Semana 7 - Ejercicios de grupo

Métodos algorítmicos en resolución de problemas II
Facultad de Informática - UCM

	Nombres y apellidos de los componentes del grupo que participan	**ID Juez**
1	Miguel Verdaguer Velazquez	MAR285
2	Daniel Hernández Martínez	MAR246
3	Jorge Arevalo Echevarria	MAR205
4		

Instrucciones:

1. Para editar este documento, es necesario hacer una copia de él. Para ello:
    - Alguien del grupo inicia sesión con la cuenta de correo de la UCM (si no la ha iniciado ya) y accede a este documento.
    - Mediante la opción *Archivo* → *Hacer una copia*, hace una copia del documento en su propia unidad de *Google Drive*.
    - Abre esta copia y, mediante el botón *Compartir* (esquina superior derecha), introduce los correos de los demás miembros del grupo para que puedan participar en la edición de la copia.

2. La entrega se realiza a través del Campus Virtual. Para ello:
    - Alguien del grupo convierte este documento a PDF (dándole como nombre el número del grupo, 1.pdf, 2.pdf, etc…). Desde *Google Docs*, puede hacerse mediante la opción *Archivo* → *Descargar* → *Documento PDF*.
    - Esta misma persona sube el fichero PDF a la tarea correspondiente del *Campus Virtual*. Solo es necesario que uno de los componentes del grupo entregue el PDF.

3,5

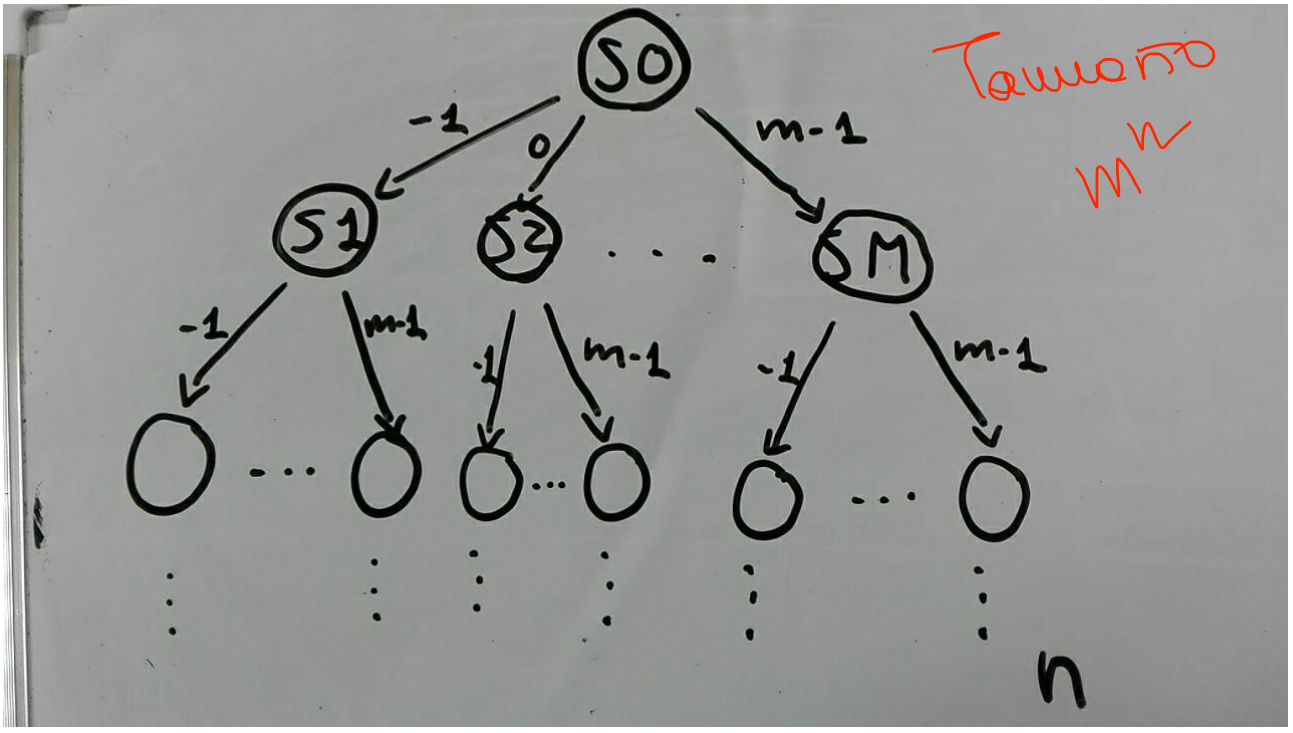
Restaurante Come Sano

El objetivo de hoy es resolver el **problema 20 Restaurante Come Sano**, del juez automático. Ahí podéis encontrar el enunciado completo del problema.

El problema consiste en sentar a *m* comensales en *n* plazas disponibles en un restaurante $(m \leq n)$ de forma que se respeten las distancias de seguridad establecidas: dos allegados pueden sentarse a cualquier distancia pero dos no allegados han de sentarse a más de dos metros de distancia. Conocidas las distancias entre las plazas del restaurante y la relación de "ser allegado" entre los comensales, y respetando la norma descrita anteriormente, se desea maximizar la cantidad de parejas de comensales allegados que están sentados a menos de dos metros de distancia.

Solución: (Explicad cómo representáis la solución al problema, cuáles son las restricciones explícitas e implícitas del problema y representad mediante un dibujo el espacio de soluciones indicando cuál es su tamaño. Explicad qué marcadores vais a utilizar para mejorar el coste del test de factibilidad. Proporcionad y comparad dos posibles cotas optimistas que puedan ser utilizados como prioridad de la cola (en la implementación utilizad la que consideréis mejor). Indicad si se puede utilizar el esquema optimista-pesimista y en caso afirmativo proporcionad dos cotas pesimistas.)

La solución la vamos a representar como un vector de enteros de 0 a n-1 donde cada posición es una silla.



Restricción explícita: El vector solución tiene que estar formado por números entre 0 y m-1 si ponemos a alguien en la silla correspondiente o si no -1.
Restricción implícitas: Entre cada dos comensales, si estos no son allegados, tiene que haber más de dos metros de distancia. En el vector solución, para dos posiciones i e j, si v[i] != -1 y v[j] != -1 y allegados[v[i]][v[j]] = 0, entonces distancia[i][j] > 2.

No puede haber dos personas en el mismo sitio

3,5

El espacio de soluciones es un árbol de profundidad n y factor de ramificación m+1. El tamaño del árbol es de (m+1)^n.

Como marcador utilizaremos un vector de enteros de longitud número de personas. Cada posición será una persona e indicaremos en qué silla se sienta o -1 si no está sentado. El vector se inicializará a 0.

Cotas optimistas:

- Sumar al número de allegados a menos de 2 metros de la solución parcial, el número de allegados para cada persona que no hayamos asignado todavía. De esta forma asumimos que cada comensal sin asignar se puede sentar a menos de 2 metros de todos sus allegados. El coste en tiempo es lineal respecto al número de comensales no sentados todavía, ya que al principio de la función haríamos un precálculo que asignase a cada persona el número de allegados.
- Sumar al número de allegados a menos de 2 metros de la solución parcial el siguiente valor: Para cada persona que quede por asignar obtendremos sus allegados. Si este allegado está ya sentado, sumaremos 1 si hay una silla sin asignar que esté a menos de 2 metros de la suya. Si el allegado no ha sido sentado todavía, sumaremos 1 si existen 2 sillas sin asignar a menos de dos metros. El coste en tiempo será de orden cuadrático, concretamente en el orden del número de personas por asignar por el número de personas - 1. Aunque el coste es mayor, hemos decidido que es asumible, ya que la cota será más cercana a la mejor solución.

En este problema no podemos asegurarnos de que existe una solución para todas las soluciones parciales, ya que hay casos en los que no vamos a poder llegar a un final que cumpla la restricción implícita, por ejemplo si nos quedan dos personas por asignar que no sean allegados y solo quedan 2 sillas a menos de dos metros. Como no existe siempre una solución no podemos utilizar el esquema optimista-pesimista, ya que no podemos obtener una cota pesimista.

Solución: (Escribid aquí las explicaciones necesarias para contar de manera comprensible la implementación del algoritmo de ramificación y poda. Incluid la definición del tipo de los nodos, y el código. Extended el espacio al que haga falta.)

```
struct Nodo{
        vector<int> solucion;
        int m; // El numero de sillas procesadas
        int parejas_asignadas; // Numero de parejas de allegados asignadas a menos de 2 metros
        int prioridad;
        vector<int> personas_asignadas; // Marcador que nos dice donde esta asignada cada persona o -1
        bool operator<(Nodo const& otro) const {
                return otro.prioridad > prioridad;
        }
};
```

```
void comensales(matriz<int> distancias, matriz<bool> allegados , double M, double N, vector<int> & sol_mejor,
double& valor_mejor) {
  Nodo Y;
  Y.sol= vector<int>(N);
  Y.k= -1;
  Y.parejas_asignadas = 0;
```

```
Y.personas_asignadas = vector<int>(N);
Y.prioridad= cota_optimista(distancias, allegados, M, Y);
priority_queue<Nodo> cola;
cola.push(Y);
valor_mejor= -1;
// búsqueda mientras pueda haber algo mejor
while(!cola.empty() && cola.top().prioridad> valor_mejor) {
        Y = cola.top();
        cola.pop();
        Nodo X(Y);
        ++X.k;
        // probamos a no sentar a nadie
        // hacemos un bucle for probando a sentar a todos los comensales
        // El resto de la función seguiría el esquema de ramificación y poda para un problema de maximización
        // No nos ha dado tiempo a escribir más código
```

Resolved el problema completo del juez, que ahora debe ser ya muy sencillo.

Número de envío: