Lab 4: Problem Solving by Searching: A* Search

Informed Search: Informed Search algorithms have information on the goal state which helps in more efficient searching. This information is obtained by a function that estimates how close a state is to the goal state. In this lab, we are going to implement A* search.

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). A* search algorithm finds the shortest path through the search space using the heuristic function

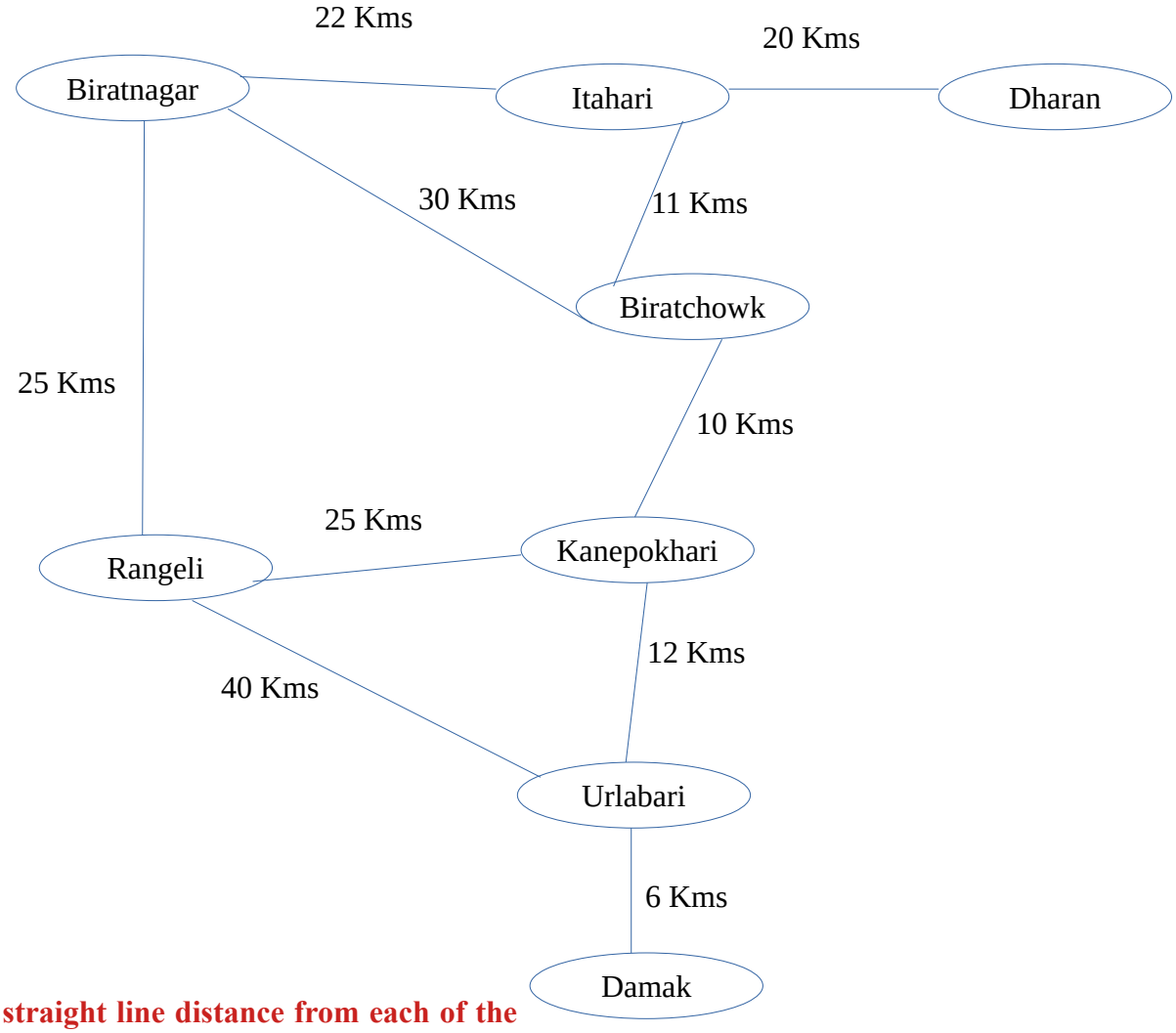
$$f(n) = g(n) + h(n).$$

This search algorithm expands less search tree and provides optimal result faster.

Algorithm

1. Declare the visited list
2. Declare the unvisited list
3. For each node in graph:
   – Add the node to the unvisited list with a g-score of infinity, an f-score of infinity and previous node of null.
4. Set the start node's g-score to 0 in the unvisited list
5. Set the start node's f-score to its h-score in the unvisited list
6. Set finished to False
7. While finished is False:
   – Set current node to the node in the unvisited list with the lowest f-score
   – If the current node is the target node
      – Set finished to True
      – Copy the values for the current node from the unvisited list to the visited list
   – Else
      – For each neighbor of current node:
         – If neighbor is not in the visited list
            – Calculate new g-score = weight of edge + g-score of current node
            – If new g-score is less than neighbor's g-score in unvisited list
               – Update the neighbor's g-score with the new g-score
               – Update the neighbor's f-score to new g-score + h_score
               – Update the neighbor's previous node to the current node
      – Copy the values for the current node from the unvisited list to the visited list
      – Remove the current node from the unvisited list
8. Return the visited list

We will use the A* algorithm to find the best path from Biratnagar to Damak in the following map.



The straight line distance from each of the city to Damak is shown below:

Heuristic function h(n)	
Biratnagar	46 Kms
Itahari	39 Kms
Dharan	41 Kms
Rangeli	28 Kms
Biratchowk	29 Kms
Kanepokhari	17 Kms
Urlabari	6 Kms
Damak	0 Kms

Source Code:

```
gScore = 0 #use this to index g(n)
fScore = 1 #use this to index f(n)
previous = 2 #use this to index previous node

inf = 10000 #use this for value of infinity

#We represent the graph using adjacent list
#as dictionary of dictionaries
G = {
  'biratnagar' : {'itahari' :  22, 'biratchowk' : 30, 'rangeli': 25},
  'itahari' : {'biratnagar' : 22, 'dharan' : 20, 'biratchowk' : 11},
  'dharan' : {'itahari' : 20},
  'biratchowk' : {'biratnagar' : 30, 'itahari' :  11, 'kanepokhari' :10},
  'rangeli' : {'biratnagar' : 25, 'kanepokhari' : 25, 'urlabari' : 40},
  'kanepokhari' : {'rangeli' : 25, 'biratchowk' : 10, 'urlabari' : 12},
  'urlabari' : {'rangeli' : 40, 'kanepokhari' : 12, 'damak' : 6},
  'damak' : {'urlabari' : 6}
  }

def h(city):
  #returns straight line distance from a city to damak
  h = {
    'biratnagar' : 46,
    'itahari' : 39,
    'dharan' : 41,
    'rangeli' : 28,
    'biratchowk' : 29,
    'kanepokhari' : 17,
    'urlabari' : 6,
    'damak' : 0
    }
  return h[city]

def getMinimum(unvisited):
  #returns city with minimum f(n)
  currDist = inf
  leastFScoreCity = ''
  for city in unvisited:
    if unvisited[city][fScore] < currDist:
      currDist = unvisited[city][fScore]
      leastFScoreCity = city
  return leastFScoreCity
```

```
def aStar(G, start, goal):
  visited = {} #we declare visited list as empty dict
  unvisited = {} #we declare unvisited list as empty dict

  #we now add every city to the unvisited
  for city in G.keys():
    unvisited[city] = [inf, inf, ""]

  hScore = h(start)
  #for starting node, the g(n) is 0, so f(n) will be h(n)
  unvisited[start] = [0, hScore, ""]

  finished = False
  while finished == False:
    #if there are no nodes to evaluate in unvisited
    if len(unvisited) == 0:
      finished = True

    else:
      #find the node with lowest f(n) from open list
      currentNode = getMinimum(unvisited)
      if currentNode == goal:
        finished = True
        #copy data to visited list
        visited[currentNode] = unvisited[currentNode]

      else:
        #we examine the neighbors of currentNode
        for neighbor in G[currentNode]:
          #we only check unvisited neighbors
          if neighbor not in visited:
            newGScore = unvisited[currentNode][gScore] + G[currentNode][neighbor]
            if newGScore < unvisited[neighbor][gScore]:
              unvisited[neighbor][gScore] = newGScore
              unvisited[neighbor][fScore] = newGScore + h(neighbor)
              unvisited[neighbor][previous] = currentNode

        #we now add currentNode to the visited list
        visited[currentNode] = unvisited[currentNode]
        #we now remove  the currentNode from unvisited
        del unvisited[currentNode]
  return visited
```

```
def findPath(visitSequence, goal):
  path = goal
  currCity = goal
  while visitSequence[currCity][previous] != '':
    prevCity = visitSequence[currCity][previous]
    path = prevCity + "->" + path
    currCity = prevCity
  return path

start = 'biratnagar'
goal = 'damak'

if __name__ == "__main__":
  visitSequence = aStar(G, start, goal)
  print(findPath(visitSequence, goal))
```

Output

```
biratnagar->biratchowk->kanepokhari->urlabari->damak
```