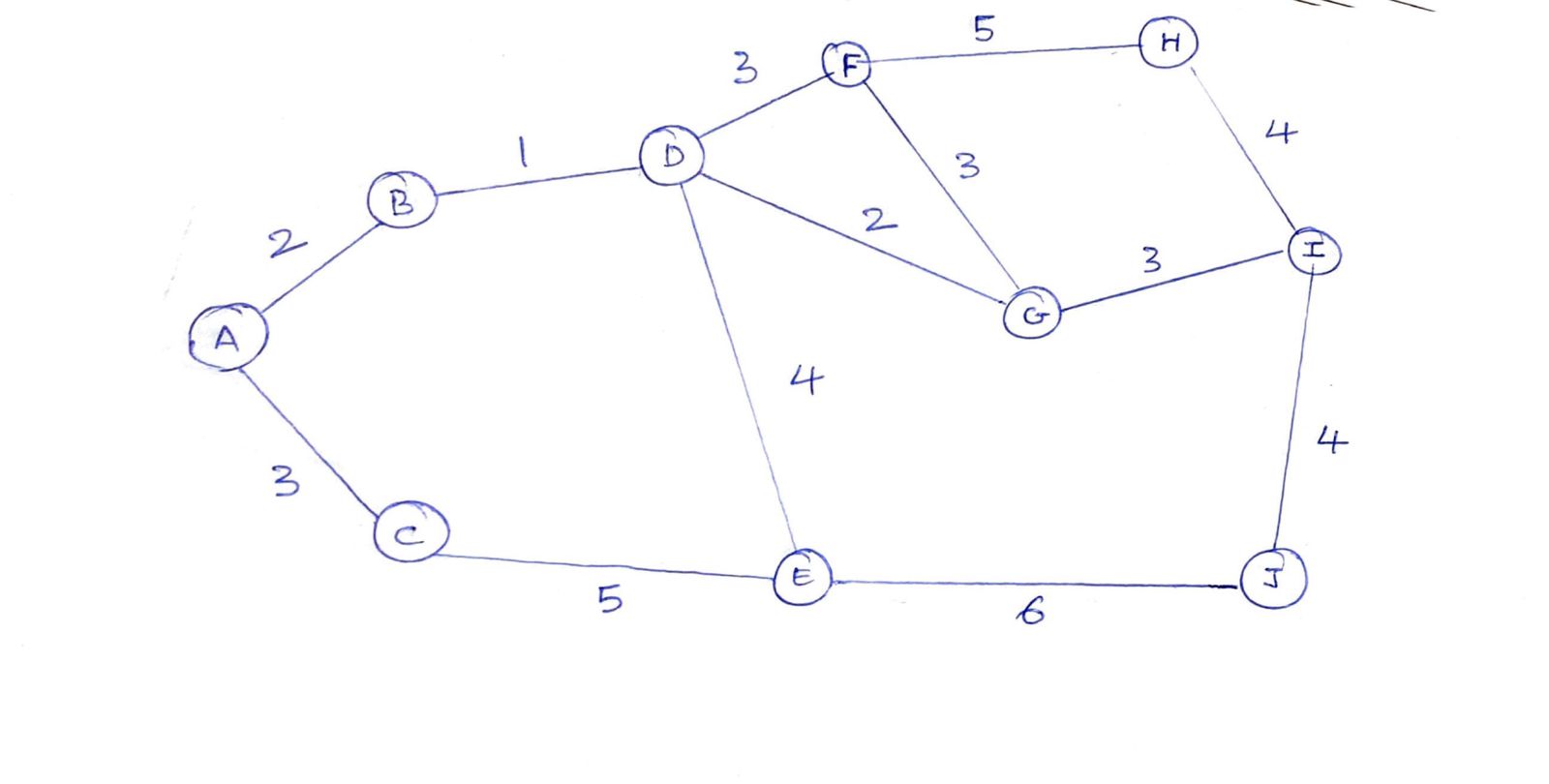
**Artificial Intelligence Project Report**

1.

**Problem**: Going from Manipal University Jaipur Gate to Railway Station in an Automated Car

**Initial state:** Manipal University Jaipur Gate (Node A)

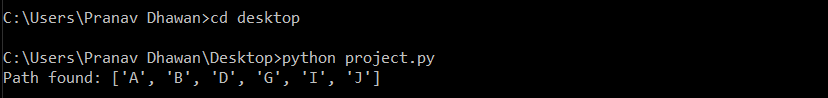
**Final state:** Railway Station (Node J)



Map from MUJ (A) to Railway Station (J)

**Path cost:** Sum of the edges

When the problem is optimized, below is the solution.



Since the most efficient path from A to J is A -> B -> D -> G -> I -> J and hence the Path cost comes out to be 0 + 2 + 1 + 2 + 3 + 4 = 12.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Automated Car Drive | The comfortable trip, Safety, Maximum Distance | Roads, Traffic, Vehicles | Steering wheel, Accelerator, Brake, Mirror | Camera, GPS, Odometer |

**PEAS** (Performance, Environment, Actuators, Sensors)

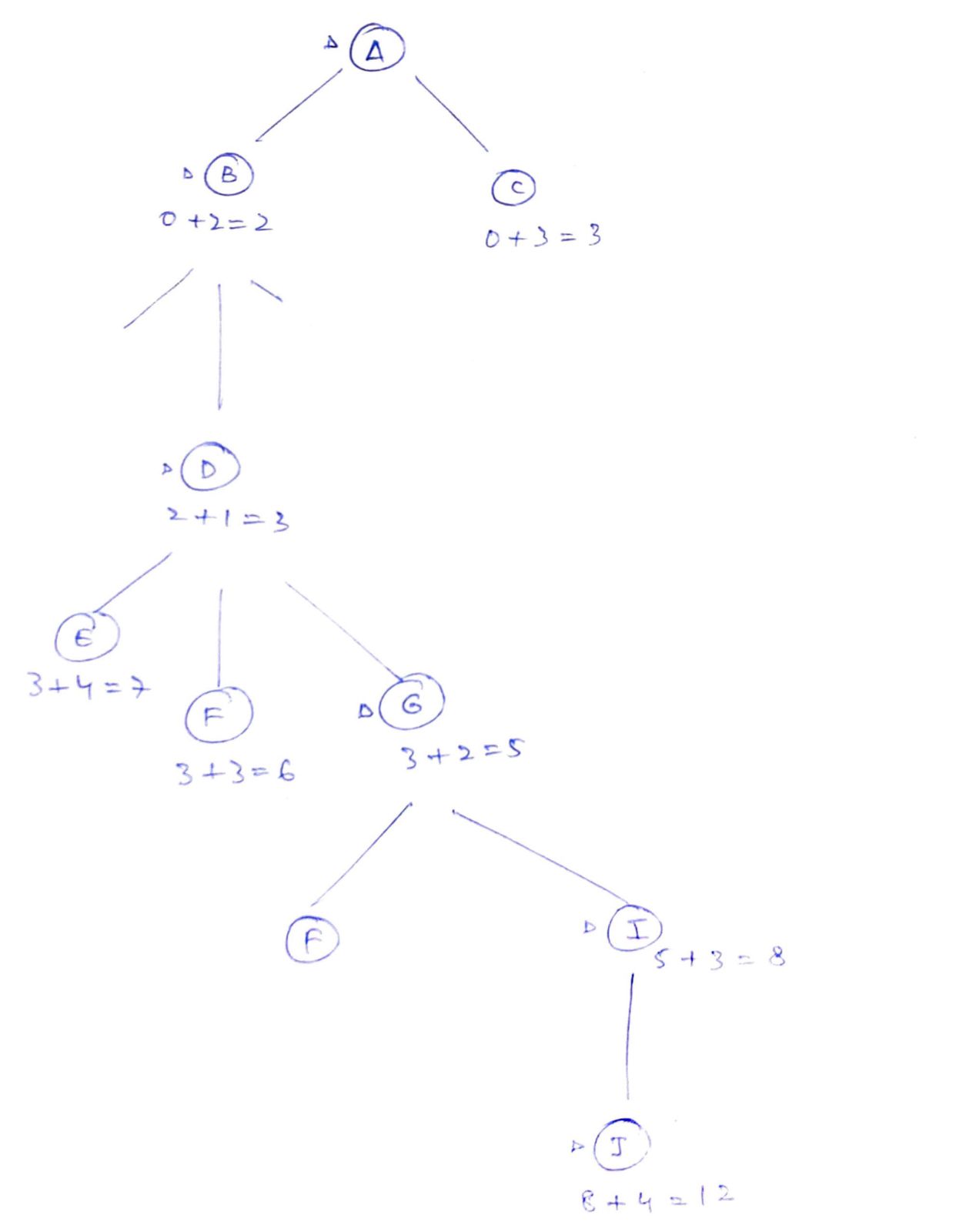
2.

**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). It has combined features of UCS and greedy best-first search, by which it solves the problem efficiently. A\* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A\* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

**Time Complexity:** The time complexity of A\* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So, the time complexity is O(b^d), where b is the branching factor.

**Space Complexity:** The space complexity of A\* search algorithm is **O(b^d).**

3.



**Code for A\* Search:**

def aStarAlgo(start\_node, stop\_node):

        open\_set = set(start\_node)

        closed\_set = set()

        g = {}

        parents = {}

        g[start\_node] = 0

        parents[start\_node] = start\_node

        while len(open\_set) > 0:

            n = None

            for v in open\_set:

                if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):

                    n = v

            if n == stop\_node or Graph\_nodes[n] == None:

                pass

            else:

                for (m, weight) in get\_neighbors(n):

                    if m not in open\_set and m not in closed\_set:

                        open\_set.add(m)

                        parents[m] = n

                        g[m] = g[n] + weight

                    else:

                        if g[m] > g[n] + weight:

                            g[m] = g[n] + weight

                            parents[m] = n

                            if m in closed\_set:

                                closed\_set.remove(m)

                                open\_set.add(m)

            if n == None:

                print('Path does not exist!')

                return None

            if n == stop\_node:

                path = []

                while parents[n] != n:

                    path.append(n)

                    n = parents[n]

                path.append(start\_node)

                path.reverse()

                print('Path found: {}'.format(path))

                return path

            open\_set.remove(n)

            closed\_set.add(n)

        print('Path does not exist!')

        return None

def get\_neighbors(v):

    if v in Graph\_nodes:

        return Graph\_nodes[v]

    else:

        return None

def heuristic(n):

        H\_dist = {

            'A': 1,

            'B': 1,

            'C': 1,

            'D': 1,

            'E': 1,

            'F': 1,

            'G': 1,

            'H': 1,

            'I': 1,

            'J': 1,

        }

        return H\_dist[n]

Graph\_nodes = {

    'A': [('B', 2), ('C', 3)],

    'B': [('D', 1)],

    'C': [('E', 5)],

    'D': [('F', 3), ('G', 2), ('E', 4)],

    'E': [('J', 6)],

    'F': [('H', 5)],

    'G': [('F', 3),('I',3)],

    'H': [('I', 4)],

    'I': [('J', 4)],

}

aStarAlgo('A', 'J')