***Practical 4***

**Aim:** *Implement recursive best-first search algorithm for Romanian map problem.*

***Theory:***

It is simple recursive algorithm that resembles the operation of standard best first search but uses only linear space. It is similar to recursive DFS and differs from Recursive DFS as follows,

It keeps track of the f value of the best alternative path available from any ancestor of the current node. Instead of continuing indefinitely down the current path.

***Algorithm:***

{

function RECURSIVE\_BEST\_FIRST\_SEARCH (Problem)

    returns a solution, or failure

    RBFS (problem, MAKE\_NODE (INITIAL\_STATE problem), Ұ)

function RBFS (Problem, Node, f-limit) returns a solution or failure and a new f-cost limit

    if GOAL\_TEST [Problem] [State] then return node

Successors ← EXPAND (node, problem)

    if successors is empty then return failure, ¥

for each S in successors do

    f(s) ← max (g(s) + h(s), f[node])

repeat

    best ← the lowest f value node in successors

        if [best] > f-limit then return failure, f[best]

    alternative ← the second lowest f-value among successors

    result, f[best] ← BFS (Problem, best, min (f-limit, alternative)

        if result # failure then return result

}

**Advantages**

* *More efficient than IDA\**
* *It is an optimal algorithm if h(n) is admissible*
* *Space complexity is O(bd).*

**Disadvantages**

* *It suffers from excessive node regeneration.*
* *Its time complexity is difficult to characterize because it depends on the accuracy of h(n) and how often the best path changes as the nodes are expanded.*

***Code:***

dict\_hn={'Arad':336,'Bucharest':0,'Craiova':160,'Drobeta':242,'Eforie':161,

         'Fagaras':176,'Giurgiu':77,'Hirsova':151,'Iasi':226,'Lugoj':244,

         'Mehadia':241,'Neamt':234,'Oradea':380,'Pitesti':100,'Rimnicu':193,

         'Sibiu':253,'Timisoara':329,'Urziceni':80,'Vaslui':199,'Zerind':374}

dict\_gn=dict(

Arad=dict(Zerind=75,Timisoara=118,Sibiu=140),

Bucharest=dict(Urziceni=85,Giurgiu=90,Pitesti=101,Fagaras=211),

Craiova=dict(Drobeta=120,Pitesti=138,Rimnicu=146),

Drobeta=dict(Mehadia=75,Craiova=120),

Eforie=dict(Hirsova=86),

Fagaras=dict(Sibiu=99,Bucharest=211),

Giurgiu=dict(Bucharest=90),

Hirsova=dict(Eforie=86,Urziceni=98),

Iasi=dict(Neamt=87,Vaslui=92),

Lugoj=dict(Mehadia=70,Timisoara=111),

Mehadia=dict(Lugoj=70,Drobeta=75),

Neamt=dict(Iasi=87),

Oradea=dict(Zerind=71,Sibiu=151),

Pitesti=dict(Rimnicu=97,Bucharest=101,Craiova=138),

Rimnicu=dict(Sibiu=80,Pitesti=97,Craiova=146),

Sibiu=dict(Rimnicu=80,Fagaras=99,Arad=140,Oradea=151),

Timisoara=dict(Lugoj=111,Arad=118),

Urziceni=dict(Bucharest=85,Hirsova=98,Vaslui=142),

Vaslui=dict(Iasi=92,Urziceni=142),

Zerind=dict(Oradea=71,Arad=75)

)

import queue as Q

start='Arad'

goal='Bucharest'

result=''

def get\_fn(citystr):

    cities=citystr.split(',')

    hn=gn=0

    for ctr in range(0,len(cities)-1):

        gn=gn+dict\_gn[cities[ctr]][cities[ctr+1]]

    hn=dict\_hn[cities[len(cities)-1]]

    return(hn+gn)

def printout(cityq):

    for i in range(0,cityq.qsize()):

        print(cityq.queue[i])

def expand(cityq):

    global result

    tot,citystr,thiscity=cityq.get()

    nexttot=999

    if not cityq.empty():

        nexttot,nextcitystr,nextthiscity=cityq.queue[0]

    if thiscity==goal and tot<nexttot:

        result=citystr+'::'+str(tot)

        return

    print("Expanded city------------------------------",thiscity)

    print("Second best f(n)------------------------------",nexttot)

    tempq=Q.PriorityQueue()

    for cty in dict\_gn[thiscity]:

            tempq.put((get\_fn(citystr+','+cty),citystr+','+cty,cty))

    for ctr in range(1,3):

        ctrtot,ctrcitystr,ctrthiscity=tempq.get()

        if ctrtot<nexttot:

            cityq.put((ctrtot,ctrcitystr,ctrthiscity))

        else:

            cityq.put((ctrtot,citystr,thiscity))

            break

    printout(cityq)

    expand(cityq)

def main():

    cityq=Q.PriorityQueue()

    thiscity=start

    cityq.put((999,"NA","NA"))

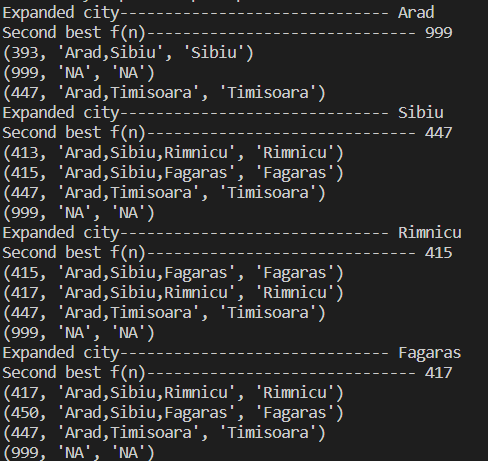
    cityq.put((get\_fn(start),start,thiscity))

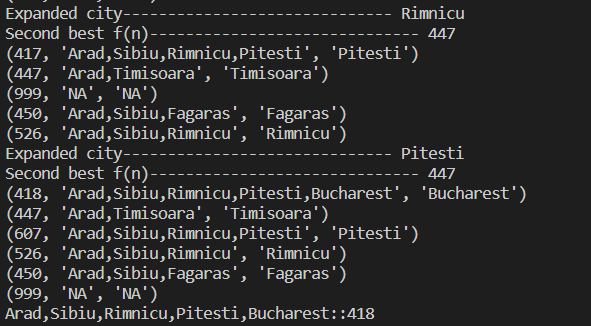
    expand(cityq)

    print(result)

main()

***Output:***

******

******

***Conclusion:***

*Implemented recursive best-first search algorithm for Romanian map problem.*