Ampliación de Programación

Práctica 5

Algoritmos Backtracking

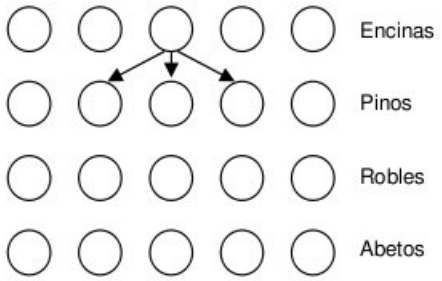
Sergio García Mondaray

Escuela Superior de Informática de Ciudad Real
Universidad de Castilla-La Mancha

1 Backtracking de arriba a abajo

1.1 Enunciado

Un leñador trabaja en una plantación en forma de cuadrícula de MxN árboles. Cada fila del bosque está formada por M árboles del mismo tipo, y el bosque tiene N filas de árboles de diferente tipo. Cada árbol tiene una anchura propia A que determina el tiempo que tarda en ser talado. Un leñador tiene que talar N árboles todos de tipos diferentes. Para ello, el leñador va talando el bosque de fila en fila, de forma ordenada. El leñador sólo puede talar los árboles de la fila siguiente que están o bien en la vertical o en la diagonal con respecto al árbol que acaba de talar. Véase la figura adjunta. El problema consiste en encontrar una solución, mediante una metodología Backtracking, que minimice el tiempo que tarda el leñador en talar N árboles, cada uno de diferente tipo.



1.2 Generación aleatoria del bosque

Nuestra forma de representar el conjunto de árboles será una matriz bidimensional, donde las filas representarán los tipos de árboles. En cada posición de la matriz se almacenará el tiempo necesario para talar el árbol que ocupa esa posición. Tanto el número de filas, el de columnas, como los valores de tiempo en talar cada árbol son generados aleatoriamente por el programa.

1.3 Planteamiento

Nuestra forma de proceder será la siguiente:

Comenzaremos en la fila de arriba, y en cada llamada recursiva de nuestro algoritmo, se seleccionará uno de los árboles de la fila inferior de entre todos los accesibles. Iremos construyendo todos los caminos posibles, y almacenando la mejor solución de todas las que vayamos construyendo. De esta forma garantizaremos que nuestra solución sea óptima.

La solución que genera nuestro algoritmo será un vector de n elementos (siendo n el número de tipos de árboles del bosque). En la posición i de la solución se encontrará la coordenada j (número de columna) del árbol elegido en la fila i. De esta manera resultará sencillo reconstruir el camino óptimo.

1.4 Algoritmo empleado

Para construir el árbol de todos los caminos posibles, emplearemos un algoritmo recursivo definido de la siguiente manera:

```
FUNCION backtracking(int fila, int n, int m, int ini, int fin, struct TSol Sol,
                                                          struct TSol* SolOpt)
INICIO

    //Podamos si no vamos a conseguir una solucion mejor:
    SI (Sol.tiempo < SolOpt->tiempo)
        SI (fila == n) { //Ya hemos terminado
            copySol(&Sol, SolOpt, n);
        SI NO
            PARA j = ini MIENTRAS j <= fin INCREMENTO j++
                Sol.camino[fila] = j;
                Sol.tiempo <- Sol.tiempo + bosque[fila][j];
                SI (j == 0)
                    nini <- 0;
                    nfin <- 1;
                SI NO, SI (j == m - 1)
                    nini <- m - 2;
                    nfin <- m - 1;
                SI NO
                    nini <- j - 1;
                    nfin <- j + 1;
                FIN SI

                //Lamada recursiva:
                backtracking(fila + 1, n, m, nini, nfin, Sol, SolOpt);
                //Y ahora deshacemos:
                Sol.camino[fila] = 0;
                Sol.tiempo <- Sol.tiempo - bosque[fila][j];
            FIN PARA
        FIN SI
    FIN SI

FIN
```

1.5 Ejemplo práctico

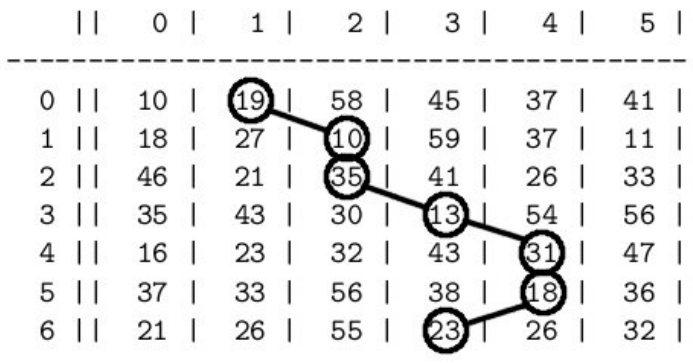
Estudiemos un ejemplo del funcionamiento de nuestro algoritmo. El bosque generado por nuestro programa, completamente de forma aleatoria, es el siguiente:

```
BOSQUE GENERADO (filas = tipos, columnas = arboles)

    ||   0 |   1 |   2 |   3 |   4 |   5 |
-----------------------------------------
  0 ||  10 |  19 |  58 |  45 |  37 |  41 |
  1 ||  18 |  27 |  10 |  59 |  37 |  11 |
  2 ||  46 |  21 |  35 |  41 |  26 |  33 |
  3 ||  35 |  43 |  30 |  13 |  54 |  56 |
  4 ||  16 |  23 |  32 |  43 |  31 |  47 |
  5 ||  37 |  33 |  56 |  38 |  18 |  36 |
  6 ||  21 |  26 |  55 |  23 |  26 |  32 |
```

Y la solución que el algoritmo da, para dicho bosque, es la siguiente (tal cual es mostrada al ejecutar programa):

```
SOLUCION BACKTRACKING (149 minutos):

  Fila  0 -> Arbol  1 ( 19 minutos)
  Fila  1 -> Arbol  2 ( 10 minutos)
  Fila  2 -> Arbol  2 ( 35 minutos)
  Fila  3 -> Arbol  3 ( 13 minutos)
  Fila  4 -> Arbol  4 ( 31 minutos)
  Fila  5 -> Arbol  4 ( 18 minutos)
  Fila  6 -> Arbol  3 ( 23 minutos)
```

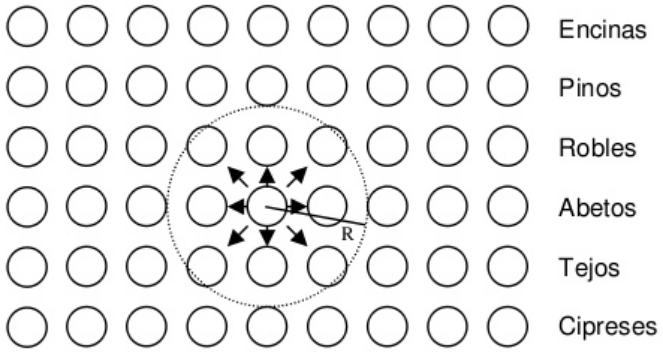
Gráficamente, el camino generado (cuya solución es óptima), queda así:



2 Backtracking modificado (con radio de alcance)

2.1 Enunciado

El leñador puede talar cualquier árbol que esté en un radio R con respecto al que acaba de talar. Para ello, los árboles están separados una distancia constante entre sí. Véase la figura que sigue. El problema consiste en encontrar una solución que minimice el tiempo que tarda el leñador en talar N árboles, cada uno de diferente tipo.



2.2 Generación aleatoria del bosque

Nuestra forma de representar el conjunto de árboles será una matriz bidimensional, donde las filas representarán los tipos de árboles. En cada posición de la matriz se almacenará el tiempo necesario para talar el árbol que ocupa esa posición. Tanto el número de filas, el de columnas, y los valores de tiempo en talar cada árbol son generados aleatoriamente por el programa.

2.3 Planteamiento

Para resolver este problema, calcularemos todos los caminos posibles desde todos los árboles del bosque. La representación del bosque no varía con respecto al problema Backtracking anterior.

Nuestra forma de proceder, será la siguiente:

Tendremos un algoritmo principal que, dado un árbol, considere todos los posibles caminos que partan de él y cumplan nuestros requisitos: sólo pase por un árbol de cada fila, siempre que la distancia entre un árbol y el siguiente no sea mayor que el radio R que consideremos. Además cada camino debe pasar por todas las filas. Por otro lado, contaremos con un algoritmo auxiliar que símplemente lance el algoritmo principal para todos los árboles del bosque.

A medida que construyamos cada camino, comprobaremos si el tiempo empleado en talar los árboles que componen dicho camino es menor que la del mejor camino encontrado hasta el momento (es decir, es una mejor solución).

La solución que devolverá nuestro algoritmo será un vector de n posiciones (siendo n el número de filas, es decir, el número de tipos de árboles del bosque). En la posición i se encontrarán las coordenadas del árbol talado en i-ésimo lugar.

2.4 Algoritmo empleado

El algoritmo principal, que dado un árbol busca todos los caminos válidos que parten de él, es el siguiente:

```
FUNCION backtracking(int x0, int y0, double R, int n, int m, struct TSol Sol,
                                                          struct TSol* SolOpt)
INICIO

    Sol.cogidas[x0] <- 1; //Marcamos la fila donde estamos como cogida
    SI (Sol.tiempo < SolOpt->tiempo) //Podamos
        SI (Sol.numar > n-2) //Hemos encontrado una solucion
            copySol(&Sol, SolOpt, n);
        SI NO
            PARA i=0 MIENTRAS i<n INCREMENTO i++
                PARA j=0 MIENTRAS j<m INCREMENTO j++
                    SI (Sol.cogidas[i]!=1 && distancia(x0,y0,i,j) <= R)
                    //Hemos encontrado un arbol talable
                        Sol.numar <- Sol.numar + 1;
                        Sol.tiempo <- Sol.tiempo + bosque[i][j];
                        Sol.camino[Sol.numar].fila <- i;
                        Sol.camino[Sol.numar].columna <- j;

                        //Llamda recursiva:
                        backtracking(i, j, R, n, m, Sol, SolOpt);

                        //Ahora deshacemos:
                        Sol.tiempo <- Sol.tiempo - bosque[i][j];
                        Sol.camino[Sol.numar].fila <- -1;
                        Sol.camino[Sol.numar].columna <- -1;
                        Sol.numar <- Sol.numar - 1;
                    FIN SI
                FIN PARA
            FIN PARA
        FIN SI
    FIN SI
    Sol.cogidas[x0] <- 0; //Desmarcamos la fila como cogida

FIN


Estructura de datos TSol{
    int tiempo;
    struct point camino[NM_MAX];
    int numar;
    int cogidas[NM_MAX];
}

Estructura de datos point{
    int fila;
    int columna;
}
```

2.5 Ejemplo práctico

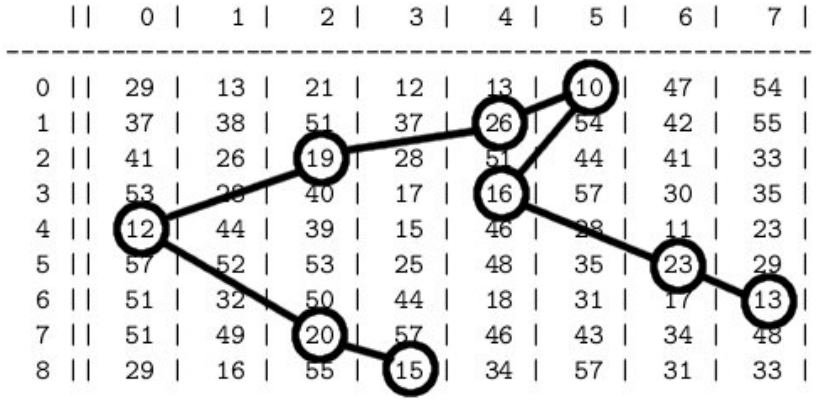
Vamos a estudiar uno de los ejemplos generados por nuestro programa. El bosque generado, aleatoriamente, es el siguiente:

```
BOSQUE GENERADO (filas = tipos, columnas = arboles)

   ||   0 |   1 |   2 |   3 |   4 |   5 |   6 |   7 |
-----------------------------------------------------
 0 ||  29 |  13 |  21 |  12 |  13 |  10 |  47 |  54 |
 1 ||  37 |  38 |  51 |  37 |  26 |  54 |  42 |  55 |
 2 ||  41 |  26 |  19 |  28 |  51 |  44 |  41 |  33 |
 3 ||  53 |  28 |  40 |  17 |  16 |  57 |  30 |  35 |
 4 ||  12 |  44 |  39 |  15 |  46 |  28 |  11 |  23 |
 5 ||  57 |  52 |  53 |  25 |  48 |  35 |  23 |  29 |
 6 ||  51 |  32 |  50 |  44 |  18 |  31 |  17 |  13 |
 7 ||  51 |  49 |  20 |  57 |  46 |  43 |  34 |  48 |
 8 ||  29 |  16 |  55 |  15 |  34 |  57 |  31 |  33 |
```

Estudiemos la solución que nos ofrece el algoritmo. La salida del programa, copiada tal cual, es la siguiente:

```
SOLUCION BACKTRACKING (154 minutos) (Radio = 4)

        Fila  6 -> Arbol  7 ( 13 minutos)
        Fila  5 -> Arbol  6 ( 23 minutos)
        Fila  3 -> Arbol  4 ( 16 minutos)
        Fila  0 -> Arbol  5 ( 10 minutos)
        Fila  1 -> Arbol  4 ( 26 minutos)
        Fila  2 -> Arbol  2 ( 19 minutos)
        Fila  4 -> Arbol  0 ( 12 minutos)
        Fila  7 -> Arbol  2 ( 20 minutos)
        Fila  8 -> Arbol  3 ( 15 minutos)
```

El orden en que se talan los árboles de las posiciones [Fila][Arbol] mostradas arriba es tal y como aparece en la salida: de arriba a abajo. Por tanto, visualmente nuestra solución sería la que se muestra a continuación:



3 Tiempo de trabajo y valoración de dificultad

Aproximadamente, he invertido unas 5 horas resolviendo esta práctica y unas 2 horas redactando la memoria. La valoración de dificultad total que le asigno a esta tarea es de 6 sobre 10.