Algoritmos y Complejidad
Tarea #3
“Recursion is beautiful”

Algorithm Knaves

15 de octubre de 2018

Un hermoso ejemplo de recursión es el programita de Pike para reconocer una versión recortada de expresiones regulares. Vea la explicación de Kernighan para detalles. Reconoce solo las siguientes construcciones:

c	Calza el caracter ’c’ (salvo los especiales a continuación)
.	Calza cualquier caracter
^	Calza el comienzo del *string*
$	Calza el final del *string*
*	Cero o más ocurrencias del caracter anterior

El programa es el del listado 1.

1. Escriba una versión rudimentaria de `grep(1)`, al que se le llama como:

   `20182t3` <*expresión*> <*archivo*>

   que escriba todas las líneas en que la <*expresión*> calza. No escriba nada más.

   (20 puntos)

2. Modifique el código dado para agregar la operación ’+’ (una o más veces lo anterior).

   (25 puntos)

3. Modifique el código dado para agregar la operación ’?’ (cero o una vez lo anterior).

   (25 puntos)

4. Al programa con ambas operaciones adicionales agregue la posibilidad de citar un caracter especial, vale decir, escribir por ejemplo ’\?’ para calzar un ’?’.

   (30 puntos)

Entregue varias versiones del código de calce, una para cada pregunta. Explique los cambios hechos al original.

Note que `grep(1)` se llama *exactamente* como se indica, y únicamente escribe las líneas que calza. Cualquier otra salida se considerará un error.

Tenga cuidado, para el *shell* los caracteres ’*’ y ’?’ tienen significado especial, para evitar accidentes se recomienda poner sus expresiones entre apóstrofes al hacer pruebas.

1. Condiciones de entrega

- La tarea se realizará *individualmente* (esto es grupos de una persona), sin excepciones.
- La entrega debe realizarse vía Moodle en un *tarball* en el área designada al efecto, bajo el formato `tarea-3-`*`rol`*`.tar.gz` (`rol` con dígito verificador y sin guión).

  Dicho *tarball* debe contener las fuentes en LaTeX (al menos `tarea.tex`) de la parte escrita de su entrega, además de un archivo `tarea-3.pdf`, correspondiente a la compilación de esas fuentes.
- En caso de haber programas, su ejecutable *debe* llamarse `tarea-3`, de haber varias preguntas solicitando programas, estos deben llamarse `tarea-3-1`, `tarea-3-2`, etc. Si hay programas compilados, incluya una `Makefile` que efectúe las compilaciones correspondientes.

  Los programas se evalúan según que tan claros (bien escritos) son, si se compilan y ejecutan sin errores o advertencias según corresponda. Parte del puntaje es por ejecución correcta con casos de prueba. Si el programa no se ciñe a los requerimientos de entrada y salida, la nota respectiva es cero.
- Además de esto, la parte escrita de la tarea debe en hojas de tamaño carta en Secretaría Docente de Informática (Piso 1, edificio F3).
- Tanto el *tarball* como la entrega física deben realizarse el día indicado en Moodle. No entregar la parte escrita en papel o no entregar en formato electrónico tiene un descuento de 50 puntos.

  Por cada día de atraso se descontarán 20 puntos. A partir del tercer día de atraso no se reciben más tareas, y la nota de la tarea es cero.
- Nos reservamos el derecho de llamar a interrogación sobre algunas de las tareas entregadas. En tal caso, la nota base (antes de descuentos por atraso y otros) es la de la interrogación. No presentarse a la interrogación sin justificación previa significa automáticamente nota cero.

LW/HvB/LaTeX 2_ε

```c
/* match: search for regexp anywhere in text */
int match(char *regexp, char *text)
{
    if (regexp[0] == '^')
        return matchhere(regexp+1, text);
    do {    /* must look even if string is empty */
        if (matchhere(regexp, text))
            return 1;
    } while (*text++ != '\0');
    return 0;
}

/* matchhere: search for regexp at beginning of text */
int matchhere(char *regexp, char *text)
{
    if (regexp[0] == '\0')
        return 1;
    if (regexp[1] == '*')
        return matchstar(regexp[0], regexp+2, text);
    if (regexp[0] == '$' && regexp[1] == '\0')
        return *text == '\0';
    if (*text!='\0' && (regexp[0]=='.' || regexp[0]==*text))
        return matchhere(regexp+1, text+1);
    return 0;
}

/* matchstar: search for c*regexp at beginning of text */
int matchstar(int c, char *regexp, char *text)
{
    do {    /* a * matches zero or more instances */
        if (matchhere(regexp, text))
            return 1;
    } while (*text != '\0' && (*text++ == c || c == '.'));
    return 0;
}
```

Listing 1: Código C para reconcer expresiones regulares de Pike