Laboratorio Tarea B
Reformas al congreso

Andrés Monetta
Alejandro Clara
Sebastián Daloia

Contenido

1 Declaración de estructuras **3**

2 Solución empleando las estructuras **3**

3 BackTracking **5**

4 Resto de funciones **7**

1 Declaración de estructuras

```
//Variables globales

int *solucion;
int *tupla;
int *indices;
int *ultimacharla;

charla *charlas_globales;
sala *salas_globales;
```

Descripción de cada una de ellas:

tupla arreglo que contendra la tupla generada para cada estado del arbol de soluciones en [0...n-1]
en [n] contendra los asistenes de las charlas hasta ese momento asignadas
en [n+1] contendra el resto de los asistentes del total de charlas

solucion arreglo que contendra la tupla solucion en [0...n-1]
[n] contendra los asistentes de la solucion

indices arreglo que mantiene los indices del arreglo charla original una vez que este es ordenado por fecha de fin

ultimacharla arreglo que mantiene la ultima charla asignada a una sala. [0...m-1] representan los indices de cada sala y ultimacharla[i] contendrá la ultima charla agregada a la sala i con $0 \leq i < m$.

2 Solución empleando las estructuras

Las *charlas* que vienen como parámetros de la función `max_asistentes_m` son ordenadas por fecha de fin y luego hacemos apuntar a *`charlas_globales`* a las charlas ordenadas para manipularlas desde las diferentes funciones.

Para manipular las salas hacemos apuntar *`salas_globales`* al arreglo de *salas* proveniente de la función `max_asistentes_m` el puntero.

Luego haciendo uso del problema de la tarea semanal **2.3 Programación Greedy** en el cual se demostraba por inducción que si las charlas estaban ordenadas de forma creciente por el atributo *fecha de fin* se llegaba a una solución óptima, con el algoritmo propuesto en la letra.
Implementamos una versión de este algoritmo, en lugar de para una sala,

para varias salas.
Otra de las cosas que vimos en la tarea antes mencionada es que para *fecha de fin* bastaba para cada nueva charla, a evaluar del arreglo ordenado, comparar la disponibilidad con la ultima charla agregada a la solución.
Esta última idea se ve reflejada en el arreglo **ultimacharla**[1]
A medida que vamos recorriendo el arbol de soluciones vamos actualizando **ultimacharla** con la ultima charla agregada a la sala respectiva.

¿Que ganamos con esto?

Cuando asignamos una sala a una charla y queremos saber si será solución solamente comparamos la compatibilidad de la charla con *la ultima charla agregada*.

[1] véase definición de ultimacharla sección 1

3 BackTracking

```
void backtracking(int n, int m, int i, int xi)
{
  if(predicado(tupla[n], tupla[n+1], solucion[n])){
    if(verificaRest(n, i-1)){
          int *aux= new int;
          *aux=ultimacharla[xi];
          ultimacharla[xi] = i-1;

          if( esMejor(n)){
            copiar(n);
          }

          for(int j=i; j<n; j++) {
            for(int xi=0; xi < m; xi++)
            {
              tupla[j]=xi;

              tupla[n]+=charlas_globales[j].asistentes;
              tupla[n+1]-=charlas_globales[j].asistentes;

              backtracking(n, m, j + 1, xi);

              tupla[n]-=charlas_globales[j].asistentes;
              tupla[n+1]+=charlas_globales[j].asistentes;

              tupla[j]=-1;
            }
          }

          ultimacharla[xi] = *aux;
          delete aux;
        }
   }
}
```

Observaciones:
predicado: Actualmente compara sí la cantidad de asistentes de las charlas *tupla generada* más la cantidad de asistentes de las charlas que aún no han

sido consideradas es mayor que la cantidad de asistentes de la actual *tupla solución*.
verificarRest: Se verifica si la *charla i* es compatible con la ultima charla asignada a la sala. En caso de que no haya charla asignada aún se verifica si los horarios de la sala están disponibles para la *charla i*.
esMejor: Compara si la tupla es mejor solución que la *tupla solución actual*
copiar: Copia como nueva solución óptima la tupla actual.

4 Resto de funciones

```
bool predicado(int tn, int tn1, int sn)
{
        if(tn + tn1 <= sn)
                return false;
        return true;
}
```

El predicado compara la cantidad de asistentes de la solucion actual con la cantidad de asistentes de las charlas de la tupla que tienen sala asignada mas los asistentes de aquellas charlas que aun no han sido procesadas ([i...n-1]) y que o bien son compatibles con las charlas que tiene sala asignada o bien tienen una sala disponible (que no tiene charla asignada aún).

Si el resultado de la segunda expresión es mayor a la cantidad de asistentes de la solucion actual, entonces es posible que haya una mejor solucion, se sigue el recorrido. Si el resultado de la segunda expresión es menor a la cantidad de asistentes de la solución actual, entonces no es posible que haya una mejor solucion se realiza poda.

```
//Verifica si las restricciones se cumplen
bool verificaRest (int n, int i)
{
  bool boolean = true;

  //Verifico si hay disponibilidad y compatibilidad para la charla i
  if (i!=-1 && tupla[i]!=-1 &&
  ((!disponible(charlas_globales[i].inicio, charlas_globales[i].fin,
   salas_globales[tupla[i]].inicio,salas_globales[tupla[i]].fin)) \
||( ultimacharla[tupla[i]]!=-1 &&
!compatibles( charlas_globales[ultimacharla[tupla[i]]].inicio,
 charlas_globales[ultimacharla[tupla[i]]].fin, \
charlas_globales[i].inicio,charlas_globales[i].fin) ) ))
        {
                boolean = false;
        }
        return boolean;
}
```

Se debe evaluar la compatibilidad de la charla actual con la charla más próxima que tiene la misma sala asignada

```
bool compatibles (int chiini, int chifin, int chjini, int chjfin)
{
        if ((chiini>=chjfin) || (chifin<=chjini))
                return true;
        return false;
}



 //Disponibilidad charla-sala
bool disponible (int chini, int chfin, int sini, int sfin)
{
        if ((chini>=sini) && (chfin<=sfin))
                return true;
        return false;
}



//Determina si la cantidad de asistentes de la tupla es mayor
//que la cantidad de asistentes de la tupla solucion
bool esMejor(int n){
        if(tupla[n]<=solucion[n])
                return false;
        return true;
}

bool esSolucion(int *tupla, int i, int n){

        if(i==n)
                return true;
        return false;
}


//Se actualiza la tupla solucion
```

```
void copiar(int n){
        for(int j=0; j<=n; j++){
                solucion[j]=tupla[j];
        }
}
```