Algoritmos
y
Estructuras de Datos

Cursada 2011

Grafos

Un equipo posee 11 posiciones:

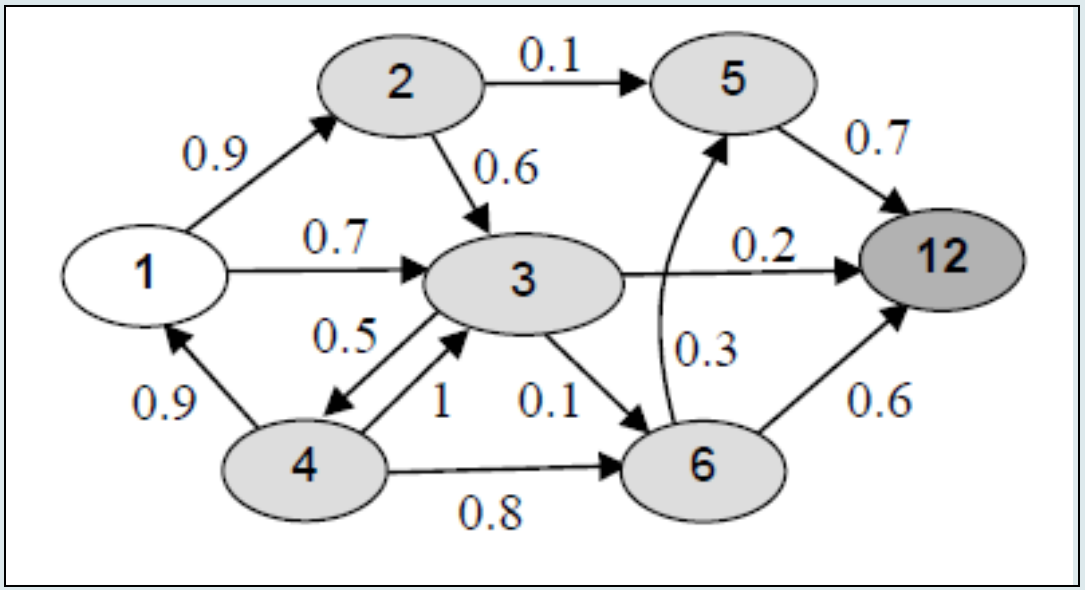
- El arquero es el número 1.
- Una por cada jugador.

Se define una posición más para el arquero contrario (que es el número 12).

Para cada par de jugadores **(a, b)**, tenemos la probabilidad, **P[a, b] ε (0, 1)**, de que un pase desde **a** hasta **b** salga bien.

P[a, 12] indica la probabilidad de que **a** marque un gol al patear.

Grafos



- Esquema del equipo con probabilidades **P[a, b]** de que los pases salgan bien.

- Si no aparece una flecha es que la probabilidad es 0.

La probabilidad de que la secuencia de pases **1 -> 3 -> 12** salga bien es:

P[1, 3] * P[3, 12] = 0.7 * 0.2.

Grafos

- Escribir un algoritmo eficiente que encuentre la estrategia de juego óptima, es decir, la secuencia de pases entre 1(nuestro arquero) y 12 (arco contrario) que maximice la probabilidad de salir bien.

Grafos


```
package tp6.equipo;

public interface Posicionable {
   public int getPosicion();
   public void setPosicion(int posicion);
}

package tp6.equipo;

public class Jugador implements Posicionable{
   private int posicion;
   private int camiseta;

   public Jugador(int i) {
         camiseta = i;
   }

//setters y getters
}
```

Grafos


```
package tp6.equipo;

//imports

public class GrafoImplMatrizAdy<T extends Posicionable> implements Grafo <T> {

   private int maxVertices;
   ListaGenerica<Vertice<T>> vertices;
   double[][] matrizAdy;

  public GrafoImplMatrizAdy(int maxVert){
      maxVertices = maxVert;
      vertices = new ListaEnlazadaGenerica<Vertice<T>>();
      matrizAdy = new double[maxVertices][maxVertices];
      for(int i = 0; i < maxVertices; i++){
        for(int j = 0; j < maxVertices; j++){
          matrizAdy[i][j] = 0;
        }
      }
  }
```

Grafos


```
public void inicializarPesosRandom() {
   for(int i = 0; i< maxVertices; i++){
      for(int j = 0; j < maxVertices; j++){
        matrizAdy [i][j] = Math.random();
      }
   }
}
public void agregarVertice(Vertice<T> v) {
 //Si el vertice ya existe no lo agrega
   if( !vertices.incluye(v)){
       v.setPosicion(vertices.tamanio());
       vertices.agregar(v);
   }
}

private int posicion(Vertice<T> v) {
  return v.getPosicion();
}
```

Grafos


```
 public double getPeso(int pos1, int pos2) {
   return matrizAdy[pos1][pos2];
 }

 public Vertice<T> getVerticePorPos(int pos) {
   return vertices.elemento(pos);
 }
}
 public class VerticeImplMatrizAdy<T> implements Vertice<T> {
    private T dato;
    private int posicion;

    public VerticeImplMatrizAdy(T d) {
     dato = d;
    }

    public T dato() {
       return this.dato;
    }
    //setters y getters
 }
```

Grafos


```
package tp6.equipo;

public class TablaDeProbabilidad {
  public static final int FILAS = 12;
  public static final int COLUMNAS = 12;
  private Grafo<Jugador> grafo =  new GrafoImplMatrizAdy <Jugador>(FILAS);

  public TablaDeProbabilidad(Jugador[] jugadores) {
    grafo.inicializarPesosRandom();
    for (Jugador jugador: jugadores) {
        Vertice<Jugador> v = new VerticeImplMatrizAdy <Jugador>(jugador);
        grafo.agregarVertice(v);
        jugador.setPosicion(grafo.posicion(v));
    }
  }

  public double p(int fila, int col) {
    return grafo.getPeso(fila,col);
  }

  public Jugador getJugadorPorPosicion(Integer posJugador) {
    return grafo.getVerticePorPos(posJugador).dato();
  }
}
```

Grafos


```
package tp6.equipo;

public class EquipoDeFutbol {

   private TablaDeProbabilidad tabla;
   private  boolean[] visitado = new boolean[TablaDeProbabilidad.FILAS];
   private ArrayList<Integer> secuenciaDePases = new ArrayList<Integer>();
   private double probabilidadMaxima = Double.MIN_VALUE;
   private ArrayList<Integer> mejorSecuenciaDePases = new ArrayList<Integer>();

   public void calcularMejorEstrategia() {
        Jugador[] jugadores = new Jugador[12];
        for (int i = 0; i < jugadores.length; i++) {
          jugadores[i] = new Jugador(i+1);
        }
        tabla = new TablaDeProbabilidad(jugadores);
        dfs(jugadores[0], jugadores[11]);
        System.out.println("probabilidad Máxima:"+probabilidadMaxima);
        System.out.println("Mejor camino:");
        for (Integer posJugador : mejorSecuenciaDePases) {

    System.out.println("Jugador:"+tabla.getJugadorPorPosicion(posJugador));
        }
      }
```

Grafos


```
private void dfs(Jugador jugador1, Jugador jugador2) {
   secuenciaDePases.add(jugador1.getPosicion());
   visitado[jugador1.getPosicion()]= true;
   dfs(jugador1.getPosicion(), jugador2.getPosicion(), 1.0);
}
```

Grafos


```
private void dfs(Integer origen, Integer destino, Double probabilidad) {
    if (origen.equals(destino)) {
        if (probabilidadMaxima < probabilidad) {
            probabilidadMaxima = probabilidad;
            mejorSecuenciaDePases.borrarTodo();
            for (Integer jugador : secuenciaDePases) {
                mejorSecuenciaDePases.agregar(jugador);
            }
        }
    } else {
        for (int i = 0; i < TablaDeProbabilidad.COLUMNAS; i++) {
            if (tabla.p(origen, i) > 0 && !visitado[i]) {
                secuenciaDePases.agregar(i);
                visitado[i]=true;
                dfs(i, destino, probabilidad * tabla.p(origen, i));
                visitado[i]=false;
                secuenciaDePases.eliminar(i);
            }
        }
    }
}
```