K. J. Somaiya College of Engineering, Mumbai-77

Batch: A2	**Roll No.: 1911027**
Experiment / assignment / tutorial No. 5	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Implementation of uninformed search algorithm(BFS/DFS/DLS/IDS)

Objective: Comparison and analysis of uninformed search algorithms

Expected Outcome of Experiment:

Course Outcome	After successful completion of the course students should be able to
CO 2	Analyse and solve problems for goal based agent architecture (searching and planning algorithms).

Books/ Journals/ Websites referred:

1. **"Artificial Intelligence: a Modern Approach" by Russell and Norving, Pearson education Publications**
2. **"Artificial Intelligence" By Rich and knight, Tata Mcgraw Hill Publications**
3. **http://people.cs.pitt.edu/~milos/courses/cs2710/lectures/Class4.pdf**
4. **http://cs.williams.edu/~andrea/cs108/Lectures/InfSearch/infSearch.html**
5. **http://www.cs.mcgill.ca/~dprecup/courses/AI/Lectures/ai-lecture02.pdf**
   http://homepage.cs.uiowa.edu/~hzhang/c145/notes/04a-search.pdf
6. **http://wiki.answers.com/Q/Informed_search_techniques_and_uninformed_search_techniques**
7. **www.cs.swarthmore.edu/~eeaton/teaching/cs63/.../**UninformedSearch**.ppt**

Pre Lab/ Prior Concepts: Problem solving, state-space trees, problem formulation, goal based agent architecture

---

Historical Profile:

The AI researchers have come up many algorithms those operate on state space tree to give the result. Goal based agent architectures solve problems through searching or planning. Depending on availability of more information other than the problem statement decides if the solution can be obtained with uninformed search or informed search.

Its fact that not all search algorithms end up in giving the optimal solution. So, it states the need to have a better and methodological approach which guarantees optimal solution.

New Concepts to be learned: Uninformed (blind) search, iterative deepening, greedy best first search, A* search

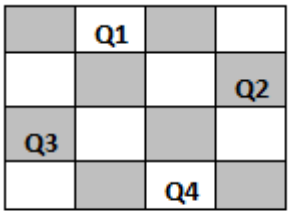
---

Uninformed searching techniques:

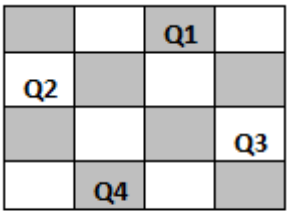
- Breadth first search
- Depth first search
- Iterative deepening search
- Depth limit search

Chosen Problem statement:

4 Queens problem: The 4-Queens consists in placing four queens on a 4 x 4 chessboard so that no two queens can capture each other. That is, no two queens are allowed to be placed on the same row, the same column or the same diagonal.

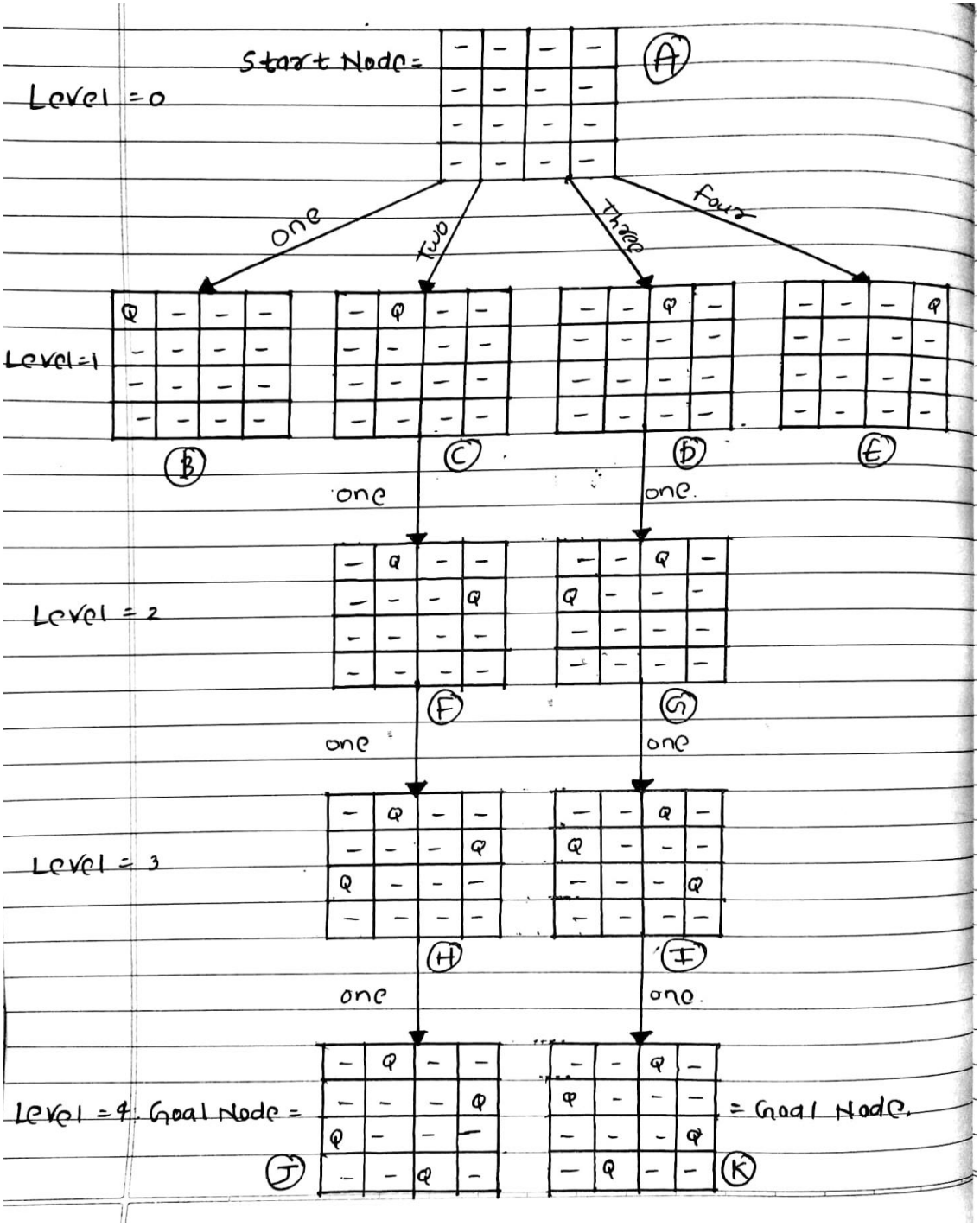


Solution 1



Solution 2

State-space tree:



Solution with of chosen algorithm on the state-space tree:

Code:

```
class Tree:
    def __init__(self,val,name,one=None,two=None,three=None,four=None):
```

```
        self.val=val
        self.one=one
        self.two=two
        self.three=three
        self.four=four
        self.name=name

def bfs(root):
    queue=[root]
    bfs_order=[]
    while(len(queue)!=0):
        visited=queue.pop(0)
        bfs_order.append(visited)
        if(visited.one!=None):
            queue.append(visited.one)
        if(visited.two!=None):
            queue.append(visited.two)
        if(visited.three!=None):
            queue.append(visited.three)
        if(visited.four!=None):
            queue.append(visited.four)
    return bfs_order

def dfs(root):
    stack=[root]
    dfs_order=[]
    while(len(stack)!=0):
        visited=stack.pop(len(stack)-1)
        dfs_order.append(visited)
        if(visited.four!=None):
            stack.append(visited.four)
        if(visited.three!=None):
            stack.append(visited.three)
        if(visited.two!=None):
            stack.append(visited.two)
        if(visited.one!=None):
            stack.append(visited.one)
    return dfs_order

def dls(root,depth):
    stack=[[root,0]]
```

```
    dls_order=[]
    while(len(stack)!=0):
        visited=stack.pop(len(stack)-1)
        dls_order.append(visited[0])
        depth_temp=visited[1]+1
        if(depth_temp<=depth):
            if(visited[0].four!=None):
                stack.append([visited[0].four,depth_temp])
            if(visited[0].three!=None):
                stack.append([visited[0].three,depth_temp])
            if(visited[0].two!=None):
                stack.append([visited[0].two,depth_temp])
            if(visited[0].one!=None):
                stack.append([visited[0].one,depth_temp])
    return dls_order

root=Tree([["-","-","-","-"],
           ["-","-","-","-"],
           ["-","-","-","-"],
           ["-","-","-","-"]], "A")

root.one=Tree([["Q","-","-","-"],
               ["-","-","-","-"],
               ["-","-","-","-"],
               ["-","-","-","-"]], "B")

root.two=Tree([["-","Q","-","-"],
               ["-","-","-","-"],
               ["-","-","-","-"],
               ["-","-","-","-"]], "C")

root.three=Tree([["-","-","Q","-"],
                 ["-","-","-","-"],
                 ["-","-","-","-"],
                 ["-","-","-","-"]], "D")

root.four=Tree([["-","-","-","Q"],
                ["-","-","-","-"],
                ["-","-","-","-"],
                ["-","-","-","-"]], "E")
```

```
root.two.one=Tree([["-","Q","-","-"],
                   ["-","-","-","Q"],
                   ["-","-","-","-"],
                   ["-","-","-","-"]], "F")

root.three.one=Tree([["-","-","Q","-"],
                     ["Q","-","-","-"],
                     ["-","-","-","-"],
                     ["-","-","-","-"]], "G")

root.two.one.one=Tree([["-","Q","-","-"],
                       ["-","-","-","Q"],
                       ["Q","-","-","-"],
                       ["-","-","-","-"]], "H")

root.three.one.one=Tree([["-","-","Q","-"],
                         ["Q","-","-","-"],
                         ["-","-","-","Q"],
                         ["-","-","-","-"]], "I")

root.two.one.one.one=Tree([["-","Q","-","-"],
                           ["-","-","-","Q"],
                           ["Q","-","-","-"],
                           ["-","-","Q","-"]], "J")

root.three.one.one.one=Tree([["-","-","Q","-"],
                             ["Q","-","-","-"],
                             ["-","-","-","Q"],
                             ["-","Q","-","-"]], "K")

goal_node=[Tree([["-","Q","-","-"],
                 ["-","-","-","Q"],
                 ["Q","-","-","-"],
                 ["-","-","Q","-"]], "J"),
           Tree([["-","-","Q","-"],
                 ["Q","-","-","-"],
                 ["-","-","-","Q"],
                 ["-","Q","-","-"]], "K")]

print("-----------------------------------------------------------------------------------------------
-------------------------------------")
```

```
print("------------------------------------------------------ UNINFORMED SEARCH ------------------------------------------------------")
print("-----------------------------------------------------------------------------------------------------------------------------")
print("-----------------------------------------------------------------------------------------------------------------------------")
print("Breadth first search traversal: ")
print("-----------------------------------------------------------------------------------------------------------------------------\n")
bfs_order=bfs(root)
first=[]
second=[]
third=[]
fourth=[]
for i in bfs_order:
    first.append(i.val[0])
    second.append(i.val[1])
    third.append(i.val[2])
    fourth.append(i.val[3])
i=0
while(i<len(first)):
    count=0
    j=i
    while(j<len(first) and count<5):
        print(first[j],end="  ")
        j=j+1
        count=count+1
    print()
    count=0
    k=i
    while(k<len(second) and count<5):
        if(k==len(second)-1):
            print(second[k],end="  ")
        else:
            print(second[k],end="->")
        k=k+1
        count=count+1
    print()
    count=0
    l=i
    while(l<len(third) and count<5):
```

```
        print(third[l],end="  ")
        l=l+1
        count=count+1
    print()
    count=0
    while(i<len(third) and count<5):
        print(fourth[i],end="  ")
        i=i+1
        count=count+1
    print()
    print()
print("-----------------------------------------------------------------------------------------------------
--------------------------------------")
for i in range(0,len(bfs_order)):
    if(i==len(bfs_order)-1):
        print(bfs_order[i].name, end="")
    else:
        print(bfs_order[i].name, end=" -> ")
print("\n-----------------------------------------------------------------------------------------------------
----------------------------------------")
print("Traversal upto goal node using BFS: ",end="")
for i in range(0,len(bfs_order)):
    if(bfs_order[i].name==goal_node[0].name or
bfs_order[i].name==goal_node[1].name):
        print(bfs_order[i].name, end=" ")
        break
    else:
        print(bfs_order[i].name, end=" -> ")
print("\n-----------------------------------------------------------------------------------------------------
----------------------------------------")
print("-----------------------------------------------------------------------------------------------------
--------------------------------------")
print("Depth first search traversal: ")
print("-----------------------------------------------------------------------------------------------------
--------------------------------------\n")
dfs_order=dfs(root)
first=[]
second=[]
third=[]
fourth=[]
for i in dfs_order:
```

```
    first.append(i.val[0])
    second.append(i.val[1])
    third.append(i.val[2])
    fourth.append(i.val[3])
i=0
while(i<len(first)):
    count=0
    j=i
    while(j<len(first) and count<5):
        print(first[j],end="  ")
        j=j+1
        count=count+1
    print()
    count=0
    k=i
    while(k<len(second) and count<5):
        if(k==len(second)-1):
            print(second[k],end="  ")
        else:
            print(second[k],end="->")
        k=k+1
        count=count+1
    print()
    count=0
    l=i
    while(l<len(third) and count<5):
        print(third[l],end="  ")
        l=l+1
        count=count+1
    print()
    count=0
    while(i<len(third) and count<5):
        print(fourth[i],end="  ")
        i=i+1
        count=count+1
    print()
    print()
print("-----------------------------------------------------------------------------------------------
-------------------------------------")
for i in range(0,len(dfs_order)):
    if(i==len(dfs_order)-1):
```

```
        print(dfs_order[i].name, end="")
    else:
        print(dfs_order[i].name, end=" -> ")
print("\n-----------------------------------------------------------------------------------------------------
---------------------------------------")
print("Traversal upto goal node using DFS: ",end="")
for i in range(0,len(dfs_order)):
    if(dfs_order[i].name==goal_node[0].name or
dfs_order[i].name==goal_node[1].name):
        print(dfs_order[i].name, end=" ")
        break
    else:
        print(dfs_order[i].name, end=" -> ")
print("\n-----------------------------------------------------------------------------------------------------
---------------------------------------")
print("-----------------------------------------------------------------------------------------------------
---------------------------------------")
print("Depth limited search traversal: ")
print("-----------------------------------------------------------------------------------------------------
---------------------------------------")
depth=int(input("Enter depth: "))
print("-----------------------------------------------------------------------------------------------------
---------------------------------------\n")
dls_order=dls(root,depth)
first=[]
second=[]
third=[]
fourth=[]
for i in dls_order:
    first.append(i.val[0])
    second.append(i.val[1])
    third.append(i.val[2])
    fourth.append(i.val[3])
i=0
while(i<len(first)):
    count=0
    j=i
    while(j<len(first) and count<5):
        print(first[j],end="  ")
        j=j+1
        count=count+1
```

```
    print()
    count=0
    k=i
    while(k<len(second) and count<5):
        if(k==len(second)-1):
            print(second[k],end="  ")
        else:
            print(second[k],end="->")
        k=k+1
        count=count+1
    print()
    count=0
    l=i
    while(l<len(third) and count<5):
        print(third[l],end="  ")
        l=l+1
        count=count+1
    print()
    count=0
    while(i<len(third) and count<5):
        print(fourth[i],end="  ")
        i=i+1
        count=count+1
    print()
    print()
print("------------------------------------------------------------------------------------------
-------------------------------------")
for i in range(0,len(dls_order)):
    if(i==len(dls_order)-1):
        print(dls_order[i].name, end="")
    else:
        print(dls_order[i].name, end=" -> ")
print("\n------------------------------------------------------------------------------------------
---------------------------------------")
print("Traversal upto goal node using DLS: ",end="")
for i in range(0,len(dls_order)):
    if(dls_order[i].name==goal_node[0].name or
dls_order[i].name==goal_node[1].name or i==len(dls_order)-1):
        if(dls_order[i].name==goal_node[0].name or
dls_order[i].name==goal_node[1].name):
            print(dls_order[i].name, end=" (Goal found)")
```

```
        else:
            print(dls_order[i].name, end=" (Goal not found)")
        break
    else:
        print(dls_order[i].name, end=" -> ")
print("\n-----------------------------------------------------------------------------------------------------
-----------------------------------------")
```

Output:

Breadth first search:

```
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------- UNINFORMED SEARCH ----------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
Breadth first search traversal:
-------------------------------------------------------------------------------------------------------------------------

['-', '-', '-', '-'] ['Q', '-', '-', '-'] ['-', 'Q', '-', '-'] ['-', '-', 'Q', '-'] ['-', '-', '-', 'Q']
['-', '-', '-', '-']->['-', '-', '-', '-']->['-', '-', '-', '-']->['-', '-', '-', '-']->['-', '-', '-', '-']->
['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-']
['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-']

['-', 'Q', '-', '-'] ['-', '-', 'Q', '-'] ['-', 'Q', '-', '-'] ['-', '-', 'Q', '-'] ['-', 'Q', '-', '-']
['-', '-', '-', 'Q']->['Q', '-', '-', '-']->['-', '-', '-', 'Q']->['Q', '-', '-', '-']->['-', '-', '-', 'Q']->
['-', '-', '-', '-'] ['-', '-', '-', '-'] ['Q', '-', '-', '-'] ['-', '-', '-', 'Q'] ['Q', '-', '-', '-']
['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', 'Q', '-']

['-', '-', 'Q', '-']
['Q', '-', '-', '-']
['-', '-', '-', 'Q']
['-', 'Q', '-', '-']

-------------------------------------------------------------------------------------------------------------------------
A -> B -> C -> D -> E -> F -> G -> H -> I -> J -> K
-------------------------------------------------------------------------------------------------------------------------
Traversal upto goal node using BFS: A -> B -> C -> D -> E -> F -> G -> H -> I -> J
-------------------------------------------------------------------------------------------------------------------------
```

Depth first search:

```
-------------------------------------------------------------------------------------------------------------------------
Depth first search traversal:
-------------------------------------------------------------------------------------------------------------------------

['-', '-', '-', '-'] ['Q', '-', '-', '-'] ['-', 'Q', '-', '-'] ['-', 'Q', '-', '-'] ['-', 'Q', '-', '-']
['-', '-', '-', '-']->['-', '-', '-', '-']->['-', '-', '-', '-']->['-', '-', '-', 'Q']->['-', '-', '-', 'Q']->
['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['Q', '-', '-', '-']
['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-']

['-', 'Q', '-', '-'] ['-', '-', 'Q', '-'] ['-', '-', 'Q', '-'] ['-', '-', 'Q', '-'] ['-', '-', 'Q', '-']
['-', '-', '-', 'Q']->['-', '-', '-', '-']->['Q', '-', '-', '-']->['Q', '-', '-', '-']->['Q', '-', '-', '-']->
['Q', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', 'Q'] ['-', '-', '-', 'Q']
['-', '-', 'Q', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', 'Q', '-', '-']

['-', '-', '-', 'Q']
['-', '-', '-', '-']
['-', '-', '-', '-']
['-', '-', '-', '-']

-------------------------------------------------------------------------------------------------------------------------
A -> B -> C -> F -> H -> J -> D -> G -> I -> K -> E
-------------------------------------------------------------------------------------------------------------------------
Traversal upto goal node using DFS: A -> B -> C -> F -> H -> J
-------------------------------------------------------------------------------------------------------------------------
```

Depth limited search:

```
Depth limited search traversal:
Enter depth: 1

['-', '-', '-', '-'] ['Q', '-', '-', '-'] ['-', 'Q', '-', '-'] ['-', '-', 'Q', '-'] ['-', '-', '-', 'Q']
['-', '-', '-', '-']->['-', '-', '-', '-']->['-', '-', '-', '-']->['-', '-', '-', '-']->['-', '-', '-', '-']
['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-']
['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-']

A -> B -> C -> D -> E
Traversal upto goal node using DLS: A -> B -> C -> D -> E (Goal not found)
```

```
Depth limited search traversal:
Enter depth: 4

['-', '-', '-', '-'] ['Q', '-', '-', '-'] ['-', 'Q', '-', '-'] ['-', 'Q', '-', '-'] ['-', 'Q', '-', '-']
['-', '-', '-', '-']->['-', '-', '-', '-']->['-', '-', '-', '-']->['-', '-', '-', 'Q']->['-', '-', '-', 'Q']->
['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['Q', '-', '-', '-']
['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-']

['-', 'Q', '-', '-'] ['-', '-', 'Q', '-'] ['-', '-', 'Q', '-'] ['-', '-', 'Q', '-'] ['-', '-', 'Q', '-']
['-', '-', '-', 'Q']->['-', '-', '-', '-']->['Q', '-', '-', '-']->['Q', '-', '-', '-']->['Q', '-', '-', '-']->
['Q', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', 'Q'] ['-', '-', '-', 'Q']
['-', '-', 'Q', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', '-', '-', '-'] ['-', 'Q', '-', '-']

['-', '-', '-', 'Q']
['-', '-', '-', '-']
['-', '-', '-', '-']
['-', '-', '-', '-']

A -> B -> C -> F -> H -> J -> D -> G -> I -> K -> E
Traversal upto goal node using DLS: A -> B -> C -> F -> H -> J (Goal found)
```

Comparison of performance of uninformed Algorithm:

BFS	DFS	DLS
1) Number of nodes traversed: 10	1) Number of nodes traversed: 6	1) Number of nodes traversed: 6 (depth=4)
2) Time: 0.15988 ms	2) Time: 0.10223 ms	2) Time: 0.11213 ms

Criterion	Breadth-First	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	No	No	Yes

Post Lab Objective questions

1. **Which search algorithm imposes a fixed depth limit on nodes?**
   a. Depth-limited search
   b. Depth-first search
   c. Iterative Deepening search
   d. Only (a) and (b)
   e. Only (a), (b) and (c).

Answer: a. Depth-limited search

2. **Optimality of BFS is**
   a. When all step costs are equal
   b. When all step costs are unequal
   c. When there is less number of nodes
   d. Both a & c

Answer: a. When all step costs are equal

Post Lab Subjective Questions:

1. **Mention the criteria for the evaluation of search Algorithm.**

ANS) 1) Completeness: A search algorithm is said to be complete when it gives a solution or returns any solution for a given random input.
2) Optimality: If a solution found is best (lowest path cost) among all the solutions identified, then that solution is said to be an optimal one.
3) Time complexity: The time taken by an algorithm to complete its task is called time complexity. If the algorithm completes a task in a lesser amount of time, then it is an efficient one.
4) Space complexity: It is the maximum storage or memory taken by the algorithm at any time while searching.

2. **State the properties of BFS,DFS, DLS and IDS.**

ANS) Breadth first search (BFS): 1) BFS algorithm is complete that is complete if the shallowest goal node is at some finite depth d.
2) BFS is optimal if the path cost is a nondecreasing function of the depth of the node.
3) Time complexity of BFS is $O(b^{d+1})$.
4) Space complexity of BFS is $O(b^{d+1})$.

Depth first search (DFS): 1) If the graph is finite in length, then DFS is complete.
2) DFS is not optimal (if the goal has more than one path from the source that leads to it, then DFS is not guaranteed to find the shortest one).
3) Time complexity of DFS is $O(b^m)$.

4) Space complexity of DFS is O(bxm).

Depth limited search (DLS): 1) First of all DLS is not complete in general. If the optimal solution node is still deeper than the depth limit L, DLS will never find the solution.
2) Since DLS is not complete, it is not optimal, too.
3) Time complexity of DLS is $O(b^{\ell})$.
4) Space complexity of DLS is $O(b \times \ell)$.

Iterative deepening search (IDS): 1) If the graph is finite in space, then IDS is complete.
2) IDS is also optimal when the edge costs are the same since it will find the solution that is the 'closest' to the root just like what BFS does.
3) Time complexity of IDS is $O(b^{d})$.
4) Space complexity of IDS is $O(b \times d)$.

Where, b = branching factor
d = depth of goal node
m = max number of connections between the start node and any node

3. Explain why BFS is worst approach when the branching factor and solution depth in state-space tree is large (value =10 or more).

ANS) As we know that the time and the space complexity of breadth first search is directly proportional to branching factor(b) and solution depth(d). The time and space complexity of BFS is given by

Time complexity: $O(b^{d+1})$ Space complexity: $O(b^{d+1})$

So when b and d increase space and time complexity of the BFS increases exponentially.

Depth	Nodes	Time	Memory
2	110	1.1 milliseconds	107 kilobytes
4	11,110	111 milliseconds	10.6 megabytes
6	10^6	11 seconds	1 gigabytes
8	10^8	19 minutes	103 gigabytes
10	10^{10}	31 hours	10 terabytes
12	10^{12}	129 days	1 petabytes
14	10^{14}	35 years	99 petabytes
16	10^{16}	3,500 years	10 exabytes

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 100,000 nodes/second; 1000 bytes/node.

As seen from the above table as d increase the time and memory required increases exponentially. And at certain depth and branching factor it becomes impossible to find a solution using BFS.