Algoritmos y Estructuras de Datos.
Trabajo Práctico de Laboratorio 1. [2019-09-05]

PASSWD PARA EL ZIP: 364C 29H6 RGV8

Ejercicios

[Ej. 1] [Pareto] (lista)

Se dice que un punto $x = (x_1, x_2, .., x_n)$ n-dimensional domina a otro punto $y = (y_1, y_2, .., y_n)$ si se cumple que:

$$\forall k \in [1, n] : x_k \leq y_k, \text{ y } \exists k_0 \ \in \ [1, n] \ / \ x_{k_0} < y_{k_0} \tag{1}$$

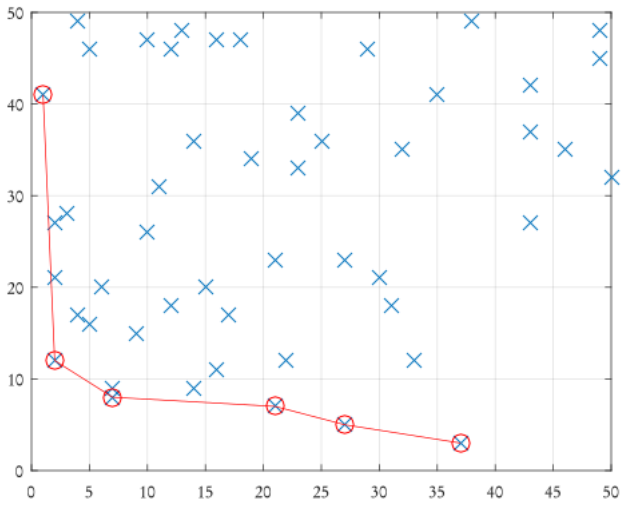
Por ejemplo **x=[2,1,5]** domina a **y=[2,2,5]**, pero dado **z=[1,2,5]** no es cierto ni que **x** domina a **z** ni **z** domina a **x**.

Entonces, dada una lista de puntos de coordenadas enteras positivas **`list<vector<int>> L`**, implemente una función **`list<vector<int>> Pareto(list<vector<int>>&L);`** que retorne la lista de puntos no-dominados (aquellos que no son dominados por ningún otro punto). Se garantiza que cada **`vector<int>`** tiene **n** coordenadas. La lista de retorno debe devolver los vectores en el mismo orden en que se encontraban en **L**.

Nota: Formalmente, lo que se solicita es que retorne los puntos que pertenecen a la frontera de Pareto, muy utilizada en problemas de optimización con múltiples objetivos.

Ejemplo: Si **L=[[2,1,5],[2,2,5],[1,2,5]]** en tonces **pareto(L)** debe retornar **[[2,1,5],[1,2,5]]**, ya que **[2,2,5]** es dominado por **[2,1,5]** pero **[2,1,5]** y +[1,2,5]+ no se dominan entre sí.

Ayuda: Se sugiere implementar una función auxiliar **`bool domina(vector<int> x, vector<int> y);`** que reciba dos puntos n-dimensionales y que retorne verdadero en caso de que **x** domine a **y**.



[Ej. 2] [merge-kway] (cola)

Implementar una función void **`merge_kway(vector< queue<int> > &qords,queue<int> &merged)`** que dado un vector de colas ordenadas **ordqs**, obtener la cola **merged** resultante de la fusión de todas las colas en una sola, de forma de que los elementos siguen ordenadas. Por ejemplo si **`ordqs=((1,3,5,6),(0,3,5,8),(6,9,10))`** entonces debe dar **`merged=(0,1,3,3,5,5,6,6,8,9,10)`**. El algoritmo puede ser destructivo sobre **ordqs**.

Ayuda: Se sugiere seguir el siguiente algoritmo:

- Eliminar todas las colas vacías del vector.
- Si no quedaron colas (estaban todas vacías) el proceso terminó.
- Recorrer todos los frentes de la colas, buscando el frente menor. Retener el índice de la cola que contiene el frente menor. Por ejemplo en el caso anterior daría el índice 1 ya que el frente menor es el elemento 0 que está en la cola 1.
- Sacar el elemento de la cola correspondiente e insertarlo en **`merged`**.
- Repetir hasta que todas las colas estén vacías.

[Ej. 3] **[is-balanced] (stack)** Dado un string **`expr`**, implementar una función **`bool is_balanced(string &expr);`** que determine si el par y orden de **`{`**, **`}`**, **`(`**, **`)`**, **`[`**, **`]`** está **balanceado**. Ejemplos:

- **`[ ( ) ] { } { [ ( ) ( ) ] ( ) }`** está **balanceado**
- **`[ ( ] )`** no está **balanceado**.

Ayuda: Se propone el siguiente algoritmo:

- Declarar una pila de **`char S`**.
- Recorrer el string **`expr`**.
- Si el caracter actual es un símbolo de apertura, entonces apilarlo.
- Si el caracter actual es un símbolo de cierre y matchea con su símbolo de apertura, entonces quitarlo de la pila. Si no retornar falso. Atención: Chequear que la pila no esté vacía.
- Una vez recorrido todo el string, si la pila está vacía, entonces la expresión está balanceada.

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`**.
- 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.