Universidad de Murcia

Algoritmos y Estructuras de Datos II

PCEO

Avance Rápido y Backtracking

Autores:
Jose Antonio Lorencia Abril
Pablo Miralles González

Profesor:
Domingo Giménez Cánovas

Mayo 2019

Índice

1. Introducción **2**

2. Lista de problemas resueltos **3**

3. Avance Rápido **4**

3.1. Pseudocódigo del algoritmo y explicación, justificando las decisiones de diseño, la utilización de las variables y las funciones básicas del esquema algorítmico. 4

3.2. Programación del algoritmo (debe funcionar). El programa debe ir documentado, con explicación de qué es cada variable, qué realiza cada función y su correspondencia con las funciones básicas del esquema algorítmico correspondiente. 5

3.3. Estudio teórico del tiempo de ejecución del algoritmo. 6

3.4. Estudio experimental del tiempo de ejecución para distintos tamaños de problema. 8

3.5. Contraste de estudio teórico y experimental, buscando justificación a las discrepancias entre los dos estudios. 10

4. Backtracking **12**

4.1. Pseudocódigo del algoritmo y explicación, justificando las decisiones de diseño, la utilización de las variables y las funciones básicas del esquema algorítmico. 12

4.2. Programación del algoritmo (debe funcionar). El programa debe ir documentado, con explicación de qué es cada variable, qué realiza cada función y su correspondencia con las funciones básicas del esquema algorítmico correspondiente. 13

4.3. Estudio teórico del tiempo de ejecución del algoritmo. 14

4.4. Estudio experimental del tiempo de ejecución para distintos tamaños de problema. 14

4.5. Contraste de estudio teórico y experimental, buscando justificación a las discrepancias entre los dos estudios. 18

5. Conclusión **19**

1. Introducción

En este proyecto trataremos de resolver un problema con la técnica de diseño de Avance rápido y otro problema con la técnica de Backtracking. Estudiaremos el problema, diseñaremos una solución y resolveremos las cuestiones pedidas en el enunciado de la práctica. Resumimos aquí los problemas para mayor facilidad:

- AR_C. Nos dan n nodos con cierta distancia entre ellos, no necesariamente simétrica. La distancia de un nodo a sí mismo es 0. Escoger m nodos de forma que se maximice la suma de las distancias entre todos ellos.
- BT_H. Nos dan un jardín y una serie de aspersores en él. Cada aspersor tiene un caudal máximo, y riega alrededor suyo llegando a la distancia del caudal que seleccionemos. Hay también un caudal máximo que podemos emplear entre todos los aspersores. El objetivo es maximizar las zonas regadas.

2. Lista de problemas resueltos

- Avance Rápido (AR_C): En lo que respecta a Avance Rápido, el programa definitivo se encuentra en el envío 242 del Mooshak. Realizamos bastantes envíos para este problema, aunque la gran mayoría de ellos son debidos a errores de compilación provocados por la indentación que hace nuestro programa de edición de texto. Hay tres respuestas erróneas, pues en un principio atacamos el problema de una forma, que no resultó muy satisfactoria. Por último, hay 4 envíos aceptados, esto es debido a algunas mejoras de legibilidad, en su mayoría, y algunos cambios que vimos experimentalmente que mejoraban la eficiencia del programa.
- BackTracking (BT_H): En esta ocasión, el envío definitivo es el 248. En esta ocasión, realizamos pocos envíos. Hay un error de compilación, debido a la razón anteriormente explicada. Los demás envíos son todos aceptados, y se deben a mejoras en el criterio, como explicaremos en la sección correspondiente.

3. Avance Rápido

3.1. Pseudocódigo del algoritmo y explicación, justificando las decisiones de diseño, la utilización de las variables y las funciones básicas del esquema algorítmico.

Lo primero que haremos es explicar nuestra solución. Partimos del conjunto completo de elementos, y queremos ir eliminando hasta quedarnos con nuestra solución. Dado un estado del problema, es decir, un conjunto entre el total y nuestra solución, cada nodo añade de distancia a la solución la suma de su distancia con el resto de nodos (en ambos sentidos), es decir, si lo eliminamos, reduciríamos la distancia total en esa cantidad. Para minimizar esta pérdida, siempre cogeremos el menos valioso según el criterio anterior. Pasemos ahora a describir las variables utilizadas, son las siguientes:

- **seleccionados**: entero que indica el número de nodos eliminados de la solución en un momento dado.
- **totalDist**: entero que indica la suma de las distancias de los nodos que pertenecen a la solución en un momento dado.
- **nodoDescartado**: par de enteros, cuyo primer valor indica el siguiente nodo a eliminar, y su segundo valor indica el valor que resta a totalDist.
- **contribucion**: array de enteros que indica el valor que aporta un nodo concreto a la solución.
- **solucion**: array de booleanos que indica qué nodos pertenecen a la solución (true) y cuáles no (false).
- **distancias**: array bidimensional de enteros que guarda la distancia entre todo par de nodos.

Y veamos, por último, las funciones básicas del esquema algorítmico quedan reflejadas de la siguiente manera:

- **seleccionar**: selecciona el nodo que menos aporta a la solución parcial actual.
- **factible**: no es necesaria en nuestro problema, pues cualquier nodo es susceptible de ser eliminado. Además, seleccionar ya se asegura de que el nodo a eliminar no haya sido previamente eliminado.
- **insertar**: el nodo seleccionado por seleccionar, se pone a false en el array solución. Se resta la distancia de este a cada nodo, en ambos sentidos, actualizando el array Contribucion.

```
1  function Seleccionar(array [1..n][1..n] of integer} distancias, array [1..n] of boolean solucion, array
       [1..n] of integer contribucion) : pair of (integer, integer)
2  begin
3      {Cogemos el nodo mínimo en el array contribucion}
4      return pair(nodoMinimo, contribucionMinima);
5  end
6
7  procedure Insertar(array [1..n] of integer distancias, array [1..n] of boolean solucion, var integer
       totalDist, array [1..n] of integer contribucion, pair of (integer, integer) nodoDescartado, var
       integer seleccionados)
8  begin
9      solucion[nodoDescartado.first] = false; {Lo sacamos de la solución}
10     totalDist -= nodoDescartado.second; {Quitamos lo que aportaba ese nodo al total de la distancia}
11     for(int i=0; i<n; i++)
12         if(!solucion[i])
13             contribucion[i] -= (distancia[nodoDescartado.first][i]+distancia[i][nodoDescartado.first]);
       {Actualizamos lo que aporta cada nodo}
14     seleccionados++; {aumentamos el contador de seleccionados}
15 end
16
17 main
18 var
19     integer n, m, seleccionados, totalDist;
20     pair of (integer, integer) nodoDescartado;
21     array [1..n] of integer contribucion; {Array con el valor que cada nodo anade a la solucion}
22     array [1..n] of boolean solucion;
23     array [1..n][1..n] of integer distancias;
```

```
begin
    seleccionados = 0; {Indica el nmero de nodos eliminados, debe llegar a n-m}
    totalDist = {suma de todos los elementos de distancias, si ambos índices están en true en la solución; Indica el valor total actual de la solucion}

    while(seleccionados < n-m)
    begin
        nodoEliminado = seleccionar(distancias, solucion, contribucion);
        insertar(distancias, solucion, totalDist, contribucion, seleccionados, nodoEliminado);
    end

    {Ahora la solucion la componen los nodos que están a true en solucion y el valor total sumado es totalDist}
end
```

ar/codigo/ar_pseudo

3.2. Programación del algoritmo (debe funcionar). El programa debe ir documentado, con explicación de qué es cada variable, qué realiza cada función y su correspondencia con las funciones básicas del esquema algorítmico correspondiente.

```
#include <iostream>
#include <vector>
#include <set>
#include <algorithm>
#define infinity 1000000000
using namespace std;

int main()
{
    int cases, n, m;
    cin >> cases;
    while (cases--)
    {
        cin >> n >> m;
        // declaramos las estructuras de datos que necesitamos
        vector<vector<long long> > dist(n, vector<long long>(n, 0));
        vector<long long> adds(n, 0);
        vector<bool> sol(n, false);
        long long selected = 0;
        long long total_dist = 0;
        // leemos entrada
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                cin >> dist[i][j], total_dist += dist[i][j], adds[i] += dist[i][j], adds[j] += dist[i][j];

        // algoritmo en sí
        // seleccionamos el primer nodo (primera llamada a Seleccionar())
        pair<int, long long> x = pair<int, long long>(0, infinity);
        for (int i = 0; i < n; i++)
            if (adds[i] < x.second)
                x.first = i, x.second = adds[i];
        while (selected < n - m)
        {
            // insertar(dist, sol, total_dist, adds, selected, x, n);
            pair<int, long long> y = pair<int, long long>(0, infinity);
            // Insertar()
            sol[x.first] = true;
            total_dist -= x.second;
            for (int i = 0; i < n; i++)
                if (!sol[i])
                {
                    adds[i] -= (dist[x.first][i] + dist[i][x.first]);
                    /*
                    *   Aquí estamos guardando ya el mínimo de los nodos, para el Seleccionar()
                    */
```

```
                    if (adds[i] < y.second)
                        y.first = i, y.second = adds[i];
                }
            selected++;
            // Seleccionar() en sí
            x = y;
        }
        cout << total_dist << endl;
        for (int i = 0; i < n; i++)
            cout << !sol[i] << ((i < n - 1) ? " " : "");
        cout << endl;
    }
}
```

ar/codigo/ar.cpp

3.3. Estudio teórico del tiempo de ejecución del algoritmo.

Para el conteo de instrucciones utilizamos el código de c++, pues el pseudocódigo es más abstracto y la cuenta no sería fiel a la realidad. De esta forma, el algoritmo en sí se encuentra en el main, concretamente en el bucle while(selected <n-m).

- **Peor caso:** el peor caso es aquel en que adds[i] <y.second, $\forall i \in \{1..n\}$. En este caso ejecutamos:

$$t_M(n,m) = \sum_{i=0}^{n-m}(4 + \sum_{j=0}^{i} 1 + \sum_{j=i+1}^{n} 7 + 2) = \sum_{i=0}^{n-m}(6 + i + (n-i)\cdot 7)$$

$$= 6(n-m) + 7n(n-m) + \sum_{i=0}^{n-m} -6i = 7n(n-m) + 6(n-m) - 6\frac{(n-m)(n-m+1)}{2} =$$

$$= 4n(n-m) + 6(n-m) - 3n + 3m(n-m+1)$$

  Y, así:

$$t_M(n,m) \in \theta(n(n-m))$$

  Más aún,

$$t_M(n,m) \in o(4 \cdot n(n-m))$$

- **Mejor caso:** este corresponde con la situación en que solo se verifica adds[i] <y.second una única vez en todo el bucle. Quedaría:

$$t_m(n,m) = \sum_{i=0}^{n-m}(4 + \sum_{j=0}^{i} 1 + 7 + \sum_{j=i+2}^{n} 5 + 2) = \sum_{i=0}^{n-m}(13 + i + (n-i+1)\cdot 5) =$$

$$= \sum_{i=0}^{n-m}(13 + 5n - 4i + 5) = 18n + 5n(n-m) - 4\frac{(n-m)(n-m+1)}{2} =$$

$$= 16n + 3n(n-m) + 2m(n-m+1)$$

  Es decir:

$$t_m(n,m) \in \theta(n(n-m))$$

  Además

$$t_m(n,m) \in o(3 \cdot n(n-m))$$

- **Caso promedio:** el caso promedio es bastante complicado de calcular, pues debemos saber la probabilidad de que adds[i] <y.second. Vamos a llamar a esta probabilidad p{i,j}, de tal forma que queda (siendo EYE_i ={elementos ya excluidos antes del loop i} y ENE_i ={elementos no excluidos antes del loop i}), y teniendo en cuenta que sus cardinales son i y n-i respectivamente en cada iteración:

$$t_p(n,m) = \sum_{i=1}^{n-m}(6 + \sum_{j=1}^{i} 1 + \sum_{j=1}^{n-i} p_{i,j} \cdot 7 + \sum_{j=1}^{n-i}(1-p_{i,j}) \cdot 5) =$$

$$= \sum_{i=1}^{n-m}(6 + i + 7 \cdot \sum_{j=1}^{n-i} p_{i,j} + 5 \cdot \sum_{j=1}^{n-i}(1-p_{i,j})) =$$

$$= \sum_{i=1}^{n-m}(6 + i + 7 \cdot \sum_{j=1}^{n-i} p_{i,j} + 5 \cdot n - i - 5 \cdot \sum_{j=1}^{n-i}(p_{i,j})) =$$

$$= \sum_{i=1}^{n-m}(6 + i + 7 \cdot \sum_{j=1}^{n-i} p_{i,j} + 5 \cdot (n-i) - 5 \cdot \sum_{j=1}^{n-i}(p_{i,j})) =$$

$$= \sum_{i=1}^{n-m}(5n - 4i + 6 + 2 \cdot \sum_{j=1}^{n-i} p_{i,j}) =$$

$$= 5n(n-m) + 6(n-m) - 2(n-m)(n-m+1) + \sum_{i=1}^{n-m}(2 \cdot \sum_{j=1}^{n-i} p_{i,j}) =$$

$$= 3n(n-m) + 6(n-m) + 2m(n-m+1) + 2n + \sum_{i=1}^{n-m}(2 \cdot \sum_{j=1}^{n-i} p_{i,j})$$

De esta forma vemos, como ya sabíamos, que

$$t_p(n,m) \in \theta(n(n-m))$$

Y, más aún, si $p := \sum_{i=1}^{n-m}(\sum_{j=1}^{n-i} p_{i,j})$,

$$t_p(n,m) \in o(3n(n-m) + 2p)$$

Donde 2p puede afectar a la constante, pero siempre estará acotado por ese orden, es decir, no puede crecer más rápidamente que n(n-m).

Vamos a intentar afinar un poco más, recordemos que adds[i] guarda la contribución del nodo i al valor actual de la solución. Teniendo en cuenta que cuando esto ocurre, y.second pasa a ser adds[i], y suponiendo que este suceso es independiente de lo ocurrido en loops anteriores, se tiene que P(adds[i] <y.second)=$\frac{1}{j}$, recorriendo j los enteros entre 1 y la cantidad de elementos aun no tomados (card(ENE_i)). De esta forma:

$$t_p(n,m) = 3n(n-m) + 6(n-m) + 2m(n-m+1) + 2n + \sum_{i=0}^{n-m}(2 \cdot \sum_{j=0}^{card(ENE_i)} p_{i,j}) =$$

$$= 3n(n-m) + 6(n-m) + 2m(n-m+1) + 2n + \sum_{i=0}^{n-m}(2 \cdot \sum_{j=1}^{card(ENE_i)} \frac{1}{j}) =$$

Ahora bien

$$\sum_{j=1} card(ENE_i)\frac{1}{j} \approx \int_1^{card(ENE_i)+1} \frac{1}{j} dj = \int_1^{n-i+1} \frac{1}{j} dj =$$

$$= log(n-i+1)$$

Así,

$$t_p(n,m) \approx 3n(n-m) + 6(n-m) + 2m(n-m+1) + 2n + \sum_{i=0}^{n-m} (2 \cdot log(n-i+1)$$

De nuevo, debemos aproximar mediante una integral, siendo

$$\sum_{i=0}^{n-m} (log(n-i+1)) \approx \int_1^{n-m+1} log(n-i+1)di =$$

$$= (-(n-m+1) - log(n+1-(n-m+1))(n+1-(n-m+1))) - (-1 - log(n+1-1)(n+1-1)) =$$

$$= m - n - 1 - mlog(m) + 1 + nlog(n) = nlog(n) + m - n - mlog(m)$$

O sea, que

$$t_p(n,m) \approx 3n(n-m) + 6(n-m) + 2m(n-m+1) + n(1+logn) + m(1-logm)$$

Y este resultado nos da más información que el anterior, bajo las suposiciones antes expuestas, y, evidentemente, presenta el mismo orden de complejidad.

3.4. Estudio experimental del tiempo de ejecución para distintos tamaños de problema.

Comenzamos el apartado observando que el generador de casos debe producir matrices n^2, por lo que generar casos muy grandes sería muy costoso. Por motivos de tiempo, solo podemos producir casos de hasta $n = 10^4$. A pesar de esto, trataremos de esbozar el crecimiento en el tiempo del algoritmo respecto a las dos variables que nos incumben, n y m, y respecto a los distintos casos. Adjuntamos toda la tabla de datos obtenida, y luego procederemos a estudiarla en profundidad.

		Mejor caso		Caso promedio		Peor caso	
n	m	time	extra ops	time	extra ops	time	extra ops
10	1	0.004	9	0.011	22	0.009	44
10	2	0.004	8	0.013	12	0.009	43
10	3	0.003	7	0.01	15	0.006	42
10	4	0.004	6	0.01	15	0.005	39
10	5	0.003	5	0.009	16	0.005	35
10	6	0.002	4	0.007	10	0.003	30
10	7	0.002	3	0.005	12	0.003	24
10	8	0.002	2	0.005	6	0.002	17
10	9	0.001	1	0.003	2	0.001	9
10	10	0	0	0.002	0	0.001	0
100	14	0.205	86	0.521	430	0.241	4859
100	28	0.177	72	0.445	327	0.205	4572
100	42	0.147	58	0.37	237	0.173	4089
100	56	0.124	44	0.225	222	0.236	3410
100	70	0.083	30	0.13	140	0.101	2535
100	84	0.047	16	0.07	65	0.056	1464
100	98	0.009	2	0.012	7	0.011	197
1000	142	23.328	858	36.954	5986	39.702	489489
1000	284	20.62	716	34.216	5075	25.699	459314
1000	426	17.072	574	30.192	4301	18.889	408975
1000	568	13.441	432	24.545	3568	14.973	338472
1000	710	9.315	290	17.737	2018	10.986	247805
1000	852	5.158	148	10.228	1547	5.21	136974
1000	994	0.276	6	0.546	35	0.278	5979
10000	1428	3268.91	8572	6513.07	74966	3367.44	48976122
10000	2856	2720.27	7144	6271.31	64851	3073.37	45918060
10000	4284	2511.2	5716	7076.2	52648	2274.41	40820814
10000	5712	2121.93	4288	4173.53	42947	1828.25	33684384
10000	7140	1431.39	2860	3133.29	27524	1291.56	24508770
10000	8568	769.833	1432	1589.02	13258	689.84	13293972
10000	9996	7.266	4	4.995	20	3.61	39990
10	8	0.003	2	0.003	6	0.002	17
100	8	0.217	92	0.261	403	0.253	4922
1000	8	25.725	992	70.068	6663	26.822	499472
10000	8	3232.4	9992	7756.32	83016	3806.44	49994972

Cuadro 1: Tabla de datos completa

La primera observación es que los tiempos de ejecución entre los distintos casos no parecen tener mucho sentido. Observamos que el mejor caso siempre tiene menos tiempo de ejecución, pero el caso promedio parece tardar siempre más que el peor caso. Como ya vimos en el estudio teórico, la única diferencia entre los distintos casos es el número de veces que se ejecuta el cambio de mínimo en la búsqueda del mínimo. Es por esto que decidimos obtener el número de veces que el algoritmo entra en ese condicional. Estos datos los guardamos en las columnas 'extra ops'. En relación a este dato, desde luego más fiable, no hay contradicción alguna, las instrucciones ejecutadas en cada uno de los casos cumple exactamente la relación de orden que debe.

- n. Analicemos ahora como evoluciona el tiempo con m = 8.

		Mejor caso		Caso promedio		Peor caso	
n	m	time	extra ops	time	extra ops	time	extra ops
10	8	0.003	2	0.003	6	0.002	17
100	8	0.217	92	0.261	403	0.253	4922
1000	8	25.725	992	70.068	6663	26.822	499472
10000	8	3232.4	9992	7756.32	83016	3806.44	49994972

Cuadro 2: Crecimiento con m=8.

Podemos observar que cada vez que multiplicamos n por 10, se incremente el tiempo en aproximadamente 100, al menos en la mayoría de casos. Todo parece indicar entonces un crecimiento cuadrático.

- m. Fijamos ahora n=10000, y hacemos variar m.

		Mejor caso		Caso promedio		Peor caso	
n	m	time	extra ops	time	extra ops	time	extra ops
10000	1428	3268.91	8572	6513.07	74966	3367.44	48976122
10000	2856	2720.27	7144	6271.31	64851	3073.37	45918060
10000	4284	2511.2	5716	7076.2	52648	2274.41	40820814
10000	5712	2121.93	4288	4173.53	42947	1828.25	33684384
10000	7140	1431.39	2860	3133.29	27524	1291.56	24508770
10000	8568	769.833	1432	1589.02	13258	689.84	13293972
10000	9996	7.266	4	4.995	20	3.61	39990

Cuadro 3: Crecimiento con n=10000

Es difícil ver un patrón aquí, pues a pesar de parecer que en líneas generales hay un decrecimiento lineal, hay cierta arbitrariedad en los descensos. Esto es probablemente debido a las infinitas variables de las que depende el tiempo de ejecución de un programa. Nos podemos volver a ayudar del campo 'extra ops', pues en el mejor caso, ese contador se aumenta una sola vez por iteración. Esto nos confirma que efectivamente se hacen n-m iteraciones en total, aunque era obvio dado el código.

3.5. Contraste de estudio teórico y experimental, buscando justificación a las discrepancias entre los dos estudios.

Observamos primero que los casos mejor, peor y promedio esperados en el estudio teórico concuerdan con los que hemos observado en el estudio experimental, al menos respecto al número de operaciones, que representamos mediante el número de entradas al condicional en el cual se cambia el mínimo buscado. Como hemos visto, en el mejor caso se entra una vez en el condicional en cada iteración, que es lo mínimo que se puede entrar, pues hay que seleccionar uno. Respecto al peor caso, en cada iteración i se realizan n-i entradas en el condicional, que es lo máximo que se puede entrar, pues solo quedan n-i elementos entre los cuales seleccionar. Esto es especialmente fácil de comprobar en la tabla Tabla 1, en n=10, si comenzamos en m=10 y vamos subiendo, vemos que aumentamos en 9,8,7,6,... como habíamos calculado. El caso promedio se mantiene entre ambos en esta cuestión.

Respecto a tiempo de ejecución hemos observado cierta disparidad. Esto se puede deber a problemas de caché, pues en el peor caso se accede en escritura a las variables que llevan el mínimo continuamente, de forma que seguramente tengan más prioridad en caché que en el caso promedio, pudiendo generar más accesos a disco el caso promedio. El criterio más fiable sigue siendo, como decimos, el número de operaciones.

La Tabla 2 muestra, como habíamos previsto en el estudio teórico, que si n aumenta en un factor de k (en este caso k=3), t(n) aumentará, aproximadamente, en un factor k^2, pues:

$$t(n,m) \approx n(n-m) \Rightarrow t(kn,m) \approx kn(kn-m) \approx k^2 n(n-m)$$

Es más dificil confirmar nuestras especulaciones acerca del comportamiento respecto a m, pues parece decrecer linealmente, pero no siempre en la misma medida, lo cual es bastante normal dada la cierta arbitrariedad en el tiempo de

ejecución de un programa. Esta es la explicación más probable para encontrar un crecimiento casi lineal, pero con ciertos saltos en los decrementos.

4. Backtracking

4.1. Pseudocódigo del algoritmo y explicación, justificando las decisiones de diseño, la utilización de las variables y las funciones básicas del esquema algorítmico.

```
procedure Generar (nivel : integer, var s : TuplaSolución, var valorActual : integer,var caudalActual : integer, var jardinActual : array[1..L] of integer)
begin
    {añadimos al jardin la nueva zona regada en los extremos, y si regamos zonas nuevas las sumamos a valorActual}
    caudalActual++;
    s[nivel]++;
end

function MasHermanos(s: TuplaSolución, nivel: integer, c: Array[1..S] of integer) : boolean
begin
    return s[nivel] < c[nivel];
end

function Solución(nivel : integer) : boolean
begin
    return (nivel == L && caudalActual<=CaudalMax);
end

procedure Retroceder(nivel : integer, var s : TuplaSolución, var valorActual : integer, var caudalActual : integer, var jardinActual : array[1..L] of integer)
begin
    {eliminamos las cantidades correspondientes en valorActual y jardinActual}
    caudalActual -= s[nivel];
    s[nivel] = -1;
    nivel--;
end

function Criterio(nivel : integer, valorActual : integer, voa : integer, jardinActual : array[1..L] of integer, caudal : array[1..S] of integer) : boolean
var
    posibleMejora : integer
begin
    posibleMejora = valorActual + AR(jardinActual, caudal); {AR es un avance rápido, que calcula la mayor cantidad posible de riego que obtendremos si seguimos profundizando, no teniendo en cuenta el caudal máximo, y teniendo en cuenta las intersecciones de riego}
    return nivel < S and posibleMejora > voa and valorActual <= CaudalMax;
end

Backtracking (var s: TuplaSolución, caudal: Array[1..S] of integer, CaudalMax : integer)
var
    valorActual, caudalActual : integer;
    jardinActual : array[1..L] of integer;
begin
    nivel:= 1
    s:= sINICIAL;
    valorActual = caudalActual = 0;
    {jardin actual a cero, lleva el número de aspersores que riegan una zona}
    voa = -infinity;
    repetir
        Generar (nivel, s, valorActual, caudalActual, jardinActual)
        si Solución (nivel) and valorActual>voa entonces
            voa:=valorActual;
        si Criterio (nivel, valorActual, voa, jardinActual, caudal) entonces
            nivel:= nivel + 1
        sino
            mientras nivel>=1 && NOT MasHermanos (nivel, s, caudal)
                hacer Retroceder (nivel, s, valorActual, caudalActual, jardinActual)
        finsi
    hasta nivel==0
end
```

bt/codigo/pseudocodigo_backtracking

4.2. Programación del algoritmo (debe funcionar). El programa debe ir documentado, con explicación de qué es cada variable, qué realiza cada función y su correspondencia con las funciones básicas del esquema algorítmico correspondiente.

```
#include <iostream>
#include <vector>
#define infinity 10000
using namespace std;

bool criterio_4(const int &valorActual, const int &valor_max, const int &nivel, const vector<int> &
    caudal, const vector<int> &posicion, vector<int> jardinActual)
{
    int posible_mejora = 0;
    for (int i = nivel + 1; i < (int)caudal.size(); i++)
        for (int j = posicion[i] - caudal[i]; j <= posicion[i] + caudal[i]; j++)
            if (jardinActual[j]++ == 0)
                posible_mejora++;

    return valorActual + posible_mejora > valor_max;
}

int main()
{
    int cases, L, S, C;
    cin >> cases;
    while (cases--)
    {
        cin >> L >> S;
        vector<int> posicion(S);
        vector<int> caudal(S);
        for (int i = 0; i < S; i++)
            cin >> posicion[i], posicion[i]--;
        cin >> C;
        for (int i = 0; i < S; i++)
            cin >> caudal[i];

        // BACKTRACKING
        vector<int> solucion(S, -1);
        vector<int> jardinActual(L, 0);
        int valorActual = 0, nivel = 0, valorMax = -infinity, caudalUsado = 0;
        while (nivel >= 0)
        {
            // Generar()
            if (++solucion[nivel] == 1 && jardinActual[posicion[nivel]]++ == 0)
                valorActual++;
            if (solucion[nivel] > 0)
            {
                if (jardinActual[posicion[nivel] + solucion[nivel]]++ == 0)
                    valorActual++;
                if (jardinActual[posicion[nivel] - solucion[nivel]]++ == 0)
                    valorActual++;
                caudalUsado++;
            }
            // fin Generar()
            if (nivel == S - 1 && caudalUsado <= C /*Solucion()*/ && valorActual > valorMax /*
    Comprobamos si esta solucion es mejor*/)
            {
                valorMax = valorActual;
                if (valorMax >= L)
                    break;
            }
            if (nivel < S - 1 && caudalUsado <= C && criterio_4(valorActual, valorMax, nivel, caudal,
    posicion, jardinActual))
                nivel++;
            // fin Criterio()
            else
                // MasHermanos()
                while (nivel >= 0 && !(solucion[nivel] < caudal[nivel]))
                {
                    // Retroceder()
```

```
64                      for (int i = posicion[nivel] - solucion[nivel]; i < posicion[nivel] + solucion[
    nivel] + 1; i++)
65                          if (--jardinActual[i] == 0)
66                              valorActual--;
67                      caudalUsado -= solucion[nivel];
68                      solucion[nivel] = -1;
69                      nivel--;
70                  }
71          }
72          cout << valorMax << endl;
73      }
74  }
```

bt/codigo/codigo.cpp

4.3. Estudio teórico del tiempo de ejecución del algoritmo.

Para abordar el estudio teórico del algoritmo de backtracking, vamos a estimar el número máximo de nodos generados, y la complejidad de las operaciones. La forma de nuestro árbol no se corresponde de forma exacta con las vistas en las clases de teoría, ya que no tiene una cantidad fija de ramificaciones por nivel, pero tampoco es combinatorio ni permutacional. Así, el árbol tiene N niveles y en el nivel i, se obtendrán r_i ramificaciones, de tal forma que el número máximo de nodos es:

$$\sum_{k=1}^{N}(\prod_{i=1}^{k}(r_i+1))$$

Veamos ahora el orden de complejidad de cada una de las operaciones:

- Generar: Son entre 2 y 8 operaciones, es decir, un orden constante.
- Solución: Se ejecutan entre 1 y 3 operaciones, de nuevo, orden constante.
- Criterio: Hacemos unas comprobaciones de orden constante, y después utilizamos un algoritmo de avance rápido. Este algoritmo tiene un orden lineal en cada caudal de los niveles que restan. Podemos mirar el tiempo, sin tener en cuenta constantes:
$$t_{Criterio} \approx \sum_{i\in(nivelesrestantes)} (2 \cdot r_i + 1)$$
- MasHermanos: Es una simple comprobación, por tanto, de orden constante.
- Retroceder: Depende del nodo en el que nos encontramos, así, si estamos en el aspersor i, que tiene caudal r_i, quedaría:
$$t_{Retroceder} = 2 \cdot r_i \cdot (numOps) + 3$$
Con numOps igual a 1, o 2, dependiendo de la zona del jardín, en cualquier caso, obtenemos un orden de r_i.

4.4. Estudio experimental del tiempo de ejecución para distintos tamaños de problema.

Gracias a los casos generados por nuestro programa generador_de_casos.cpp, podemos introducirlos en el programa generador_de_tiempos.cpp y obtener una tabla con la información necesaria para comparar la relación del tiempo de ejecución con ciertas variables.

Las variables que estudiaremos son el número de nodos, el número de nodos máximo (teóricos), el número de nodos por milisegundo, y obtendremos también el número de zonas en el jardín y el número de aspersores, aunque solo este último influye sobre el número de nodos.

¿Qué compararemos en este estudio? Sobre todo el número de nodos frente a distintos criterios, para saber cuanto nos ayudan estos criterios, y también profundizaremos en el criterio obtenido.

- Tiempo de ejecución contra número de nodos. A continuación adjuntamos una tabla en la cual comparamos tres criterios distintos, para distintos tamaños del problema. Los resultados más destacados los resaltamos en verde, para que sea más fácil buscar en una tabla tan grande.

  - Criterio 1. Con criterio 1 designamos un criterio que devuelve siempre true, es decir, no hay criterio.
  - Criterio 2. Ahora utilizamos el criterio dado por el problema, es decir, que no sobrepasemos el límite de caudal total utilizado.
  - Criterio 3. Utilizamos un algoritmo de avance rápido para obtener una cota superior del máximo beneficio que podemos obtener en esta rama. Lo que hacemos es avanzar por los niveles que quedan, y sumamos a una posible solución el doble del caudal y uno más si el caudal es mayor que 0. Esta cota superior es muy mejorable, pues no tiene en cuenta la intersección. Sin embargo, tiene una orden lineal en el número de niveles restantes. Si la cota superior es peor que lo que ya tenemos, dejamos de avanzar.
  - Criterio 4. Utilizamos otro algoritmo de avance rápido, esta vez sí tenemos en cuenta las intersecciones, luego tendremos una mejor cota. Sin embargo, para cada nivel realizaremos un número lineal en el caudal del aspersor de operaciones, haciendo más costoso el criterio.

  Pasamos ahora a estudiar los datos de la tabla. Se puede observar que el número de nodos teóricos (NT en la tabla) corresponde con el número de nodos en el criterio vacío. Por otra parte, observamos una gran mejora en el criterio 2 respecto al 1 cuando el caudal máximo es bastante pequeño.

  Finalmente, observamos las grandes mejorías en varios casos gracias a nuestros criterios con algoritmos de avance rápido. Los resultado más significativos están marcados en verde. En varios casos obtenemos una gran mejora con el Criterio 3, y casi siempre la obtenemos con el criterio 4, si el tamaño no es trivial. Hay algunos casos en el que el número de nodos eliminados con el criterio 4 es tan grande que el algoritmo resulta mucho más rápido. Sin embargo, cuando esta diferencia no es muy grande, el mayor coste de tratar cada nodo acaba haciendo el algoritmo más lento, a pesar de tratar menos nodos. Al final, hemos decidido optar por el criterio 4, pues cuando empeora el tiempo, lo hace en una cantidad mucho menor que las mejoras que produce en otros casos. Las mejoras son mucho mayores, y con los casos de entrada del Mooshak iba desde luego más rápido.

L	S	C	NT	Criterio 1			Criterio 2			Criterio 3			Criterio 4		
				Nodos	Ms	Nodos/ms	Nodos	Ms	Nodos/ms	Nodos	Ms	Nodos/ms	Nodos	Ms	Nodos/ms
10	2	2	4	4	0.002	2000	4	0.001	4000	4	0.001	4000	4	0.002	2000
10	2	4	6	6	0.001	6000	6	0.001	6000	6	0.002	3000	6	0.002	3000
10	2	8	3	3	0.002	1500	3	0.002	1500	3	0.002	1500	3	0.002	1500
10	2	16	2	2	0.001	2000	2	0.001	2000	2	0.001	2000	2	0.001	2000
10	5	2	20	20	0.002	10000	16	0.002	8000	16	0.003	5333.33	16	0.005	3200
10	5	4	9	9	0.002	4500	9	0.002	4500	9	0.002	4500	9	0.004	2250
10	5	8	38	38	0.004	9500	38	0.003	12666.7	32	0.004	8000	14	0.006	2333.33
10	5	16	5	5	0.002	2500	5	0.001	5000	5	0.001	5000	5	0.003	1666.67
20	2	2	8	8	0.005	1600	7	0.002	3500	7	0.001	7000	7	0.002	3500
20	2	4	8	8	0.002	4000	8	0.002	4000	8	0.002	4000	8	0.003	2666.67
20	2	8	6	6	0.001	6000	6	0.002	3000	6	0.001	6000	6	0.002	3000
20	2	16	12	12	0.002	6000	12	0.002	6000	12	0.002	6000	12	0.003	4000
20	2	32	14	14	0.003	4666.67	14	0.002	7000	14	0.002	7000	14	0.002	7000
20	5	2	15	15	0.002	7500	15	0.002	7500	15	0.003	5000	15	0.006	2500
20	5	4	162	162	0.012	13500	126	0.009	14000	96	0.009	10666.7	96	0.024	4000
20	5	8	268	268	0.022	12181.8	268	0.02	13400	268	0.021	12761.9	60	0.014	4285.71
20	5	16	15	15	0.002	7500	15	0.002	7500	15	0.003	5000	15	0.006	2500
20	5	32	116	116	0.009	12888.9	116	0.01	11600	74	0.008	9250	46	0.015	3066.67
20	8	2	127	127	0.011	11545.5	76	0.015	5066.67	76	0.01	7600	76	0.025	3040
20	8	4	648	648	0.05	12960	316	0.027	11703.7	198	0.026	7615.38	112	0.049	2285.71
20	8	8	354	354	0.028	12642.9	354	0.029	12206.9	314	0.031	10129	37	0.017	2176.47
20	8	16	6604	6604	0.477	13844.9	6604	0.416	15875	6604	0.476	13873.9	67	0.032	2093.75
20	8	32	93	93	0.009	10333.3	93	0.008	11625	54	0.007	7714.29	25	0.011	2272.73
30	2	2	2	2	0.004	500	2	0.001	2000	2	0.001	2000	2	0.002	1000
30	2	4	28	28	0.003	9333.33	22	0.003	7333.33	22	0.003	7333.33	19	0.004	4750
30	2	8	24	24	0.003	8000	24	0.003	8000	24	0.003	8000	24	0.006	4000
30	2	16	4	4	0.001	4000	4	0.001	4000	4	0.002	2000	4	0.002	2000
30	2	32	20	20	0.002	10000	20	0.003	6666.67	20	0.003	6666.67	20	0.005	4000
30	5	2	1269	1269	0.084	15107.1	98	0.009	10888.9	86	0.009	9555.56	86	0.018	4777.78
30	5	4	403	403	0.026	15500	152	0.012	12666.7	134	0.012	11166.7	134	0.024	5583.33
30	5	8	138	138	0.012	11500	126	0.01	12600	86	0.01	8600	70	0.025	2800
30	5	16	90	90	0.006	15000	90	0.008	11250	45	0.006	7500	45	0.016	2812.5
30	5	32	265	265	0.019	13947.4	265	0.019	13947.4	125	0.012	10416.7	59	0.018	3277.78
30	8	2	5136	5136	0.309	16621.4	155	0.013	11923.1	139	0.015	9266.67	133	0.04	3325
30	8	4	4272	4272	0.258	16558.1	436	0.033	13212.1	436	0.038	11473.7	420	0.1	4200
30	8	8	4510	4510	0.267	16891.4	1775	0.11	16136.4	992	0.104	9538.46	896	0.205	4370.73
30	8	16	62	62	0.006	10333.3	62	0.005	12400	38	0.005	7600	38	0.016	2375
30	8	32	2084	2084	0.14	14885.7	2084	0.13	16030.8	1730	0.116	14913.8	206	0.082	2512.2
30	11	2	34029	34029	1.237	27509.3	520	0.036	14444.4	460	0.033	13939.4	460	0.078	5897.44
30	11	4	180496	180496	6.997	25796.2	2921	0.154	18967.5	2508	0.163	15386.5	1908	0.456	4184.21
30	11	8	1854	1854	0.136	13632.4	1718	0.13	13215.4	1501	0.13	11546.2	716	0.156	4589.74
30	11	16	29875	29875	1.597	18707	29689	1.152	25771.7	21347	1.033	20665.1	117	0.038	3078.95
30	11	32	2993	2993	0.198	15116.2	2993	0.2	14965	1813	0.166	10921.7	55	0.033	1666.67
30	14	2	91486192	91486192	3642.8	25114.2	1757	0.107	16420.6	1684	0.109	15449.5	1652	0.336	4916.67
30	14	4	369306	369306	17.165	21515.1	7199	0.414	17388.9	6878	0.374	18390.4	6041	0.893	6764.84
30	14	8	2145672	2145672	105.838	20273.2	64062	4.374	14646.1	52224	2.927	17842.2	51610	7.927	6510.66
30	14	16	28787450	28787450	1180.47	24386.5	3288923	140.688	23377.4	2153399	114.95	18733.4	901	0.394	2286.8
30	14	32	462474	462474	22.362	20681.2	462474	22.002	21019.6	286300	21.421	13365.4	253	0.146	1732.88

Cuadro 4: Comparación de los criterios 1, 2, 3 y 4

- Análisis de datos para el criterio 4 (Cuadro 5). Haremos algunas observaciones, aunque ya se han dicho varias cosas en el apartado anterior.

  - Si observamos los datos en verde en la tabla, podemos deducir que el número de nodos sin poda no tiene nada que ver con el caudal máximo, aunque este sí interviene en la poda con el criterio 2.
  - En los datos en naranja encontramos un ejemplo de crecimiento en el tamaño del jardin y decrecimiento en del número de nodos sin poda. Esto nos indica que no tienen por qué estar relacionados, aunque el tamaño de jardin acota los caudales de los aspersores, por lo que un jardin mayor permite más nodos.
  - Podemos buscar cierto patrón sobre en qué condiciones se da una poda muy grande. Por supuesto, solo nos fijamos en tamaños considerables. En gris marcamos algunos casos que consideramos representativos del patrón que parece existir. Si los mayores caudales se concentran el la primera parte, y los menores al final, es más probable que se pode con el criterio 4, pues una vez hayamos seleccionado una solución buena, y estemos por los últimos niveles, la suma de los restante no añadiría mucho, y la cota superior sería muy cercana a la solución que estemos tratando en ese momento.

L	S	C	Nodos sin criterio	Listado de caudales	Nodos	Ms	Nodos/ms
10	2	2	6	1, 1	6	0.007	857.143
10	2	4	10	1, 3	6	0.003	2000
10	2	8	4	1, 0	4	0.002	2000
10	2	16	6	1, 1	6	0.002	3000
10	5	2	178	1, 3, 2, 1, 1	51	0.016	3187.5
10	5	4	38	1, 2, 0, 1, 0	18	0.006	3000
10	5	8	31	0, 1, 1, 1, 1	25	0.006	4166.67
10	5	16	135	2, 1, 2, 1, 1	31	0.007	4428.57
20	2	2	20	1, 8	20	0.005	4000
20	2	4	12	2, 2	9	0.002	4500
20	2	8	8	1, 2	5	0.002	2500
20	2	16	24	2, 6	10	0.003	3333.33
20	2	32	24	2, 6	10	0.003	3333.33
20	5	2	3927	2, 5, 6, 2, 8	228	0.027	8444.44
20	5	4	60	2, 0, 1, 1, 2	40	0.009	4444.44
20	5	8	3020	4, 2, 7, 3, 4	100	0.02	5000
20	5	16	2064	5, 6, 2, 2, 3	59	0.014	4214.29
20	5	32	952	3, 2, 5, 3, 1	51	0.013	3923.08
20	8	2	37658	1, 8, 2, 5, 0, 5, 2, 4	382	0.057	6701.75
20	8	4	4772	1, 2, 1, 3, 1, 2, 4, 1	697	0.13	5361.54
20	8	8	5418	5, 1, 1, 1, 2, 3, 1, 2	421	0.121	3479.34
20	8	16	18494	1, 2, 2, 5, 9, 0, 2, 3	76	0.024	3166.67
20	8	32	1868	1, 2, 0, 2, 1, 1, 5, 2	77	0.02	3850
30	2	2	14	1, 5	14	0.002	7000
30	2	4	3	0, 1	3	0.002	1500
30	2	8	6	1, 1	6	0.003	2000
30	2	16	12	1, 4	12	0.003	4000
30	2	32	10	1, 3	10	0.002	5000
30	5	2	1135	4, 1, 3, 2, 7	170	0.022	7727.27
30	5	4	1175	4, 1, 3, 6, 2	318	0.056	5678.57
30	5	8	2082	2, 6, 1, 7, 4	516	0.106	4867.92
30	5	16	1267	0, 5, 1, 7, 11	29	0.012	2416.67
30	5	32	7730	9, 3, 2, 6, 7	106	0.033	3212.12
30	8	2	188363	6, 6, 2, 4, 4, 4, 2, 1	422	0.062	6806.45
30	8	4	84036	5, 4, 6, 2, 1, 4, 3, 1	1812	0.299	6060.2
30	8	8	59306	1, 1, 4, 11, 1, 1, 9, 4	5376	0.975	5513.85
30	8	16	88550	9, 1, 1, 6, 6, 1, 2, 5	184	0.053	3471.7
30	8	32	31974	2, 0, 1, 6, 3, 6, 1, 11	82	0.035	2342.86
30	11	2	5601426	2, 0, 8, 2, 7, 2, 8, 0, 5, 3, 11	1245	0.199	6256.28
30	11	4	1780776	7, 10, 4, 1, 1, 4, 0, 7, 5, 0, 1	3968	0.675	5878.52
30	11	8	2046716	1, 2, 2, 4, 2, 12, 1, 1, 3, 6, 3	68641	10.958	6264.01
30	11	16	1766847	2, 11, 1, 1, 3, 0, 4, 1, 4, 11, 3	158	0.067	2358.21
30	11	32	16618495	4, 1, 7, 9, 2, 6, 3, 5, 3, 2, 1	208	0.079	2632.91
30	14	2	282653787	2, 7, 1, 10, 7, 14, 1, 2, 1, 1, 1, 0, 14, 4	2618	0.422	6203.79
30	14	4	440852242	1, 7, 3, 2, 7, 1, 8, 7, 7, 1, 2, 3, 2, 1	20834	3.343	6232.13
30	14	8	22231386388	3, 5, 14, 4, 2, 1, 9, 14, 1, 9, 4, 7, 3, 2	569177	88.453	6434.8
30	14	16	69374027	0, 1, 1, 4, 3, 5, 5, 2, 2, 4, 1, 13, 5, 1	233	0.096	2427.08
30	14	32	127250784	3, 4, 5, 1, 4, 12, 1, 8, 0, 10, 3, 0, 3, 0	311	0.15	2073.33

Cuadro 5: Datos para el criterio 4

4.5. Contraste de estudio teórico y experimental, buscando justificación a las discrepancias entre los dos estudios.

En este caso solo podemos comparar un par de cosas, pues el estudio teórico en Backtracking es bastante difícil, sobre todo por las podas.

Sí observamos, como ya hemos dicho, que el número de nodos que habíamos calculado estaba bien, como podemos comprobar en los nodos generados sin criterio. También observamos que nuestro criterio hace que se recorran más lentamente los nodos, pues cada vez que se aplica se hacen operaciones lineales respecto a los caudales y el número de niveles por delante.

5. Conclusión

Con este proyecto hemos aprendido varias cosas. Una de ellas es el balance entre obtener la solución óptima, y resolver el problema en un tiempo aceptable. Sin duda, antes de este proyecto no nos habíamos planteado el hecho de programar algo que no nos fuese a dar la solución exacta, pero ahora entendemos las motivaciones que pueden llevar a esta decisión. Sobre todo, hemos aprendido lo que facilita el diseño y programación de una solución el hecho de pensar un problema con detenimiento y con un esquema algorítmico. Esto te aleja del problema y te ayuda a tener una visión panorámica.

Técnica	Apartado	Tiempo empleado
Avance Rápido	Diseño	1h
	Programación	1h
	Estudio	4h
Backtracking	Diseño	1h
	Programación	1h
	Estudio	6h