Laboratorio 2 - Backtracking

JUAN RETAMALES

Profesora: Mónica Villanueva

Ayudante: Patricio Vargas

Compiled April 23, 2017

En este documento se da a conocer conceptos básicos del algoritmo Backtracking y el lenguaje de programación C, ya que, mediante un problema planteado se pretende comprender el funcionamiento del algoritmo además de mostrar las conclusiones realizadas.

1. INTRODUCCIÓN

En el presente documento se analiza como funciona un algoritmo de combinatoria llamado Backtracking usando el lenguaje de programación C con un problema propuesto. Por ello este documento consistirá en esta introducción, un enunciado del problema anteriormente mencionado, luego se detallaran los conceptos asociados en el marco teórico para comenzar la descripción de una solución, una vez llego a este punto se procederá con el análisis de los resultados para posteriormente ir a una traza de la solución y resumir las conclusiones del documento.

Como objetivo planteado es el comprender el funcionamiento del método de combinatoria Backtracking.

2. DESCRIPCIÓN DEL PROBLEMA

Se debe resolver el enunciado de la subsección A, antes del plazo de entrega informado en la subsección D.

A. Enunciado

La empresa Cuidate SA crea sistemas de seguridad básicos basados en com- binaciones de números. Se sabe que el largo de las contraseñas es de 12 unidades y además se deben cumplir ciertos criterios para formar las claves. En primer lugar no se pueden generar códigos en donde existan 3 números iguales consecutivos, la segunda condición indica que si la contraseña comienza con un número impar no puede terminar con un número impar, por último, la combinación no puede contener secuencias de más de 3 números, por ejemplo, si en la contraseña se forma la secuencia 1234 significa que la clave es inválida. Considerando que los números están entre el 0 al 9 diseñe e implemente un programa que encuentre todas la posibles combinaciones dado un conjunto de entrada, para esto utilice la técnica de **Backtracking** y el **lenguaje de programación C** [1].

B. Entrada

En un archivo de texto titulado entrada.in se encuentran listados todos números que se deben utilizar para formar las contraseñas.

Table 1. Ejemplo de entrada y salida

entrada	salida
0	
1	125433221002
2	431023312115
3	652323119011
4	514173319930
5	001122334455
6	.....

C. Salida

En un archivo de texto titulado salida.out se debe mostrar separadas por saltos de líneas todas las contraseñas que se pueden crear con el archivo de entrada. Si con el contenido de la entrada no se pueden construir un conjunto de claves se debe informar en el archivo que se necesita otro conjunto.

D. Entrega

Domingo 23 de abril del 2017 hasta las 23:59, se descontará 1 punto por cada hora de retraso, al no entregar se reprueba el laboratorio.

3. MARCO TEORICO

Se pretende mediante el enunciado anterior, comprender sobre el funcionamiento del algoritmo de fuerza bruta usando el lenguaje de programación C.

A. Método de Backtracking

Dado un conjunto S, se debe construir un subconjunto $S' \subset S$ que satisfaga una propiedad P dada. Al construir S´ paso a

paso seleccionando en cada paso un elemento candidato a ser incorporado al conjunto construido.

Por cada elemento seleccionado, se comprueba si el conjunto construido hasta ese momento más el nuevo elemento satisface P. Si es así entonces se incorpora y se repite el proceso. Caso contrario, se retira el último elemento incorporado a S´ y se intenta con otro elemento de $S - S$.

Si en algún momento no existen más candidatos, se retira el elemento anterior de S´ y se repite el proceso. Siempre se obtiene una solución, el problema es que el número de posibilidades a examinar puede ser muy grande. [2].

B. Lenguaje de programación C

El lenguaje C es un lenguaje estructurado, en el mismo sentido que lo son otros lenguajes de programación tales como el lenguaje Pascal, el Ada o el Modula-2, pero no es estructurado por bloques, o sea, no es posible declarar subrutinas (pequeños trozos de programa) dentro de otras subrutinas, a diferencia de como sucede con otros lenguajes estructurados tales como el Pascal. Además, el lenguaje C no es rígido en la comprobación de tipos de datos, permitiendo fácilmente la conversión entre diferentes tipos de datos y la asignación entre tipos de datos diferentes [3].

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
int main(int argc, char** argv) {
    ...
    return (EXIT_SUCCESS);
}
```

C. Análisis de Algoritmos: Complejidad

A efectos prácticos o ingenieriles, nos deben preocupar los recursos físicos necesarios para que un programa se ejecute. Aunque puede haber muchos parametros, los mas usuales son el tiempo de ejecución y la cantidad de memoria (espacio). Ocurre con frecuencia que ambos parametros están fijados por otras razones y se plantea la pregunta inversa: ¿cual es el tamano del mayor problema que puedo resolver en T segundos y/o con M bytes de memoria? En lo que sigue nos centraremos casi siempre en el parametro tiempo de ejecución, si bien las ideas desarrolladas son fácilmente aplicables a otro tipo de recursos.

Para cada problema determinaremos un medida N de su tamaño (por número de datos) e intentaremos hallar respuestas en función de dicho N. El concepto exacto que mide N depende de la naturaleza del problema. Así, para un vector se suele utizar como N su longitud; para una matriz, el número de elementos que la componen; para un grafo, puede ser el número de nodos (a veces es mas importante considerar el número de arcos, dependiendo del tipo de problema a resolver); en un fichero se suele usar el número de registros, etc. Es imposible dar una regla general, pues cada problema tiene su propia lógica de coste [4].

4. DESCRIPCIÓN DE LA SOLUCIÓN

A. Etorno

El entorno de trabajo se realiza en el editor Netbeans IDE 8.2 en su versión de 64 bits el cual funciona con el software Cygwin que proporciona un compilador del lenguaje C ademáss de otras herramientas usadas por el editor. Todas las pruebas se realizan en un equipo AMD de 4 núcleos de 2.9 Ghz con 16Gb de memoria ram el cual tiene instalado el sistema operativo Windows 7 en su versión Ultimate de 64bits y más de 500Gb de espacio libre en el disco duro principal.

De los archivos del laboratorio se destacan los siguientes:

-**main.c**: Guarda el código principal del algoritmo así como su función principal (main).

-**entrada.in**: Es el utilizado para que el software pueda leer los números para generar las combinaciones, cabe destacar que para no incidir a un error se utilizara el archivo en formato UTF sin BOM. En este archivo de texto se leen todos los caracteres, de los cuales se extraen todos los números entre 0 y 9, de ser un numero de dos dígitos quedaran como dos de un digito, y de los dígitos guardados no se podrán repetir, es decir si ingreso 33 solo se guardara una vez 3.

-**salida.out**: Es el archivo el cual el software guarda las combinaciones validas con un salto de línea entre cada combinación.

-**funciones.h**: Contiene las declaraciones y una breve explicación del funcionamiento de las validaciones (funciones) usadas.

-**funciones.c**: Este archivo contiene el código de las validaciones declaradas en el archivo “funciones.h” las cual se encargan de revisar si los números cumplen ciertas condiciones explicadas en la subsección “Condiciones”.

B. Condiciones

Del enunciado se pueden extraer tres condiciones que se debe cumplir para que la combinatoria sea válida:

1. No se pueden generar códigos en donde existan 3 números iguales consecutivos.
2. Si la contraseña comienza con un número impar no puede terminar con un número impar.
3. La combinación no puede contener secuencias de más de 3 número.

Para cada una de estas condiciones se utilizó una función llamada “validadorIgualesConsecutivos”, “validadorImparImpar” y “validadorConsecutivos” respectivamente, que retornaban 1 de ser válido y 0 de no serlo.

```
/*
* validadorImparImpar - verifica que si el primer numero es
      impar, no termine en impar
*
*Entrada: 2 numeros de formato int.
*Salida: Numero int 1 si ambos son impar o 0 de lo
     contrario.
*/
int validadorImparImpar(int c1, int c12)
{
    if(c1%2==1 && c12%2==1)
    {
        return 1;
    }
    return 0;
}
```

C. Logica

Para generar las combinaciones se utilizaron 12 ciclos anidados que comprendía el largo de caracteres que podía utilizar la combinación, es decir, si la combinación el largo fuera 7, solo hubieran sido necesarios 7 ciclos anidados. Cada ciclo “for” usa una variable del abecedario para identificarla y utilizarla, la cual consto de la letra “a” a la letra “l”. Para usar backtracking es necesario que se destaque del código las siguientes observaciones de los validadores: Después del tercer bucle de la combinación en adelante se utiliza el validador “validadorIgualesConsecutivos”.

Table 2. Tabla del orden - validadorIgualesConsecutivos

For N°	validadorIgualesConsecutivos
3	2
4	4
5	6
6	8
7	10
8	12
9	14
10	16
11	18
12	20

Posteriormente en el siguiente “for” en adelante se utiliza el validador “validadorConsecutivos”. Finalmente en el último ciclo de la combinación se incluye el validador “validadorImparImpar”. Al fallar cualquiera, en dicho “for” salta inmediatamente a la siguiente combinación posible y no termina la combinación que intentaba realizar. Cuando termina la combinación en el doceavo “for” y es válido, se guarda en el archivo salida.out. Además cada combinación está separada por un salto de línea.

```
fp = fopen ( "salida.out", "a" );
fprintf(fp, "%d%d%d%d%d%d%d%d%d%d%d%d␣\n", entrada[a],
     entrada[b], entrada[c], entrada[d], entrada[e],
     entrada[f], entrada[g], entrada[h], entrada[i],
     entrada[j], entrada[k], entrada[l]);
fclose ( fp );
```

D. Complejidad y orden

Para calcular el orden empezamos calculando el orden de los validadores o funciones del archivo “funciones.c” y se puede apreciar lo siguiente: Para el validador “validadorIgualesConsecutivos” el orden seria normalmente O((m-2)*2) pero en el archivo “main.c” nosotros siempre sabemos cuáles son los valores que toma en su “único peor caso”. Lo mismo ocurre con el validador “validadorConsecutivos” con el orden O((m-3)*3). En el caso del validador “validadorImparImpar” su orden siempre es O(2).

Finalmente luego de hacer los cálculos correspondientes para calcular orden, solo en los 12 ciclos anidados tendremos un orden de $O(142285554566920N^{12})$. Teniendo un orden final de $O(142285554566932N^{12})$ de tiempo de ejecución para generar todas las combinaciones, escribirlas en salida.out y además desplegar el tiempo que se demora en realizar lo anterior.

5. ANÁLISIS DE LOS RESULTADOS

En la **Table 4.** Ejemplos de salida, se muestran 6 de los resultados aleatorios obtenidos con los números de entrada 1,3 y 8.

En la **Table 5.**Tiempo usado según el software, para las combinaciones de entrada "12345" se detuvo manualmente luego de tener más de 6 minutos en marcha el software con la cantidad de registros indicados, y lo mismo en el último caso que se detuvo manualmente a los 16 minutos aproximadamente.

Para el tan grande “orden” que se mencionó, este tiempo de ejecución contrasta en gran medida el porqué de su valor.

Table 3. Tabla del orden - validadorConsecutivos

For N°	validadorConsecutivos
4	3
5	6
6	9
7	12
8	15
9	18
10	21
11	24
12	27

Table 4. Ejemplos de salida

N°	Resultado
1	113113113118
51	113113188138
86	113113383318
147	113113883388
184	113118138838
122454	883883883883

Table 5. Tiempo usado según el software

Entrada	N° de combinaciones	Tiempo
1	0	270ms
1l0	350	0.187s
asderfa	0	0.015s
12345*	544707	6m30s
13579	0	26.239s
0123456789*	1103680	16m46s

Además en el último caso, luego de 1103680 combinaciones, la última formada es "001002156734" y se puede deducir que llevábamos solo la una centésima parte del tiempo además de que para realizar el total de combinaciones requerirá como mínimo una hora.

Es eficaz al ser más rápido y eficiente al usar mejor el recurso tiempo, en comparación con el método de fuerza bruta que termina la combinación para recién comprobar si es válido o no.

Se detiene automáticamente al terminar de realizar las combinaciones posibles y desplegar un mensaje, sin embargo de otra forma se tiene que forzar un cierre.

Se podría mejorar quitando las líneas que muestran el tiempo que se demora en realizar las combinaciones pero así como esta última, solo en pequeñas medidas.

6. TRAZA DE LA SOLUCIÓN

Primero revisa un archivo de texto llamado entrada.in del cual se leen todos los caracteres, de los cuales se extraen todos los números entre 0 y 9 inclusive, y se guardan en un arreglo, además de un contador para saber el largo de este arreglo. Luego crea o sobrescribe el archivo salida.out para guardar las combinaciones, borrando cualquier registro anterior que tuviera.

Luego pasa por los 12 ciclos "for" anidados con variables de la "a" a "l", donde al igual que se resuelve un candado de combinación, se cada vez que un ciclo se termina, el anterior avanza una vez y llama al arreglo de combinación por cada digito para así formar cada combinatoria. Por ejemplo si de entrada tenemos el número 1 y 2, la primera combinación seria "111111111111" y la segunda "111111111112", la tercera "111111111121" y así sucesivamente.

Después del tercer bucle de la combinación en adelante se utiliza el validador "validadorIgualesConsecutivos".

Posteriormente en el siguiente "for" en adelante se utiliza el validador "validadorConsecutivos".

Finalmente en el último ciclo de la combinación se incluye el validador "validadorImparImpar".

Al fallar cualquiera, en dicho "for" salta inmediatamente a la siguiente combinación posible y no termina la combinación que intentaba realizar. Cuando termina la combinación en el doceavo "for" y es válido, se guarda en el archivo salida.out. Además cada combinación está separada por un salto de línea para diferenciarlos entre cada combinación. En este caso una combinación que cumpliera las condiciones se escribiría "112211221122" y después un salto de línea.

```
fp = fopen ( "salida.out", "a" );
fprintf(fp, "%d%d%d%d%d%d%d%d%d%d%d%d␣\n", entrada[a],
     entrada[b], entrada[c], entrada[d], entrada[e],
     entrada[f], entrada[g], entrada[h], entrada[i],
     entrada[j], entrada[k], entrada[l]);

/*entrada[a] seria 1*/
/*entrada[b] seria 1*/
/*entrada[c] seria 2*/
/*entrada[d] seria 2*/
/*entrada[e] seria 1*/
/*entrada[f] seria 1*/
/*entrada[g] seria 2*/
/*entrada[h] seria 2*/
/*entrada[i] seria 1*/
/*entrada[j] seria 1*/
/*entrada[k] seria 2*/
/*entrada[l] seria 2*/
/*y realiza el salto de linea.*/

fclose ( fp );

     salida++;
/* Y suma en una unidad a la variable salida. */
```

7. CONCLUSIÓN

El método backtracking es un método de combinación que se empleó para realizar todas las combinaciones validas de números ingresados en el archivo entrada.in, dependiendo de los números distintos ingresados puede demorarse varios minutos y un gran gasto de procesamiento. Sin embargo no se pudo resolver el peor caso por el tiempo requerido de entrada.in que tenía todos los números posibles, sin embargo se pudo estimar el tiempo requerido y contrastarlo con el orden calculado justificando en cierta medida su valor. Además la actividad demostró ser más eficiente que el método de fuerza bruta y se cumplió el objetivo planteado en el documento.

REFERENCES

1. M. Villanueva, "Algoritmos avanzados 1.2017 - laboratorio ii," (2017).
2. M. Villanueva, "Algoritmos: Teoria y aplicaciones," (2002).
3. E. V. B. Esteban, "Lenguaje c," (2009).
4. J. A. Mañas, "Análisis de algoritmos: Complejidad," (1997).

FULL REFERENCES

1. M. Villanueva, "Algoritmos avanzados 1.2017 - laboratorio ii," (2017).
2. M. Villanueva, "Algoritmos: Teoria y aplicaciones," (2002).
3. E. V. B. Esteban, "Lenguaje c," (2009).
4. J. A. Mañas, "Análisis de algoritmos: Complejidad," (1997).