MEMORIA PRÁCTICA 2

Grupo: PCEO

Curso: 2022-2023

Autores
Jose Guillén Riquelme
Gonzalo Lucena Vázquez
PCEO 10

ÍNDICE

Introducción.

En esta documentación de la práctica 2 se muestran los pasos realizados para llegar a completarlos ejercicios E de voraces y K de backtracking. Se comentará pues: los pseudocódigos que reflejan los esqueletos de los códigos, el estudio teórico de los tiempos de ejecución, el estudio experimental, el contraste entre ambos estudios y, finalmente, las conclusiones. A continuación, también dejamos presente el registro de entregas al mooshak:

ENTREGAS MOOSHAK (VORACES)

#	Tiempo de Concurso	País	Equipo	Problema	Lenguaje	Resultado	Estado
361	1034:52:47		PCEO PCEO 10	E_AR	C++	3 Accepted	final
360	1034:51:17		PCEO PCEO 10	E_AR	C++	0 Compile Time Error	final
359	1034:49:55		PCEO PCEO 10	E_AR	C++	0 Compile Time Error	final

Entrega: 361

ENTREGAS MOOSHAK (BACKTRACKING)

#	Tiempo de Concurso	País	Equipo	Problema	Lenguaje	Resultado	Estado
445	1070:23:33		PCEO PCEO 10	K_Ba	C++	3 Accepted	final
444	1070:22:46		PCEO PCEO 10	K_Ba	C++	0 Compile Time Error	final
287	1036:03:06		PCEO PCEO 10	K_Ba	C++	3 Accepted	final

Entrega: 445

Pseudocódigos.

→ Pseudocódigo del ejercicio de voraces (sin incluir el main):

```
//void para guardar el número de mecánicos de una columna en la última fila de la matriz
procedure sumaColumna(int mecanicos, int columna, int**C);
begin
        int suma = 0;
        for(int i =0; i< mecanicos; i++){
                suma += C[i][columna];
        }
        C[mecanicos][columna] = suma;
end;

//void para hacerlo al momento para todas las columnas
procedure sumaTotal(int mecanicos, int averias, int**C)
begin
        for(int j =0; j<averias; j++){
        sumaColumna(mecanicos, j, C);
        }
```

```
end;


function asignarMecanicos(int mecanicos, int averias, int** C, int solucion[]): int;
begin
	int mecanicosRestantes = mecanicos;
	int averiasRestantes = averias;

	//Inicializar solucion a 0
	for(int i = 0; i< averias; i++){
		solucion[i] = 0;
	}

	int averiasReparadas = 0;
	int min = INT_MAX;
	int posicionAveria = -1;
	int candidato = 0;
	int contador = 0;



	mientras (mecanicosRestantes>0 && averiasRestantes>0){

	//Elegimos la avería con menos mecánicos disponibles
	for(int i = 0; i< averias; i++){
		si (C[mecanicos][i] !=0 && min > C[mecanicos][i] && solucion[i] == 0) {
			min = C[mecanicos][i];
			posicionAveria = i;
		}
	}

	min = INT_MAX;



	//Recorremos la columna (averia) y tomamos el primer mecanico disponible (1er uno)
	mientras (posicionAveria!= -1 && candidato == 0 && contador< mecanicos ){
		si (C[contador][posicionAveria] == 1) {
			candidato = contador +1;
			solucion[posicionAveria] = candidato;
			averiasReparadas++;
			mecanicosRestantes--;
			averiasRestantes--;
		}
		contador++;
	}
	contador = 0;

	si (posicionAveria == -1)  averiasRestantes--;

	si (posicionAveria != -1){
		for(int i = 0; i<= mecanicos; i++) {
```

```
                        C[i][posicionAveria] = 0;
                }
                for(int j = 0; j< averias; j++) {
                        si (C[candidato-1][j] == 1) {
                                C[mecanicos][j]--;
                                C[candidato-1][j] = 0;
                        }
                }
        }

        posicionAveria = -1;
        candidato = 0;

        }
        devuelve averiasReparadas;
end;
```

→ Pseudocódigo del ejercicio de backtracking (sin incluir el main):

```
const MAX_PARTICIPANTES = 30

estructura SolucionOptima{
        int array[MAX_PARTICIPANTES];
        int pesoEquipo1;
        int pesoEquipo2;

};

//usamos el esquema de optimización
estructura SolucionOptima Backtracking(int S[],  int P[], int nl){
        int nivel = 0;
        int voa = INT_MAX;
        struct SolucionOptima solucion;
        int pEquipo1 = 0;                   //peso equipo1
        int pEquipo2 = 0;                   //peso equipo2
        int nEquipo1 = 0;                   //numero de personas en equipo 1
        int nEquipo2 = 0;                   //numero de personas en equipo 2


        hacer {
                //Generar
                S[nivel] = S[nivel] +1;

                si (S[nivel] == 0){
                        pEquipo1 = pEquipo1 + P[nivel];
                        nEquipo1++;
                } sino si (S[nivel] == 1){
                        pEquipo1 = pEquipo1 - P[nivel];
                        pEquipo2 = pEquipo2 + P[nivel];
                        nEquipo1--;
                        nEquipo2++;
```

```
		}

		//si Solución and condición entonces…
		si (nivel==n-1 && abs(nEquipo1 - nEquipo2)<= 1 && abs(pEquipo1 - pEquipo2)<
voa){
			voa = abs(pEquipo1 - pEquipo2);
			solucion.pesoEquipo1 = pEquipo1;
			solucion.pesoEquipo2 = pEquipo2;
			for(int i = 0; i<n; i++){
				solucion.array[i] = S[i];
			}

		}
		//si Criterio se cumple, avanza
		si (nivel<n-1){
			nivel = nivel + 1;

		} sino {
			//mientras not MasHermanos… retrocede
			mientras (!(S[nivel]<1) && nivel > -1) {
				si (S[nivel] == 0){
					pEquipo1 = pEquipo1 - P[nivel];
					nEquipo1--;
				} sino si (S[nivel] == 1){
					pEquipo2 = pEquipo2 - P[nivel];
					nEquipo2--;
				}

				S[nivel] = -1;
				nivel = nivel -1;
			}

		}

	} mientras (nivel != -1);

	devolver solucion;
}
```

Breve explicación de los algoritmos

Para el algoritmo de voraces hemos implementado un par de funciones void para facilitarnos algunos procesos, como el conteo de mecánicos disponibles por avería y tras ellas tenemos la función asignarMecánicos, que es en la que reside el grueso de nuestro código. En ella, primero hacemos las inicializaciones pertinentes, tras eso, hacemos un recorrido buscando la columna con menos mecánicos disponibles (ya que daremos prioridad a las que menos tengan), y guardamos su índice en la variable posicionAveria. Tras eso, recorremos la columna guardada en posicionAveria y nos quedamos con el primer mecánico (uno) que encontremos. Por último, hacemos los decrementos que toquen y dejamos fila y columna de la posición de dicho mecánico a 0, ya que la avería ha sido reparada y ese mecánico ya no podrá ser asignado de nuevo.

Para el de backtracking simplemente hemos seguido el esquema de optimización aplicado a un árbol binario. Manejamos la estructura SolucionOptima con los pesos de los equipos 1 y 2 y el array con los participantes. Lo siguiente fue seguir el esquema teniendo cuidado con cómo se iban añadiendo los participantes a los equipos y las condiciones que íbamos estableciendo en las distinciones de casos.

Estudio teórico del tiempo de ejecución.

En el caso de voraces, ya partíamos con la idea de que tiempo peor y tiempo mejor serían del mismo orden, ya que el tiempo resultante nunca iba a variar dependiendo del contenido de la matriz de entrada, lo cual sería lo que plantearía las situaciones peor caso y mejor caso. Al darnos cuenta de esto, estudiamos en torno al orden exacto que podría darnos, y finalmente obtuvimos Orden exacto(n^2).

Posteriormente, para el de backtracking solo se estudia el tiempo para el caso peor ya que es el único posible al no incluir ninguna poda. Esto fue más sencillo, ya que de partida sabíamos que solo se podía dar este y, tras hacer el estudio, resultó que este tiempo pertenece a $O(2^n)$ (orden exponencial).

Estudio experimental.

ESTUDIO GENERAL (VORACES)

Primeramente, al ver repetidas veces cómo el caso peor y el caso mejor nos daban casos prácticamente iguales, decidimos estudiar los tiempos en base a un generador de entradas aleatorias. También queríamos expresar el resultado en un gráfico de dispersión, pero debido a que al final nos consumiría mucho tiempo el estudio para cada n (mecánicos) y m (averías), hemos optado finalmente por adjuntar capturas de

los tiempos para las entradas aleatorias con distintos tamaños. Adicionalmente, hemos añadido un gráfico para los casos en los que hemos fijado n y m a ser iguales, sabemos que lógicamente no aporta un resultado muy revelador, ya que no abarca una gran cantidad de casos, pero lo dejamos para que quede constancia de algún estudio gráfico.

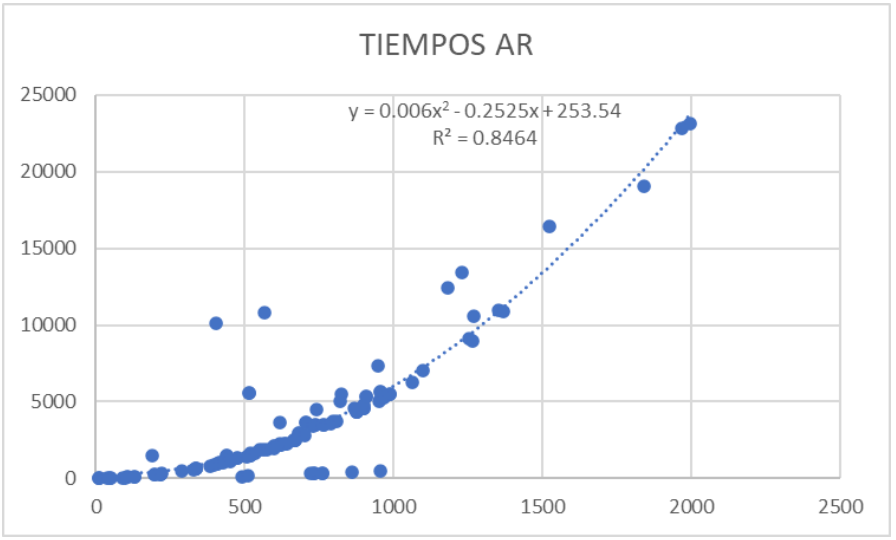
Capturas de tiempos con n y m aleatorios

```
Mecánicos: 872 Averías: 986 Tiempo: 1.757
Mecánicos: 487 Averías: 106 Tiempo: 0.021
Mecánicos: 317 Averías: 637 Tiempo: 0.733
Mecánicos: 616 Averías: 546 Tiempo: 0.543
Mecánicos: 411 Averías: 94 Tiempo: 0.016
Mecánicos: 576 Averías: 723 Tiempo: 0.943
Mecánicos: 541 Averías: 381 Tiempo: 0.266
Mecánicos: 236 Averías: 488 Tiempo: 0.431
Mecánicos: 347 Averías: 699 Tiempo: 0.882
Mecánicos: 525 Averías: 4 Tiempo: 0
Mecánicos: 367 Averías: 499 Tiempo: 0.451
Mecánicos: 601 Averías: 172 Tiempo: 0.054
Mecánicos: 380 Averías: 98 Tiempo: 0.018
Mecánicos: 8 Averías: 748 Tiempo: 1.009
Mecánicos: 381 Averías: 406 Tiempo: 0.298
Mecánicos: 354 Averías: 252 Tiempo: 0.117
Mecánicos: 169 Averías: 838 Tiempo: 1.268
Mecánicos: 133 Averías: 483 Tiempo: 0.52
Mecánicos: 252 Averías: 746 Tiempo: 1.009
Mecánicos: 28 Averías: 661 Tiempo: 0.795
Mecánicos: 616 Averías: 380 Tiempo: 0.261
Mecánicos: 383 Averías: 156 Tiempo: 0.045
Mecánicos: 758 Averías: 394 Tiempo: 0.281
Mecánicos: 642 Averías: 881 Tiempo: 1.399
Mecánicos: 92 Averías: 942 Tiempo: 1.609
Mecánicos: 882 Averías: 456 Tiempo: 0.377
Mecánicos: 440 Averías: 260 Tiempo: 0.123
Mecánicos: 625 Averías: 595 Tiempo: 0.735
Mecánicos: 356 Averías: 409 Tiempo: 0.303
Mecánicos: 342 Averías: 512 Tiempo: 0.476
```

```
Mecánicos: 245 Averías: 284 Tiempo: 0.147
Mecánicos: 215 Averías: 331 Tiempo: 0.199
Mecánicos: 80 Averías: 732 Tiempo: 17.234
Mecánicos: 188 Averías: 522 Tiempo: 0.493
Mecánicos: 377 Averías: 994 Tiempo: 1.799
Mecánicos: 79 Averías: 894 Tiempo: 1.44
Mecánicos: 735 Averías: 125 Tiempo: 0.032
Mecánicos: 537 Averías: 322 Tiempo: 0.189
Mecánicos: 647 Averías: 724 Tiempo: 0.95
Mecánicos: 264 Averías: 868 Tiempo: 1.62
Mecánicos: 448 Averías: 374 Tiempo: 0.254
Mecánicos: 407 Averías: 169 Tiempo: 0.053
Mecánicos: 926 Averías: 651 Tiempo: 0.763
Mecánicos: 48 Averías: 920 Tiempo: 1.552
Mecánicos: 395 Averías: 663 Tiempo: 0.797
Mecánicos: 877 Averías: 415 Tiempo: 0.315
Mecánicos: 944 Averías: 868 Tiempo: 1.385
Mecánicos: 743 Averías: 799 Tiempo: 1.151
Mecánicos: 599 Averías: 928 Tiempo: 1.56
Mecánicos: 320 Averías: 751 Tiempo: 1.019
Mecánicos: 699 Averías: 174 Tiempo: 0.063
Mecánicos: 423 Averías: 212 Tiempo: 0.082
Mecánicos: 74 Averías: 957 Tiempo: 1.651
Mecánicos: 531 Averías: 718 Tiempo: 0.932
Mecánicos: 459 Averías: 792 Tiempo: 1.149
Mecánicos: 363 Averías: 904 Tiempo: 1.481
Mecánicos: 165 Averías: 767 Tiempo: 1.065
Mecánicos: 848 Averías: 866 Tiempo: 1.35
Mecánicos: 195 Averías: 671 Tiempo: 0.83
Mecánicos: 785 Averías: 365 Tiempo: 0.241
```

```
Mecánicos: 782 Averías: 451 Tiempo: 0.369
Mecánicos: 548 Averías: 453 Tiempo: 0.371
Mecánicos: 791 Averías: 340 Tiempo: 0.224
Mecánicos: 744 Averías: 584 Tiempo: 0.62
Mecánicos: 256 Averías: 389 Tiempo: 0.274
Mecánicos: 197 Averías: 485 Tiempo: 0.426
Mecánicos: 366 Averías: 175 Tiempo: 0.056
Mecánicos: 158 Averías: 564 Tiempo: 0.76
Mecánicos: 386 Averías: 658 Tiempo: 0.928
0 Mecánicos: 969 Averías: 222 Tiempo: 0.09
1 Mecánicos: 193 Averías: 856 Tiempo: 1.316
2 Mecánicos: 40 Averías: 287 Tiempo: 0.149
3 Mecánicos: 926 Averías: 828 Tiempo: 1.255
4 Mecánicos: 414 Averías: 466 Tiempo: 0.396
5 Mecánicos: 271 Averías: 862 Tiempo: 1.366
6 Mecánicos: 120 Averías: 52 Tiempo: 0.006
7 Mecánicos: 91 Averías: 443 Tiempo: 0.355
8 Mecánicos: 280 Averías: 879 Tiempo: 1.398
9 Mecánicos: 558 Averías: 23 Tiempo: 0.001
0 Mecánicos: 463 Averías: 811 Tiempo: 1.211
1 Mecánicos: 409 Averías: 435 Tiempo: 0.344
2 Mecánicos: 74 Averías: 773 Tiempo: 1.082
3 Mecánicos: 385 Averías: 7 Tiempo: 0
4 Mecánicos: 114 Averías: 768 Tiempo: 1.08
5 Mecánicos: 663 Averías: 858 Tiempo: 1.329
6 Mecánicos: 766 Averías: 853 Tiempo: 1.313
7 Mecánicos: 491 Averías: 803 Tiempo: 1.191
8 Mecánicos: 140 Averías: 416 Tiempo: 0.317
9 Mecánicos: 408 Averías: 329 Tiempo: 0.197
0 Mecánicos: 657 Averías: 454 Tiempo: 0.378
```

Estudio con mecánicos = averías (Gráfica de abajo)



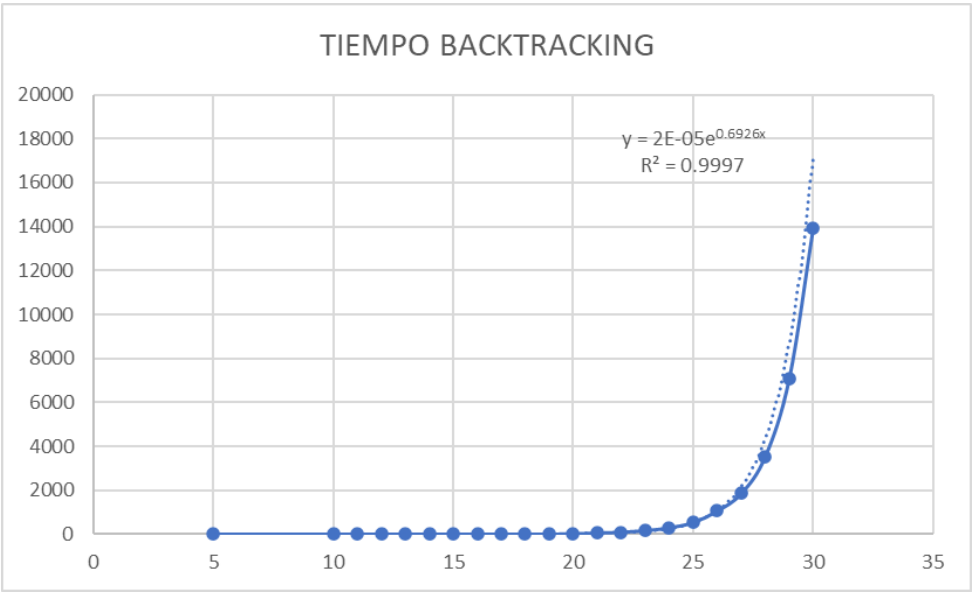
Se puede apreciar en este caso que la función que hemos obtenido se aproxima bastante a la que nos ofrece Excel (una polinómica de orden 2) con un R también cercano a 1.

Los generadores empleados se adjuntarán en la entrega también.

CASO PEOR (BACKTRACKING)

En el caso de Backtracking, al recorrerse todo pase lo que pase, solo se puede analizar en función del tiempo peor.

n	tiempo
5	0.0007
10	0.0135
11	0.0273
12	0.052
13	0.1084
14	0.2509
15	0.4794
16	0.8412
17	2.1507
18	4.8937
19	8.5201
20	24.2145
21	56.9298
22	95.6471
23	170.8182
24	301.1281
25	539.431
26	1061.489
27	1856.274
28	3542.642
29	7099.594
30	13947.89



Como podemos observar en la gráfica obtenida, el algoritmo tiene un orden exponencial. La R ofrecida por Excel nos ayuda a comprobarlo ya que es muy cercana a 1. También se podía visualizar durante la recogida de los tiempos que esto iba a ser así.

Contraste entre estudios.

Gracias a realizar el estudio previamente sabíamos que, posteriormente al hacer la gráfica, estas serían muy similares a las funciones cuadráticas (en el caso de voraces) y a la de una exponencial (en el caso de backtracking).

Y, efectivamente, al comprobarlas se ha dado el caso en el que coinciden, bastante más la de backtracking ya que se ve a la perfección y algo menos la de voraces, aunque principalmente esto se debe a la ausencia de puntos en el tramo final.

Así que, en resumen, podemos decir que se han obtenido los resultados esperados.

Conclusiones.

Gonzalo: Esta práctica nos ha mostrado un poco el contraste entre dos formas de afrontar los problemas, una que sacrifica optimalidad en su solución con tal de encontrarla de forma más directa y otra que sigue más al pie de la letra un esquema a modo de esqueleto. Realmente lo he encontrado útil y, aunque diría que el estudio teórico, al menos a mí me ha parecido un poco extraño de aplicar (especialmente en el ejercicio de voraces), he quedado bastante satisfecho con el proceso y resultados de esta práctica.

Jose: Esta 2ª práctica me ha gustado más que la primera y creo que ha sido más sencillo aplicar los algoritmos que se exigen de una mejor manera. Por eso, me ha servido bastante para conocer aún mejor el funcionamiento de voraces y backtracking. Estoy muy contento con los resultados que hemos obtenido ya que no nos han surgido grandes problemas a la hora de programar. Lo que menos me gusta de la actividad es el estudio de tiempos.

En cuanto a tiempo dedicado en realizar la práctica rondará las 20 horas.