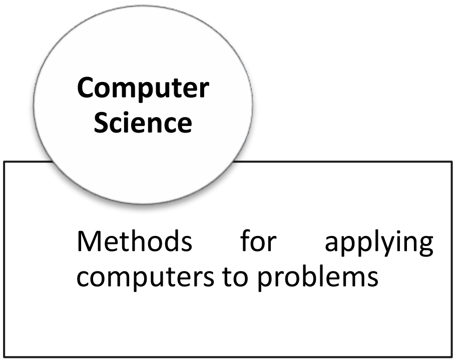
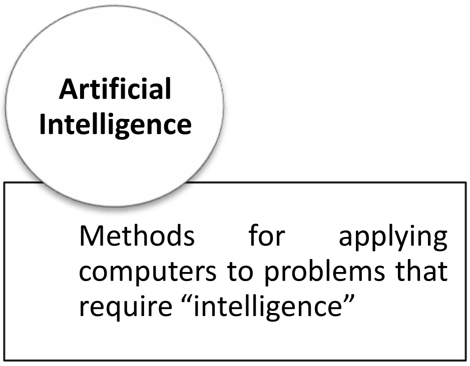
**Introduction to Artificial Intelligence**
**Semester I, 2016-2017**
**Instructor: Dr. Rawaa D. Al-Dabbagh**
**Third Class – Department of Computer Science**
**University of Baghdad**

# INTRODUCTION TO ARTIFICIAL INTELLIGENCE

## 1. Artificial Intelligence Overview

Artificial Intelligence (AI) is the branch or field of computer science that is concerned with the automation of intelligent behavior. Major AI researchers and textbooks define this field as “the study and design of intelligent agents”, in which an intelligent agent is a system that perceives its environment and takes actions that maximize its chances of success. John McCarthy, who coined the term in 1955, defines it as “the science and engineering of making intelligent machines”. AI has a long history but is still constantly and actively growing and changing. It has become an essential part of the technology industry, providing the heavy lifting for many of the most challenging problems in computer science and many other fields.





The two most fundamental concerns of AI researchers are *knowledge representation* and *search*. The first of these, which is also called *Natural Language Processing* (NLP), addresses the problem of capturing in a formal language. The second is a problem-solving technique that systematically explores a space of problem states.

In this course you will learn the basics and applications of AI, including: Different types of search techniques and natural language processing.

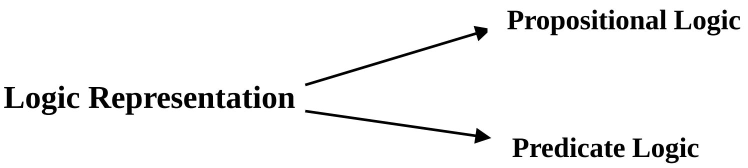
## 2. Overview of AI Application Areas

There are many areas of study in AI. We will try to list some of them:

- Game Playing
    - Playing games using a well-defined set of rules such as checkers, chess and 15-puzzel.
- Knowledge Representation and Automated Reasoning
    - Representing information about the world in a form that a computer system can utilize to solve complex tasks such as diagnosing a medical condition, having a dialog in a natural language, intelligent assistants, real-time problem solving and internet agents.
- Expert Systems
    - A program that address the problem of reasoning with uncertain or incomplete information, such as expert system for expert doctor or engineer.
- Natural Language Processing
    - information retrieval, summarization, understanding, generation, and translation
- Vision
    - Image analysis, Pattern recognition, and scene understanding.
- Robotics
    - Grasping/manipulation, locomotion, motion planning, and mapping.
- Search and Optimization
    - Planning, Airline scheduling, and resource allocation.

# LOGIC REPRESENTATION

In order to determine appropriate actions to take to achieve goals, an intelligent system needs to compactly represent information about the world and draw conclusions based on general world knowledge and specific facts.



## Propositional Logic

Propositional symbols are used to represent facts. Each symbol can mean what we want it to be. Each fact can be either true or false. Propositional symbols: P, Q, etc. representing specific facts about the world. For example,

P1 = “Water is a liquid”.
P2= “ Today is Monday”.
P3= “It is hot”
Q1= “The goround is wet”
Q2=”It is raining”

Propositions are combined with logical connectives to generate sentences with more complex meaning. The connectives are:

| | |
|---|---|
| ∧ | AND |
| ∨ | OR |
| ⌉ | NOT |
| ⟶ | Implies |
| ⇔ | Mutual implication |

For example : if Q2 then Q1 ⇔ Q2 → Q1

**The truth tables for the connectives**

| **p** | **q** | **⌉ p** | **p∧q** | **p∨q** | **p⟶q** |
|---|---|---|---|---|---|
| T | T | F | T | T | T |
| T | F | F | F | T | F |
| F | T | T | F | T | T |
| F | F | T | F | F | T |

## Logic Notes

$\neg p (\neg p) \Leftrightarrow p$

$( p \vee q) \Leftrightarrow (\neg p \rightarrow q)$

$\neg (p \vee q) \Leftrightarrow \neg p \wedge \neg q$

$\neg (p \wedge q) \Leftrightarrow \neg p \vee \neg q$

$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$

$p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$

$p \wedge q \Leftrightarrow q \wedge p$

$p \vee q \Leftrightarrow q \vee p$

$(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$

$(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$

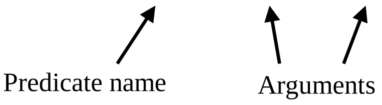
$p \rightarrow q \Leftrightarrow \neg p \vee q$

## Predicate Logic

A predicate names a relationship between zero or more objects. Predicate logic allows us to deal with the component of a sentence. For example,

P= “It rained on Tuesday”
Predicate representation: weather( tuesday, rain)

Q= “ Water is liquid”
Predicate representation: property(water, liquid)



For generality, predicate logic representation allows us to use variables, for example,

| | |
|---|---|
| P1 = “It rained on Tuesday” | weather( tuesday, rain) |
| P2= “It rained on Wednesday” | weather( wedensday, rain) |
| P3= “It rained on Thursady” | weather( thursday, rain) |
| . | . |
| . | . |
| . | . |
| P2= “It rained on Monday” | weather( monday, rain) |

✖

It is more efficient to use variables in the representation format of the predicate.

weather(X, rain)
where $X \in$ { Sunday, Monday, … , Saturady} ✓

**<u>Constant:</u>** A constant refers to a specific object. A constant starts with a lower case letter.

**<u>Variable:</u>** A variable is used to refer to a general classes of ojects. A variable strats with an upper case letter.

**<u>Clauses:</u>** A clause is one or more predicates combined using the connectives above. A clause with one predicate is called a <u>unite clause</u>.

**<u>Horn Clause:</u>** A horn clause has the following form:

$$b1(\ ) \wedge b2\ (\ ) \wedge \ldots\ bn\ (\ ) \longrightarrow a(\ )$$

where b1( ), …, bn( ) and a( ) are all positive predicates. a( ) is called the head of the horn clause. b1( ), …, bn( ) is called the body of the horn clause. There are three cases of the horn clause:

1- a( ) (horn clause has no body)
   In this case the clause is called a <u>Fact.</u>
2- $b1(\ ) \wedge b2\ (\ ) \wedge \ldots\ bn\ (\ )$ (horn clause has no head)
   In this case the clasue is considered as a set of <u>Subgoals</u>.
3- $b1(\ ) \wedge b2\ (\ ) \wedge \ldots\ bn\ (\ ) \longrightarrow a(\ )$ ( the standard form of the horn clause)
   In this case the clasue is called a <u>Rule</u>.

**<u>Qualifications:</u>** Each variable must be associated with one of the two quantifiers ($\forall$ , $\exists$) depending on the meaning required

$\forall$ for all [universal quantifier]
$\exists$ there exist [ existential quantifier]

There are some common identities:

$\neg\ \exists X\ p(X) \iff \forall X\ \neg\ p(X)$
$\neg\ \forall X\ p(X) \iff \exists X\ \neg\ p(X)$
$\exists X\ p(X) \iff \exists Y\ p(Y)$
$\forall X\ p(X) \iff \forall Y\ p(Y)$

**Some examples of knowledge representation:**

**(1)** If it does not rain tomorrow, Zeki will go to the lake.
⌉ weather(tomorrow, rain) ⟶ go ( zeki, lake)

**(2)** All basketball players are tall.
∀X [ player(X) ⋀ play (X, basketball) ⟶ tall (X) ]

**(3)** Some students like AI.
∃X [ student(X) ⋀ like (X, ai) ]

**(4)** Nobody like taxes.
⌉ ∃X like (X, taxes) OR ∀X ⌉ like (X, taxes)

**<u>Homework:</u>**

**Convert the following sentences into their correponding predicate logic:**

1- All vertebrates are animals.
2- Everyone in the purchasing department over 30 years is married.
3- There is a cub on top of every red cylinder.
4- Every city has a dogcatcher who has been bitten by every dog in town.

**Reasoning with logic (Inference rule):**

A reasoning or inference rule is a mechanism for producing new sentences from other sentences. There many types of reasoning mechanisms. The most common mechanism is called <u>Resolution.</u> Resolution is the process of choosing two clauses in normal form such that one contains (p) predicate and the negation (⌉ p) of this predicate in the other clause. The result is a clause called the <u>resolvent</u> which consists of the disjunction of all the predicates of the two clauses except the predicate (p) and its negation (⌉ p). This procedure continues until we reach to contradiction or no contradiction. A contradiction is obtained when the empty clause is generated. If contradiction is reached, this means that a clause with its negative cannot be true and considered with the other clauses that are involved in the same context, and the clause should be true; otherwise, if no contradiction then the negative clause is correct. Resolution mechanism has three main parts.

1- Unification
Unification is the process of making a set of predicates with the same name matches

each other exactly. Assume we have a set of predicates to be unified $\{P1, P2 \ldots Pn\}$, we seek a substitution that matches these predicates $F = \{(t_1, v_1) \ldots (t_k, v_k)\}$; where $v_i$ is replaced by term $t_i$. Such that,

$$P1F = P2F = \cdots = PnF$$

where term is a variable, constant, or a function. For example,
L1 = P(X, Y, b)
L2 = P(Z, W, b)
F={(X, Z), (Y, W)}

Another example

L1 = P(a, f(b), c)
L2 = P(Z, W, b)
F= {(a, Z), (f(b), W), ??} **fail to unify**

2- Skolemization

Skolemization is the process of eliminating existential quantifiers and their corresponding variables. For example,

$\exists X$ father(X, ali) $\xrightarrow{skolemization}$ father (zeki, ali)

$\forall X \ \exists Y$ father (Y, X) $\xrightarrow{skolemization}$ $\forall X$ father (f(X), X)

$\exists Y \ \forall X$ father (Y, X) $\xrightarrow{skolemization}$ $\forall X$ father (a, X)

3- Clause normal form

A predicate logic expression which is in its well formed formula (WFF) is in clause normal form if it consists of a disjunction of predicates.

**Steps to convert a WFF clause to a normal form clause:**

=========================================

$\forall X \{ [ p(X) \wedge q(X) ] \longrightarrow [ r(X, a) \wedge \exists Y (\exists Z\ r(Y, Z) \longrightarrow s(X, Y) ) ] \} \vee \forall X\ t(X)$

**Step1: Eliminate ⟶ by using the identity**

$$\mathbf{p \longrightarrow q \equiv \urcorner p \vee q}$$

$\forall X \{ \urcorner [ p(X) \wedge q(X) ] \vee [ r(X, a) \wedge \exists Y ( \urcorner \exists Z\ r(Y, Z) \vee s(X, Y) ) ] \} \vee \forall X\ t(X)$

**Step 2: Reduce the scope of negation**

$\forall X \{ [ \neg p(X) \vee \neg q(X) ] \vee [ r(X,a) \wedge \exists Y (\forall Z \neg r(Y,Z) \vee s(X,Y) ) ] \} \vee \forall X\, t(X)$

**Step 3: Standarize variables so that each quantifier uses a different variable**

$\forall X \{ [ \neg p(X) \vee \neg q(X) ] \vee [ r(X,a) \wedge \exists Y (\forall Z \neg r(Y,Z) \vee s(X,Y) ) ] \} \vee \forall W\, t(W)$

**Step 4: Move all quantifiers to the left without changing the order**

$\forall X \; \exists Y \; \forall Z \; \forall W \{ [ \neg p(X) \vee \neg q(X) ] \vee [ r(X,a) \wedge ( \neg r(Y, Z) \vee s(X, Y) ) ] \} \vee t(W)$

**Step 5: Skolemization**

$\forall X \; \forall Z \; \forall W \{ [ \neg p(X) \vee \neg q(X) ] \vee [ r(X, a) \wedge ( \neg r(f(X), Z) \vee s(X, f(X) ) ) ] \} \vee t(W)$

**Step 6: Drop all $\forall$ quantifiers**

$\{ [ \neg p(X) \vee \neg q(X) ] \vee [ r(X, a) \wedge ( \neg r(t(X), Z) \vee s(X, f(X) ) ) ] \} \vee t(W)$

**Step 7: Convert the expression into a conjunction of disjunctions**

$$\mathbf{p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)}$$

$[ [ \neg p(X) \vee \neg q(X) ] \vee r(X, a) ] \wedge [ [ \neg p(X) \vee \neg q(X) ] \vee ( \neg r(f(X), Z) \vee s(X, f(X) ) ) ] ] \vee t(W)$

$\{ [ [ \neg p(X) \vee \neg q(X) ] \vee r(X, a) ] \vee t(W) \} \wedge \{ [ [ \neg p(X) \vee \neg q(X) ] \vee ( \neg r(f(X), Z) \vee s(X, f(X) ) ) ] \vee t(W)$

$[ \neg p(X) \vee \neg q(X) \vee r(X, a) \vee t(W) ] \wedge [ \neg p(X) \vee \neg q(X) \vee \neg r(f(X), Z) \vee s(X, f(X)) \vee t(W) ]$

**Step 8: Write each conjunction as a separate clause**

i) $\neg p(X) \vee \neg q(X) \vee r(X, a) \vee t(W)$
ii) $\neg p(X) \vee \neg q(X) \vee \neg r(f(X), Z) \vee s(X, f(X)) \vee t(W)$

**Step 9: Rename variables so that each clause assume a different set of variables**

i) $\neg p(X_1) \vee \neg q(X_1) \vee r(X_1, a) \vee t(W_1)$
ii) $\neg p(X_2) \vee \neg q(X_2) \vee \neg r(f(X_2), Z) \vee s(X_2, f(X_2)) \vee t(W_2)$

## Examples for resolution:

**Example 1 : Given the following information “ Max is a dog” , “All dogs are animals”, “All animals will die”; find out whether Max will die or no ? using resolution method.**

**Example 2 : Consider the following story: “Anyone passing his history exams and winning the lottery is happy. Anyone who studies or is lucky pass all his exams. Ali did not study but he is lucky. Anyone who is lucky wins the lottery.” Answer the following question using resolution method “ Who is happy ?”**

## STRUCTURES AND STRATEGIES FOR STATE SPACE SEARCH

Search is a universal problem-solving mechanism in AI. In AI problems, the sequence of steps required to solve a problem are not known a priori, but often must be determined by a systematic trial-and-error exploration of alternatives. There are three general classes of problems that have been addressed by AI search algorithms: single-agent path finding problems (e.g. 8-puzzle game), two-player games (e.g. tic-tac-toe), and constraint-satisfaction problems (e.g. 8-queens puzzle).

### 1. Problem Space Model

A *problem space* is the environment in which a search takes place. A problem space consists of a set of *states* of the problem, and a set of *operators* that change the states. For example, in the 8-puzzle game (see Figure 1), the states are the different possible permutation of the tiles, and the operators slide the tile into the empty position.

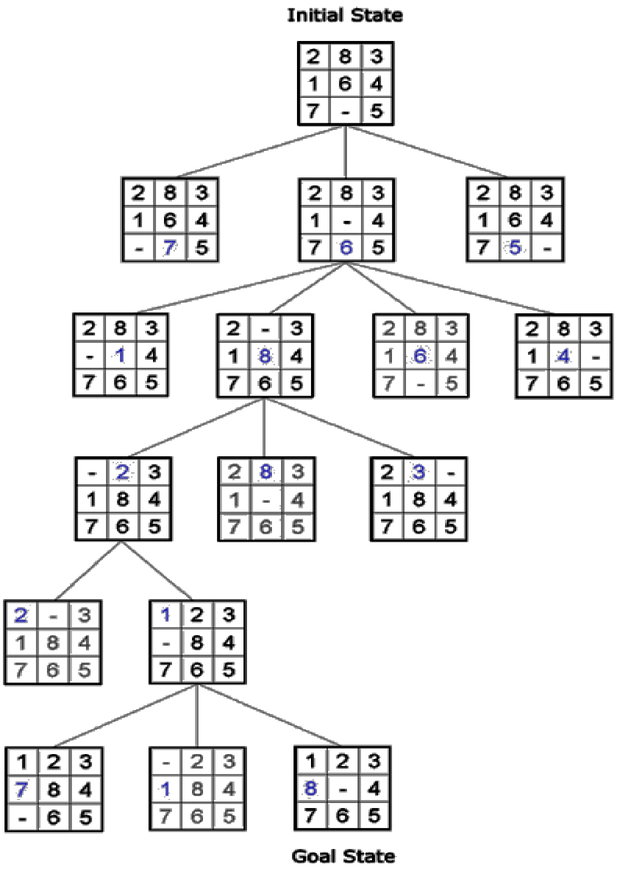


**Figure 1:** Single player 8-Puzzle game

## 2. Search Terminologies

- **Problem Space** − It is the environment in which the search takes place. (A set of states and set of operators to change those states)
- **Problem Instance** − It is Initial state + Goal state.
- **Problem Space Graph** − It represents the problem space. States are shown by nodes and operators are shown by edges.
- **Depth of a problem** − Length of the shortest path or the shortest sequence of operators from initial state to goal state.
- **Space Complexity** − The maximum number of nodes that are stored in memory.
- **Time Complexity** − The maximum number of nodes that are created.

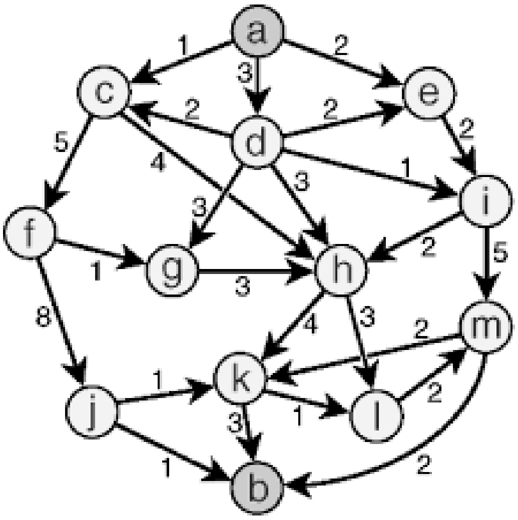


**Figure 2:** State space graph (Problem space graph)

## 3. Search Algorithms

- Systematic Search Algorithms
- Heuristic Search Algorithms

### 1- Systematic Search Algorithms

Determines the orders in which states are examined in the tree or graph, there are many possibilities, such as depth-first search and breadth-first search algorithms.

❖ **Depth-First-Search Algorithm**

In depth first search, when a state is examined, all of its children and their descendants are examined before any of its siblings (see Figure 3). Depth-first search goes deeper into the search space whenever this is possible.

```
Algorithm Depth-First-Search
  Begin
      Initialization: $open = [Start], close = []$, $parent [Start] = "null"$,
      $found = No.$
      While $open <> []$ do
          Begin
                      -  remove the first state from left of open, call it $X$;
                      -  if $X$ is a goal then $found = true$, break;
                      -  generate all children of $X$ and put them in list $L$;
                      -  put $X$ in $close$;
                      -  eliminate from $L$ any states already in $close$;
                      -  eliminate from $open$ any states already in $L$;
                      -  append $L$ to the left of $open$;
                      -  for each child $Y$ in $L$ set $parent[Y] = X$;
                      -  empty $L$
              End;
          If ($found = true$) compute the solution path;
          Else return fail;
    End.
```

## Tracing and Returning a Path in Depth First Search

**Consider the following state space graph with Initial State: a and Goal State: j**

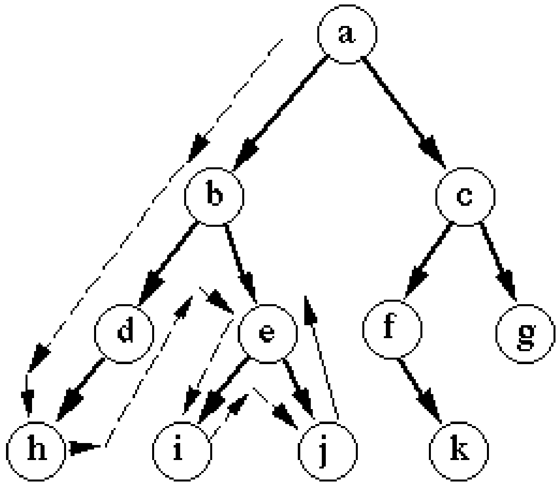


**Figure 3:** Order of node generation for Depth-First-Search algorithm

#0 $open = [a],\ close = [],\ found = No$

<u>database</u>

parent[a] = 'null'
parent[b] = a
parent[c] = a
parent[d] = b
parent[e] = b
parent[h] = d
parent[i] = e
parent[j]= e

#1 $X = a$
$a$ not the goal
$L = [\ b\ c\ ]$
$close = \ [\ a\ ]$
$open\ = \ [\ b\ c\ ]$

#2 $X = b$
$b$ not the goal
$L = [\ d\ e\ ]$
$close = \ [\ a\ b]$
$open\ = \ [\ d\ e\ c\ ]$

#3 $X = d$
$d$ not the goal
$L = [\ h\ ]$
$close = \ [\ a\ b\ d\ ]$
$open\ = \ [\ h\ e\ c\ ]$

#4 $X = h$
$h$ not the goal
$L = [\ ]$
$close = \ [\ a\ b\ d\ h\ ]$
$open\ = \ [\ e\ c\ ]$

#5 $X = e$
$e$ not the goal
$L = [\ i\ j\ ]$
$close = \ [\ a\ b\ d\ h\ e\ ]$
$open\ = \ [\ i\ j\ c\ ]$

#6 $X = i$
$i$ not the goal
$L = [\ ]$
$close = \ [\ a\ b\ d\ h\ e\ i\ ]$
$open\ = \ [\ j\ c\ ]$

#7 $X = j$
$j$ the goal is found, stop the search

The path is: **a** → **b** → **e** → **j**

### ❖ Breadth-First-Search Algorithm

Breadth-first search expands nodes in order of their distance from the root, generating one level of the tree at a time until a solution is found (see Figure 4). It is most easily implemented by maintain a queue of nodes, initially containing just the root, and always removing the node at the head of the queue, expanding it, and adding its children to the end of the queue.

```
Algorithm Breadth-First-Search
  Begin
     Initialization: $open = [Start], close = []$, $parent[Start] = "null"$,
     $found = No$.
     While $open <> []$ do
        Begin
              -  remove the first state from left of open, call it $X$;
              -  if $X$ is a goal then $found = true$, break;
              -  generate all children of $X$ and put them in list $L$;
              -  put $X$ in $close$;
              -  eliminate from $L$ any states already in $open$ or $close$;
              -  append $L$ to the right of $open$;
              -  for each child $Y$ in $L$ set $parent[Y] = X$;
              -  empty $L$
         End;
        If ($found = true$) compute the solution path;
        Else return fail;
    End.
```

## Tracing and Returning a Path in Breadth-First-Search

**Consider the following state space graph with Initial State: a and Goal State: j**

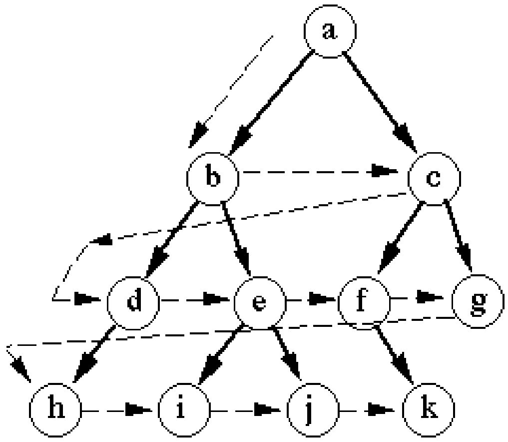


**Figure 4:** Order of node generation for Breadth-First-Search algorithm

#0 $open = [a],\ close = [],\ found = No$

#1 $X = a$
$a$ not the goal
$L = [\ b\ c\ ]$
$close =\ [\ a\ ]$
$open\ =\ [\ b\ c\ ]$

#2 $X = b$
$b$ not the goal
$L = [\ d\ e\ ]$
$close =\ [\ a\ b]$
$open\ =\ [\ c\ d\ e\ \ ]$

#3 $X = c$
$c$ not the goal
$L =\ [\ f\ g\ ]$
$close\ =\ [a\ b\ c\ ]$
$open\ =\ [\ d\ e\ f\ g\ ]$

#4 $X = d$
$d$ not the goal
$L =\ [\ h\ ]$
$close\ =\ [a\ b\ c\ d]$
$open\ =\ [\ \ e\ f\ g\ h\ ]$

#5 $X = e$
$e$ not the goal
$L =\ [\ i\ \ j\ ]$
$close\ =\ [a\ b\ c\ d\ e]$
$open\ =\ [\ \ f\ g\ h\ \ i\ j\ ]$

#6 $X = f$
$f$ not the goal
$L =\ [\ k\ ]$
$close\ =\ [a\ b\ c\ d\ e\ f\ ]$
$open\ =\ [\ \ g\ h\ i\ j\ k\ ]$

#7 $X = g$
$g$ not the goal
$L =\ [\ \ ]$
$close\ =\ [\ a\ b\ c\ d\ e\ f\ \ g\ ]$
$open\ =\ [\ h\ i\ j\ k\ ]$

database

parent[a] = ’null’
parent[b] = a
parent[c] = a
parent[d] = b
parent[e] = b
parent[f] = c
parent[g] = c
parent[h]= d
parent[i]= e
parent[j]= e
parent[k]=f

#8 $X = h$
$h$ not the goal
$L = [\ ]$
$close = [a\ b\ c\ d\ e\ f\ g\ h]$
$open = [\ i\ j\ k]$

#9 $X = i$
$i$ not the goal
$L = [\ ]$
$close = [a\ b\ c\ d\ e\ f\ g\ h\ i]$
$open = [\ j\ k]$

#10 $X = j$
$j$ the goal is found, stop the search

The path is : **a → b → e → j**

## Homework:

Draw the problem space graph of the following 8-puzzel game, and then find the path using Depth-first-search and Breadth-first- search algorithms.

Initial State

| | | |
|---|---|---|
| 1 | 4 | 3 |
| 7 | | 6 |
| 5 | 8 | 2 |

Goal State

| | | |
|---|---|---|
| 1 | 4 | 3 |
| 7 | 8 | 6 |
| | 5 | 2 |

Game operations: Move the blank tile to (up, down, left, right)

## Comparison between depth- and breadth- first search algorithms:

Depth-first search gets quickly into a deep search space. If it is known that the solution path will be long, depth-first search won't waste time searching a large number of "shallow" states in the graph. On the other hand, depth-first search can get "lost" deep in a graph, missing shorter paths to a goal or even becoming stuck in an infinitely long path that does not lead to a goal. Depth-first search is much more efficient for searching spaces with many branches since it does not have to keep all the nodes at a given level on the memory "open list", so it requires less memory space. Unlike breadth-first search, a depth first search does not guarantee to find the shortest path to a state the first time it is encountered. Whereas, breadth-first search always finds the shortest path to a goal node. Breadth-first search will not get trapped into along unfruitful path.

## 2- Heuristic Search Algorithms

All of the search methods in the preceding section are uninformed in that they did not take into account the goal. They do not use any information about where they are trying to get to unless they happen to stumble on a goal. One form of heuristic information about which nodes seem the most promising is a *heuristic function $h(n)$*, which takes a node $n$ and returns a non-negative real number that is an estimate of the path cost from node $n$ to a goal node. The heuristic function is a way to inform the search about the direction to a goal. It provides an informed way to guess which neighbor of a node will lead to a goal. There is no general theory for finding heuristics, because every problem is different.

Another way to measure the cost from the start state to the goal state is the *evaluation function $f(n)$ (cost function)*. Cost function can be measured as,

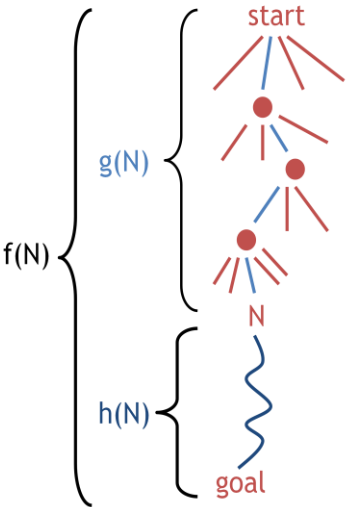
$$f(n) = g(n) + h(n)$$

where

$h(n)$ is the heuristic function that estimates the distance between node $n$ and a goal node.

$g(n)$ is the (known) distance from the start state to a goal node $n$.

$f(n)$ gives you the (partially estimated) distance from the start node to a goal node $n$.



We will consider two types of heuristic algorithms: Heuristic search algorithms (with no cost function) and heuristic search algorithm (with cost function).

## ❖ Heuristic Search Algorithms (with no cost function)

- Hill Climbing Algorithm
- Best – First Search Algorithm

### - Hill Climbing Algorithm

Hill climbing (HC) algorithm is a technique for certain classes of optimization problems. The idea is to start with a sub-optimal solution to a problem (i.e., start at the base of a hill) and then repeatedly improve the solution (walk up the hill) until some condition is maximized (the top of the hill is reached).

**Hill Climbing Algorithm**

```
Begin
   $CS = Start,\ open = [Start],\ Stop = false,\ path = []$
   While ($Not\ Stop$) do
      Begin
         -   Add $CS$ to $path$;
         -   If ( $CS = Goal$ ) then return ($path$), break; /* reached */
         -   Empty $open$;
         -   Generate all possible children of $CS$ and put them in $open$;
         -   If $open = []$ then $stop = true$; /* dead end */
             Else Let $X = CS$
             For each state $Y$ in $open$ do begin
               - Compute the heuristic value of $Y$, $h(Y)$
               - If $Y$ is better than $X$ then $X = Y$;
             EndFor
             If $X$ is better than $CS$ then $CS = X$
             Else $stop = true$    /* local optima */
             EndIF
       EndWhile
       Return (Fail)
End.
```

## Tracing and Returning a Path in Hill-Climbing Algorithm

**Consider the following state space graph in Figure 4 with Initial State: A and Goal State: M. Find the path using Hill-Climbing algorithm.**

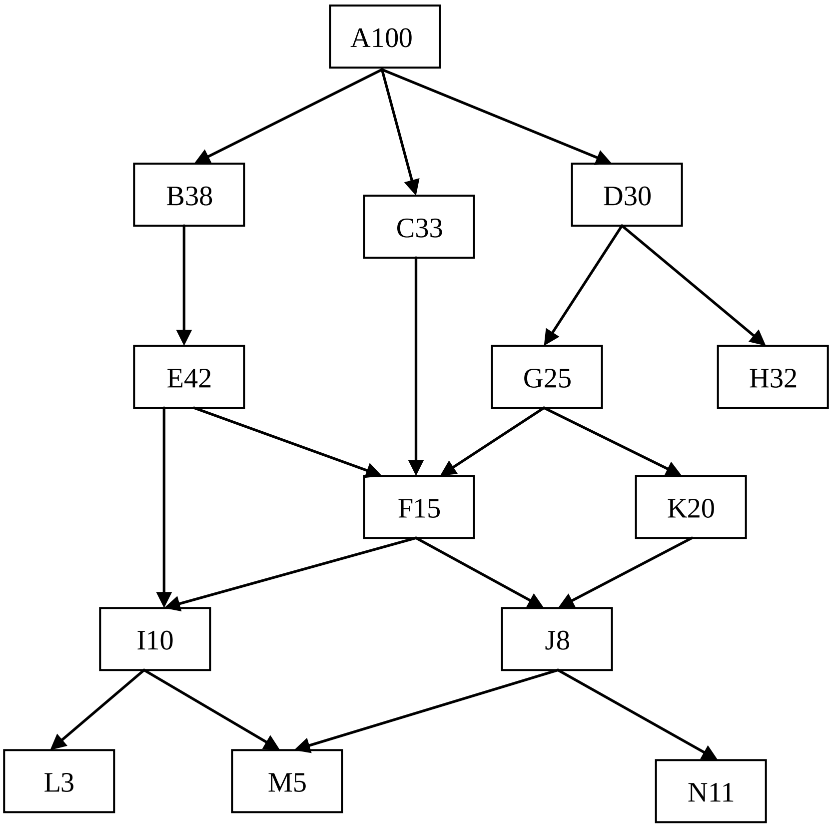


**Figure 4:** State space graph with heuristic values

#0 $CS = A,\ open = [A],\ stop = false,\ path = []$

#1 $path = [A]$
$openA = [\ B38\ C33\ D30\ ]$
$X = D30$
$CS = D30$

#2 $path = [\ D\ A\ ]$
$openD = \ [\ G25\ H32\ ]$
$X = \ G25$
$CS = G25$

#3 $path = [\ G\ D\ A\ ]$
$openG = \ [\ F15\ K20\ ]$
$X = F15$
$CS = F15$

#4 $path = [\ F\ G\ D\ A\ ]$
$openF = [\ I10\ J8]$

$X = J8$
$CS = J8$

#5 $path = [J\ F\ G\ D\ A\ ]$
$openJ = [\ M5\ N11]$
$X = M5$
$CS = M5$

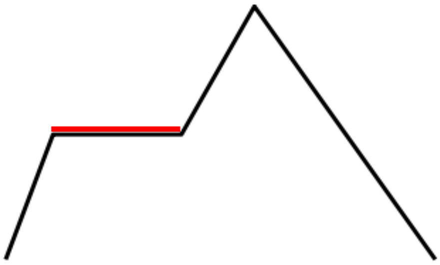
#6 $path = [\ M\ J\ F\ G\ D\ A\ ]$
$CS = goal$
Stop

Path: **A ⟶ D ⟶ G ⟶ F ⟶ J ⟶ M**

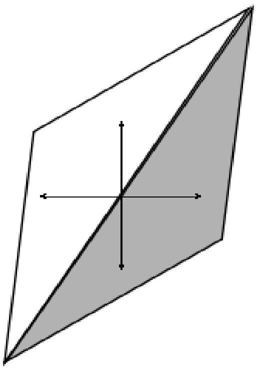
## Hill Climbing: Disadvantages

1- **Local maximum**
A state that is better than all of its neighbors, but not better than some other states far away.

2- **Plateau**
A flat area of the search space in which all neighboring states have the same value.



3- **Ridge**
The orientation of the high region, compared to the set of available moves, makes it impossible to climb up.



**Ways Out**

- For Local maximum problem, backtrack to some earlier node and try going in a different direction.
- For Plateau problem, make a big jump to try to get in a new section.
- For Ridge, moving in several directions at once.

## Implementation of Heuristic Evaluation Function:

We now evaluate the performance of several different heuristics for solving the 8-puzzel.
The simplest heuristic is to count the tiles out of place in each state when it is compared with the goal. This is intuitively appealing, since it would seem that, all else being equal, the state had fewest tiles out of place is probably closer to the desired goal and would be the best to examine next.
However, this heuristic does not use all of the information available in a board configuration; since it does not take into account the distance the tiles must be moved. A "better" heuristic would sum all the distances by which the tiles are out of place, one for each square a tile must be moved to reach its position in the goal state, Table 2 shows the result of applying each of these two heuristics to the three children states with comparison to a goal state.

**Table 2: Different cases of calculating the heuristic function in the 8-puzzle game**

| State | Goal state | Tiles out of place | Sum of distances out of place |
|---|---|---|---|
| 2 8 3<br>1 6 4<br>  7 5 | 1 2 3<br>8   4<br>7 6 5 | 5 | 6 |
| 2 8 3<br>1   4<br>7 6 5 | 1 2 3<br>8   4<br>7 6 5 | 3 | 4 |

<table>
<tr>
<td><table><tr><td>2</td><td>8</td><td>3</td></tr><tr><td>1</td><td>6</td><td>4</td></tr><tr><td>7</td><td>5</td><td></td></tr></table></td>
<td><table><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>8</td><td></td><td>4</td></tr><tr><td>7</td><td>6</td><td>5</td></tr></table></td>
<td>5</td>
<td>6</td>
</tr>
</table>

- **Best-First Search Algorithm**

Best first search simply chooses the unvisited node with the best heuristic value to visit next. It can be implemented in the same algorithm as lowest-cost Breadth First Search. This time the priority of each node added to the queue as its heuristic value.

**Algorithm Best-First Search (with no cost)**

```
Begin
Initialization: $open = [S]$; $closed = [\,]$; $pred\ [S] = "null"$; $found = false$
  While ( $open <> []$ ) and ( $found = false$ ) Do
      Begin
            - Reordered the $open$;
            - Remove the first element from the left of open, call it $X$;
            - If $X$ is the goal then $found = true$
              Else
                Begin
                  - Generate all children of $X$ and put them in $L$;
                 - Remove $X$ from open and put it in $close$;
                 - For each child $Y$ Do
                - If $Y$ is not already in $open$ or $closed$ then
                  Begin
                - Compute $h[Y]$;
                - $pred\ [Y] = X$;
                - Insert $Y$ in $open$;
               End
              Else remove $Y$ from $L$;
         EndFor
   - Reordered open;
   Endif
Endwile
If $found = false$ then output failure
Else
Trace the $pred$ list from $X$ to the start node $S$ to form the path;
End.
```

## Tracing and Returning a Path in Best-First Search Algorithm

**Consider the following state space graph in Figure 5 with Initial State: A and Goal State: M. Find the path using Best-First search algorithm.**

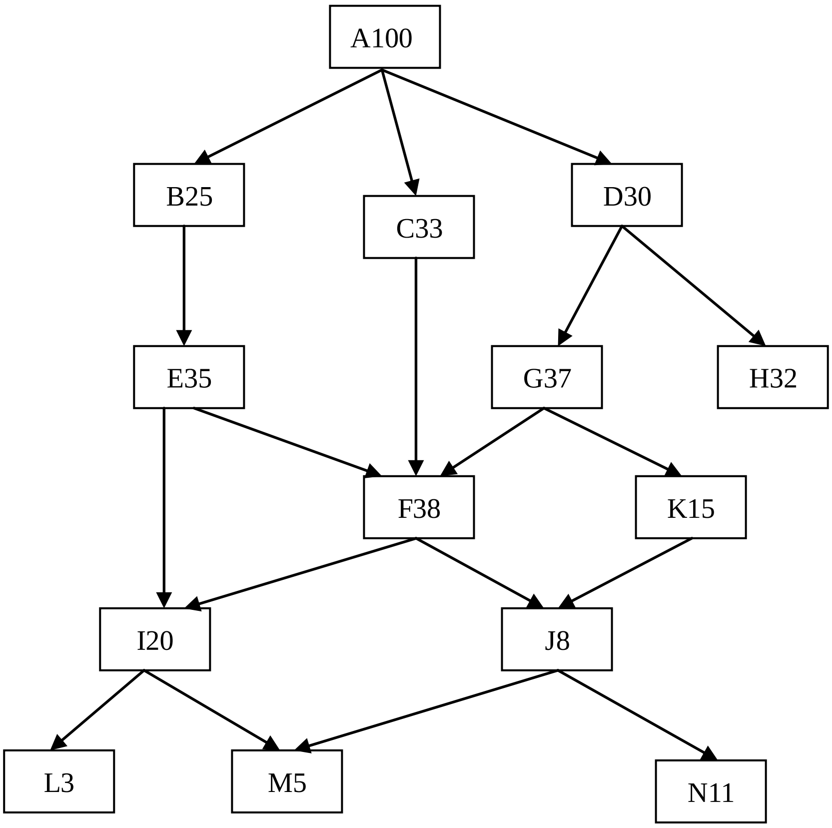


**Figure 5:** State space graph with heuristic values

#0 $open = [A100],\ close = [], found = no$

#1 $X = A100$

$Found = no$

$L = [B25\ \ C33\ \ D30]$

$Close = [A100]$

$Open = [\cancel{B25}\ D30\ C33\ ]$

#2 $X = B25$

$Found = no$

$L = [E35]$

$Close = [A100\ B25]$

$Open = [\cancel{D30}\ C33\ E35\ ]$

$p[A] = "null"$

$p[B] = A$

$p[C] = A$

$p[D] = A$

$p[E] = B$

$p[G] = D$

$p[H] = D$

$p[F] = C$

$p[I] = E$

$p[L] = I$

$p[M] = I$

#3 $X = D30$
$Found = no$
$L = [G37\ H32]$
$Close = [A100\ B25\ D30]$
$Open = [\cancel{H32}\ C33\ E35\ G37\ ]$

#4 $X = H32$
$Found = no$
$L = []$
$Close = [A100\ B25\ D30\ H32]$
$Open = [\cancel{C33}\ E35\ G37\ ]$

#5 $X = C33$
$Found\ = no$
$L = [F38]$
$Close = [A100\ B25\ D30\ H32\ C33]$
$Open = [\cancel{E35}\ G37\ F38]$

#6 $X = E35$
$Found = no$
$L = [I20\ \cancel{F38}]$
$Close = [A100\ B25\ D30\ H32\ C33\ E35]$
$Open = [\cancel{I20}\ \ G37\ F38]$

#7 $X = I20$
$Found = no$
$L = [L3\ M5]$
$Close = [A100\ B25\ D30\ H32\ C33\ E35\ I20]$
$Open = [\cancel{L3}\ M5\ \ G37\ F38\ ]$

#8 $X = L3$
$Found = no$
$L = []$
$Close = [A100\ B25\ D30\ H32\ C33\ E35\ I20\ L3]$
$Open = [\ \cancel{M5}\ \ G37\ F38\ ]$

#9 $X = M5$
$Found = true$

**The Path is:** A → B → E → I → M

## ❖ Heuristic Search Algorithms (with cost function)

- A* Algorithm ( Best-First Search Algorithm with cost ).

```
Algorithm Best-first search (with cost function)
Begin
- Initialization: $open = [Start], closed = [], g[Start] = 0, pred[Start] = null, found = false$;
- While ($open <> []$) and ($found = false$) Do
  Begin
   - Remove the best element from $open$ and call it X;
   - If $X$ is the goal then $found = true$;
    Else
     Begin
      - Generate the children of $X$;
      - Remove $X$ from $open$ and put it in $close$;
      - For each child $Y$ of $X$ Do
      Begin
        - If ($Y \notin open$) and ($Y \notin close$) then
          Begin
           - $g[Y] = g[X] + cost(X,Y)$;
           - $f[Y] = g[Y] + h[Y]$;
           - $pred[Y] = X$;
           - Insert $Y$ in $open$;
        End if
        Else /* $Y$ in $open$ or $close$ */
         Begin
          - $Temp = h[Y] + g[X] + cost(X,Y)$;
          - If $temp < f[Y]$ then
            Begin
             - $g[Y] = g[X] + cost(X,Y)$;
             - $f[Y] = Temp$;
             - $pred[Y] = X$;
             - If $Y$ is in $close$ then insert $Y$ in $open$ and remove it from $close$
            End if
        End else
      End else
   End while
   - If found is false then output "failure";
    Else
   Trace pointer in $pred$ fields to construct the path;
End.
```

## Tracing and Returning a Path in A* Algorithm

**Consider the following state space graph in Figure 6 with Initial State: A and Goal State: M. Find the path using A* algorithm.**

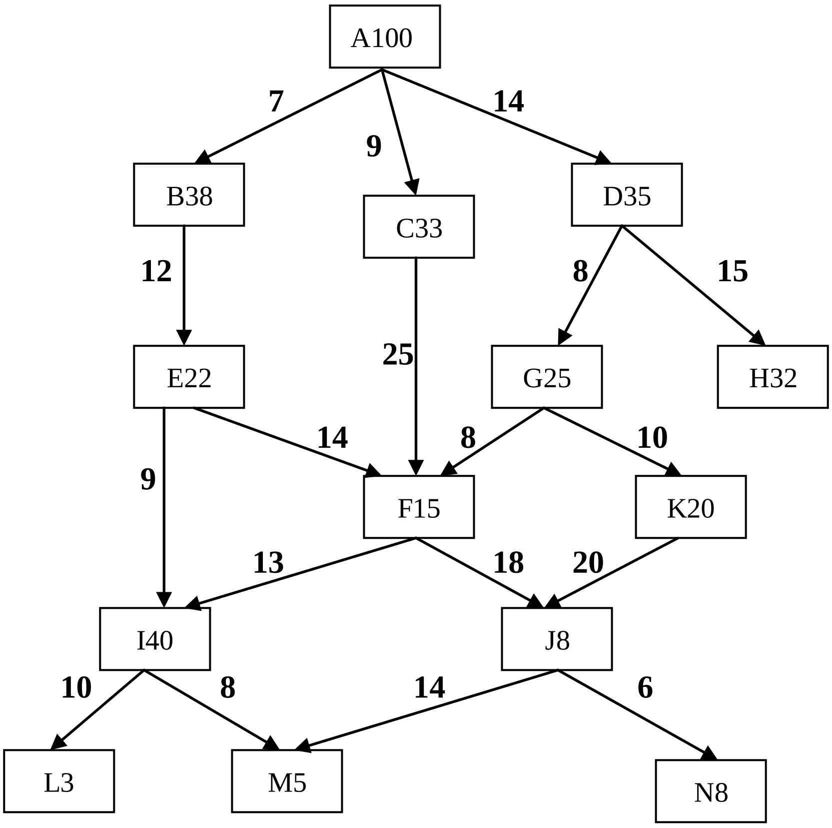


**Figure 6:** State space graph with heuristic and cost values

#0 Initialization: $open = [A100]$, $close = []$, $found = false$

#1 $X = A100$

$found\ = false$

$L = [\ B38\ C33\ D35\ ]$

$Close = [\ A100]$

$g(B) = 0 + 7 = 7 \quad f(B) = 38 + 7 = 45$

$g(C) = 0 + 9 = 9 \quad f(C) = 33 + 9 = 42$

$g(D) = 0 + 14 = 14 \quad f(D) = 35 + 14 = 49$

$Open = [\ B45\ \ \cancel{C42}\ \ D49\ ]$

#2 $X = C42$
$found = false$
$L = [F15]$
$close = [A100\ C42]$
$g(F) = 9 + 25 = 34 \quad f(F) = 15 + 34 = 49$
$open = [\cancel{B45}\ F49\ \ D49]$

#3 $X = B45$
$found = false$
$L = [E22]$
$close = [A100\ \ C42\ \ B45]$
$g(E) = 7 + 12 = 19 \quad f(E) = 22 + 19 = 41$
$open = [\cancel{E41}\ \ F49\ \ D49\ ]$

#4 $X = E41$
$found = false$
$L = [I40\ \ F15]$
$close = [A100\ \ C42\ \ B45\ \ E41]$
$g(I) = 19 + 9 = 28 \quad f(I) = 40 + 28 = 68$
$Temp(F) = 15 + 19 + 14 = 48$
**(* change the current value of $F$ in $open$ and database *)**
$open = [\cancel{F48}\ \ D49\ \ I68]$

#5 $X = F48$
$found = false$
$L = [I40\ \ J8]$
$close = [A100\ \ C42\ \ B45\ \ E41\ \ F48]$
$Temp(I) = 40 + 33 + 13 = 86$ **(* ignore *)**
$g(J) = 33 + 18 = 51 \quad f(J) = 51 + 8 = 59$
$open = [\cancel{D49}\ \ J59\ I68]$

#6 $X = D49$
$found = false$
$L = [G25\ \ H32]$
$close = [A100\ \ C42\ \ B45\ \ E41\ \ F48\ \ D49]$
$g(G) = 14 + 8 = 22 \quad f(g) = 25 + 22 = 47$
$g(H) = 14 + 15 = 29 \quad f(H) = 32 + 29 = 61$
$open = [\ \cancel{G47}\ J59\ \ H61\ \ I68]$

$\#7\ X = G47$

$found = false$

$L = [F15\ \ K20]$

$close = [A100\ \ C42\ \ B45\ \ E41\ \ F48\ \ D49\ G47]$

$Temp(F) = 15 + 22 + 8 = 45$

**(* remove $F$ from $close$ and put it in $open$, then change the database *)**

$g(K) = 22 + 10 = 32\ \ f(K) = 20 + 32 = 52$

$open = [\cancel{F45}\ \ K52\ \ J59\ \ H61\ \ I68]$

#8 $X = F45$

$found = false$

$L = [I40\ \ J8]$

$close = [A100\ \ C42\ \ B45\ \ E41\ \ F48\ \ D49\ G47\ F45]$

$Temp(I) = 40 + 13 + 30 = 83$ **(* ignore *)**

$Temp(J) = 8 + 18 + 30 = 56$

**(* change the current value of $J$ in $open$, then change the database *)**

$open = [\cancel{K52}\ J56\ \ H61\ \ I68]$

#9 $X = K52$

$found = false$

$L = [J8]$

$close = [A100\ \ C42\ \ B45\ \ E41\ \ F48\ \ D49\ G47\ F45\ \ K52]$

$Temp(J) = 8 + 32 + 20 = 60$ **(* ignore *)**

$open = [\cancel{J56}\ H61\ \ I68]$

#10 $X = J56$

$found = false$

$L = [M5\ \ N8]$

$close = [A100\ \ C42\ \ B45\ \ E41\ \ F48\ \ D49\ G47\ F45\ \ K52\ J56]$

$g(M) = 48 + 14 = 62\ \ f(M) = 5 + 62 = 67$

$g(N) = 48 + 6 = 54\ \ f(N) = 8 + 54 = 62$

$open = [\cancel{H61}\ N62\ \ M67\ \ I68]$

#11 $X = H61$

$found = false$

$L = []$

$close = [A100\ \ C42\ \ B45\ \ E41\ \ F48\ \ D49\ G47\ F45\ \ K52\ J56\ \ H61]$

$open = [\cancel{N62}\ M67\ \ I68]$

#12 $X = N62$

$found = false$

$L = []$

$close = [A100\ \ C42\ \ B45\ \ E41\ \ F48\ \ D49\ G47\ F45\ \ K52\ J56\ \ H61\ \ N62]$

$open = [\cancel{M}67\ \ I68]$

#13 $X = M67$

$found = true$

**The Path is:** $\mathbf{A_{100} \longrightarrow D_{49} \longrightarrow G_{47} \longrightarrow F_{45} \longrightarrow J_{56} \longrightarrow M_{67}}$

**(* To reach the shortest path we must continue search the state space *)**

**Table 1: Database Table**

| **State** | $\boldsymbol{pred\ (state)}$ | $\boldsymbol{h\ (state)}$ | $\boldsymbol{g\ (state)}$ | $\boldsymbol{f\ (state)}$ |
|---|---|---|---|---|
| A | Null | 100 | 0 | 100 |
| B | A | 38 | 7 | 45 |
| C | A | 33 | 9 | 42 |
| D | A | 35 | 14 | 49 |
| F | ~~C~~ ~~E~~ G | 15 | ~~34~~ ~~33~~ 30 | ~~49~~ ~~48~~ 45 |
| E | B | 22 | 19 | 41 |
| I | E | 40 | 28 | 68 |
| J | F | 8 | ~~51~~ 48 | ~~59~~ 56 |
| G | D | 25 | 22 | 47 |
| H | D | 32 | 29 | 61 |
| K | G | 20 | 32 | 52 |
| M | J | 5 | 62 | 67 |
| N | J | 8 | 54 | 62 |

#14 $X = I68$

$found = false$

$L = [L3\ M5]$

$close = [A100\ \ C42\ \ B45\ \ E41\ \ F48\ \ D49\ G47\ F45\ \ K52\ J56\ \ H61\ \ N62\ \ M67\ \ I68]$

$g(L) = 28 + 10 = 38 \quad f(L) = 3 + 38 = 41$

$Temp(M) = 5 + 28 + 8 = 41$

$open = [\cancel{L4}1\ \ M41]$

$\#15\ X =\ L41$
$Found = false$
$L = []$
$close =\ [A100\ \ C42\ \ B45\ \ E41\ \ F48\ \ D49\ G47\ F45\ \ K52\ J56\ \ H61\ \ N62\ \ M67\ \ I68\ \ L41]$
$open = [\cancel{M4}1]$

$\#16\ X =\ M41$
$Found = yes$

**The Path is:** $\mathbf{A_{100}} \longrightarrow \mathbf{B_{45}} \longrightarrow \mathbf{E_{41}} \longrightarrow \mathbf{I_{68}} \longrightarrow \mathbf{M_{41}}$

## References

1- **Artificial Intelligence: Structures and Strategies for Complex Problem Solving. George F. Lugar. 2008**

2- **Artificial Intelligence. Elain Rich and Kevin Knight. 1991.**

3- **Logic-Based Artificial Intelligence. Jack Minker. 2000.**