ARTIFICIAL INTELLIGENCE

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

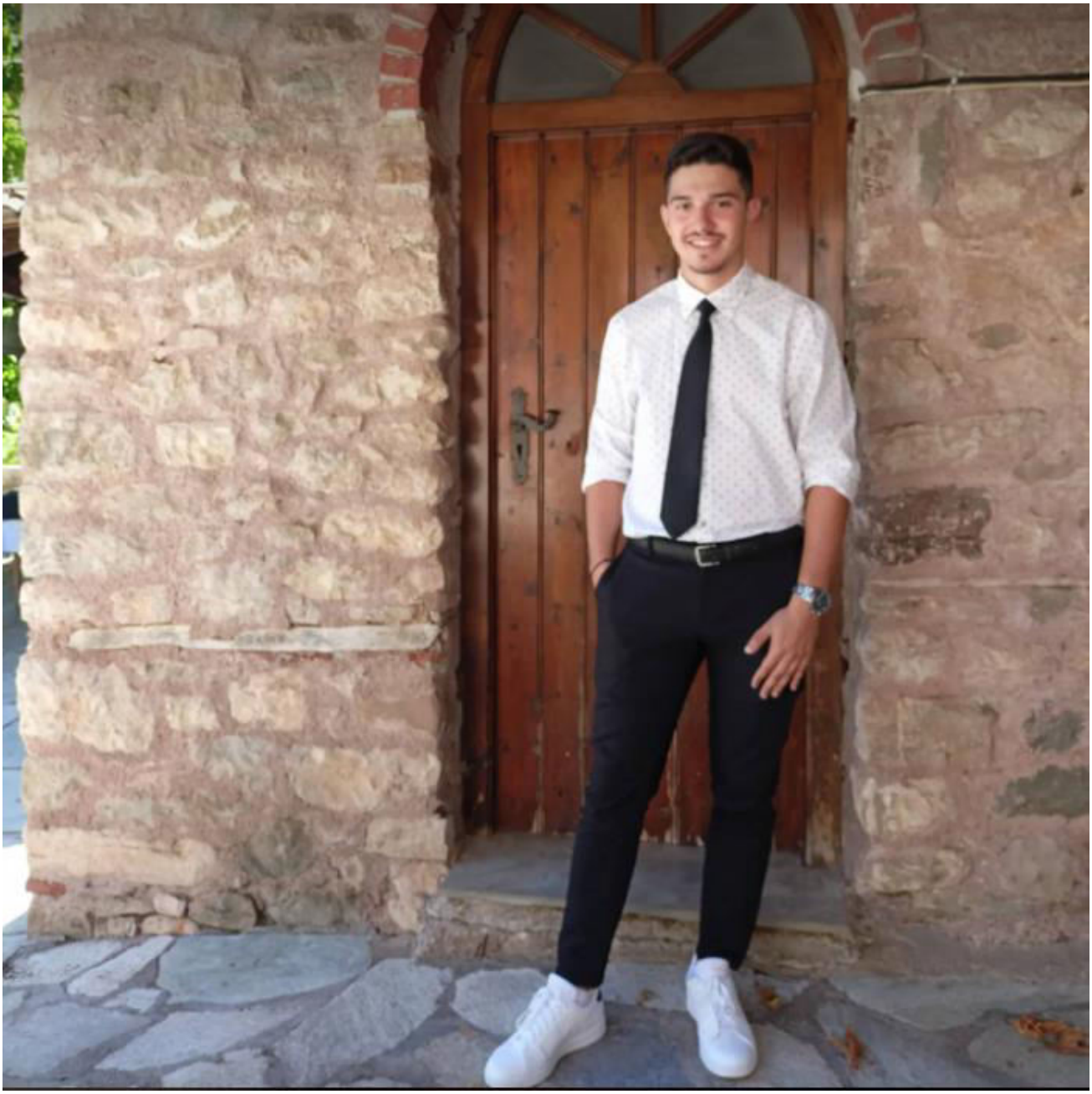
DEPARTMENT OF INFORMATION AND COMPUTER ENGINEERING

APPLICATION OF SEARCH ALGORITHMS TO THE PACMAN PROBLEM

STUDENT DETAILS

NAME: ATHANASIOU VASILEIOS EVANGELOS
STUDENT ID: 19390005
SEMESTER: 7th
STUDENT STATUS : UNDERGRADUATE
STUDY PROGRAM : UNIWA
THEORY INSTRUCTOR: MASTOROKOSTAS PARIS
LAB INSTRUCTOR : TSELENTI PANAGIOTA
DELIVERY DATE : 10/1/2023

STUDENT PHOTO:



1. The world of the problem

The world of pacman includes the following:

Items: Pacman , Fruits, Cells , Cell Grid and Cell Grid Grid

Properties : Pacman can move right, left, up, down or even eat fruit . Fruits coexist in cells as long as they are not in the same cell as pacman .

Relationships: Both fruits and pacman are in cells and each cell belongs to a grid. Every single one mesh belongs to another larger mesh. Pacman can only eat fruit if he is in a cell containing fruit . A fruit can be included in a cell with pacman or stand alone.

1.1 The initial and final state of the problem

Initial state of the problem can be defined as any state in which pacman and fruit are in a cell within the grid that includes grids of cells. For example in a grid with dimensions 4 x 4 initial state can be defined as follows:

Table 1 . 1.1 The initial state

	0	**1**	**2**	**3**
0	[[[' ', 'f']	[' ', 'f']	[' ', ' ']	[' ', 'f']]
1	[[' ', ' ']	[' ', ' ']	[' p' , ' ']	[' ', 'f']]
2	[[' ', ' ']	[' ', 'f']	[' ', 'f']	[' ', ' ']]
3	[[' ', 'f']	[' ', 'f']	[' ', 'f']	[' ', 'f']]]

A final state of the problem can be any state in which pacman has eaten all the fruits, that is, he is alone in a cell of the grid. For example in a grid with dimensions 4 x 4 final state can be defined as follows:

Table 1.1 . 2 The final state

	0	**1**	**2**	**3**
0	[[[' ', ' ']	[' ', ' ']	[' ', ' ']	[' ', ' ']]
1	[[' ', ' ']	[' ', ' ']	[' ', ' ']	[' ', ' ']]
2	[[' ', ' ']	[' ', ' ']	[' ', ' ']	[' ', ' ']]
3	[[' ', ' ']	[' ', ' ']	[' ', ' ']	[' p' , ' ']]]

The world of the problem based on the instance of Table 1 can be represented as follows:

Table 1.1.3 The problem world

Objects	Properties	Relationships
Pacman	It can be moved to the right or left or top or bottom cell.	The pacman is in the cell of the 2nd ^{cell} grid, i.e. at position (1, 2).
Fruit	Can't move unless eaten by pacman .	Ten fruits are free in cells (0, 0), (3, 0), (0, 1), (2, 1), (3, 1), (2, 2), (3, 2), (0 , 3), (1, 3) and (3, 3).
Cell	The cell provides two slots for the fruit and the pacman .	The cell can contain fruit or pacman or fruit together with pacman or nothing.
Grid cells	The cell grid provides four cells.	The cell grid contains the cells.
Grid of grid cells	The cell mesh grid provides four grids that include cells.	The cell grid grid contains the cell grids.

2 . The transition operators

The transition operators are those that allow the use of pacman properties within the grid and that create new states of the problem. In total, in the pacman problem there are 5 operators, that of move right (move _ right), move left (move _ left), move up (move _ up), move down (move _ down) and of eating fruit (eat). Each operator is executed based on some conditions that are presented in detail in the table below. These conditions are checked by the corresponding functions can _ x (where x is the corresponding operator).

Table 2 . 1 The transition operators

Operator	User	Capacity	Condition
move_right	Pacman	Pacman moves to a cell that is to his right.	Pacman should not be in the last cell of a grid of cells .
move_left	Pacman	Pacman moves to a cell that is to his left.	Pacman should not be in the first cell of a grid of cells .
move_up	Pacman	Pacman moves in a cell that is above him.	Pacman should not be in the first grid of cells .
move_down	Pacman	Pacman moves to a cell below him.	Pacman should not be in the last grid of cells .
eat	Pacman	The pacman eats fruit	Pacman should be in the same cell as fruit.

3 . The state space

The state space of the problem is every admissible state created by the intervention of the transition operators. An admissible state is one where at least one cell contains one and only pacman . For example:

Table 3 . 1 The initial state

	0	1	2	3
0	[[[' ', 'f']]	[' ', 'f ']	[' ', ' ']	[' ', 'f']]

1	[[' ', ' ']]	[' ', ' ']	[' p' , ' ']	[' ' , 'f']]
2	[[' ', ' ']]	[' ' , 'f']	[' ' , 'f']	[' ', ' ']]
3	[[' ', 'f']]	[' ' , 'f']	[' ' , 'f']	[' ' , 'f']]]

Table 3.2 The situation after the intervention of the operator move _ left

	0	**1**	**2**	**3**
0	[[[' ', 'f']	[' ' , 'f']	[' ', ' ']	[' ' , 'f']]
1	[[' ', ' ']	[' p ' , ' ']	[' ', ' ']	[' ' , 'f']]
2	[[' ', ' ']	[' ' , 'f']	[' ' , 'f']	[' ', ' ']]
3	[[' ', 'f']	[' ' , 'f']	[' ' , 'f']	[' ' , 'f']]]

Table 3.3 The situation after the intervention of the move _ down operator

	0	**1**	**2**	**3**
0	[[[' ', 'f']	[' ' , 'f']	[' ', ' ']	[' ' , 'f']]
1	[[' ', ' ']	[' ', ' ']	[' ', ' ']	[' ' , 'f']]
2	[[' ', ' ']	[' p' , 'f']	[' ' , 'f']	[' ', ' ']]
3	[[' ', 'f']	[' ' , 'f']	[' ' , 'f']	[' ' , 'f']]]

Table 3.4 The state after the intervention of the eat operator

	0	**1**	**2**	**3**
0	[[[' ', 'f']	[' ' , 'f']	[' ', ' ']	[' ' , 'f']]
1	[[' ', ' ']	[' ', ' ']	[' ', ' ']	[' ' , 'f']]
2	[[' ', ' ']	[' p' , ' ']	[' ' , 'f']	[' ', ' ']]
3	[[' ', 'f']	[' ' , 'f']	[' ' , 'f']	[' ' , 'f']]]

Table 3.5 The situation after the intervention of the move _ right operator

	0	**1**	**2**	**3**
0	[[[' ', 'f']	[' ' , 'f']	[' ', ' ']	[' ' , 'f']]
1	[[' ', ' ']	[' ', ' ']	[' ', ' ']	[' ' , 'f']]
2	[[' ', ' ']	[' ', ' ']	[' p' , 'f']	[' ', ' ']]
3	[[' ', 'f']	[' ' , 'f']	[' ' , 'f']	[' ' , 'f']]]

Table 3.6 The state after the intervention of the eat operator

	0	**1**	**2**	**3**
0	[[[' ', 'f']	[' ' , 'f']	[' ', ' ']	[' ' , 'f']]
1	[[' ', ' ']	[' ', ' ']	[' ', ' ']	[' ' , 'f']]
2	[[' ', ' ']	[' ', ' ']	[' p' , ' ']	[' ', ' ']]
3	[[' ', 'f']	[' ' , 'f']	[' ' , 'f']	[' ' , 'f']]]

Table 3.7 The situation after the intervention of the move _ up operator

	0	1	2	3
0	[[[' ', 'f']	[' ' , 'f']	[' ', ' ']	[' ' , 'f']]
1	[[' ', ' ']	[' ', ' ']	[' p' , ' ']	[' ' , 'f']]
2	[[' ', ' ']	[' ', ' ']	[' ', ' ']	[' ', ' ']]
3	[[' ', 'f']	[' ' , 'f']	[' ' , 'f']	[' ' , 'f']]]

x 4 list with four lists of cells. Each cell list contains four cell lists each cell list contains two strings, " p " or " f " or "" "", where " p " is pacman , " f " is fruit, and "" "" is blank. The cell list cannot contain two ' p 's, like all cell list lists.

find _ children function in Python

4 . 1 The world of the problem

The world of the problem in Python is represented by a list containing four lists (arrays of cells), where each list contains four list-cells which in turn contain two strings, " p " or " f " or " " " , where " p " is pacman , " f " is fruit, and """ is space. The initial state of table 1.1.1 is stored in the init_state variable . The final state of table 2.2 is stored in the variable goal_state . The states are defined in a main function " main " which is called first by the program.

Initial and final state

```
def main ():
   # Initial and final state of the problem in 4x4 dimensions (DFS is preferred for
main search algorithm)

   init_state = [[[ ' ' , 'f' ], [ ' ' , 'f' ], [ ' ' , ' ' ], [ ' ' , 'f' ]],
[[ ' ' , ' ' ], [ ' ' , ' ' ], [ 'p' , ' ' ], [ ' ' , 'f' ]],
[[ ' ' , ' ' ], [ ' ' , 'f' ], [ ' ' , 'f' ], [ ' ' , ' ' ]],
[[ ' ' , 'f' ], [ ' ' , 'f' ], [ ' ' , 'f' ], [ ' ' , 'f' ]]]

   goal_state = [[[ ' ' , ' ' ], [ ' ' , ' ' ], [ ' ' , ' ' ], [ ' ' , ' ' ]],
[[ ' ' , ' ' ], [ ' ' , ' ' ], [ ' ' , ' ' ], [ ' ' , ' ' ]],
[[ ' ' , ' ' ], [ ' ' , ' ' ], [ ' ' , ' ' ], [ ' ' , ' ' ]],
[[ ' ' , ' ' ], [ ' ' , ' ' ], [ ' ' , ' ' ], [ 'p' , ' ' ]]]
```

A search algorithm is then selected by the user, that of Depth Traversal (DFS) or that of Breadth Traversal (BFS).

Selection search algorithm

```
   method = 'Blind searching algorithm'
   # Select one of two search algorithms DFS or BFS for the main search algorithm
solving the problem with pacman
   while method ! = 'DFS' and method != 'BFS' :
      method = input ( 'Choose between DFS and BFS as main blind searching algorithm
: ' )
```

find _ solution is called which stores the initial state in a list for the search front (calling make _ front) and a list of lists for the search queue of the default search algorithm (calling make _ queue). Additionally, it takes as arguments an empty set which is the closed set (closed) with the visited state-nodes, the final state (goal_state) and the default search algorithm (method).

Call find_solution

```
# Call find_solution to find a path that will lead from the default initial state to
the final state based on a default search algorithm
   print ( ' ' )
   print ( 'Begin Searching...' )
   print ( 'Searching algorithm : ' , method )
   print ( ' ' )
   find_solution ( make_front ( init_state ), make_queue ( init_state ), [],
goal_state , method )
```

Code of the make _ front and make _ queue functions

```
""" Initialize search front """
def make_front ( state ):
   return [ state ]

""" Initialize Search Queue """
def make_queue ( state ):
    return [[ state ]]
```

H find _ solution is called recursively until a path is found through the transition operators leading from the initial state to the final state. If the last node of a path is the final state, that is, if the first node-state of the search front is the final state, then the final path and the goal are found (calling is _ goal _ state).

Code of the function find _ solution

```
""" Function to find path leading from initial state to final state via DFS or BFS
search algorithms """
def find_solution ( front , queue , closed , goal , method ) :
   if not front :
      print ( 'No solution found! End of searching...' )
   elif front [ 0 ] in closed :
        new_front = copy . deepcopy ( front )
        new_front . pop ( 0 )
        new_queue = copy . deepcopy ( queue )
        new_queue . pop ( 0 )
        find_solution ( new_front , new_queue , closed , goal , method )
   elif is_goal_state ( front [ 0 ]):
      print ( 'Goal found!' )
      for i in range ( len ( queue [ 0 ])):
            for j in range ( len ( queue [ 0 ][ i ])):
```

```
                print ( queue [ 0 ][ i ][ j ]) # Node-states in the final path are
plotted two-dimensionally and one below the other
            print ( ' ' )
   else :
      closed . append ( front [ 0 ])
      front_copy = copy . deepcopy ( front )
      front_children = expand_front ( front_copy , method )
      queue_copy = copy . deepcopy ( queue )
      queue_children = extend_queue ( queue_copy , method )
      closed_copy = copy . deepcopy ( closed )
      find_solution ( front_children , queue_children , closed_copy , goal , method
)
```

Code of the is _ goal _ state function

```
""" Check to find end state """
def is_goal_state ( state ):
   for row in range ( len ( state )):
      for coll in range ( len ( state [ row ])):
         if state [ row ][ col ][ 1 ] == 'f' :
            return 0
   return 1
```

The path is found by calling expand _ front and extend _ queue , where depending on the default search algorithm the search front and search queue are expanded. More specifically, in Depth Traversal (DFS) the visited state-node is removed from the search front and added to the closed set (closed) and replaced by its children-states at the beginning of the front (LIFO), while in Crossing by Width children enter at the end of the front (FIFO).
Similarly, in Depth Traversal (DFS) the visited path is removed from the search queue and the terminal node-state is added to the closed set (closed) and the paths with terminal nodes the child states of the node that removed at the beginning of the queue (LIFO), while in Width Crossing the children are entered at the end of the queue (FIFO).

Code of the expand _ front function

```
""" Expand the search front """
def expand_front ( front , method ):
   if method == 'DFS' : # Extend the search front of the DFS search algorithm
      if front :
         print ( "Front : " )
         for i in range ( len ( front )):
            for j in range ( len ( front [ i ])):
               print ( front [ i ][ j ]) # Node-states on the search front are
displayed 2D and one below the other
            print ( ' ' )
         print ( '_____________________________________' )
         node = front . pop ( 0 ) # Remove the first state-node from the search
front
```

```
        for child in find_children ( node ): # Find states-children of the node
removed from the front, via the function find_children
            front . insert ( 0 , child ) # The child states of the removed node are
placed in a stack structure (LIFO) at the beginning of the forward search
   elif method == 'BFS' : # Extend the search front of the BFS search algorithm
      if front :
         print ( " Front : " )
         for i in range ( len ( front )):
            for j in range ( len ( front [ i ])):
               print ( front [ i ][ j ]) # Node-states on the search front are
displayed 2D and one below the other
            print ( ' ' )
         print ( '_____________________________________' )
         node = front . pop ( 0 ) # Remove the first state-node from the search
front
         for child in find_children ( node ): # Find states-children of the node
removed from the front, via the function find_children
            front . append ( child ) # The child states of the removed node are
placed in a queue structure (FIFO) at the end of the search front

   return front
```

Code of the function extend _ queue

```
""" Extension her caudal search """
def extend_queue ( queue , method ) :
# Extend the search queue of the DFS search algorithm
   if method == 'DFS' :
      print ( " Queue : " )
      for i in range ( len ( queue )):
         for j in range ( len ( queue [ i ])):
            for w in range ( len ( queue [ i ][ j ])):
               print ( queue [ i ][ j ][ w ]) # Node-states in the search queue are
displayed two-dimensionally and one below the other
            print ( ' ' ) # A path is a sequence of nodes one below the other and
ends at the node shown before the "End of path" message
         print ( '----------- End of path ------------' )
         print ( ' ' )
      print ( '_____________________________________' )
      node = queue . pop ( 0 ) # Remove the first path from the search queue
      queue_copy = copy . deepcopy ( queue )
      children = find_children ( node [ - 1 ]) # Find states-children of the node
that was at the end of the dequeued path, via the find_children call
      for child in children :
         path = copy . deep copy ( node )
         path . append ( child ) # Child states are stored in the path list by
creating a new path terminating a child node of the last node that was on the
dequeued path
```

```
        queue_copy . insert ( 0 , path ) # The path is placed in a stack structure
(LIFO), at the beginning of the search queue
# Extend the search queue of the BFS search algorithm
   elif method == 'BFS' :
      print ( " Queue : " )
      for i in range ( len ( queue )):
         for j in range ( len ( queue [ i ])):
            for w in range ( len ( queue [ i ][ j ])):
               print ( queue [ i ][ j ][ w ]) # Node-states in the search queue are
displayed two-dimensionally and one below the other
            print ( ' ' )
         print ( '----------- End of path ------------' )
         print ( ' ' )
      print ( '_____________________________________' )
      node = queue . pop ( 0 ) # Remove the first path from the search queue
      queue_copy = copy . deepcopy ( queue )
      children = find_children ( node [ - 1 ]) # Find states-children of the node
that was at the end of the dequeued path, via the find_children call
      for child in children :
         path = copy . deep copy ( node )
         path . append ( child ) # Child states are stored in the path list by
creating a new path terminating a child node of the last node that was on the
dequeued path
         queue_copy . append ( path ) # The path is placed in a queue structure
(FIFO), at the end of the search queue

   return queue_copy
```

4 . 2 The transition operators

Each transition operator in order to be used by pacman should meet some conditions. These conditions are checked by the functions can _ x (where x is the transition operator). The function and conditions of use of each operator are analyzed in table 2.1 The functions of the operators return the states after their intervention or the same state that they accepted as an argument, as the prescribed condition is not met.

Code of the can _ eat function

```
""" Checking usage of the eat fruit operator """
def can_eat ( state ):
   for row in range ( len ( state )):
      for coll in range ( len ( state [ row ])):
         if state [ row ][ col ][ 0 ] == 'p' and state [ row ][ col ][ 1 ] == 'f' :
            return 1
   return 0
```

Code of the function can _ move _ right

```
""" Check usage of shift right operator """
def can_move_right ( state ):
   for row in range ( len ( state )):
      if state [ row ][ len ( state [ row ]) - 1 ][ 0 ] == 'p' :
         return 0
   return 1
```

Code of the function can _ move _ left

```
""" Check usage of shift left operator """
def can_move_left ( state ):
   for row in range ( len ( state )):
      if state [ row ][ 0 ][ 0 ] == 'p' :
         return 0
   return 1
```

Code of the can _ move _ up function

```
""" Check usage of move up operator """
def can_move_up ( state ):
   for coll in range ( len ( state [ 0 ])):
      if state [ 0 ][ col ][ 0 ] == 'p' :
         return 0
   return 1
```

Code of the can _ move _ down function

```
""" Check usage of move down operator """
def can_move_down ( state ):
   for coll in range ( len ( state [ len ( state ) - 1 ])):
      if state [ len ( state ) - 1 ][ col ][ 0 ] == 'p' :
         return 0
   return 1
```

Code of the eat function

```
""" Fruit Eater """
def eat ( state ):
   if can_eat ( state ):
      for row in range ( len ( state )):
         for coll in range ( len ( state [ row ])):
            if state [ row ][ col ][ 0 ] == 'p' and state [ row ][ col ][ 1 ] == 'f'
:
               state [ row ][ col ][ 1 ] = ' '
               return state
   else :
      return state
```

Code of the move _ right function

```
""" Right shift operator """
def move_right ( state ):
   if can_move_right ( state ):
      for row in range ( len ( state )):
         for coll in range ( len ( state [ row ])):
            if state [ row ][ col ][ 0 ] == 'p' :
               state [ row ][ col ][ 0 ] = ' '
               state [ row ][ col + 1 ][ 0 ] = 'p'
               return state
   else :
      return state
```

Code of the function move _ left

```
""" Move left operator """
def move_left ( state ):
   if can_move_left ( state ):
      for row in range ( len ( state )):
         for coll in range ( len ( state [ row ])):
            if state [ row ][ col ][ 0 ] == 'p' :
               state [ row ][ col ][ 0 ] = ' '
               state [ row ][ col - 1 ][ 0 ] = 'p'
               return state
   else :
      return state
```

Code of the move _ up function

```
""" move up operator """
def move_up ( state ):
   if can_move_up ( state ):
      for row in range ( len ( state )):
         for coll in range ( len ( state [ row ])):
            if state [ row ][ col ][ 0 ] == 'p' :
               state [ row ][ col ][ 0 ] = ' '
               state [ row - 1 ][ col ][ 0 ] = 'p'
               return state
   else :
      return state
```

Code of the move _ down function

```
""" Move down operator """
def move_down ( state ):
   if can_move_down ( state ):
      for row in range ( len ( state )):
         for coll in range ( len ( state [ row ])):
```

```
            if state [ row ][ col ][ 0 ] == 'p' :
               state [ row ][ col ][ 0 ] = ' '
               state [ row + 1 ][ col ][ 0 ] = 'p'
               return state
   else :
      return state
```

find _ children function

find _ children function stores in a children list the child states created by the five aforementioned operators if the conditions are met. Calls are made to the transition operators that return the states they created by their intervention or the same state due to conditions not being met.

Code of the function find _ children

```
""" Function to find descendants of a state-node """
def find_children ( state ):
   children = []
   right_state = copy . deepcopy ( state )
   child_right = move_right ( right_state ) # Find descendant-state with which
pacman can move right
   left_state = copy . deepcopy ( state )
   child_left = move_left ( left_state ) # Find descendant-state with which pacman
can move left
   up_state = copy . deepcopy ( state )
   child_up = move_up ( up_state ) # Find descendant-state to which pacman can move
up
   down_state = copy . deepcopy ( state )
   child_down = move_down ( down_state ) # Find descendant-state to which pacman can
move down
   eat_state = copy . deepcopy ( state )
   child_eat = eat ( eat_state ) # Find offspring-state with which pacman can eat
fruit

   if not child_right == None : # Check for any found descendant-condition by which
pacman can move to the right
      children . append ( child_right )
   if not child_left == None : # Check for any found descendant-condition by which
pacman can move left
      children . append ( child_left )
   if not child_up == None : # Check for any found descendant-status with which
pacman can move up
      children . append ( child_up )
   if not child_down == None : # Check for any found descendant-status with which
pacman can scroll down
      children . append ( child_down )
   if not child_eat == None : # Check for any offspring-condition found by which
pacman can eat fruit
```

```
children . append ( child_eat )
```

BFS Width Traversal search algorithm

5.1 1st Example

Initial state of the problem can be defined as any state in which pacman and fruit are in a cell within the grid that includes grids of cells. For example in a grid with dimensions 2 x 2 initial state can be defined as follows:

Table 5. 1.1 The initial state

	0	**1**
0	[[[' ', 'f']	[' p' , ' ']
1	[[' ', 'f']	[' ', ' ']

A final state of the problem can be any state in which pacman has eaten all the fruits, that is, he is alone in a cell of the grid. For example in a grid with dimensions 2 x 2 final state can be defined as follows:

Table 5.1 . 2 The final state

	0	**1**
0	[[[' ', ' ']	[' ', ' ']
1	[['p', ' ']	[' ', ' ']

BFS search algorithm results for the 1st [example]

Choose between DFS and BFS as main blind searching algorithm :

Start Searching...

Searching algorithm: BFS

Front:

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

_____

queue :

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

_____

Front:

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

_____

queue :

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

---------- End of path ----------

Front:

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

---

queue :

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

_____

Front:

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

_____

queue :

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

_____

Front:

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

_____

queue :

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

---------- End of path ----------

_____

Front:

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

_____

queue :

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

---------- End of path ----------

_____

Front:

[[' ', ' '], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

_____

queue :

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[[' ', ' '], ['p', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', ' '], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', ' '], [' ', ' ']]

---------- End of path ----------

_____

Front:

[[' ', ' '], [' ', ' ']]
[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

_____

queue :

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[[' ', ' '], ['p', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', ' '], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', ' '], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', ' '], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]
[[' ', 'f'], ['p', ' ']]

---------- End of path ----------

Front:

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', ' '], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', ' ']]

---

queue :

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[[' ', ' '], ['p', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', ' '], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]
[['p', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]
[['p', ' '], [' ', ' ']]

---------- End of path ----------

_____

Front:

[['p', 'f'], [' ', ' ']]
[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]
[[' ', 'f'], ['p', ' ']]

[[' ', ' '], [' ', ' ']]
[[' ', 'f'], ['p', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', ' '], [' ', ' ']]

_____

queue :

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]
[['p', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]
[[' ', 'f'], ['p', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]
[['p', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]
[['p', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]
[['p', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[[' ', ' '], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', ' '], [' ', ' ']]

---------- End of path ----------

_____

Front:

[[' ', ' '], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', ' '], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]
[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', ' '], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]
[[' ', ' '], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]
[[' ', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', ' '], [' ', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', ' '], [' ', ' ']]

_____

queue :

[[' ', 'f'], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]
[[' ', 'f'], ['p', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]
[['p', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]
[[' ', 'f'], ['p', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]
[['p', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]
[['p', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]
[['p', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[[' ', 'f'], ['p', ' ']]

[[' ', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', ' '], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', 'f'], ['p', ' ']]
[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]
[['p', ' '], [' ', ' ']]

[['p', 'f'], [' ', ' ']]
[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', ' ']]
[[' ', ' '], [' ', ' ']]

---------- End of path ----------

_____

Goal found!

[[' ', 'f'], ['p', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[['p', 'f'], [' ', ' ']]

[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', ' ']]

5. 2 2nd Example

Initial state of the problem can be defined as any state in which pacman and fruit are in a cell within the grid that includes grids of cells. For example in a grid with dimensions 2 x 2 initial state can be defined as follows:

Table 5. 2.1 The initial state

	0	**1**
0	[[[' ', ' ']	[' ', 'f']
1	[['p', ' ']	[' ' , ' f ']

A final state of the problem can be any state in which pacman has eaten all the fruits, that is, he is alone in a cell of the grid. For example in a grid with dimensions 2 x 2 final state can be defined as follows:

Table 5.2 . 2 The final state

	0	**1**
0	[[[' ', ' ']	[' p' , ' ']
1	[[' ', ' ']	[' ', ' ']

BFS search algorithm results for the 2nd example

Choose between DFS and BFS as main blind searching algorithm :

Start Searching...

Searching algorithm: BFS

Front:

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

_____

queue :

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

---------- End of path ----------

_____

Front:

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

_____

queue :

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

---------- End of path ----------

_____

Front:

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

_____

queue :

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

---------- End of path ----------

_____

Front:

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

_____

queue :

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]
[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]
[[' ', ' '], ['p', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]
[['p', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]
[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]
[[' ', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]
[['p', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]
[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]
[['p', ' '], [' ', 'f']]

---------- End of path ----------

Front:

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

---

queue :

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

---------- End of path ----------

_____

Front:

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', ' ']]

_____

queue :

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]
[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]
[['p', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]
[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]
[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]
[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', 'f']]
[[' ', ' '], [' ', ' ']]

---------- End of path ----------

_____

Front:

[[' ', ' '], [' ', 'f']]
[['p', ' '], [' ', ' ']]

[[' ', ' '], ['p', 'f']]
[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', 'f']]

_____

queue :

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', 'f']]

---------- End of path ----------

_____

Front:

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

_____

queue :

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', ' ']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

---------- End of path ----------

_____

Front:

[['p', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', ' ']]

_____

queue :

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', ' ']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', ' ']]

---------- End of path ----------

_____

Front:

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', 'f']]

_____

queue :

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]
[['p', ' '], [' ', ' ']]

[['p', ' '], [' ', 'f']]
[[' ', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]
[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]
[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]
[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', 'f']]
[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', 'f']]
[[' ', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]
[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]
[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', 'f']]

---------- End of path ----------

_____

Front:

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

_____

queue :

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', ' ']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', ' ']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]
[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]
[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', ' ']]
[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]
[['p', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]
[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]
[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]
[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]
[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]
[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', ' ']]

[['p', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]
[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]
[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]
[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]
[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]
[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', ' ']]
[[' ', ' '], [' ', 'f']]

---------- End of path ----------

[[' ', ' '], [' ', 'f']]
[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]
[[' ', ' '], ['p', 'f']]

[[' ', ' '], ['p', 'f']]
[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

---------- End of path ----------

_____

Goal found!

[[' ', ' '], [' ', 'f']]

[['p', ' '], [' ', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', 'f']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], ['p', 'f']]

[[' ', ' '], [' ', ' ']]

[[' ', ' '], ['p', ' ']]

[[' ', ' '], [' ', ' ']]

Thank you for your attention.