Algoritmos y Estructuras de Datos.
TPL1. Trabajo Práctico de Laboratorio. [2015-08-29]

PASSWD PARA EL ZIP: ZF9 K22 3RU Y8M

Ejercicios

[Ej. 1] **[iscomb]** Dado un vector de listas `VL` y una lista `L`, determinar si `L` es una combinación de las listas de `VL` en alguna permutación dada. Cada una de las `VL[j]` debe aparecer una y sólo una vez en `L`. Por ejemplo si `VL=[(1,2,3),(4,5,6),(7,8)]`, y `L=(7,8,4,5,6,1,2,3)` entonces `iscomb(VL,L)->true`. Pero si `L=(3,2,1,7,8,4,5,6)->false`.

Ayuda:

- Escribir una funcion `bool isprefix(Lpref,L);` que retorna `true` si `Lpref` es prefijo de `L`.
- Primero chequear que la longitud de `L` debe ser igual a la suma de las longitudes de los `VL[j]`.
- Si pasa este test se procede en forma recursiva. Para cada `VL[j]` se determina si es prefijo de `L`. Si ninguna de ellas es prefijo entonces retorna `false`. Si una (o varias de ellas) lo es entonces se aplica recursivamente `iscomb()` para determinar si `L-VL[j]` es una combinación de `VL-VL[j]`. Es decir se quita `VL[j]` del comienzo de `L` y la posicion `j` de `VL`. La recursión se corta cuando `L` es vacía.

[Ej. 2] **[max-sublist]** Programar una función `list<int> max_sublist(list<int> &L);` la cual recibe una lista de enteros y encuentra y retorna la sublista consecutiva `Lmax` que obtenga la mayor suma entre todos sus elementos. Notar que debido a que algunos elementos pueden ser negativos el problema no se resuelve simplemente tomando todos los elementos. También es posible que la sublista resultado no contenga ningún elemento, en el caso de que todos los elementos de `L` sean negativos.

Por ejemplo:

```
(1, 2, -5, 4, -3, 2) -> (4)
(5, -3, -5, 1, 7, -2) -> (1,7)
(4, -3, 11, -2) -> (4, -3, 11)
(4  -4  2 2) -> (4)
```

Si hay varias sublistas que den la misma suma mínima debe retornar la primera y la más corta. En el último ejemplo de arriba hay tres sublistas que dan la máxima suma de 4: a saber `(4)`, `(4 -4 2 2)`, `(2 2)`. Por este criterio debe retornar `(4)` ya que es la que empieza primero y la más corta.

[Ej. 3] **[mergesort]** Programar una funcion `void mergesort(list<int> &L);` que reciba una lista `L` desordenada y la ordene en forma ascendente mediante la siguiente estrategia recursiva:

- Si la lista está vacía o de un elemento ya está ordenada
- Sino se parte la lista en dos sublistas y se las ordena a cada una de forma recursiva. Luego se mezclan (fusionan) cada una de las sublistas ya ordenadas.
- Para partir una lista puede utilizarse el método de la clase list `void splice (iterator position, list& x, iterator first, iterator last);` el cual transfiere los elementos desde la lista `x` al contenedor que realiza la llamada en la posición position.
- Se sugiere escribir una función `void merge(list<int> &L1, list<int> &L2, list<int>& L);` la cual recibe dos listas ordenadas de forma ascendente `L1` y `L2` y retorna una lista `L`, pasada como parámetro, con los elementos de ambas ordenados tambien en forma ascendente. Por ejemplo si `L1=(1, 3, 6, 11)` y `L2=(2, 4, 6, 10)`, la lista `L` debe quedar como `(1, 2, 3, 4, 6, 6, 10, 11)`.

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.
- 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 una función de evaluación, por ejemplo si **`fj`** son las funciones a evaluar tenemos

```
Eval ev;
int vrbs = 0;
ev.eval1(f1,vrbs);
ev.eval2(f2,vrbs);
ev.eval3(f3,vrbs);
```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las función (**`eval1`**). La primera **`ev.evalj(fj,vrbs);`** toma una serie de casos de prueba de entrada, le aplica la función del usuario **`fj`** 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

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

- 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 ev; ev.dump(VX);`**. El string **`s`** es un label opcional.
  - **`void Eval::dump(list <int> &L,string s="")`**
  - **`void Eval::dump(vector< list<int> > VL,string s="")`**.
- 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.

mstorti@galileo/aed-3.1-211-g839bcb5e/Thu Aug 23 23:00:21 2018 -0300