UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructuras de Datos I

Segundo cuatrimestre de 2018

22 de Octubre de 2018

Taller de Strings

Resolver los siguientes ejercicios en la computadora. Para los mismos utilizaremos la clase string [1]. Solo está permitido operar sobre ellos utilizando `size()`, `push_back()` y `clear()`.

Los alumnos deben:

- generar el proyecto CLion vacío;
- incorporar al mismo todos los archivos fuentes de los ejercicios;
- incorporar los archivos de tests;
- descargar de la página del laboratorio el GTest comprimido e incorporarlo al proyecto, como se vino realizando a lo largo de los laboratorios precedentes;
- definir el path en el CMakeLists.txt para que se puedan leer los archivos de los tests de risas.

1. split

```
vector<string> split(string s, char delim);
```

Separar el string según el delimitador pasado por parámetro.

Por ejemplo, `split("la casa está en orden", ' ')` debe devolver `<"la", "casa", "está", "en", "orden">`.

2. darVueltaPalabra

```
string darVueltaPalabra(string s);
```

Dado un string solo conformado por letras y espacios, devolver otro que resulte de dar vuelta el orden de las palabras.

Por ejemplo, `darVueltaPalabra("hola mundo")` debe devolver `"mundo hola"`.

3. subsecuencia

```
bool subsecuencia(string s, string t);
```

Decidir si **s** es subsecuencia de **t**. Un string **s** es subsecuencia de **t** si existen índices $i_1 < i_2 < i_3 ... < i_{|s|}$ tales que $t[i_1] = s[0],\ t[i_2] = s[1] \ldots, t[i_{|s|}] = s[|s| - 1]$

Por ejemplo, `subsecuencia("ace", "abcdec")` debe devolver **true** pero `subsecuencia("abdc", "abcecde")` debe devolver **false**.

4. agruparAnagramas

```
vector<vector<string>> agruparAnagramas(vector<string> v);
```

Dado un vector de strings devolver un `vector<vector<string>>` tal que se cumpla que:

- Todos los strings del vector de entrada están en exactamente un subvector resultado (y el resultado no tiene strings que no estén en la entrada);
- Cada elemento (vector) del resultado contiene solo anagramas;
- Se mantiene el orden relativo de cada palabra dentro de cada vector y el orden de cada vector responde a la primera aparición de un nuevo anagrama.

[1] http://www.cplusplus.com/reference/string/string/

Por ejemplo, `agruparAnagramas({"ab", "cd", "ef", "ba", "ab", "dc"})` debe devolver `{{"ab", "ba", "ab"}, {"cd", "dc"}, {"ef"}}`

5. esPalindromo

```
bool esPalindromo(string s);
```

Decidir si un string es palíndromo, es decir, si se escribe igual en ambas direcciones.

6. tieneRepetidos

```
bool tieneRepetidos(string s)
```

Decidir si un string tiene algún caracter repetido.

7. rotar

```
string rotar(string s, int j);
```

Devolver un string que esté rotado j veces hacia la derecha. Por ejemplo `rotar("hola", 7)` debe devolver `"olah"`.

8. darVueltaK

```
string darVueltaK(string s, int k);
```

Dado un string `s`, dar vuelta cada `k` caracteres el substring de longitud `k`. Si la cantidad de caracteres de la entrada no es múltiplo de `k`, el último substring (de longitud menor a `k`) también debe darse vuelta.

Por ejemplo, `darVueltaK("abracadabra", 3)` debe devolver `"rbaacabadar"`.

9. Análisis de la risa

Científicos rusos están investigando propiedades de la risa. Para ello, analizan el habla humana e identifican qué sonidos pertenecen a una risa.

Estos científicos ya hicieron un software que traduce el habla humana en texto. Consideran que la risa es una secuencia alternada de las letras "h" y "a". Por ejemplo, "ahahaha", "hah" y "a" son risas, pero "abacaba" y "hh" no.

Dado un string s encontrar la longitud de la risa más larga, es decir, un substring de s que sea risa y que sea más larga que el resto de las risas.

Deben implementar la función en el archivo `risas.cpp`:

```
int risaMasLarga(string s)
```

Ejemplos

- `risaMasLarga("ahaha")` debe ser igual a 5.
- `risaMasLarga("ahahrunawayahahsofasthah")` debe ser igual a 4.
- `risaMasLarga("ahahaahaha")` debe ser igual a 5.

Datos y Tests

En la carpeta `risas` se encuentran dos archivos de prueba como ejemplo del formato de entrada que sus funciones de E/S deben saber levantar. En el archivo `risas.cpp` hay también dos funciones que es preciso implementar para poder leer los datos. Este formato consiste en una sola línea donde cada caracter está separado por un espacio. Por ejemplo, el contenido del archivo `01.in` es:

```
a h a h a
```

La respuesta esperada para esa entrada se encuentra en el archivo `01.out` cuyo contenido es un entero indicando la risa más larga.

En la carpeta test, el archivo `risasTEST.cpp` contiene un gran número de tests que debe cumplir el código.