APUNTES ALGORÍTMICA – ALBERTO JESÚS DURÁN LOPEZ

DGIIM 16/17

Problema de la mochila

M=100

C={1,2,3,4,5}

T={0,0,0,0,0} peso actual = 0

i=3   (0,0,1,0,0)   = 30

i=5   (0,0,1,0,1)   = 80

i=4   (0,0,1,0.5,1)   (100-80)/40 = (M - Peso actual)/Wi = 0.5

Coste = 1*66 + 0.5*40 + 1*60 = 146

Como no es la solución óptima, hay que dar un contraejemplo deun coste que sea mayor.

Problema de planificación de tareas.

Lista de candidatos: Las tareas

Lista C. U. : Las tareas despachadas

Criterio de factibilidad: Se cumple siempre

Función objetivo: min{ sum(Tiempo hasta que i finalice)}

Función solución: Que todas las tareas hayan sido insertadas

Función de selección: Seleccionar tarea i con mínimo Ti

P[0...N-1] = Planificador (t[0...N-1])

```
C <-{0,1,...N-1} // Tareas
P <- 0
MIENTRAS(|C| >0){
  i <- SELECCIÓN DE TAREA EN C CON MINIMO Ti
  C <- C\{i}
  P <- P U {i}

}
DEVOLVER P
```

EJERCICIOS:

1) Coloreo de un grafo:

Sea un grafo no dirigido y conexo; y un número máximo de colores M.

Encontrar la asignación de colores mínima entre los colores y los nodos de un grafo tal que no existan dos vértices adyacentes con el mismo color.

Solución:

Nº colores M=5

k= número de colores mínimo

```
    V
  M-/ \-M
    \ /
    V
```

Solución:{M,V,M,M,V} K=2

Lista de candidatos: Los M colores

L.C.U: Los colores usados

F. Selección: Escoger un color cualquiera

F. Factibilidad: Dos nodos adyacentes no pueden tener el mismo color

F. Solución: Todos los nodos rellenos

F. Objetivo: Minimizar el nº de colores a usar

BACKTRACKING:

Cuando no se tienen información sobre el problema se usa backtracking

En que consiste: Se explora el arbol entero mediante búsqueda en profundidad

Diseño: fácil de diseñar e implementar

Eficiencia: Pocos eficientes

Diseño de algoritmo:

-Buscar una representación del tipo T=(x1,x2,...xt)

-Diseñar las restricciones implícitas

-Identificar las restricciones explícitas: restricciones del problema

  -En el problema de la mochila 0/1.. que los objetos no superen el peso de la mochila

  -En el problema de las 8 reinar que no se hagan jaque mate

-Criterio de parada = Función objetivo ==función solución de greedy

-Diseñar una función de poda Bk(x1,x2,...,xk)

-Diseñar la estructura del arbol/grafo implícito

Branch and bound:

Eficiencia: En el peor de los casos (depende de lo bueno que seas poniendo cotas

en tu solución) tiene eficiencia exponencial

Se basa en el cálculo de cotas

-Función selección

BACKTRACKING - PROBLEMA DEL VIAJANTE DEL COMERCIO

1)

T -> Solución. Array de nodos visitados de tam N, T={x1,x2,...,xn}

N -> Nº nodos

Ejemplo T={1,2,3,4,5}, T={4,2,1,5,3}

2)

Restricciones implícitas:

xi e {1,2,...,N}

3)

Restricciones explícitas (Que no se repita ningún nodo)

Para todo i,j, i!=j T[xi] != T[xj]

Ejemplo: T={1,4,1,2,3}

4)

Función objetivo

Que la solución sea óptima, es necesario, por tanto, visitar todo el árbol de estados

5)

Función de poda

Si existe xi,xk con T[xi]=T[xk] se explora en el nivel k (i<k)

6)

Estructura del árbol de estados

Como es un circuito, no hay nodo inicial por lo que el primer estado es una

posición del vector de nodos

-Estado inicial:

T={1} -> Sup. partimos desde el nodo 1 (x1=1)

-Acciones del nivel i: ¿A donde voy después de este nodo?

Se decide la i-ésima +1 ciudad a visitar

-Profundidad:

Haber llegado al nivel N-1 del árbol

Cuando T tenga N componentes

7)Adaptar backtracking al problema que tengo

Previo:

```
MejorSolución (MS) <- {}
Coste MS <- +infinito  (ya que queremos minimizar)
L <- Matriz adyacencia
N <- Nº nodos
T <- {1}
k <- 1

Llamada: BT(k,T,MS,L,N)
  MS -> Parámetro de salida

ALGORITMO BT(K,T,MS,L,N)
  PARA CADA VALOR Xk+1 e {2,...,N}
    Si no existe T[i]=Xk+1(i<=K){
      T[K+1] <- Xk+1
        Si(K=N-1){
          Si Coste(T) < Coste(MS){
            MS <- T
          }
        } en otro caso{
          BT(K+1,T,MS,L,N)
        }
```

}

```
    0   1   2   2   7
A=  10  0   10  5   1
    2   6   0   10  9
    10  10  5   0   10
    7   10  9   10  0
```

```
        {1}
A{1,2}   B{1,3}   C{1,4}   D{1,5}
```

BRANCH AND BOUND

Cálculo cota superior inicial:

Mejor solución (MS) = {1,2,3,4,5,...,N}

Cálculo cotas inferiores:

Cota(T(1...K)) = Coste(T(1...K)) + Cota(T(K)) + SUM cota(i) * NoUsado(i,T)

Cota(A)=1+1+2+5+7=16   //Nos quedamos con esta que es la menor

Cota(B)=2+1+6+10+7=24

Cota(C)=2+1+2+5+7=17

Cota(D)=7+5+2+5+9=28

```
                {1}
        A{1,2}   B{1,3}   C{1,4}   D{1,5}
E{1,2,3}   F{1,2,4}   G{1,2,5}
```

Cota(E)=1+10+9+10+7=37

Cota(F)=1+5+2+5+7=20

Cota(G)=1+1+2+5+9=18

//No nos quedamos con ninguno ya que al ser una cola con prioridad

nos quedamos con el 17 de arriba

{1}

A{1,2}   B{1,3}   C{1,4}   D{1,5}

H{1,4,2}   I{1,4,3}   J{1,4,5}

Cota(H)=2+10+1+2+7=22

Cota(I)=2+5+1+6+7=21

Cota(J)=2+10+10+2+9=33

PROGRAMACIÓN DINÁMICA

```
Funcion PlayOffs(i,j,p)
  Si i=0 entonces Devolver 1
  Si j=0 entonces Devolver 0
  Devolver p*PlayOffs(i-1,j,p) + (1-p)*PlayOffs(i,j-1,p)
```

Calcular ecuación en recurrencia de la función anterior:

      1 ; caso base

T(n) 2*T(n-1) + 1; Caso general

Principio de optimalidad de Bellman: Si una secuencia de pasos para resolver un problema es óptima, entonces cualquier subsecuencia de estos pasos también es óptima.

El problema del cambio de monedas:

- Hay monedas de n valores diferencies.
- Las monedas tipo i tienen valor $d_i>0$
- Las monedas están ordenadas por su valor, en orden creciente
- Al cliente hay que devolverle un cambio igual a N
- Función objetivo.

n = tipos de monedas

N = cambio que queremos devolver

Función objetivo: minimizar el número de monedas a devolver

Ejemplo:

$d_1=1$

$d_2=2$

$d_3=5$ Cambio=N=7

	0	1	2	3	4	5	6	7
1	0	1	2	3	4	5	6	7
2	0	1	1	2	2	3	3	4
3	0	1	1	2	2	1	2	2

La primera fila y columna son el caso base

El caso general: min {T[i][j]= 1+T[i][j-di] , T[i-1][j]}

T[3][1]=1

T[3][2]=1

T[3][5]=min{1+T[3][0] , T[2][5]} = 1

T[3][6]=min{1+T[3][1] , T[2][6]} = 2

T[3][7]=min{1+T[3][2] , t[2][7]} = 2

```
int Monedas(int i, int j, int *d){
  //Caso base
  if(j<=0) return 0;
  if(i=1) return j;

  int echandoMoneda = 1+Monedas(i, j-d[i], d);
  int SinEcharMoneda= Monedas(i-1,j, d);

  return (echandoMoneda < SinEcharMoneda) ? echandoMoneda:SinEcharMoneda;
}
```

Problema de la mochila 0/1

Objetos ordenados de menor a mayor bi/wi

T[i][j]=Maximo beneficio de haber considerado llevar hasta el objeto i

para una capacidad de mochila j

T[i][j]=Max{ bi+T[i-1][j-wi], T[i-1][j]}

//Casos bases

T[i][0]=0 Porque no te puedes llevar ningún objeto

T[1][j]=b1 Para cada j>=w1, no definido para otros casos.

Comprobamos que se cumple el principio de optimalidad de Bellman. yes!

T[2][2]=Max{6+T[i-1][2-wi],T[i-1][2]}

T[2][3]=Max{6+T[i-1][3-2], T[i-1][3]}

Solución óptima.

El problema de caminos minimos PD != greedy

En greedy, partimos de un nodo concreto inicial

En PD se calcula el camino mínimo de cualquier nodo(todos con todos)

Algoritmo de Floyd !!!MUY IMPORTANTE, APRENDER EXAMEN!!

D1[i][j]= camino óptimo de haber pasado o no por el nodo 1

D2[i][j]=haber considerado el camino de i hasta j haber cosiderado pasar

o no por el nodo 1 y 2