Inteligencia Artificial

Guía 3.2. Cibernética

4° Licenciatura En Sistemas de Información

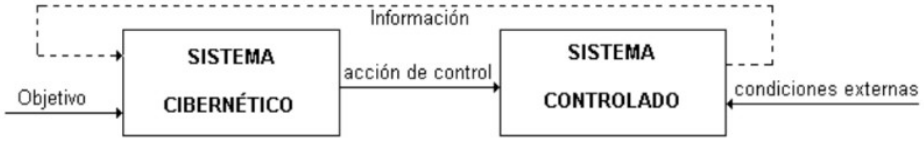
2020

Universidad: UADER FCyT Concepción del Uruguay

Profesor: Lopez De Luise Daniela, Bel Walter

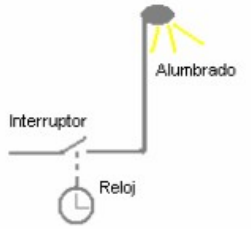
Alumnos: Exequiel Gonzalez, Cepeda Leandro

1. Dada la definición dada en clase, se podría resumir un sistema cibernético como el de la figura:

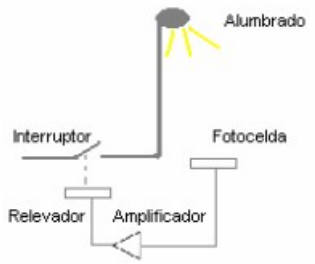


De un ejemplo de circuito cibernético (con lazo cerrado). Explique y justifique cada parte.

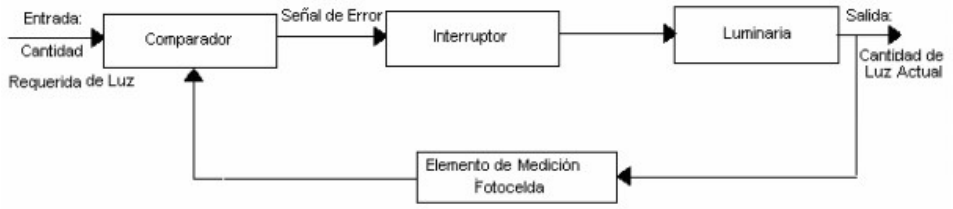
U claro ejemplo es el de alumbrado público, su objetivo es mantener un nivel mínimo de iluminación en las calles, al menor costo. Para lograr este objetivo se pueden proponer dos soluciones: la primera consiste en encender los focos del alumbrado a la hora en que comúnmente empieza a oscurecer, y apagarlos al amanecer. Así, pues se puede decidir encender el alumbrado a las 20 hs y apagarlo a las 6:30 hs. En este sistema, la entrada (cambio de posición del interruptor) es independiente de la salida (cantidad de luz en la calle). Este mecanismo, simple y económico de llevar a cabo, puede acarrear dificultades, ya que la hora en que empieza a aclarar, varían de acuerdo con las estaciones del año, además, en días nublados se puede tener una oscuridad indeseable.



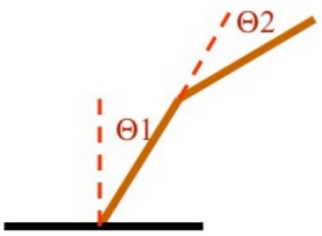
La otra solución, más efectiva, consiste en instalar un dispositivo (fotocelda, fototransistor, etc) para detectar la cantidad de iluminación y de acuerdo con esto, encender o apagar el alumbrado público. En este caso, la entrada (cantidad óptima de luz en las calles) se compararía con la salida (cantidad de luz real en las calles) a los efectos de que la señal de error generada accione o no el interruptor de luz.

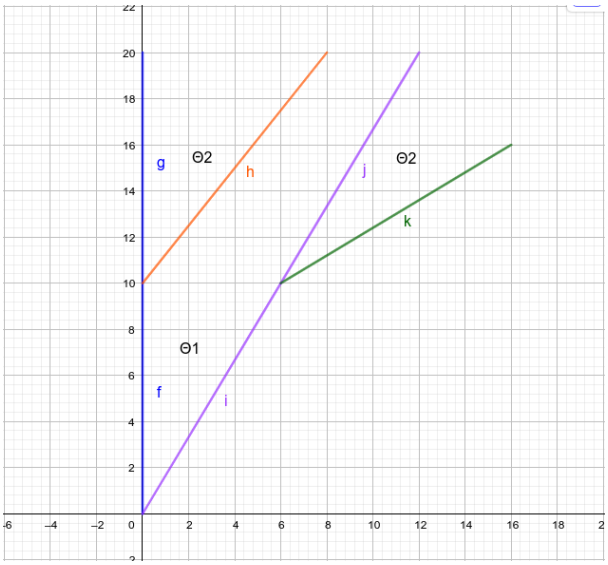


El siguiente diagrama de bloques representa el control realimentado de lazo cerrado:



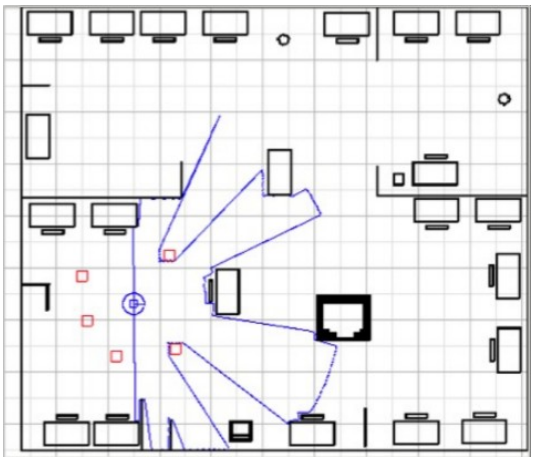
2. El siguiente es un esquema de un manipulador con dos grados de libertad. Indique en ejes cartesianos un ejemplo de configuración (elija una secuencia de 3 posiciones sucesivas del manipulador en este espacio y grafique.)





Movimiento 1; Movimiento 2; ***Movimiento 3***

3. Dado el siguiente world-map, siendo en azul el recorrido que un robot de desplazamiento autónomo realiza, siguiendo un plano de probabilidades, determine en la rejilla una distribución de probabilidades posible, previo a la navegación.

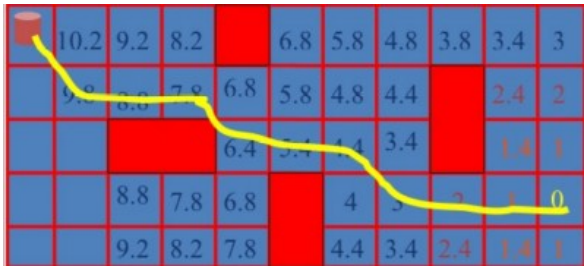


**NOTA: le pide determinar todos los a(x,y) tales que W = [A(21 x 21)]; algunos valores podrían ser, considerando la existencia de paredes:

a(0,0) =[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
a(1,0) =[0 0 0 0 0 0 0 0 0 0 0 0.3 0 0 0 0 0 0 0 0.1 0]
…
a(21,0)=[0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0]

```
w = [
    0   0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
    0   0       0       0       0       0       0       0       0       0       0       0       0.3     0       0       0       0       0       0       0       0.1     0
    0   0       0.4     0.4     0.4     0.5     0.5     0.6     0.6     0.6     0.5     0.5     0.4     0.4     0.4     0.3     0.3     0.2     0.2     0.1     0
    0   0       0.4     0.5     0.5     0.6     0.7     0.8     0.9     0.8     0.7     0.6     0.5     0.5     0.4     0.4     0.3     0.3     0.2     0       0
    0   0       0.5     0.5     0.6     0.7     0.8     0.9     1       0.9     0.8     0.7     0.6     0.5     0.5     0.4     0.4     0.4     0.3     0.2     0
    0   0       0.5     0.6     0.7     0.8     0.9     1       0.9     0.8     0       0.8     0.7     0.6     0.6     0.5     0       0       0.3     0.2     0
    0   0.4     0.5     0.5     0.6     0.7     0       1       0.9     1       0       0.9     0.8     0.7     0       0       0       0       0       0       0
    0   0       0       0       0       0       0       0       1       1       0.9     1       1       0.9     0.8     0       0       0       0       0       0
    0   0       0       0       0       0       1       1       1       0.9     0.9     1       1       0.9     0.8     0.7     0.6     0       0       0       0       0
    0   0.6     0.7     0.8     0.9     1       0       0.9     1       1       0.9     0.9     0.8     0.7     0.6     0.5     0.4     0.3     0       0       0
    0   0       0       0       0.9     1       0.9     1       0       0.9     0.8     0.8     0.7     0.6     0.5     0.4     0.3     0.2     0       0       0
    0   0       0.7     0       0.9     R       0.9     1       0       0.9     0.9     0.9     0       0       0       0.5     0.4     0.3     0       0       0
    0   0.6     0.7     0       0.9     1       0.9     0.9     1       1       1       1       0       0       0       0.6     0.5     0.4     0       0       0
    0   0.6     0.7     0.8     0       1       0       1       0.9     0.9     0.9     0.9     1       0.9     0.8     0.7     0.6     0.5     0       0       0
    0   0.6     0.7     0.8     0.9     1       0.9     1       1       1       0.9     0.9     1       0.9     0.8     0.7     0.6     0.5     0.4     0       0
    0   0       0       0       0       1       0.9     1       0.9     0.9     1       1       0.9     0.8     0.7     0       0       0.4     0       0       0
    0   0       0       0       0       1       1       0.9     1       0       0.9     0       0       0.7     0       0       0       0       0       0       0
    0   0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
    0   0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
    0   0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
    0   0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
]
```

4) Dado el siguiente recorrido (en amarillo) determine un algoritmo de recorrido posible para un robot que debe desplazando minimizando costos desde la posición inicial en el extremo izquierdo superior, hasta la casilla rotulada en 0.



a) Codifique su respuesta en seudocódigo

```
Dijkstra pseudocode:


Estructura de datos auxiliar: Q = Estructura de datos cola de prioridad (se puede implementar con un montículo)

1 Prim (Grafo G, nodo_fuente s)
2      para todo u en V[G] hacer
3             distancia[u] = INFINITO
4             padre[u] = NULL
5             visitado[u] = false
6
7      distancia[s] = 0
8      adicionar (cola, (distancia[s] , s) )
9
10     mientras que cola no sea vacia hacer
11            u = extraer_minimo(cola)
12            visitado[u] = true
13            para todo v en vecinos de u hacer
14            si no visitado[v] y distancia[v] > peso(u, v) hacer
15                   distancia[v] = peso(u, v)
16                   padre[v] = u
17                   adicionar (cola, (distancia[v] , v) )


A* pseudocode:


1      Inicializar lista ABIERTA
2      Inicializar lista CERRADA
3      Crear nodo objetivo; llámalo node_goal
4      Crear nodo de inicio; llámalo nodo_inicio
5      Agregue node_start a la lista ABIERTA
5
6      mientras la lista ABIERTA no está vacía
7             Obtenga el nodo n de la lista ABIERTA con la f (n) más baja
8             Agregar n a la lista CERRADA
9             si n es lo mismo que node_goal, hemos encontrado la solución; Solución de retorno (n)
10            Genere cada nodo sucesor n 'de n
11
12            para cada nodo sucesor n 'de n
13                   Establecer el padre de n 'a n
14                   Establezca h (n ') como la distancia estimada heurísticamente a node_goal
15                   Establezca g (n ') como g (n) más el costo para llegar a n' desde n
16                   Establezca f (n ') en g (n') más h (n ')
17
19                   si n 'está en la lista ABIERTA y la existente es tan buena o mejor, descarte n' y continuar
21
22                   si n 'está en la lista CERRADA y la existente es tan buena o mejor, descarte n 'y continuar
23
24                   Eliminar las apariciones de n 'de ABIERTO y CERRADO
25                   Agregar n 'a la lista ABIERTA
26
27     error de retorno (si llegamos a este punto, hemos buscado todos los nodos accesibles y aún no hemos
28     encontrado la solución, por lo tanto, no existe)
```

b) Codifique su respuesta en PROLOG

```
Dijkstra.pl


:-dynamic
       rpath/2.      % A reversed path

% Example of GUIA 3.1

%      1      2      3      4      5      6      7      8      9      10     11
%
% 1    -1     10.2   9.2    8.2    ∞      6.8    5.8    4.8    3.8    3.4    3
% 2    10.8   9.8    8.8    7.8    6.8    5.8    4.8    4.4    ∞      2.4    2
% 3    10.4   9.4    ∞      ∞      6.4    5.4    4.4    3.4    ∞      1.4    1
% 4    10.8   9.8    8.8    7.8    6.8    ∞      4      3      2      1      0
% 5    11.2   10.2   9.2    8.2    7.8    ∞      4.4    3.4    2.4    1.4    1
```

```
% Columna 1
edge(c1f1, c2f1, 10.2).
edge(c1f1, c1f2, 10.8).
edge(c1f1, c2f2, 9.8).
edge(c1f2, c2f1, 10.2).
edge(c1f2, c2f2, 9.8).
edge(c1f2, c1f3, 10.4).
edge(c1f2, c2f3, 9.4).
edge(c1f3, c1f4, 10.8).
edge(c1f3, c2f2, 9.8).
edge(c1f3, c2f3, 9.4).
edge(c1f3, c2f4, 9.8).
edge(c1f4, c1f5, 11.2).
edge(c1f4, c2f3, 9.4).
edge(c1f4, c2f4, 9.8).
edge(c1f4, c2f5, 10.2).
edge(c1f5, c2f4, 9.8).
edge(c1f5, c2f5, 10.2).

%Columna 2
edge(c2f1, c2f2, 9.8).
edge(c2f1, c3f1, 9.2).
edge(c2f1, c3f2, 8.8).
edge(c2f2, c2f3, 9.4).
edge(c2f2, c3f1, 9.2).
edge(c2f2, c3f2, 8.8).
edge(c2f3, c2f4, 9.8).
edge(c2f3, c3f4, 8.8).
edge(c2f3, c3f2, 8.8).
edge(c2f4, c2f5, 10.2).
edge(c2f4, c3f4, 8.8).
edge(c2f4, c3f5, 9.2).

% Columna 3
edge(c3f1, c3f2, 8.8).
edge(c3f1, c4f1, 8.2).
edge(c3f1, c4f2, 7.8).
edge(c3f2, c4f1, 8.2).
edge(c3f2, c4f2, 7.8).
edge(c3f4, c3f5, 9.2).
edge(c3f4, c4f4, 7.8).
edge(c3f4, c4f5, 8.2).
edge(c3f5, c4f4, 7.8).
edge(c3f5, c4f5, 8.2).

% Columna 4
edge(c4f1,c4f2,7.8).
edge(c4f1,c5f2,6.8).
edge(c4f2,c5f2,6.8).
edge(c4f2,c5f3,6.4).
edge(c4f4,c4f5,8.2).
edge(c4f4,c5f3,6.4).
edge(c4f4,c5f4,6.8).
edge(c4f4,c5f5,7.8).
edge(c4f5,c5f4,6.8).
edge(c4f5,c5f5,7.8).

% Columna 5
edge(c5f2,c5f3,6.4).
edge(c5f2,c6f1,6.8).
edge(c5f2,c6f2,5.8).
edge(c5f2,c6f3,5.4).
edge(c5f3,c5f4,6.8).
edge(c5f3,c6f2,5.8).
edge(c5f3,c6f3,5.4).
edge(c5f4,c5f5,7.8).
edge(c5f4,c6f3,5.4).

% Columna 6
edge(c6f1,c6f2,5.8).
edge(c6f1,c7f1,5.8).
edge(c6f1,c7f2,4.8).
edge(c6f2,c6f3,5.4).
edge(c6f2,c7f1,5.8).
edge(c6f2,c7f2,4.8).
edge(c6f2,c7f3,4.4).
edge(c6f3,c7f2,4.8).
edge(c6f3,c7f3,4.4).
edge(c6f3,c7f4,4).

% Columna 7
edge(c7f1,c7f2,4.8).
edge(c7f1,c8f1,4.8).
edge(c7f1,c8f2,4.4).
edge(c7f2,c7f3,4.4).
edge(c7f2,c8f1,4.8).
edge(c7f2,c8f2,4.4).
edge(c7f2,c8f3,3.4).
edge(c7f3,c7f4,4).
edge(c7f3,c8f2,4.4).
edge(c7f3,c8f3,3.4).
edge(c7f3,c8f4,3).
edge(c7f4,c7f5,4.4).
edge(c7f4,c8f3,3.4).
edge(c7f4,c8f4,3).
edge(c7f4,c8f5,3.4).
edge(c7f5,c8f4,3).
edge(c7f5,c8f5,3.4).
```

```
% Columna 8
edge(c8f1, c9f1, 3.8).
edge(c8f1, c8f2, 4.4).
edge(c8f2, c8f3, 3.4).
edge(c8f2, c9f1, 3.8).
edge(c8f3, c8f2, 4.4).
edge(c8f3, c8f4, 3).
edge(c8f3, c9f4, 2).
edge(c8f4, c8f5, 3.4).
edge(c8f4, c9f4, 2).
edge(c8f4, c9f5, 2.4).

% Columna 9
edge(c9f1, c10f1, 3.4).
edge(c9f1, c10f2, 2.4).
edge(c9f4, c9f5, 2.4).
edge(c9f4, c10f3, 1.4).
edge(c9f4, c10f4, 1).
edge(c9f4, c10f5, 1.4).
edge(c9f5, c10f4, 1).
edge(c9f5, c10f5, 1.4).

% Columna 10
edge(c10f1, c10f2, 2.4).
edge(c10f1, c11f1, 3).
edge(c10f1, c11f2, 2).
edge(c10f2, c10f3, 1.4).
edge(c10f2, c11f1, 3).
edge(c10f2, c11f2, 2).
edge(c10f2, c11f3, 1).
edge(c10f3, c10f4, 1).
edge(c10f3, c11f2, 2).
edge(c10f3, c11f3, 1).
edge(c10f3, c11f4, 0).
edge(c10f4, c10f5, 1.4).
edge(c10f4, c11f3, 1).
edge(c10f4, c11f4, 0).
edge(c10f4, c11f5, 1).
edge(c10f5, c11f4, 0).
edge(c10f5, c11f5, 1).



path(From,To,Dist) :- edge(To,From,Dist).
path(From,To,Dist) :- edge(From,To,Dist).

shorterPath([H|Path], Dist) :-                % path < stored path? replace it
       rpath([H|T], D), !, Dist < D,          % match target node [H|_]
       retract(rpath([H|_],_)),
       %writef('%w is closer than %w\n', [[H|Path], [H|T]]),
       assert(rpath([H|Path], Dist)).
shorterPath(Path, Dist) :-              % Otherwise store a new path
       %writef('New path:%w\n', [Path]),
       assert(rpath(Path,Dist)).

traverse(From, Path, Dist) :-               % traverse all reachable nodes
       path(From, T, D),             % For each neighbor
       not(memberchk(T, Path)),      %  which is unvisited
       shorterPath([T,From|Path], Dist+D), %   Update shortest path and distance
       traverse(T,[From|Path],Dist+D).      %  Then traverse the neighbor

traverse(From) :-
       retractall(rpath(_,_)),           % Remove solutions
       traverse(From,[],0).              % Traverse from origin
traverse(_).

go(From, To) :-
       traverse(From),                   % Find all distances
       rpath([To|RPath], Dist)→         % If the target was reached
         reverse([To|RPath], Path),      % Report the path and distance
         Distance is Dist,
         writef('Shortest path is %w with distance %w = %w\n',
              [Path, Dist, Distance]);
       writef('There is no route from %w to %w\n', [From, To]).
```

```
?- go(c1f1, c11f4).
Shortest path is [c1f1,c2f2,c3f2,c4f2,c5f3,c6f3,c7f4,c8f4,c9f4,c10f4,c11f4] with distance 0+9.8+8.8+7.8+6.4+5.4+4+3+2+1+0 = 48.2
true.
```

c) Recodifique en 3GL (java o C)

Dijkstra.java

```
package dijkstra;

import java.util.PriorityQueue;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
```

```
class Vertex implements Comparable<Vertex>
{
       public final String name;
       public Edge[] adjacencies;
       public double minDistance = Double.POSITIVE_INFINITY;
       public Vertex previous;
       public Vertex(String argName) { name = argName; }
       public String toString() { return name; }
       public int compareTo(Vertex other)
       {
           return Double.compare(minDistance, other.minDistance);
       }

}


class Edge
{
       public final Vertex target;
       public final double weight;
       public Edge(Vertex argTarget, double argWeight)
       { target = argTarget; weight = argWeight; }
}

public class Dijkstra
{
       public static void computePaths(Vertex source)
       {
           source.minDistance = 0.;
           PriorityQueue<Vertex> vertexQueue = new PriorityQueue<Vertex>();
       vertexQueue.add(source);

       while (!vertexQueue.isEmpty()) {
           Vertex u = vertexQueue.poll();

               // Visit each edge exiting u
               for (Edge e : u.adjacencies)
               {
                   Vertex v = e.target;
                   double weight = e.weight;
                   double distanceThroughU = u.minDistance + weight;
           if (distanceThroughU < v.minDistance) {
               vertexQueue.remove(v);

               v.minDistance = distanceThroughU ;
               v.previous = u;
               vertexQueue.add(v);
           }
               }
           }
       }

       public static List<Vertex> getShortestPathTo(Vertex target)
       {
           List<Vertex> path = new ArrayList<Vertex>();
           for (Vertex vertex = target; vertex ≠ null; vertex = vertex.previous)
               path.add(vertex);

           Collections.reverse(path);
           return path;
       }

       public static void main(String[] args)
       {
            // Columna 1
            Vertex C1F1 = new Vertex ("11.2");
            Vertex C1F2 = new Vertex ("10.8");
            Vertex C1F3 = new Vertex ("10.4");
            Vertex C1F4 = new Vertex ("10.8");
            Vertex C1F5 = new Vertex ("11.2");
            // Columna 2
            Vertex C2F1 = new Vertex ("10.2");
            Vertex C2F2 = new Vertex ("9.8");
            Vertex C2F3 = new Vertex ("9.4");
            Vertex C2F4 = new Vertex ("9.8");
            Vertex C2F5 = new Vertex ("10.2");
            // Columna 3
            Vertex C3F1 = new Vertex ("9.2");
            Vertex C3F2 = new Vertex ("8.8");
            Vertex C3F4 = new Vertex ("8.8");
            Vertex C3F5 = new Vertex ("9.2");
            // Columna 4
            Vertex C4F1 = new Vertex ("8.2");
            Vertex C4F2 = new Vertex ("7.8");
            Vertex C4F4 = new Vertex ("7.8");
            Vertex C4F5 = new Vertex ("8.2");
            // Columna 5
            Vertex C5F2 = new Vertex ("6.8");
            Vertex C5F3 = new Vertex ("6.4");
            Vertex C5F4 = new Vertex ("6.8");
            Vertex C5F5 = new Vertex ("7.8");
            // Columna 6
            Vertex C6F1 = new Vertex ("6.8");
            Vertex C6F2 = new Vertex ("5.8");
            Vertex C6F3 = new Vertex ("5.4");
            // Columna 7
            Vertex C7F1 = new Vertex ("5.8");
            Vertex C7F2 = new Vertex ("4.8");
            Vertex C7F3 = new Vertex ("4.4");
            Vertex C7F4 = new Vertex ("4");
            Vertex C7F5 = new Vertex ("4.4");
```

```
// Columna 8
Vertex C8F1 = new Vertex ("4.8");
Vertex C8F2 = new Vertex ("3.4");
Vertex C8F3 = new Vertex ("3.4");
Vertex C8F4 = new Vertex ("3");
Vertex C8F5 = new Vertex ("3.4");
// Columna 9
Vertex C9F1 = new Vertex ("3.8");
Vertex C9F4 = new Vertex ("2");
Vertex C9F5 = new Vertex ("2.4");
// Columna 10
Vertex C10F1 = new Vertex ("3.4");
Vertex C10F2 = new Vertex ("2.4");
Vertex C10F3 = new Vertex ("1.4");
Vertex C10F4 = new Vertex ("1");
Vertex C10F5 = new Vertex ("1.4");
// Columna 11
Vertex C11F1 = new Vertex ("3");
Vertex C11F2 = new Vertex ("2");
Vertex C11F3 = new Vertex ("1");
Vertex C11F4 = new Vertex ("0");
Vertex C11F5 = new Vertex ("1");

// set the edges and weight

////////////////////////////////////////////////////////////// Columna 1

C1F1.adjacencies = new Edge[] {
    new Edge(C1F2, 10.8),
    new Edge(C2F1, 10.2),
    new Edge(C2F2, 9.8)
};

C1F2.adjacencies = new Edge[]{
    new Edge(C1F1, 11.2),
    new Edge(C1F3, 10.4),
    new Edge(C2F1, 10.2),
    new Edge(C2F2, 9.8),
    new Edge(C2F3, 9.4)
};

C1F3.adjacencies = new Edge[]{
    new Edge(C1F2, 10.8),
    new Edge(C1F4, 10.8),
    new Edge(C2F2, 9.8),
    new Edge(C2F3, 9.4),
    new Edge(C2F4, 9.8)
};

C1F4.adjacencies = new Edge[]{
    new Edge(C1F3, 10.4),
    new Edge(C1F5, 11.2),
    new Edge(C2F3, 9.4),
    new Edge(C2F4, 9.8),
    new Edge(C2F5, 10.2)
};

C1F5.adjacencies = new Edge[]{
    new Edge(C1F4, 10.8),
    new Edge(C2F4, 9.8),
    new Edge(C2F5, 10.2)
};

//////////////////////////////////////////////
C2F1.adjacencies = new Edge[]{
    new Edge(C1F1, 11.2),
    new Edge(C1F2, 10.8),
    new Edge(C2F2, 9.8),
    new Edge(C3F1, 9.2),
    new Edge(C3F2, 8.8)
};

C2F2.adjacencies = new Edge[]{
    new Edge(C1F1, 11.2),
    new Edge(C1F2, 10.8),
    new Edge(C1F3, 10.4),
    new Edge(C2F1, 10.2),
    new Edge(C2F3, 9.4),
    new Edge(C3F1, 9.2),
    new Edge(C3F2, 8.8)
};

C2F3.adjacencies = new Edge[]{
    new Edge(C1F2, 10.8),
    new Edge(C1F3, 10.4),
    new Edge(C1F4, 10.8),
    new Edge(C2F2, 9.8),
    new Edge(C2F4, 9.8),
    new Edge(C3F2, 8.8),
    new Edge(C3F4, 8.8)
};

C2F4.adjacencies = new Edge[]{
    new Edge(C1F3, 10.4),
    new Edge(C1F4, 10.8),
    new Edge(C1F5, 11.2),
    new Edge(C2F3, 9.4),
    new Edge(C2F5, 10.2),
    new Edge(C3F4, 8.8),
    new Edge(C3F5, 9.2)
};
```

```
C2F5.adjacencies = new Edge[]{
    new Edge(C1F4, 10.8),
    new Edge(C1F5, 11.2),
    new Edge(C2F4, 9.8),
    new Edge(C3F4, 8.8),
    new Edge(C3F5, 9.2)
};

//////////////////////////////////////////////////
C3F1.adjacencies = new Edge[]{
    new Edge(C2F1, 10.2),
    new Edge(C2F2, 9.8),
    new Edge(C3F2, 8.8),
    new Edge(C4F1, 8.2),
    new Edge(C4F2, 7.8)
};

C3F2.adjacencies = new Edge[]{
    new Edge(C2F1, 10.2),
    new Edge(C2F2, 9.8),
    new Edge(C2F3, 9.4),
    new Edge(C3F1, 9.2),
    new Edge(C4F1, 8.2),
    new Edge(C4F2, 7.8)
};

C3F4.adjacencies = new Edge[]{
    new Edge(C2F3, 9.4),
    new Edge(C2F4, 9.8),
    new Edge(C2F5, 10.2),
    new Edge(C3F5, 9.2),
    new Edge(C4F4, 7.8),
    new Edge(C4F5, 8.2)
};

C3F5.adjacencies = new Edge[]{
    new Edge(C2F4, 9.8),
    new Edge(C2F5, 10.2),
    new Edge(C3F4, 8.8),
    new Edge(C4F4, 7.8),
    new Edge(C4F5, 8.2)
};

//////////////////////////////////////////////////
C4F1.adjacencies = new Edge[]{
    new Edge(C3F1, 9.2),
    new Edge(C3F2, 8.8),
    new Edge(C4F2, 7.8),
    new Edge(C5F2, 6.8)
};

C4F2.adjacencies = new Edge[]{
    new Edge(C3F1, 9.2),
    new Edge(C3F2, 8.8),
    new Edge(C4F1, 8.2),
    new Edge(C5F2, 6.8),
    new Edge(C5F3, 6.4)
};

C4F4.adjacencies = new Edge[]{
    new Edge(C3F4, 8.8),
    new Edge(C3F5, 9.2),
    new Edge(C4F5, 8.2),
    new Edge(C5F3, 6.4),
    new Edge(C5F4, 6.8),
    new Edge(C5F5, 7.8)
};

C4F5.adjacencies = new Edge[]{
    new Edge(C3F4, 8.8),
    new Edge(C3F5, 9.2),
    new Edge(C4F4, 7.8),
    new Edge(C5F4, 6.8),
    new Edge(C5F5, 7.8)
};

//////////////////////////////////////////////////
C5F2.adjacencies = new Edge[]{
    new Edge(C4F1, 8.2),
    new Edge(C4F2, 7.8),
    new Edge(C5F3, 6.4),
    new Edge(C6F1, 6.8),
    new Edge(C6F2, 5.8),
    new Edge(C6F3, 5.4)
};

C5F3.adjacencies = new Edge[]{
    new Edge(C4F2, 7.8),
    new Edge(C4F4, 7.8),
    new Edge(C5F2, 6.8),
    new Edge(C5F4, 6.8),
    new Edge(C6F2, 5.8),
    new Edge(C6F3, 5.4)
};

////////////////////////////////////////////////////////////// Columna 4

C5F4.adjacencies = new Edge[]{
    new Edge(C4F4, 7.8),
    new Edge(C4F5, 8.2),
    new Edge(C5F3, 6.4),
    new Edge(C5F5, 7.8),
    new Edge(C6F3, 5.4),
};
```

```
////////////////////////////////////////////////////////////// Columna 5

C5F5.adjacencies = new Edge[]{
    new Edge(C4F4, 7.8),
    new Edge(C4F5, 8.2),
    new Edge(C5F4, 6.8),
};

////////////////////////////////////////////////////////////// Columna 6

C6F1.adjacencies = new Edge[]{
    new Edge(C5F2, 6.8),
    new Edge(C6F2, 5.8),
    new Edge(C7F1, 5.8),
    new Edge(C7F2, 4.8),
};

C6F2.adjacencies = new Edge[]{
    new Edge(C5F2, 6.8),
    new Edge(C5F3, 6.4),
    new Edge(C6F1, 6.8),
    new Edge(C6F3, 5.4),
    new Edge(C7F1, 5.8),
    new Edge(C7F2, 4.8),
    new Edge(C7F3, 4.4),
};

C6F3.adjacencies = new Edge[]{
    new Edge(C5F2, 6.8),
    new Edge(C5F3, 6.4),
    new Edge(C5F4, 6.8),
    new Edge(C6F2, 5.8),
    new Edge(C7F2, 4.8),
    new Edge(C7F3, 4.4),
    new Edge(C7F4, 4),
};

////////////////////////////////////////////////////////////// Columna 7

C7F1.adjacencies = new Edge[]{
    new Edge(C6F1, 6.8),
    new Edge(C6F2, 5.8),
    new Edge(C7F2, 4.8),
    new Edge(C8F1, 4.8),
    new Edge(C8F2, 4.4),
};

C7F2.adjacencies = new Edge[]{
    new Edge(C6F1, 6.8),
    new Edge(C6F2, 5.8),
    new Edge(C6F3, 5.4),
    new Edge(C7F1, 5.8),
    new Edge(C7F3, 4.4),
    new Edge(C8F1, 4.8),
    new Edge(C8F2, 4.4),
    new Edge(C8F3, 3.4),
};

C7F3.adjacencies = new Edge[]{
    new Edge(C6F2, 5.8),
    new Edge(C6F3, 5.4),
    new Edge(C7F2, 4.8),
    new Edge(C7F4, 4),
    new Edge(C8F2, 4.4),
    new Edge(C8F3, 3.4),
    new Edge(C8F4, 3),
};

C7F4.adjacencies = new Edge[]{
    new Edge(C6F3, 5.4),
    new Edge(C7F3, 4.4),
    new Edge(C7F5, 4.4),
    new Edge(C8F3, 3.4),
    new Edge(C8F4, 3),
    new Edge(C8F5, 3.4),
};

C7F5.adjacencies = new Edge[]{
    new Edge(C7F4, 4),
    new Edge(C8F4, 3),
    new Edge(C8F5, 3.4),
};

////////////////////////////////////////////////////////////// Columna 8

C8F1.adjacencies = new Edge[]{
    new Edge(C7F1, 5.8),
    new Edge(C7F2, 4.8),
    new Edge(C8F2, 4.4),
    new Edge(C9F1, 3.8),
};

C8F2.adjacencies = new Edge[]{
    new Edge(C7F1, 5.4),
    new Edge(C7F2, 4.8),
    new Edge(C7F3, 4.4),
    new Edge(C8F1, 4.8),
    new Edge(C8F3, 3.4),
    new Edge(C9F1, 3.8),
};
```

```
C8F3.adjacencies = new Edge[]{
    new Edge(C7F2, 4.8),
    new Edge(C7F3, 4.4),
    new Edge(C7F4, 4),
    new Edge(C8F2, 4.4),
    new Edge(C8F4, 3),
    new Edge(C9F4, 3.4),
};

C8F4.adjacencies = new Edge[]{
    new Edge(C7F3, 4.4),
    new Edge(C7F4, 4),
    new Edge(C7F5, 4.4),
    new Edge(C8F3, 3.4),
    new Edge(C8F5, 3.4),
    new Edge(C9F4, 2),
    new Edge(C9F5, 2.4),
};

C8F5.adjacencies = new Edge[]{
    new Edge(C7F4, 4),
    new Edge(C7F5, 4.4),
    new Edge(C8F4, 3),
    new Edge(C9F4, 2),
    new Edge(C9F5, 2.4),
};

////////////////////////////////////////////////////////////// Columna 9

C9F1.adjacencies = new Edge [] {
    new Edge (C8F1, 5.8),
    new Edge (C8F2, 4.4),
    new Edge (C10F1, 3.4),
    new Edge (C10F2, 2.4),
};

C9F4.adjacencies = new Edge [] {
    new Edge (C8F4, 2),
    new Edge (C8F5, 2.4),
    new Edge (C10F3, 1.4),
    new Edge (C10F4, 1),
    new Edge (C10F5, 1.4),
};

C9F5.adjacencies = new Edge [] {
    new Edge (C8F5, 3.4),
    new Edge (C8F4, 3),
    new Edge (C9F4, 2),
    new Edge (C10F4, 1),
    new Edge (C10F5, 1.4),
};

////////////////////////////////////////////////////////////// Columna 10

C10F1.adjacencies = new Edge [] {
    new Edge (C9F1, 3.8),
    new Edge (C10F2, 2.4),
    new Edge (C11F1, 3),
    new Edge (C11F2, 2),
};

C10F2.adjacencies = new Edge [] {
    new Edge (C10F1, 3.4),
    new Edge (C10F3, 1.4),
    new Edge (C11F1, 3),
    new Edge (C11F2, 2),
};

C10F3.adjacencies = new Edge [] {
    new Edge (C10F2, 2.4),
    new Edge (C11F2, 2),
    new Edge (C10F4, 1),
    new Edge (C11F4, 0),
    new Edge (C10F4, 1),
};

C10F4.adjacencies = new Edge [] {
    new Edge (C10F3, 1.4),
    new Edge (C11F3, 1),
    new Edge (C11F4, 0),
    new Edge (C11F5, 1),
    new Edge (C10F5, 1.4),
    new Edge (C9F5, 2.4),
    new Edge (C9F4, 2),
};

C10F5.adjacencies = new Edge [] {
    new Edge (C11F5, 1),
    new Edge (C11F4, 0),
    new Edge (C10F4, 1),
    new Edge (C9F4, 2),
    new Edge (C9F5, 2.4),
};

////////////////////////////////////////////////////////////// Columna 11

C11F1.adjacencies = new Edge[]{
    new Edge(C10F1, 3.4),
    new Edge(C10F2, 2.4),
    new Edge(C11F2, 2),
};
```

```
            C11F2.adjacencies = new Edge[]{
                new Edge(C10F1, 3.4),
                new Edge(C10F2, 2.4),
                new Edge(C10F3, 1.4),
                new Edge(C11F1, 3),
                new Edge(C11F3, 1),
            };

            C11F3.adjacencies = new Edge[]{
                new Edge(C10F2, 2.4),
                new Edge(C10F3, 1.4),
                new Edge(C10F4, 1),
                new Edge(C11F2, 2),
                new Edge(C11F4, 0),
            };

            C11F4.adjacencies = new Edge[]{
                new Edge(C10F3, 1.4),
                new Edge(C10F4, 1),
                new Edge(C10F5, 1.4),
                new Edge(C11F3, 1),
                new Edge(C11F5, 1),
            };

            C11F5.adjacencies = new Edge[]{
                new Edge(C10F4, 1),
                new Edge(C10F5, 1.4),
                new Edge(C11F4, 0),
            };

            computePaths(C1F1); // run Dijkstra
            System.out.println("Distance to " + C11F4 + ": " + C11F4.minDistance);
            List<Vertex> path = getShortestPathTo(C11F4);
            System.out.println("Path: " + path);

            // Distance to 0: 48.2
            // Path: [11.2, 9.8, 8.8, 7.8, 6.4, 5.4, 4, 3, 2, 1, 0]
        }
}
```

```
Distance to 0: 48.2
Path: [11.2, 9.8, 8.8, 7.8, 6.4, 5.4, 4, 3, 2, 1, 0]
```

5) Dado el siguiente lazo cerrado de un robot, completar la actividad más probable en los recuadros en blanco.