Curso de Programación en C++.
TPL1. Trabajo Práctico de Laboratorio 1. [2015-05-20]

PASSWD PARA EL ZIP: `88I J4P P7H JKC`

Ejercicios

[Ej. 1] **[countsets]** Dado un vector de vectores `vector<vector<int> > &VV` escribir una función `int countsets(vector< vector<int> > &VV,int x);` que retorna la cantidad de vectores en los cuales `x` está presente al menos una vez. Por ejemplo si

```
VV[0]=[3 4 1 4]
VV[1]=[1]
VV[2]=[4 1]
VV[3]=[1 3 3 0]
VV[4]=[3 4 0]
VV[5]=[]
VV[6]=[]
VV[7]=[2 3 3 2]
VV[8]=[1 0 4 0]
VV[9]=[0 2 2 0]
```

entonces `countsets(VV,4)` debe retornar 4 ya que el número `x=4` aparece en los vectores `VV[0], VV[2], VV[4], VV[8]`.

[Ej. 2] **[max-seq]** Dado un vector de enteros `w` y un entero `m` calcular todas las sumas de las subsecuencias de longitud `m` y retornar el máximo de ellas. Por ejemplo, si `w=[1 3 4 1 2 3 0 2 5 5 2 2]` y `m=5`, entonces las subsecuencias de 5 elementos son `[1 3 4 1 2]`, `[3 4 1 2 3]`, ..., `[2 5 5 2 2]`. De las cuales, las sumas son 11, 13, ..., 16. Por lo tanto la función debe retornar el máximo que es 16.
Consigna: Escribir una función `int max_mseq(vector<int> &w,int m);`

[Ej. 3] **[badpwd]** Bob es perezoso y cuando el sistema le pide que cambie el password simplemente permuta las sílabas del mismo. Por ejemplo si el password era **bazinga** tratará de usar **zingaba** o **gabazin**. La administradora del sistema Alice, quiere evitar esto, ya que si un individuo malicioso tuvo acceso al viejo password podrá probar con un reducido número de combinaciones y deducir el nuevo. Sin embargo tampoco quiere prohibirle permutaciones arbitrarias del viejo password ya que sería demasiado restrictivo. Después de mucho cavilar decide no permitir passwords que sean permutaciones de hasta 3 substrings del viejo password. Es decir si el viejo password es **bazinga**, entonces **gazinba** no es aceptable ya que se puede obtener dividiendo a **bazinga** en **ba|zin|ga** y permutar sus componentes. Pero si es permitido **agnizab** (**bazinga** al revés).

Por lo tanto Alice solicita que escribas una función `int badpwd(string &oldpwd,string &newpwd);` que retorna **1** si **newpwd** se puede obtener de **oldpwd** dividiéndolo en a lo sumo 3 substrings y permutándolos y **0** caso contrario.

Ayuda:

- Sea `N` la longitud de `oldpwd`, entonces recorrer todas las posibles tripletas de enteros `n1,n2,n3` tales que `N==n1+n2+n3` (`0<=n1,n2,n3<=N`).
- Extraer los substrings correspondientes de `oldpwd` y realizar las permutaciones correspondientes.
- Si `s1,s2,s3` son los substrings, las 6 posibles permutaciones son

```
s1+s2+s3, s1+s3+s2,
s2+s1+s3, s2+s3+s1,
s3+s1+s2, s3+s2+s1
```

- Para reocorrer las tripletas mencionadas `n1,n2,n3` lo más fácil es hacer un lazo triple anidado sobre los 3 índices entre `[0,N]` y chequear si la tripleta satisface la condición `N==n1+n2+n3`.

[Ej. 4] **[larger-odd-subvec]** Dado un vector de vectores de enteros `VV` determinar para cada vector `VV[j]` la subsecuencia de elementos impares `odd(VV[j])`, y de todos ellas aquella que tiene longitud máxima `oddmax`.

```
void larger_odd_subvec(vector< vector<int> > &VV,vector<int> &oddmax);
```

```
VV[0]= [3 8 7 4 7 4 4 1 3 4 6],          odd(VV[0])= [3 7 7 1 3]
VV[1]= [4 1 8 0 6 5 1 4 8 4 4],          odd(VV[1])= [1 5 1]
VV[2]= [7 2 7 6 1 3 1 0 4 1 3 7 3 8 3],  odd(VV[2])= [7 7 1 3 1 1 3 7 3 3]

oddmax = odd(VV[2]) = [7 7 1 3 1 1 3 7 3 3]
```

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-1.2.8-61-g36ebe6a/Wed May 20 14:48:14 2015 -0300