LP2

May 28, 2023

[1]:
```
#bfs
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F', 'G'],
    'D': ['B'],
    'E': ['B'],
    'F': ['C'],
    'G': ['C']
}

def bfs(graph, initial):
    visited = []
    queue = [initial]
    while queue:
        node = queue.pop(0)
        if node not in visited:
            visited.append(node)
            neighbours = graph[node]
            for neighbour in neighbours:
                queue.append(neighbour)
    return visited

print(bfs(graph,'A'))
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

[2]:
```
#dfs
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F', 'G'],
    'D': ['B'],
    'E': ['B'],
    'F': ['C'],
    'G': ['C']
}
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start)
    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)
    return visited

dfs(graph, 'A')
```

```
A
B
D
E
C
F
G
```

[2]:
```
{'A', 'B', 'C', 'D', 'E', 'F', 'G'}
```

[3]:
```
# A* code
from queue import PriorityQueue

def word_ladder(start_word, target_word, word_list):
    def heuristic(word1, word2):
        # Calculate the number of differing letters between word1 and word2
        return sum(l1 != l2 for l1, l2 in zip(word1, word2))

    open_set = PriorityQueue()
    open_set.put((0, start_word))
    came_from = {}
    cost = {start_word: 0}

    while not open_set.empty():
        _, current_word = open_set.get()

        if current_word == target_word:
            # Reconstruct the path from the target word to the start word
            path = []
            while current_word in came_from:
                path.append(current_word)
                current_word = came_from[current_word]
            path.append(start_word)
            path.reverse()
            return path
        for next_word in word_list:
            if next_word != current_word and heuristic(next_word, current_word)␣
↪== 1:
                tentative_cost = cost[current_word] + 1

                if next_word not in cost or tentative_cost < cost[next_word]:
                    came_from[next_word] = current_word
                    cost[next_word] = tentative_cost
                    priority = tentative_cost + heuristic(next_word,␣
↪target_word)
                    open_set.put((priority, next_word))

    # No path found
    return None

# Example usage
start_word = "hit"
target_word = "cog"
word_list = ["hot", "dot", "dog", "lot", "log", "cog"]
path = word_ladder(start_word, target_word, word_list)
if path is not None:
    print("Path found:", path)
else:
    print("No path found")
```

```
Path found: ['hit', 'hot', 'dot', 'dog', 'cog']
```

[4]:
```
#prims algorithm
INF = 9999999
V= 5
G = [[0, 9, 75, 0, 0],
[9, 0, 95, 19, 42],
[75, 95, 0, 51, 66],
[0, 19, 51, 0, 31],
[0, 42, 66, 31, 0]]

selected = [0, 0, 0, 0, 0]
no_edge = 0
selected[0] = True

print("Edge : Weight\n")
while (no_edge < V - 1):
    minimum = INF
    x= 0
    y= 0
    for i in range(V):
        if selected[i]:
            for j in range(V):
                if ((not selected[j]) and G[i][j]):

        # not in selected and there is an edge
                    if minimum > G[i][j]:
                        minimum = G[i][j]
                        x= i
                        y= j
    print(str(x) + "-" + str(y) + ":" + str(G[x][y]))
    selected[y] = True
    no_edge += 1
```

```
Edge : Weight

0-1:9
1-3:19
3-4:31
3-2:51
```

[5]:
```
# N-Queen
# Backtracking
def backtracking_n_queen(n):
    def is_safe(board, row, col):
        return all(board[i] != row and board[i] != row + col - i and board[i] !
 ↪= row - col + i for i in range(col))

    def solve_util(board, col):
        if col >= n:
            return True
        for row in range(n):
            if is_safe(board, row, col):
                board[col] = row
                if solve_util(board, col + 1):
                    return True
        return False
    board = [-1] * n
    if solve_util(board, 0):
        for row in board:
            print(' '.join(['Q' if i == row else '-' for i in range(n)]))
    else:
        print("No solution Exists")
n = 8
backtracking_n_queen(n)
```

```
Q - - - - - - -
- - - - Q - - -
- - - - - - - Q
- - - - - Q - -
- - Q - - - - -
- - - - - - Q -
- Q - - - - - -
- - - Q - - - -
```

[6]:
```
#Branch and Bounds
def branch_n_queen(n):

    def is_safe(board, row, col):
        for i in range(col):
            if(board[i] == row or board[i] == row+col-1 or board[i] ==␣
↪row-col+i):
                return False
        return True

    def solve_util(board, col):
        if col >= n:
            return True
        for row in range(n):
            if is_safe(board, row, col):
                board[col] = row
                if solve_util(board, col+1):
                    return True
        return False

    board = [-1] * n
    if solve_util(board, 0):
        for row in board:
            print(' '.join(['Q' if i == row else '-' for i in range(n)]))
    else:
        print("No Solution Exists")

n = 8
branch_n_queen(n)
```

```
Q - - - - - - -
- - - Q - - - -
- - - - - Q - -
- - - - - - - Q
- Q - - - - - -
- - - - - - Q -
- - - - Q - - -
- - Q - - - - -
```

[ ]:
```
#chatbot
def greet(bot_name, birth_year):
    print("Hello! My name is {0}.".format(bot_name))
    print("I was created in {0}.".format(birth_year))

def remind_name():
    print('\nPlease, remind me your name.')
    name = input()
    print("What a great name you have, {0}!".format(name))

def guess_age():
    print('\nLet me guess your age.')
    print('Enter remainders of dividing your age by 3, 5 and 7.')

    rem3 = int(input())
    rem5 = int(input())
    rem7 = int(input())
    age = (rem3 * 70 + rem5 * 21 + rem7 * 15) % 105

    print("Your age is {0}; that's a good time to start programming!".
 ↪format(age))

def number_guess():
    import random
    import math

    print("\nHey! Here's a number guessing game for you!")
    lower = int(input("\nEnter Lower bound:- "))
    upper = int(input("Enter Upper bound:- "))

    x = random.randint(lower, upper)
    print("\n\tYou've only ",
          round(math.log(upper - lower + 1, 2)),
          " chances to guess the integer!\n")

    count = 0
    while count < math.log(upper - lower + 1, 2):
        count += 1

        guess = int(input("Guess a number:- "))

        if x == guess:
            print("Congratulations you did it in ",
                  count, " try")
            break
        elif x > guess:
            print("You guessed too small!")
        elif x < guess:
            print("You Guessed too high!")
    if count >= math.log(upper - lower + 1, 2):
        print("\nThe number is %d" % x)
        print("\tBetter Luck Next time!")


def count():
    print('\nNow I will prove to you that I can count to any number you want.')
    num = int(input())

    counter = 0
    while counter <= num:
        print("{0} !".format(counter))
        counter += 1


def test():
    print("\nLet's test your programming knowledge.")
    print("Why do we use methods?")
    print("1. To repeat a statement multiple times.")
    print("2. To decompose a program into several small subroutines.")
    print("3. To determine the execution time of a program.")
    print("4. To interrupt the execution of a program.")

    answer = 2
    guess = int(input())
    while guess != answer:
        print("Please, try again.")
        guess = int(input())

    print('Completed, have a nice day!')
    print('.................................')
    print('.................................')
    print('.................................')


def end():
    print('Congratulations, have a nice day!')
    print('.................................')
    print('.................................')
    print('.................................')
    input()

greet('Shreyas Peherkar','2002')
remind_name()
guess_age()
number_guess()
```

```
count()
test()
end()
```

```
Hello! My name is Shreyas Peherkar.
I was created in 2002.

Please, remind me your name.
Raje
What a great name you have, Raje!

Let me guess your age.
Enter remainders of dividing your age by 3, 5 and 7.
0
1
0
Your age is 21; that's a good time to start programming!

Hey! Here's a number guessing game for you!

Enter Lower bound:- 1
Enter Upper bound:- 10

        You've only  3  chances to guess the integer!

Guess a number:- 6
You guessed too small!
```