**Integrantes: GRUPO 1 Fecha:** 2019-06-03

Borja Grace

Cevallos Franciso

Inchiglema Nicolas

Parra Jonathan

**Objetivos**

* Conocer las estrategias de búsqueda no informada que realizan los agentes para llegar a una solución.

**Marco Teórico:**

**BÚSQUEDAS NO INFORMADAS**

Los algoritmos de búsqueda ciega o no informada no dependen de información propia del problema a la hora de resolverlo, sino que proporcionan métodos generales para recorrer los árboles de búsqueda asociados a la representación del problema, por lo que se pueden aplicar en cualquier circunstancia. Se basan en la estructura del espacio de estados y determinan estrategias sistemáticas para su exploración, es decir, que siguen una estrategia fija a la hora de visitar los nodos que representan los estados del problema. Se trata también de algoritmos exhaustivos, de manera que, en el peor de los casos, pueden acabar recorriendo todos los nodos del problema para hallar la solución. [1]

Existen, básicamente, dos estrategias de recorrido de un espacio de búsqueda, en anchura y en profundidad. Los algoritmos que veremos a continuación se basan en una de las dos (o incluso en las dos). El problema principal que tienen es que, al ser exhaustivos y sistemáticos su coste puede ser prohibitivo para la mayoría de los problemas reales, por lo tanto solo serán aplicables en problemas pequeños, pero presentan la ventaja de que no es necesario ningún conocimiento adicional sobre el problema, por lo que siempre son aplicables. [1]

**Búsqueda primero en anchura**

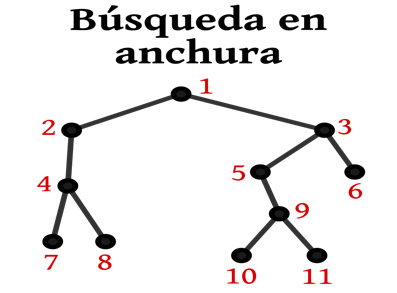


Figura 1.

La búsqueda en anchura consiste en buscar horizontalmente los nodos, es decir expandir de izquierda a derecha, todos los nodos de un nivel, para así poder pasar al siguiente nivel, es una búsqueda óptima cuando el espacio de estados no es infinito, sin embargo esta búsqueda tiene complejidad de espacio ya que se deben guardar en memoria, los nodos extendidos. [2]

La búsqueda en anchura tiene una estructura de cola FIFO, es decir, el primero en extender es el primero en extender a nuevos nodos y así sucesivamente. [2]

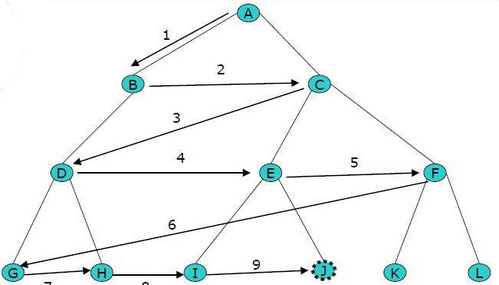


Figura 2.

**Desarrollo:**

Se expande primero el nodo raíz, a continuación, se expanden todos los sucesores del nodo raíz, después sus sucesores, etc.

Usa una estructura FIFO, es decir, los nuevos sucesores van al final.

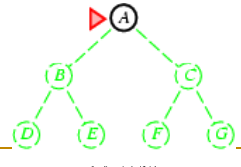


Figura 3.

Imagen que contiene objeto, reloj

Descripción generada automáticamente

Figura 4.

Imagen que contiene reloj, objeto

Descripción generada automáticamente

Figura 5.

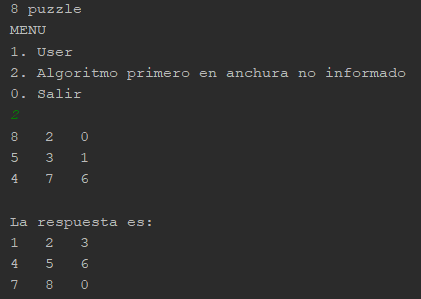
**Ejecución:**

public puzzle() {  
 int n;  
 for(int i = 0; i < puz.length; i++)  
 for(int j = 0; j < puz[i].length; j++) {  
 n = (int) Math.*floor*(Math.*random*() \* (10 ));  
 while (verificar(n)==true){  
 n = (int) Math.*floor*(Math.*random*() \* (10));  
 }  
 puz[i][j] = n;  
 }  
 for(int i = 0; i < puz.length; i++)  
 for (int j = 0; j < puz[i].length; j++)  
 puz[i][j]--;  
}

public void mover(int i){  
 int[] zero = find();  
 int aux;  
 switch (i){  
 case 1:  
 if(zero[0]!=2){  
 puz[zero[0]][zero[1]] = puz[zero[0]+1][zero[1]];  
 puz[zero[0]+1][zero[1]] = 0;  
 }  
 break;  
 case 2:  
 if(zero[0]!=0){  
 puz[zero[0]][zero[1]] = puz[zero[0]-1][zero[1]];  
 puz[zero[0]-1][zero[1]] = 0;  
 }  
 break;  
 case 3:  
 if(zero[1]!=2){  
 puz[zero[0]][zero[1]] = puz[zero[0]][zero[1]+1];  
 puz[zero[0]][zero[1]+1] = 0;  
 }  
 break;  
 case 4:  
 if(zero[1]!=0){  
 puz[zero[0]][zero[1]] = puz[zero[0]][zero[1]-1];  
 puz[zero[0]][zero[1]-1] = 0;  
 }  
 break;  
 }  
}

private static void user() {  
 puzzle p = new puzzle();  
 System.*out*.println(p.toString());  
 System.*out*.println();  
 boolean r=true;  
 int op;  
 while(p.validar() && r){  
 System.*out*.println("Movimiento: 1(arriba), 2 (abajo), 3 (izquierda), 4 (derecha), 0 (rendirse)");  
 op=*in*.nextInt();  
 if(op!=0)  
 p.mover(op);  
 else if(op==0)  
 r=false;  
 System.*out*.println(p.toString());  
 }  
 System.*out*.println("Fin");  
}

private static Nodo BNI(){  
 Tree arbol = new Tree();  
 arbol.generarTree();  
 Nodo nodoActual = null;  
 boolean acabar = false;  
 System.*out*.println(arbol.raiz.p.toString());  
 *pushLN*(arbol.raiz);  
  
 while (!*listaNodos*.isEmpty() && !acabar){  
 nodoActual = *popLN*();  
 *pushLN*(nodoActual);  
 if(nodoActual.p.validar())  
 acabar = true;  
 else  
 *expandir*(nodoActual);  
 }  
  
 return nodoActual;  
}



**Conclusiones:**

Las estrategias de búsqueda no informada se utilizan cuando no se conoce el problema, más que solo su estado inicial y su estado objetivo, estas estrategias sirven para encontrar una solución a un problema.

Las búsquedas no informada no dan al agente información adicional sobre el problema que busca resolver, sin embargo este tipo de búsqueda puede ser útil cuando el agente no necesita estar pendiente de lo que sucede a su alrededor.

**Bibliografía:**

1. F. Caparrini and W. Work, "Búsquedas No Informadas - Fernando Sancho Caparrini", *Cs.us.es*, 2019. [Online]. Available: http://www.cs.us.es/~fsancho/?e=95. [Accessed: 02- Jun- 2019].
2. "TEMA 3: ESTRATEGIAS DE BÚSQUEDA NO INFORMADA | Inteligencia Artificial2 por Raquel Murillo", *Inteligencia Artificial2 por Raquel Murillo*, 2019. [Online]. Available: https://jraquelm2.wixsite.com/ia2raquelmurillo/single-post/2015/05/04/TEMA-3-ESTRATEGIAS-DE-B%C3%9ASQUEDA-NO-INFORMADA. [Accessed: 02- Jun- 2019].