Curso de Programación en C++.
TPL2-2015. Trabajo Práctico de Laboratorio 2. [2015-06-17]

PASSWD PARA EL ZIP: `67P 1LY IPM K1N`

Ejercicios

[Ej. 1] **[ppt-game]** Consideremos el jugo de **piedra, papel, tijera** (**ppt**). Los jugadores se basan en la siguiente clase

```
class ppt_player_t {
  public:
  virtual int play(vector<int> &you,vector<int> &me)=0;
};
```

- Las jugadas son un entero representando 0=piedra, 1=papel, 2=tijera. Dada una jugada `j` la ganadora corriente es `(j+1)%3` y la perdedora es `(j+2)%3`.
- El jugador recibe la historia de jugadas del oponente `vector<int> you` y propias `vector<int> me`. Debe devolver su nueva jugada en base a estas dos o puede ser aleatoria. Escribir las siguientes implementaciones
  - `random_t`: juega random (usar la función `rand()` de `<cstdlib>`
  - `fixed_t`: juega siempre lo mismo (piedra por ejemplo).
  - `repeater_t`: piensa (ingenuamente) que es mejor no jugar la jugada anterior. Es decir si en la jugada anterior jugó piedra, entonces en la siguiente elegirá random entre papel y tijera.
  - `repeater_killer_t`: Si el oponente es un `repeater_t` entonces puede basarse en su anterior jugada para mejorar su estrategia. Por ejemplo, si el oponente P1 jugó `j1=0` (piedra), entonces en la siguiente P2 sabe que jugará 1=papel o 2=tijera, por lo tanto le conviene jugar 2=tijera, ya que con esa jugada gana o empata. Es decir la jugada ganadora para P2 es `(j1+2)%3`.
- Implementar una función

  ```
  vector<int>
  ppt_game(ppt_player_t &p1,ppt_player_t &p2,int Nrounds=1000);
  ```

  que toma dos jugadores `p1` y `p2` y los enfrenta en `Nrounds` rondas y devuelve un vector de 3 enteros que contabilizan las veces que ganó `p1`, `p2` y empates. `ppt_game()` además debe mantener la historia de las jugadas de `p1` y `p2` y pasárselas apropiadamente `(his2,his1)` a `p1` y `(his1,his2)` a `p2`.
- Enfrentar jugadores de diferente tipo y verificar los siguientes resultados (los porcentajes son: gana P1, gana P2, empate):

  ```
  random vs. random: 33% 33% 33%
  repeater vs. repeater: 33% 33% 33%
  repeater vs. repeater_killer: 0% 50% 50%
  repeater_killer vs. repeater_killer: 0% 0% 100%
  ```

[Ej. 2] **[map2freq]** Dado una lista de conjuntos, encontrar la frecuencia con la cual un elemento dado aparece en los conjuntos (es decir la cantidad de conjuntos a los cuales pertenece).

`void map2freq(list< set<int> > &LS, map<int,int> &freq);`
La correspondencia **freq** es tal que **freq[x]** es la cantidad de conjuntos en los cuales **x** aparece. Por ejemplo

```
LS[0]: 2 3 4
LS[1]: 1 4
LS[2]: 2 4
LS[3]: 2 3
LS[4]: 0 4
LS[5]:
LS[6]: 0
freq:
M[0] = 2
M[1] = 1
M[2] = 3
M[3] = 2
M[4] = 4
```

[Ej. 3] **[is-perm-fun]** Dado un conjunto **S** y una función de mapeo **map_fun_t f** determinar si el conjunto **fS** resultado de aplicar **f** a los elementos de **S** es una permutación de los mismos, es decir si **fS==S**. Por ejemplo si **S={0,1,2,3...,9}** entonces la función **int f(x) { return 9-x; }** es una permutación ya que genera los elementos **fS={9,8,...,0}** que es igual a **S**.

```
typedef int (*map_fun_t)(int k);
bool ispermfun(set<int> &S,map_fun_t f);
```

Los conjuntos utilizados en los ejemplos tienen todos elementos en **[0,10)** y las funciones de mapeo son:

```
const int N1=10;
typedef int (*map_fun_t)(int k);
int f1(int x) { return N1-x-1; }
int id(int x) { return x; }
int shift2(int x) { return (x+2)%N1; }
```

[Ej. 4] **[filt-by-img]** Dada una correspondencia (**map<int,int> M**) hacer una copia de la misma con sólo aquellos pares de asignación para los cuales la **imagen** satisface un cierto predicado
`bool pred(int x);`
`void filtbyimg(map<int,int> &M,map<int,int> &M2,pred_t pred);`
Por ejemplo, si el predicado es **even()** entonces debemos tener:

```
M[0] = 4
M[5] = -2
M[6] = 1
M[7] = 4
M[8] = 1
M[9] = 2
predicate: even ->
M2[0] = 4
```

```
M2[5] = -2
M2[7] = 4
M2[9] = 2
```

Los predicados utilizados en los ejemplos son:

```
typedef bool (*pred_t)(int x);
bool even(int x) { return x%2==0; }
bool odd(int x) { return x%2>0; }
bool positive(int x) { return x>0; }
```

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.
- Hay una función de evaluación, por ejemplo si **`f`** es la función a evaluar tenemos

  ```
  ev.eval1(f,vrbs);
  ```

  `ev.eval1(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

mstorti@galileo/texstuff-1.4.0-68-g058255cf/Tue Jun 22 10:34:49 2021 -0300