Iterative Deepening Search

```
"""
UVA 155 - All Squares
Searching strategy: IDS
"""

cx, cy, total = [0] * 3
true, false = True, False


def scan(t=int):
    scanned = input().split()
    len_scan = len(scanned)
    if len_scan is 1:
        return t(scanned[0])

    return [t(val) for val in scanned]


def depth_limited_search(x, y, k, level, max_level):
    if k is 0:
        return true
    if level is max_level:
        return false

    global cx, cy, total
    if (x - k <= cx <= x + k) and (y - k <= cy <= y + k):
        total += 1

    return (
            depth_limited_search(x + k, y + k, int(k / 2), level + 1, max_level) and
            depth_limited_search(x + k, y - k, int(k / 2), level + 1, max_level) and
            depth_limited_search(x - k, y + k, int(k / 2), level + 1, max_level) and
            depth_limited_search(x - k, y - k, int(k / 2), level + 1, max_level)
    )


def main():
    global cx, cy, total
    while true:
        k, cx, cy = scan()
        if k is cx is cy is 0:
            break
```

```
    depth_limit = 11  # log2(1024) = 10
    for max_depth in range(depth_limit):
        total = 0
        if depth_limited_search(1024, 1024, k, 0, max_depth):
            print('%3d' % total)
            break


main()
```

A* Searching Algorighm

```
"""
A* searching algorithm
Road to Bucharest
"""

from queue import PriorityQueue

Arad = "Arad"
Bucharest = "Bucharest"
Craiova = "Craiova"
Dobreta = "Dobreta"
Eforie = "Eforie"
Fagaras = "Fagaras"
Giurgiu = "Giurgiu"
Hirsova = "Hirsova"
Iasi = "Iasi"
Lugoj = "Lugoj"
Mehadia = "Mehadia"
Neamt = "Neamt"
Oradea = "Oradea"
Pitesti = "Pitesti"
RimnicuVilcea = "Rimnicu Vilcea"
Sibiu = "Sibiu"
Timisoara = "Timisoara"
Urziceni = "Urziceni"
Vaslui = "Vaslui"
Zerind = "Zerind"

dist_to_bucharest = {
    Arad: 366,
```

```
    Bucharest: 0,
    Craiova: 160,
    Dobreta: 242,
    Eforie: 161,
    Fagaras: 176,
    Giurgiu: 77,
    Hirsova: 151,
    Iasi: 226,
    Lugoj: 244,
    Mehadia: 241,
    Neamt: 234,
    Oradea: 380,
    Pitesti: 100,
    RimnicuVilcea: 193,
    Sibiu: 253,
    Timisoara: 329,
    Urziceni: 80,
    Vaslui: 199,
    Zerind: 374
}

cost_so_far = {
    Arad: 0,
    Bucharest: 0,
    Craiova: 0,
    Dobreta: 0,
    Eforie: 0,
    Fagaras: 0,
    Giurgiu: 0,
    Hirsova: 0,
    Iasi: 0,
    Lugoj: 0,
    Mehadia: 0,
    Neamt: 0,
    Oradea: 0,
    Pitesti: 0,
    RimnicuVilcea: 0,
    Sibiu: 0,
    Timisoara: 0,
    Urziceni: 0,
    Vaslui: 0,
    Zerind: 0
}
```

```
map_of_romania = {
   Arad: {
      Zerind: 75,
      Timisoara: 118,
      Sibiu: 140
   },
   Bucharest: {
      Fagaras: 211,
      Pitesti: 101,
      Giurgiu: 90,
      Urziceni: 85
   },
   Craiova: {
      Dobreta: 120,
      RimnicuVilcea: 146,
      Pitesti: 138
   },
   Dobreta: {
      Craiova: 120,
      Mehadia: 75
   },
   Eforie: {
      Hirsova: 86
   },
   Fagaras: {
      Sibiu: 99,
      Bucharest: 211
   },
   Giurgiu: {
      Bucharest: 90
   },
   Hirsova: {
      Eforie: 86,
      Urziceni: 98
   },
   Iasi: {
      Neamt: 87,
      Vaslui: 92
   },
   Lugoj: {
      Timisoara: 111,
      Mehadia: 70
   },
   Mehadia: {
```

```
    Lugoj: 70,
    Dobreta: 75
  },
  Neamt: {
    Iasi: 87
  },
  Oradea: {
    Zerind: 71,
    Sibiu: 151
  },
  Pitesti: {
    RimnicuVilcea: 97,
    Craiova: 138,
    Bucharest: 101
  },
  RimnicuVilcea: {
    Pitesti: 97,
    Sibiu: 80,
    Craiova: 146
  },
  Sibiu: {
    RimnicuVilcea: 80,
    Oradea: 151,
    Arad: 140,
    Fagaras: 99
  },
  Timisoara: {
    Arad: 118,
    Lugoj: 111
  },
  Urziceni: {
    Bucharest: 85,
    Vaslui: 142,
    Hirsova: 98
  },
  Vaslui: {
    Urziceni: 142,
    Iasi: 92
  },
  Zerind: {
    Arad: 75,
    Oradea: 71
  }
}
```

```
def h(n):
    global dist_to_bucharest
    return dist_to_bucharest[n]


def g(n):
    global cost_so_far
    return cost_so_far[n]


def f(n):
    return h(n) + g(n)


def a_star(root, goal):
    q = PriorityQueue()
    q.put((h(root), root, [root]))

    while not q.empty():
        front = q.get()
        city = front[1]
        if city is goal:
            print("Reached with {} cost".format(front[0]))
            print("Path:", front[2])
            return

        adjacent_cities = map_of_romania[city]
        for adjacent_city, cost in adjacent_cities.items():
            cost_so_far[adjacent_city] = cost_so_far[city] + cost
            q.put((f(adjacent_city), adjacent_city, front[2] + [adjacent_city]))


def main():
    a_star(Arad, Bucharest)


main()
```