Algoritmos y Estructuras de Datos.
TPL1R. Recuperatorio Trabajo Práctico de Laboratorio. [2015-09-12]

PASSWD PARA EL ZIP: `Y69 8LP BPM EP9`

Ejercicios

[Ej. 1] **[mochila]** Se necesita llenar una mochila, incapaz de soportar más de un peso determinado `K`, con todo o parte de un conjunto de objetos, cada uno con un peso y valor específicos. Los objetos colocados en la mochila deben maximizar el valor total sin exceder el peso máximo. En esta versión simplificada, se supone que cada objeto tiene el mismo valor, pero pueden tener diferentes pesos. Por lo tanto se trata de **elegir la mayor cantidad de objetos sin exceder** `K`. El peso de los objetos (enteros positivos) está en una lista `P`.

Consigna: Escribir una función `list<int> mochila(list<int> &P, int K);` que reciba una lista `P` con el peso de cada uno de los objetos que se podrían llevar en la mochila y un peso máximo permitido `K` y que retorne una lista `M` con una lista con un subconjunto de los elementos de `P` cuya suma es `<=K` y tiene la máxima cantidad de elementos. Si hay varias posibilidades debe retornar cualquiera de ellas.

Ejemplos:

- Si `P=(3,2,7,11,2,5,9,4), K=11 -> M=(3,2,2,4)`
- Si `K=10 -> M=(3,2,2)` o `M=(4,2,2)` o `M=(5,2,2)`
- Si `K=12 -> M=(2,2,3,4)`

Ayuda:

- Escribir el algoritmo en forma recursiva. Si `K<=0` o `P=()` entonces la solución es la lista vacía.
- Tomar el primer elemento `a0` de `P` y formar la lista `Pm0=P-(a0)` es decir `P` pero sin el elemento `a0`.
- Considerar dos casos, que `a0` esté o no en `M`. Si `a0>K` no hace falta considerar la posibilidad de que `a0` esté en `M`.
- En el caso que `a0` **NO** esté en `M` llamar a `mochila(Pm0,K)`.
- En el caso que `a0` **SI** esté en `M` la suma de los elementos de `Pm0` debe sumar `K-a0` de manera que se debe llamar a `mochila(Pm0,K-a0)`. A la `M` resultante debe agregársele `a0`.
- De las dos posibilidades quedarse con la que tiene la mochila con más elementos.

Notas:

- Los elementos en `M` pueden estar en cualquier orden.
- Puede haber elementos iguales en `P`.
- Puede darse `K==0` en ese caso debe retornar `M=()`.

[Ej. 2] **[enunosolo]** Dada una lista de listas `LL` y una lista `L`, buscar que cada elemento de `L` esté contenido en una y sólo una lista de `LL`. La función propuesta es
`bool contenido(list <list<int> > &LL, list <int> &L);` Si el elemento de `L` no se encuentra en ninguna lista contenida en `LL`, deberá retornar `false`.

Ejemplos: Considerando `LL=((1 2 3 4) (1 2 3 5) (1 2 3 6))`.

- Si **L=(6 5 4) -> true** ya que 6 está sólo en la lista **L2**, 5 en **L1** y **4** en **L0**
- Si **L=(6 5) -> true** ya que 6 está sólo en la lista **L2**, 5 en **L1**. No hay ningún elemento único para la lista **L0** pero no importa.
- Si **L=(6 5 1) -> false**, ya que 1 está en todas las listas.
- Si **L=(6 5 3) -> false**, ya que 3 está en todas las listas.
- Si **L=(6 5 3 7) -> false**, ya que 3 está en todas las listas y 7 en ninguna.

Ayuda:

- Escribir una función auxiliar
  bool contiene(list<int>&L,int x), que retorna **true** si **x** está en **L**.
- Para cada elemento **x** de **L** recorrer todas las listas de **LL** y contar en cuantas de ellas está **x**. Si no está exactamente una vez retornar **false**.
- Si todos los elementos de **L** pasan la condición entonces retorna **true**.

[Ej. 3] **[ppt]** Escriba una función
int ppt(list<int> &H, int n); que reciba una lista de enteros que representan todas las elecciones previas del jugador oponente en el juego **piedra-papel-tijera** y devuelve la siguiente más probable **nextplay**. La lista solo tiene los enteros 1, 2 y 3, que se corresponden con las elecciones de “piedra”, “papel” y “tijera” respectivamente. El último elemento de la lista corresponde a la última jugada. El objetivo es tratar de predecir la próxima jugada **nextplay** del oponente. Para ello debe buscar en su historial **H** todas las veces que el jugador jugó la misma secuencia que en las últimas **n** partidas y retornar el valor siguiente a dicha vez que apareció la secuencia.

Por ejemplo, si el historial es **H=(1,1,2,2,1,2,3,3,2,1,2,3,1,2)** y **n=2** la función debería retornar 3. Las últimas dos jugadas fueron **last=(1,2)**, y la anterior vez que jugó **(1,2)** eligió a continuación **nextplay=3**.

En caso de que la secuencia **last** no aparezca debe retornar **nextplay=0**.

Ayuda:

- Copie en una lista auxiliar **last** las últimas **n** jugadas.
- Recorriendo **H** para cada posición **p** verificar si a partir de **p** se encuentra una copia de **last**.
- Si se encuentra y hay al menos un elemento adicional en **H** quedarse con ese candidato **nextplay**.
- De todos los posibles **nextplay** quedarse con el último.

Notas:

- Tener en cuenta que dos apariciones de la secuencia buscada pueden solaparse, por ejemplo si **H=(1 2 1 2 1 2 3 1 2 1 2)** y **n=4** tenemos las secuencias **(1 2 1 2 1)** y **(1 2 1 2 3)**. Se recomienda utilizad un iterador **p** que va recorriendo la lista y para cada **p** recorrer con otro iterador **q** desde **p** hacia atrás.
- Tener en cuenta que al final de **H** aparece la secuencia buscada, pero no debe tenerse en cuenta porque no tiene un elemento siguiente.

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.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **`f`** es la función a evaluar tenemos

  ```
  ev.evalj(f,vrbs);
  hj = ev.evalrj(f,seed); // para SEED=123 debe dar Hj=170
  ```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**`eval1`** y **`evalr1`**). La primera **`ev.evalj(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 **`evalrj`** 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 **`evalrj()`** 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="")`**
  - **`void Eval::dump(list< list<int> > &LL,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.