8 Queens

```
eight_queens(Board) :- length(Board, 8), place_queens(Board), safe(Board).

place_queens([]).

place_queens([Column | Columns]) :-
   place_queens(Columns),
   between(1, 8, Column),
   \+ member(Column, Columns).

safe([]).

safe([Q | Queens]) :-
   safe_from(Q, Queens, 1),
   safe(Queens).

safe_from(_, [], _).

safe_from(Q, [Q1 | Queens], D) :-
   Q =\= Q1,
   abs(Q - Q1) =\= D,
   D1 is D + 1,
   safe_from(Q, Queens, D1).
```

OUTPUT

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.3.14-16-gf1d555c05)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- ["D:/AI-Practical/8queens.pl"].
true.
2 ?- eight_queens(Board).
Board = [4, 2, 7, 3, 6, 8, 5, 1]
```

Depth First Search (DFS)

```
% Define the edges of a simple graph.
edge(a, b).
edge(a, c).
edge(b, d).
edge(b, e).
edge(c, f).
edge(e, g).
edge(f, g).
% Depth First Search rule
dfs(Start, Goal, Path) :-
   dfs_recursive(Start, Goal, [Start], Path).
dfs_recursive(Goal, Goal, Path, Path).
dfs_recursive(Start, Goal, Visited, Path) :-
   edge(Start, NextNode),
   \+ member(NextNode, Visited),
   dfs_recursive(NextNode, Goal, [NextNode | Visited], Path).
```

OUTPUT

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.3.14-16-gf1d555c05)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- ["D:/AI-Practical/dfs.pl"].
true.
?- dfs(a, g, Path).
Path = [g, e, b, a] ;
Path = [g, f, c, a].
```

Best First Search

```
% Define the graph edges and their costs.

edge(a, b, 1).

edge(a, c, 3).

edge(b, d, 1).

edge(b, e, 6).

edge(c, f, 5).

edge(d, g, 3).

edge(e, g, 2).

edge(f, g, 1).


% Define heuristic values for each node.

heuristic(a, 6).

heuristic(b, 5).

heuristic(c, 4).

heuristic(d, 3).

heuristic(e, 2).

heuristic(f, 3).

heuristic(g, 0).  % Goal node has heuristic 0


% Best First Search rule

best_first_search(Start, Goal, Path, Cost) :-

   best_first_recursive([[Start, 0]], Goal, [], Path, Cost).


% Recursive Best First Search

best_first_recursive([[Goal, Cost] | _], Goal, _, [Goal], Cost).

best_first_recursive([[Current, CurrentCost] | RestQueue], Goal, Visited, [Current | Path], Cost) :-

   findall(

      [Next, NewCost],

      (edge(Current, Next, StepCost),

       \+ member(Next, Visited),
```

```
        heuristic(Next, H),
        NewCost is CurrentCost + StepCost + H),
       NextNodes),
    append(RestQueue, NextNodes, Queue),
    sort(2, @=<, Queue, SortedQueue),
    best_first_recursive(SortedQueue, Goal, [Current | Visited], Path, Cost).
```

OUTPUT

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.3.14-16-gf1d555c05)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- ["D:/AI-Practical/best_first_search.pl"].
true.

?- best_first_search(a, g, Path, Cost).
Path = [a, b, d, g],
Cost = 5 ;
Path = [a, c, f, g],
Cost = 7.
```

8-Puzzle

```
% Define the goal state.
goal([1, 2, 3, 4, 5, 6, 7, 8, 0]).

% Calculate Manhattan distance as a heuristic.
manhattan(State, Distance) :-
   goal(Goal),
   manhattan_distance(State, Goal, 0, Distance).

manhattan_distance([], [], D, D).
manhattan_distance([Tile | RestTiles], [TileGoal | RestGoal], Acc, Distance) :-
   ( Tile =\= 0 ->
      index(Tile, Goal, IndexTile),
      index(TileGoal, Goal, IndexGoal),
      PosX is IndexTile mod 3,
      PosY is IndexTile // 3,
      GoalX is IndexGoal mod 3,
      GoalY is IndexGoal // 3,
      StepDistance is abs(PosX - GoalX) + abs(PosY - GoalY),
      NewAcc is Acc + StepDistance
   ;   NewAcc = Acc ),
   manhattan_distance(RestTiles, RestGoal, NewAcc, Distance).

index(Element, List, Index) :- nth0(Index, List, Element).

% Define possible moves for tiles in the puzzle.
move([0, B, C, D, E, F, G, H, I], [B, 0, C, D, E, F, G, H, I]).
move([A, 0, C, D, E, F, G, H, I], [A, C, 0, D, E, F, G, H, I]).
move([A, B, 0, D, E, F, G, H, I], [A, B, F, D, E, 0, G, H, I]).
% Additional moves here...
```

```
% Best First Search for solving 8-puzzle
best_first_puzzle(Start, Path, Cost) :-
   manhattan(Start, StartH),
   best_first([[Start, [], StartH]], [], Path, Cost).
best_first([[State, PathSoFar, _] | _], _, [State | PathSoFar], 0) :-
   goal(State).
best_first([[State, PathSoFar, _] | Rest], Visited, Solution, Cost) :-
   findall(
      [NextState, [State | PathSoFar], NextH],
      (move(State, NextState),
       \+ member(NextState, Visited),
       manhattan(NextState, NextH)),
      Moves),
   append(Rest, Moves, Queue),
   sort(3, @=<, Queue, SortedQueue),
   best_first(SortedQueue, [State | Visited], Solution, Cost).
```

OUTPUT

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.3.14-16-gf1d555c05)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- ["D:/AI-Practical/8puzzle.pl"].
true.

?- best_first_puzzle([1, 2, 3, 4, 0, 5, 6, 7, 8], Path, Cost).
Path = [[1, 2, 3, 4, 0, 5, 6, 7, 8], ..., [1, 2, 3, 4, 5, 6, 7, 8, 0]],
Cost = 5.
```

Robot Traversal

```
% Define the goal position.
goal([3, 3]).

% Calculate Manhattan distance between two points.
manhattan_distance([X1, Y1], [X2, Y2], Distance) :-
   Distance is abs(X1 - X2) + abs(Y1 - Y2).

% Possible moves for the robot (up, down, left, right).
move([X, Y], [X, Y1]) :- Y1 is Y - 1, Y1 >= 0.  % Move up
move([X, Y], [X, Y1]) :- Y1 is Y + 1, Y1 =< 3.  % Move down
move([X, Y], [X1, Y]) :- X1 is X - 1, X1 >= 0.  % Move left
move([X, Y], [X1, Y]) :- X1 is X + 1, X1 =< 3.  % Move right

% Means-End Analysis for finding a path from Start to Goal.
means_end(Start, Path) :-
   goal(Goal),
   means_end_recursive(Start, Goal, [Start], Path).

means_end_recursive(Goal, Goal, Visited, Path) :-
   reverse(Visited, Path).
means_end_recursive(Current, Goal, Visited, Path) :-
   findall(
      Next,
      (move(Current, Next),
       \+ member(Next, Visited)),
      Moves),
   sort_moves_by_heuristic(Moves, Goal, SortedMoves),
   SortedMoves = [BestMove | _],
   means_end_recursive(BestMove, Goal, [BestMove | Visited], Path).
```

```
% Sort moves by Manhattan distance to goal.
sort_moves_by_heuristic(Moves, Goal, SortedMoves) :-
    maplist(add_heuristic(Goal), Moves, MovesWithHeuristic),
    sort(2, @=<, MovesWithHeuristic, SortedMovesWithHeuristic),
    pairs_keys(SortedMovesWithHeuristic, SortedMoves).

add_heuristic(Goal, Move, Move-Distance) :-
    manhattan_distance(Move, Goal, Distance).
```

OUTPUT

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.3.14-16-gf1d555c05)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- ["D:/AI-Practical/robot_traversal.pl"].
true.

?- means_end([0, 0], Path).
Path = [[0, 0], [1, 0], [2, 0], [3, 0], [3, 1], [3, 2], [3, 3]].
```

Traveling Salesman Problem

```
% Define distances between pairs of cities.
distance(a, b, 10).
distance(a, c, 15).
distance(a, d, 20).
distance(b, c, 35).
distance(b, d, 25).
distance(c, d, 30).
distance(b, a, 10).
distance(c, a, 15).
distance(d, a, 20).
distance(c, b, 35).
distance(d, b, 25).
distance(d, c, 30).

% Calculate the total distance of a given path.
path_distance([_], 0).
path_distance([City1, City2 | Rest], Distance) :-
    distance(City1, City2, D),
    path_distance([City2 | Rest], RestDistance),
    Distance is D + RestDistance.

% Find all possible tours and select the one with the minimum distance.
tsp(StartCity, MinPath, MinDistance) :-
    findall(
        Path,
        (permute_cities([a, b, c, d], StartCity, Path)),
        Paths),
    maplist(path_distance, Paths, Distances),
    min_member(MinDistance, Distances),
    nth0(Index, Distances, MinDistance),
```

```
    nth0(Index, Paths, MinPath).

% Helper to generate permutations starting with StartCity.
permute_cities(Cities, Start, [Start | Perm]) :-
    select(Start, Cities, RemainingCities),
    permutation(RemainingCities, Perm).
```

OUTPUT

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.3.14-16-gf1d555c05)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- ["D:/AI-Practical/tsp.pl"].
true.

?- tsp(a, MinPath, MinDistance).
MinPath = [a, b, d, c],
MinDistance = 80.
```

Study of prolog

```
father(john, mary).

mother(mary, alice).

parent(X, Y) :- father(X, Y).

parent(X, Y) :- mother(X, Y).

factorial(0, 1).

factorial(N, Result) :-

   N > 0,

   N1 is N - 1,

   factorial(N1, SubResult),

   Result is N * SubResult.

member(X, [X|_]).

member(X, [_|Tail]) :- member(X, Tail).


append([], List, List).

append([Head|Tail], List, [Head|Result]) :-

   append(Tail, List, Result).
```

OUTPUT

```
?- parent(john, alice).

false.


?- factorial(5, Result).

Result = 120.


?- member(2, [1, 2, 3]).

true.


?- append([1,2], [3,4], Result).

Result = [1, 2, 3, 4].
```