Problema 2: Col-tris

a)

```
int Recursio( int N, int NIter,int direccioN, int[] peces, int[][][][] combinacio){
    // En la primera llamada NIter = N

    int alçada = 0;
    if(NIter != 0){
            peçaTop = peces[NIter];
            peçaBot = peces[NIter-1];

            // Comprobamos mejor combinacion, forzamos a que la ultima pieza sea siempre Direccion N
            int millor_pes;

            // min() es una funcio auxiliar que retorna el valor minim dels parametres donats.

            if(N==NIter){
                int pes1 = combinacio[peçaBot][PesaTop][0][DireccioN];
                int pes2 = combinacio[peçaBot][PesaTop][1][DireccioN];

                millor_pes = Min(pes1, pes2);
            }
            else{
                int pes1 = combinacio[peçaBot][PesaTop][0][0];
                int pes2 = combinacio[peçaBot][PesaTop][0][1];
                int pes3 = combinacio[peçaBot][PesaTop][1][0];
                int pes4 = combinacio[peçaBot][PesaTop][1][1];

                millor_pes = Min(pes1, pes2, pes3, pes4);
            }


            alçada = Recursio(N, NIter-1, direccioN, peces, combinacio);

            //Hacemos la suma a la altura actual
            alçada = millor_pes + alçada;
    }

    return alçada;

}
```

b)

En este caso tendríamos que generar un bucle y guardar todas las soluciones de cada movimiento en un array tal que así.

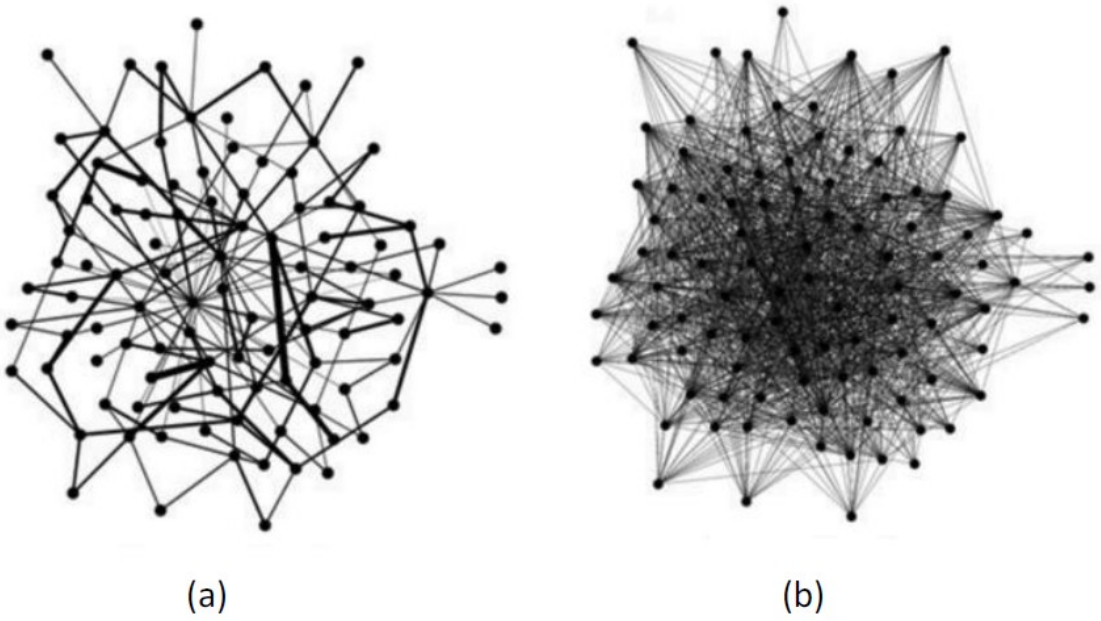
Por cada pieza que haya hacemos la combinacion con la siguiente y se la sumamos con lo que generiamos un array con todas las soluciones.

Solucions[0] [altura de terra + peça1] [combinacio peça2 i peça1 + solucions[1]]…. [N-1]

c)

```
int Coltris( int[] peces, int[][][][] combinacio){

    int solucions[peces.size()];
    solucions[0] = 0;

    if(peces.size()>0){

        for(int i = 1;i<peces.size();i++){

            // Combinaciones posibles
            int pes1 = combinacio[i-1][i][0][0];
            int pes2 = combinacio[i-1][i][0][1];
            int pes3 = combinacio[i-1][i][1][0];
            int pes4 = combinacio[i-1][i][1][1];

            millor_pes = Min(pes1, pes2, pes3, pes4);

            //Con esto obtenemos el mejor movimiento + la suma de los anteriores mejores
            // movimientos.
            solucion[i] = millor_pes + solucions[i-1];

            // El acceso a la solucion seria solucions[N]
            // Siendo N el numero de piezas que hay en peces[]
        }

    }

    return solucions[peces.size()-1];
}
```

Problema 3: Teoria I [1 punt] Donats els grafs següents, amb el mateix nombre de vèrtex, quina implementació de l'algorisme de Prim és la més convenient en cada cas? Justifica la resposta.



(a) (b)

a) En este caso utilizaríamos la implementación con el binary heap ya que no es tan denso como en el caso b, como se pude observar las aristas estan enlazadas con pocos vértices.

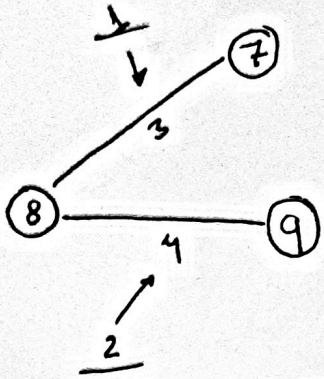
b) En este caso es mejor utilizar la implementación clásica, que es la que utiliza matriz adjunta y un array, porque es un grafo muy denso, es decir que las aristas están enlazadas todas con todas como se puede observar.

Problema 4. Teoría II.

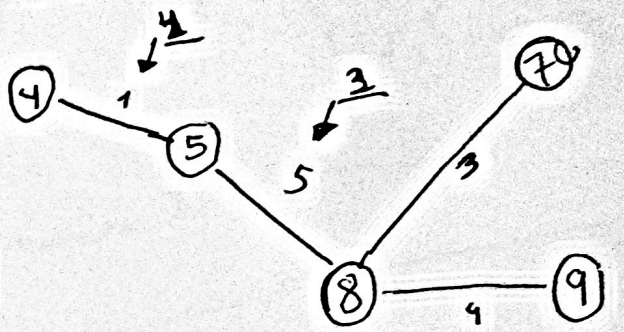
NIF = 55156888J

Inici = 8

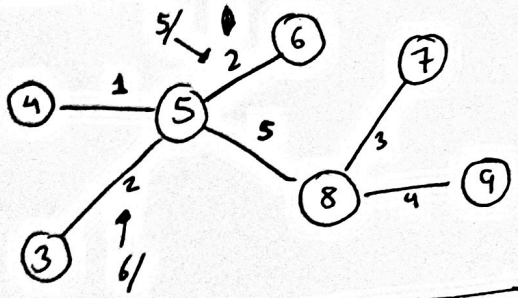
Iteración 1, 2



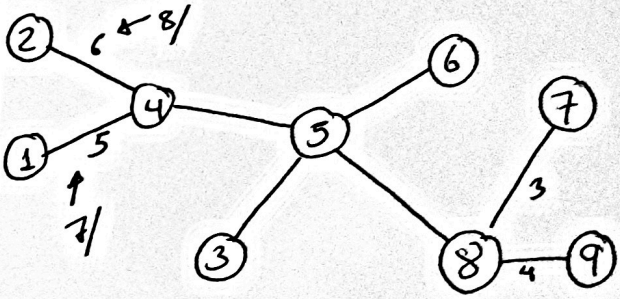
Iteración: 3, 4



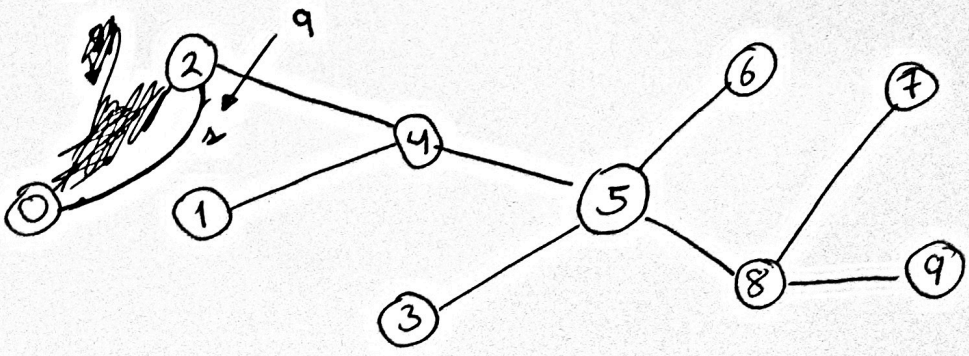
Iteración; ~~3, 4~~ 5, 6



It: 7, 8



It 9



Problema 5.

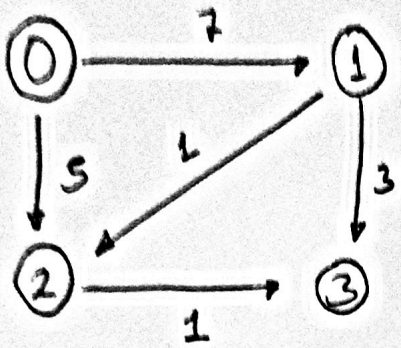


tabla de iteracions.

de 0 a todos. It 1.

	0	1	2	3
0	0	7	5	∞
1	∞	0	∞	∞
2	∞	∞	0	∞
3	∞	∞	∞	0

It 2

	0	1	2	3
0	0	7	5	∞
1	∞	0	1	3
2	∞	∞	∞	∞
3	∞	∞	∞	∞

It 3

	0	1	2	3
0	0	7	5	∞
1	∞	0	1	3
2	∞	∞	0	1
3	∞	∞	∞	∞

It final.

	0	1	2	3
0	0	7	5	∞
1	∞	0	1	3
2	∞	∞	0	1
3	∞	∞	∞	0

Siendo p[2][3] = 1, si el camino de 2 → 3 cambia significa que se ha encontrado un camino con menor peso pa llegar a esta.

Problema 6: Prova de validació pràctica

1. Que és el que fa més complicat el joc de les Amazones que el Connecta-4?

Los más complido es que en el juego de las amazonas hay mas casillas en el tablero y mas opciones de movimiento haciendo que el min i max tenga que recorrer muchos mas nodos que el conecta 4, el nivel de profundidad a la que puede bajar es muy limitado.

2. Explica quina és la heurística que heu fet servir.

Nuestra heurisca lo hicimos mediante la medida de territorio y movilidad, dado un tablero calcula:
movilidad: Calcula todos los posibles movimientos de cada una de nuestras amazonas y las suma, haciendo lo mismo para los enemigos.

Ma = Todos los posibles moviemientos de nuestas amazonas.
Me = Todos los posibles moviemientos de las amazonas enemigas.

Territorio: calcula por cada casilla a que jugador pertenece dicha casilla, una casilla le perteneces al jugado que puede llegar en menos movimientos a ella, en caso de que los dos lleguen en el mismo numero de movimientos, la casilla no pertenecerá a nadie.

Ta= Numero de casillas que pertenecen al aliado.
Te= Número de casillas que pertenece al enemigo.

f(n) = Ta*4+Ma - Te*4 – Te

3. Heu fet algun canvi respecte el minimax estàndard per reduir-ne la complexitat i/o augmentar-ne la intel·ligència?

Implementamos la poda alfa y beta para cortar los subarboles que no nos interesa recorrer aumento así el numero de nodos que podemos recorrer.

Aparte de esto no hicimos ningún cambio que aumente la inteligencia del mín i max.

4. En que pot ajudar el Zobrist hashing a la implementació de la IA?

El zobrist nos ayuda a aumentar el numero de nodos que podemos visitar. Guarda el posible mejor movimiento por cada iteración, por lo que en la siguiente iteración empezara por el mejor posible movimiento ayudando a la poda alfa y beta.

Con esto podemos visitar mas nodos y bajar más la profundidad porque ayuda a mejorar la poda.

5. Quina diferència en la “intel·ligència” de la vostra IA hi ha entre l’inici del joc (tauler complet) i la fase final amb tauler residual petit ?

La inteligencia se basa en dos partes:

Casillas del tablero libres > 20(mas o menos):

En este caso no podemos dejar que el algoritmo min y max elija la posicion donde poner la flecha y donde se tiene que mover, por eso lo que nosotros hicimos es que nosotros decidimos donde poner la flecha y dejar el min i max diga donde moverse. Al hacer eso mejor mucho el nivel de profundidad ya que el min i max no se encarga de ver el mejor tiro de flecha.

Casillas del tablero libres < 21(mas o menos):

Aquí si dejamos que el min i max se encargue de elegir el mejor tiro de flecha y el mejor movimiento, esto hace que el min i max tenga que recorrer muchos mas nodos y lo ralentiza, pero aumenta mucho su inteligencia.