Laboratorio Tarea B Implementación

Paolo Montero
Obdulio Varela
Indio Arispe

Contents

1 Formalización

- Forma de Solución

$< x_0, x_1, ..., x_{n-1} >$ tupla de largo **n**, donde **n** es la cantidad de charlas.

- Restricciones explícitas

$-1 \leq x_i \leq m-1$ con $0 \leq i \leq n-1$,
donde **n** es la cantidad de charlas, **m** es la cantidad de salas

- Restricciones implícitas

Dos charlas pertenecen la misma sala si son compatibles.

$$compatibles(i,j) = \begin{cases} true, & \text{if } charlas[i].fin \leq charlas[j].inicio \\ \vee\ charlas[j].fin \leq charlas[i].inicio & \\ false, & \text{otro caso} \end{cases}$$

Una charla es asignada a una sala solo si la sala tiene disponibilidad horaria para abarcar la charla.

$$disponible(x_i,i) = \begin{cases} true, & \text{if } charlas[i].fin \leq salas[x_i].fin \\ \wedge\ charlas[i].inicio \geq salas[x_i].inicio & \\ false, & \text{otro caso} \end{cases}$$

- Función objetivo

Devuelve aquella solución que asegura la máxima asistencia.

$$T = \{t =< x_0, x_1, ..., x_{n-1} >\ \ t\ es\ solucion\}$$

$$f(t) = max_{t \in T}\{cantAsistentes(t)\}$$

$$cantAsistentes(t) = \sum_{i=0}^{i=n-1} \delta(x_i) charlas[i].asistentes$$

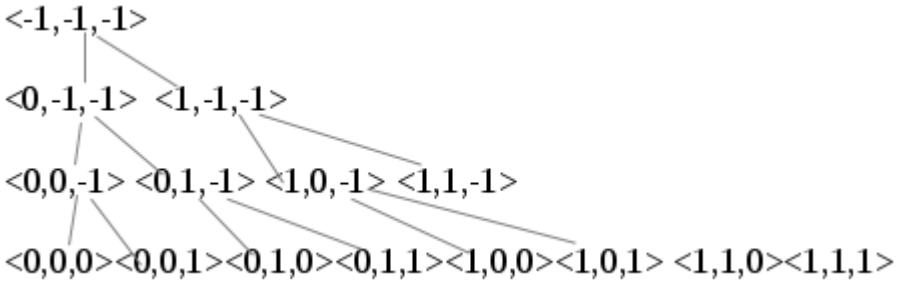
$$\delta(x_i) = \begin{cases} 0, & \text{if } x_i = -1 \\ 1, & \text{otro caso} \end{cases}$$

2 Recorrida Backtracking

2.1 Comentarios sobre recorridas

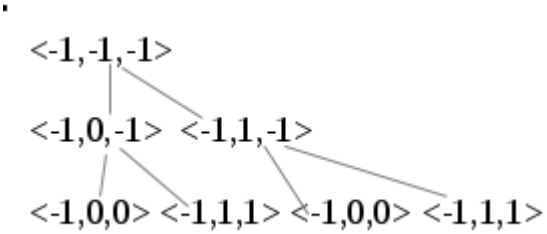
- **Caso tres charlas dos salas**

Una recorrida que nos sirve es esta



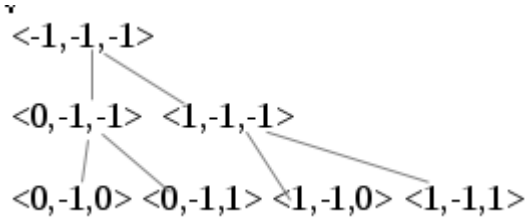
Pero dadas las restricciones de que una sala cualquiera puede estar o no debemos hacer recorridas anulando salas.

Por ejemplo un recorrido que no considere la primera sala.



Estas soluciones no son contempladas en la primera recorrida, por lo que para llegar a estas hay que meter una recorrida imponiendo que la primera charla no sea considerada.

También puede ser de interés considerar las combinaciones de la primera y tercera charla y ver que soluciones salen sin tener en cuenta a la segunda.



El recorrido debería considerar estas soluciones también.

2.2 Propuesta de algoritmo

El algoritmo adjunto trata de cubrir el arbol de soluciones teniendo en cuenta las consideraciones de la sección 2.1 de la siguiente manera.

```
//Declaracion de funcion
void backtracking(charla* charlas, int n, sala* salas, int m,
int *solucion, int i, int *tupla, int parche);
//Extracto del codigo sucesivas llamadas a BackTracking
        printf("BackTrackin!! \n");
        //Se trata de cubrir los agujeros con un parche
        //que lo que hace es para cada pasada va silenciando
        //a las salas.
        for(int i=0; i < n; i++)
        {
                for(int j=0; j < n; j++)
                {
                        tupla[j]=-1;

                }

                for(int k=i-1; k < n; k++){

                        backtracking(charlas, n, salas, m,
                         solucion, i, tupla, k);
                }
        }
```

```
void backtracking(charla* charlas, int n, sala* salas, int m,
int *solucion, int i, int *tupla, int parche)
{
//Recorrida correcta de cada componente
//probando con diferentes salas para cada charla
//obviando a las charlas silenciadas por el parche

                for(int xi=0; xi < m; xi++)
                        {
                                        if(i==parche)
                                                i=i+1;
                                        if(i<n){
                                                tupla[i]=xi;
                                                backtracking(charlas,
                                                n, salas, m,
                                                solucion, i + 1,
                                                tupla, parche);
                                                if(tupla[i]=m)
                                                        tupla[i]=-1;
                                        }
                }
}
```