Universidad
Rey Juan Carlos

Análisis de algoritmos: Practica 3.

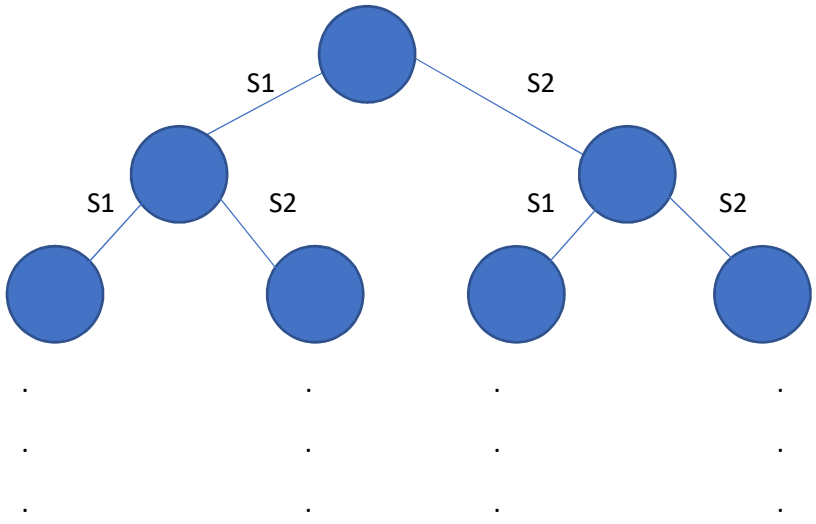
Algoritmos de Vuelta atrás.

Daniel Lois Nuevo

Adrián García Oller

1. Técnica de vuelta atrás.

a)



El árbol tendrá tantos niveles como meses se tengan en cuentan (número de niveles = tamaño de los arrays de costes). En cada nodo del arbol puedes elegir entre coger la sede 1 o la sede 2.

b)

Hemos usado dos funciones, una principal que recoge los datos de entrada y transforma los dos arrays en una matriz, inicializa la solución optima como infinito, pero como se puede hemos elegido el valor más grande posible, inicializamos la solAux , anterior (la sede donde ha estado el anterior mes) e i (i=mes) a cero y llamamos a la función auxiliar que es el algoritmo de vuelta atrás.

```
public static int sedes (int[] c0, int[] c1, int f) {
	int i = 0;
	int aux;
	int anterior = 0;
	int sol = 2147483647;
	int solAux = 0;
	int[][] cAux = new int[2][c0.length];
	for (int k = 0; k < c0.length; k++){
		cAux[0][k] = c0[k];
	}
	for (int k = 0; k < c0.length; k++){
		cAux[1][k] = c1[k];
	}
	aux = sedesAux(i,sol,solAux , cAux ,f, anterior);
	return aux;
}
```

En este algoritmo por cada una de las sedes decidimos si es el primer mes no usaremos el coste de traslado al escoger sede, y si es el único mes a tener en cuenta comprobara si la solución óptima es peor que solAux y si es el caso asignara a sol(solución óptima) el valor de solAux y devolverá la solución, si no, llamará de nuevo a esta función con el siguiente mes (i++) , la solAux mas el coste de la sede k en el mes i, la misma sol , la misma matriz de costes y el mismo coste de traslado y la sede k, y recogerá la sol devuelta.

Si no es el primer mes comprueba si es el último mes, si lo es comprobara si la solución óptima es peor que solAux y si es el caso asignara a sol (solución óptima) el valor de solAux y devolverá la solución.
Si es un mes intermedio comprobara si la anterior sede y la actual son iguales, si lo son hará lo mismo que para el primer mes, si no, además de lo que hace para el primer mes añadirá a solAux el coste de traslado.

```
private static int sedesAux(int i,int sol, int solAux, int[][] cAux, int
f,int anterior){

		for (int k = 0; k < 2; k++){
			if (i==0){
				if (i == (cAux[k].length)-1){
					solAux=cAux[k][i];
					if(sol>solAux) {
						sol=solAux;
					}
					return sol;
				}
				else{
				anterior =k;

				solAux += cAux[k][i];
				i++;
				sol=sedesAux(i, sol, solAux, cAux, f, anterior);
				i--;
				solAux -= cAux[k][i];

				}
			}
			else{

					if (i ==  (cAux[k].length)){
						if(sol>solAux){
							sol=solAux;
						}
						return sol;
					}
					else{
						if(anterior==k){
							solAux+=cAux[k][i];
							i++;
					sol=sedesAux(i, sol, solAux, cAux, f, k);
							i--;
							solAux-=cAux[k][i];

						}
						else{
							//anterior=k;
							solAux+=cAux[k][i]+f;
							i++;
					sol=sedesAux(i, sol, solAux, cAux, f, k);
							i--;
							solAux-=cAux[k][i]+f;
						}
					}


			}
```

```
}
return sol;
}
```

2. Comparación de optimalidad.

a) El método sedes es el vuelta atrás.
El método sedesC es un heurísticos que compara los costes de cada mes más el coste de traslado si cambiara de sede y elige el menor.
El método sedesS es un heurísticos que compara los costes de cada mes y elige el menor.

b) Según los resultados recogidos como todos los resultados de sedes (el algoritmo de vuelta atrás) son siempre óptimos ese algoritmo es exacto en cambio los otros dos no lo son porque en cierto porcentaje no encuentra resultados óptimos.

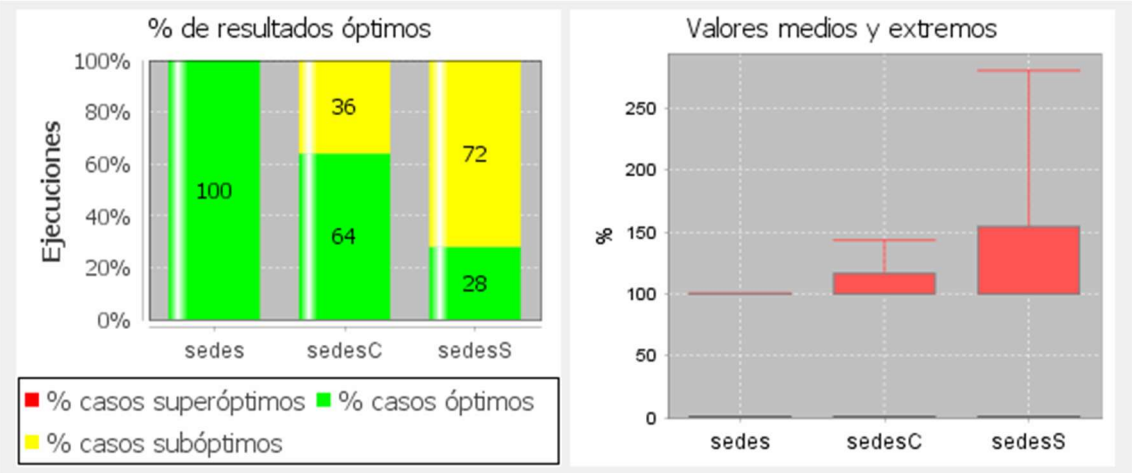
c) Tabla histórica:
Como se puede ver en la tabla de la siguiente hoja todos los resultados de sedes (vuelta atrás) son óptimos en cambio en sedesC hay una menor cantidad de resultados óptimos y en sedesS hay todavía menos.

Núm.	sedes	sedesC	sedesS
1	35	35	74
2	56	75	107
3	61	61	171
4	53	53	64
5	55	55	57
6	62	74	69
7	61	84	61
8	44	44	52
9	32	32	87
10	59	60	113
11	78	78	78
12	54	72	57
13	52	57	58
14	55	55	55
15	68	71	73
16	76	76	134
17	58	58	104
18	69	75	115
19	65	71	99
20	69	69	69
21	64	76	108
22	20	20	20
23	43	43	43
24	55	61	108
25	68	82	72
26	78	78	121
27	48	48	102
28	75	75	110
29	65	65	77
30	21	21	35
31	67	96	75
32	29	29	29
33	100	106	100
34	73	73	98
35	101	101	167
36	56	56	56
37	77	93	166
38	70	71	119
39	63	63	94
40	76	76	92
41	67	67	94
42	47	50	50
43	69	69	73
44	70	70	70
45	54	54	54
46	57	57	57
47	66	66	76
48	58	58	86
49	44	44	44
50	63	68	63

Tabla de resumen numérico:
Como se puede ver en la tabla el algoritmo de sedes el 100% de las soluciones son optimas al contrario que el 64% de sedesC y el 28% de sedesC.

Medidas	sedes	sedesC	sedesS
Núm. ejecuciones	50	50	50
Núm. ejec. válidas del método	50	50	50
Núm. ejec. válidas en total	50	50	50
% soluciones subóptimas	0 %	36 %	72 %
% soluciones óptimas	100 %	64 %	28 %
% soluciones sobreóptimas	0 %	0 %	0 %
% valor medio no óptimo	0 %	116,33 %	154,25 %
% valor extremo subóptimo	0 %	143,28 %	280,33 %
% valor medio sobreoptimo	0 %	0 %	0 %
% valor extremo sobreóptimo	0 %	0 %	0 %

Tabla de resumen gráfico:
Como podemos ver en la siguiente imagen, en el primer grafico observamos de forma grafica lo mismo que en la anterior tabla, pero en el segundo podemos observar que en sedes todas las soluciones entran dentro de una media, pero en sedesC los valores extremos se alejan de la media y en sedesS se alejan aún más todavía.



d) Debido a que había ejecuciones en las que daba mejor solución en los heurísticos que en el vuelta atrás hemos tenido que hacer dos cambios en los heurísticos.

El primero ha sido en el heurístico de sedesS que no asignaba donde había estado el mes anterior en el caso del de ser el primer mes.

```
public static int sedesSinCostedeTraslado(int[] c0, int[] c1, int f){

    int coste = 0;
    int anterior  = 0;
    if (c0[0] < c1[0]){
        coste = c0[0];
        anterior=0;
    } else {
        coste = c1[0];
```

```
        anterior=1;
    }
```

El segundo cambio ha sido en ambos heurísticos ya que cuando comparaban los costes no tenían en cuanta el caso de que ambos costes fueran iguales, para eso quitamos los else if (sede1<sede2).

```
public static int sedesS (int[] c0, int[] c1, int f){
    int coste = 0;
    int anterior  = 0;
    if (c0[0] < c1[0]){
        coste = c0[0];
        anterior=0;
    } else {
        coste = c1[0];
        anterior=1;
    }
    for (int i = 1; i < c0.length;i++){
        if ((anterior == 0)){
            if (c0[i] < c1[i]){
                coste = coste + c0[i];
                anterior = 0;
            } else {
                coste = coste + c1[i] + f;
                anterior = 1;
            }
        } else if (anterior == 1) {
            if (c0[i] < c1[i]){
                coste = coste + c0[i] + f;
                anterior = 0;
            } else {
                coste = coste + c1[i];
                anterior = 1;
            }
        }
    }
    return coste;
}
public static int sedesC (int[] c0, int[] c1, int f){
    int coste = 0;
    int anterior  = 0;
    if (c0[0] < c1[0]){
        coste = c0[0];
        anterior = 0;
    } else {
        coste = c1[0];
        anterior = 1;
    }
    for (int i = 1; i < c0.length;i++){
        if ((anterior == 0)){
            if (c0[i] < (c1[i] + f)){
```

```
                coste = coste + c0[i];
                anterior = 0;
            } else{
                coste = coste + c1[i] + f;
                anterior = 1;
            }
        } else if (anterior == 1) {
            if ((c0[i]+ f) < c1[i]){
                coste = coste + c0[i] + f;
                anterior = 0;
            } else{
                coste = coste + c1[i];
                anterior = 1;
            }
        }
    }
    return coste;
}
```

3. Conclusiones.

Esta practica ayuda a comprender los algoritmos de vuelta atrás y a practicar con herramientas de estudio de algoritmos. En cuanto a la dificultad de la práctica creemos que tiene una complejidad notablemente mayor a la anterior por el tipo de algoritmo empleado, pero gracias a la practica hemos podido solventar algunos problemas de la anterior práctica. Aunque el programa empleado, optimEx ha presentado dos problemas, el primero es que con java 10.0.2 no te permite seleccionar problema y el segundo es que si el tamaño de los nombres de los métodos es notablemente diferente el programa no te deja hacer más que una ejecución de datos.