H3=453
S=622 -> H1=317 H2=279 H3=870
S=969 -> H1=463 H2=413 H3=328
S=893 -> H1=504 H2=253 H3=871
S=311 -> H1=334 H2=373 H3=195
S=529 -> H1=097 H2=428 H3=338
S=721 -> H1=295 H2=875 H3=372
S=384 -> H1=877 H2=778 H3=479
S=798 -> H1=141 H2=952 H3=650
S=615 -> H1=377 H2=849 H3=096
S=386 -> H1=626 H2=046 H3=324
S=215 -> H1=317 H2=507 H3=361
S=935 -> H1=106 H2=260 H3=285
S=292 -> H1=685 H2=918 H3=178
S=821 -> H1=156 H2=863 H3=884
S=527 -> H1=017 H2=709 H3=557
S=359 -> H1=851 H2=797 H3=847
S=626 -> H1=719 H2=119 H3=618
S=226 -> H1=978 H2=893 H3=823
S=236 -> H1=755 H2=306 H3=748
S=468 -> H1=300 H2=191 H3=308
S=529 -> H1=097 H2=428 H3=338
S=630 -> H1=939 H2=208 H3=956
S=923 -> H1=533 H2=709 H3=323
S=335 -> H1=364 H2=155 H3=784
S=802 -> H1=568 H2=778 H3=859
S=958 -> H1=067 H2=936 H3=424
S=967 -> H1=915 H2=338 H3=882
S=656 -> H1=950 H2=354 H3=546
S=242 -> H1=331 H2=261 H3=679
S=973 -> H1=274 H2=680 H3=658
S=219 -> H1=400 H2=620 H3=268
S=437 -> H1=391 H2=673 H3=449
S=624 -> H1=900 H2=712 H3=185
S=670 -> H1=231 H2=947 H3=166
```