Optimization in ArtificialIntelligence

Food Delivery System with the Utilization of Vehicle Using Geographical Information System(GIS) and A Star Algorithm

Group 3

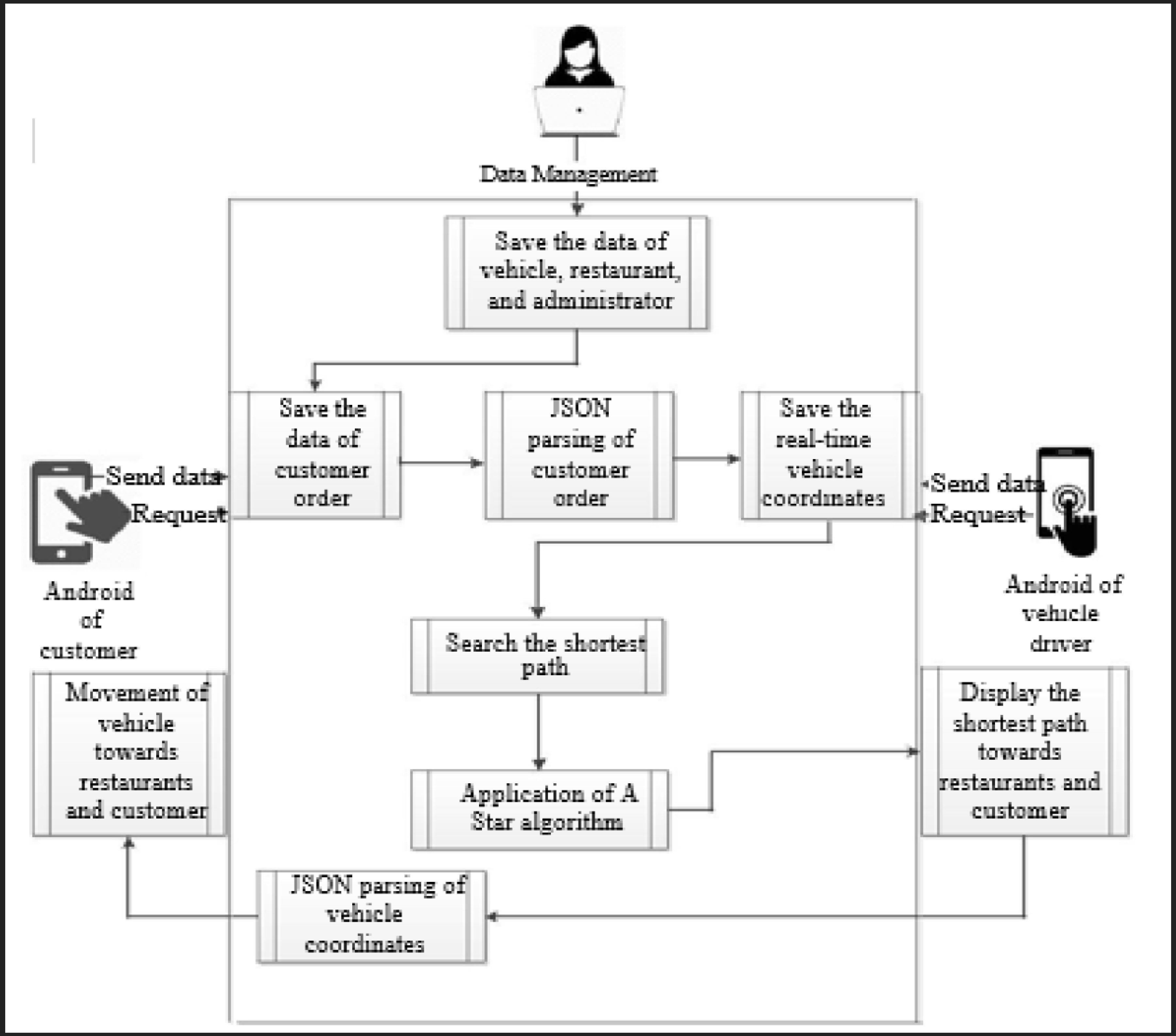
2020147051 JANG GEON
2021111012 LEE SEUNG YEON

Contents

Paper Abstract

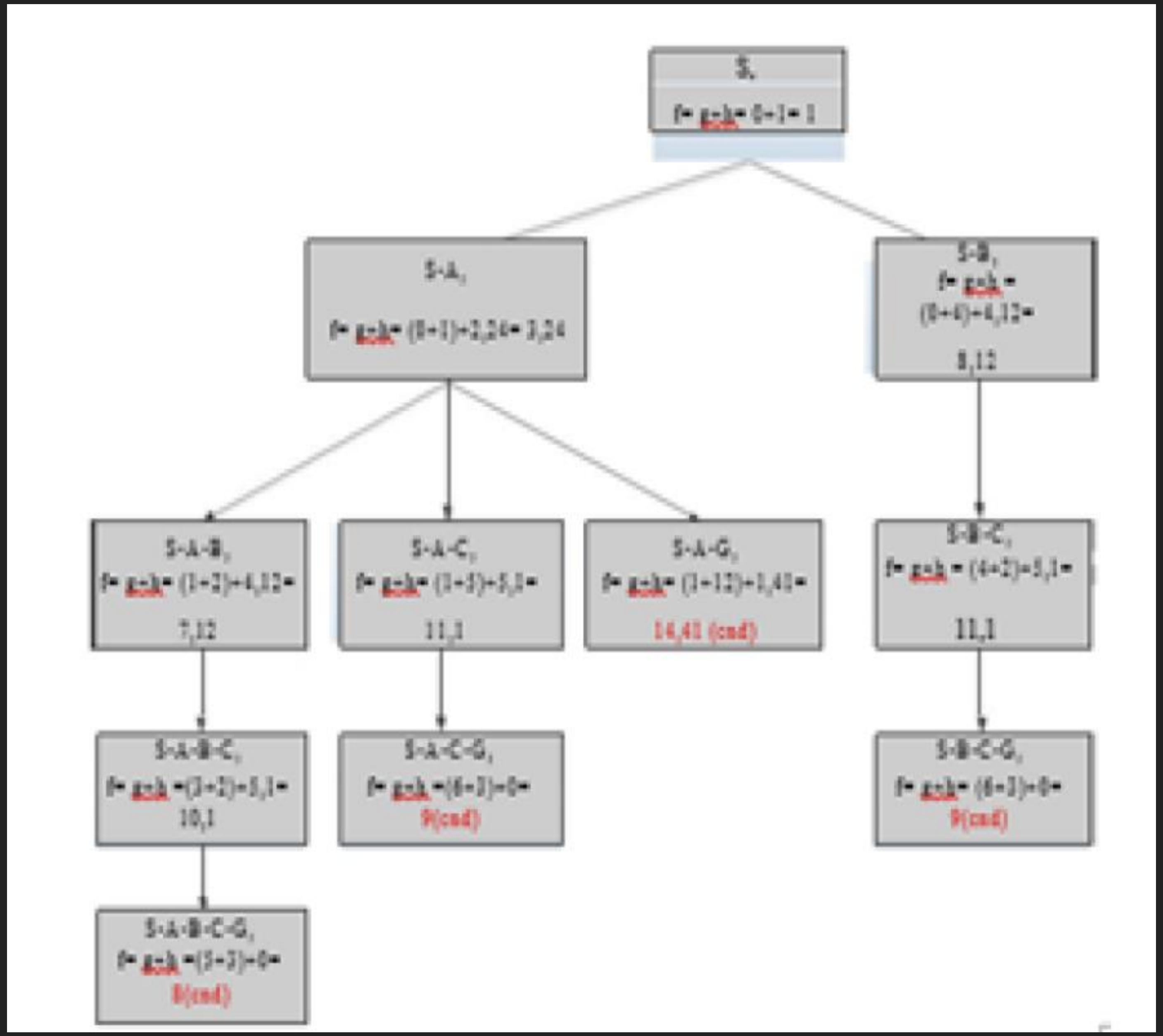
- Food delivery system is one kind of geographical information systems (GIS) that can be applied through digitation process. The main case in food delivery system is the way to determine the shortest path and food delivery vehicle movement tracking. Therefore, to make sure that the digitation process of food delivery system can be applied efficiently, it is needed to add shortest path determination facility and food delivery vehicle tracking. This research uses A Star (A*) algorithm for determining shortest path and location-based system (LBS) programming for moving food delivery vehicle object tracking. According to this research, it is generated the integrated system that can be used by food delivery driver, customer, and administrator in terms of simplifying the food delivery system. Through the application of shortest path and the tracking of moving vehicle, thus the application of food delivery system in the scope of geographical information system (GIS) can be executed.

Methodology



General Architecture

Methodology



- The Process of Shortest Path Calculation

- Shortest Path Calculation by A* algorithm
  -> f(x) = g(x) + h(x)

  g(x) - the total distance from the initial position into current location

  h(x) - the heuristic function used to estimate the distance from current location into the destination location

Methodology

No.	Node	Coordinate	Name
1	S	1,1	Pempek Palembang Setiabudi
2	A	3,1	Intersection of Dr. Sumarsono Street
3	B	2,6	Intersection of Politeknik/ Tri Dharma Street
4	C	6,1	Intersection of Gate3, University of North Sumatera (USU)
5	G	6,5	Faculy of Medicine, University of North Sumatera (USU)

- Explanation of Shortest Path Element

Methodology

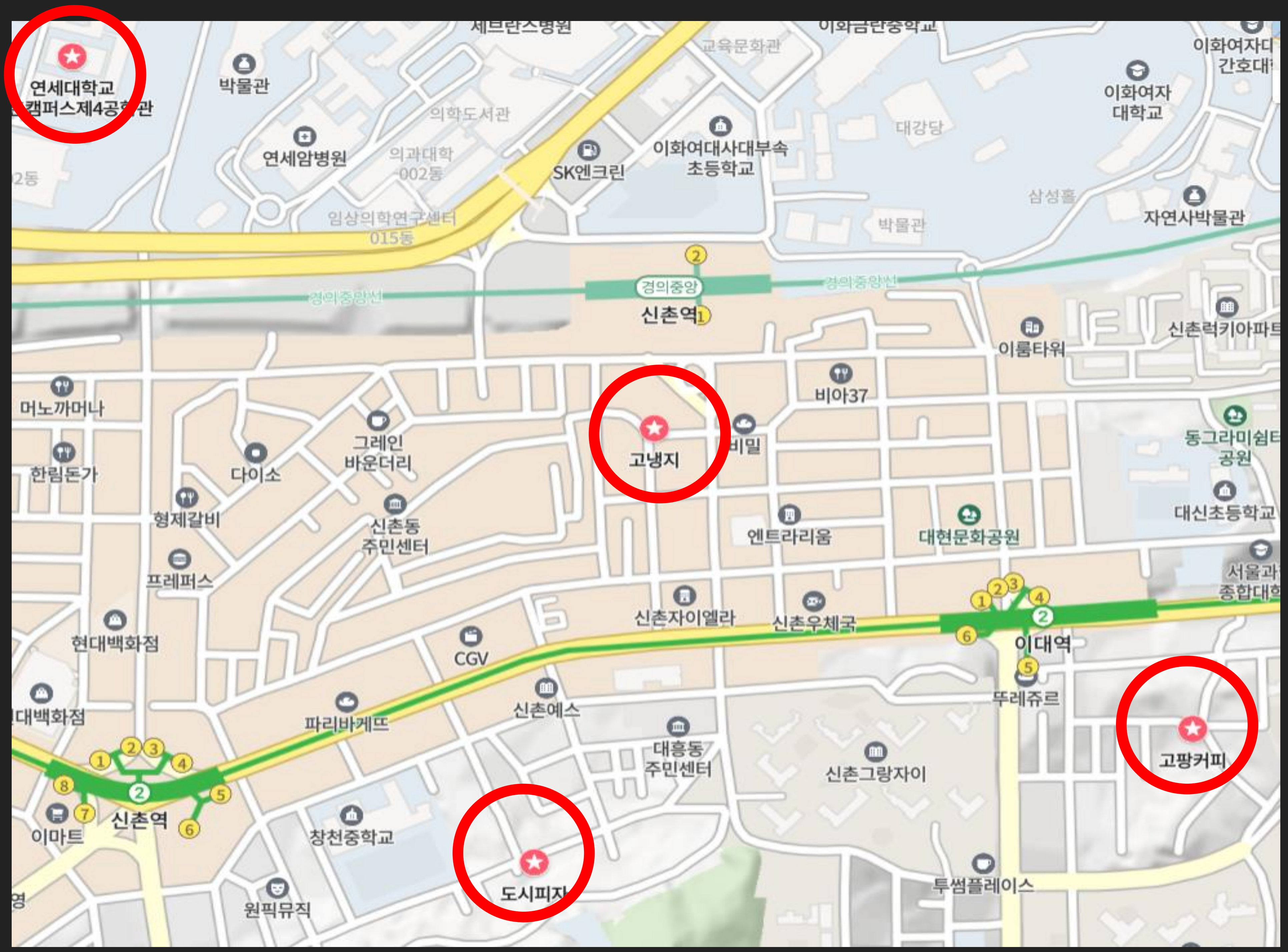
No.	Node Relation	Distance in Kilometers (km)
1	S-A	1
2	S-B	4
3	A-B	2
4	A-C	5
5	A-G	12
6	B-C	2
7	C-G	3

- Relation and Distance among Nodes

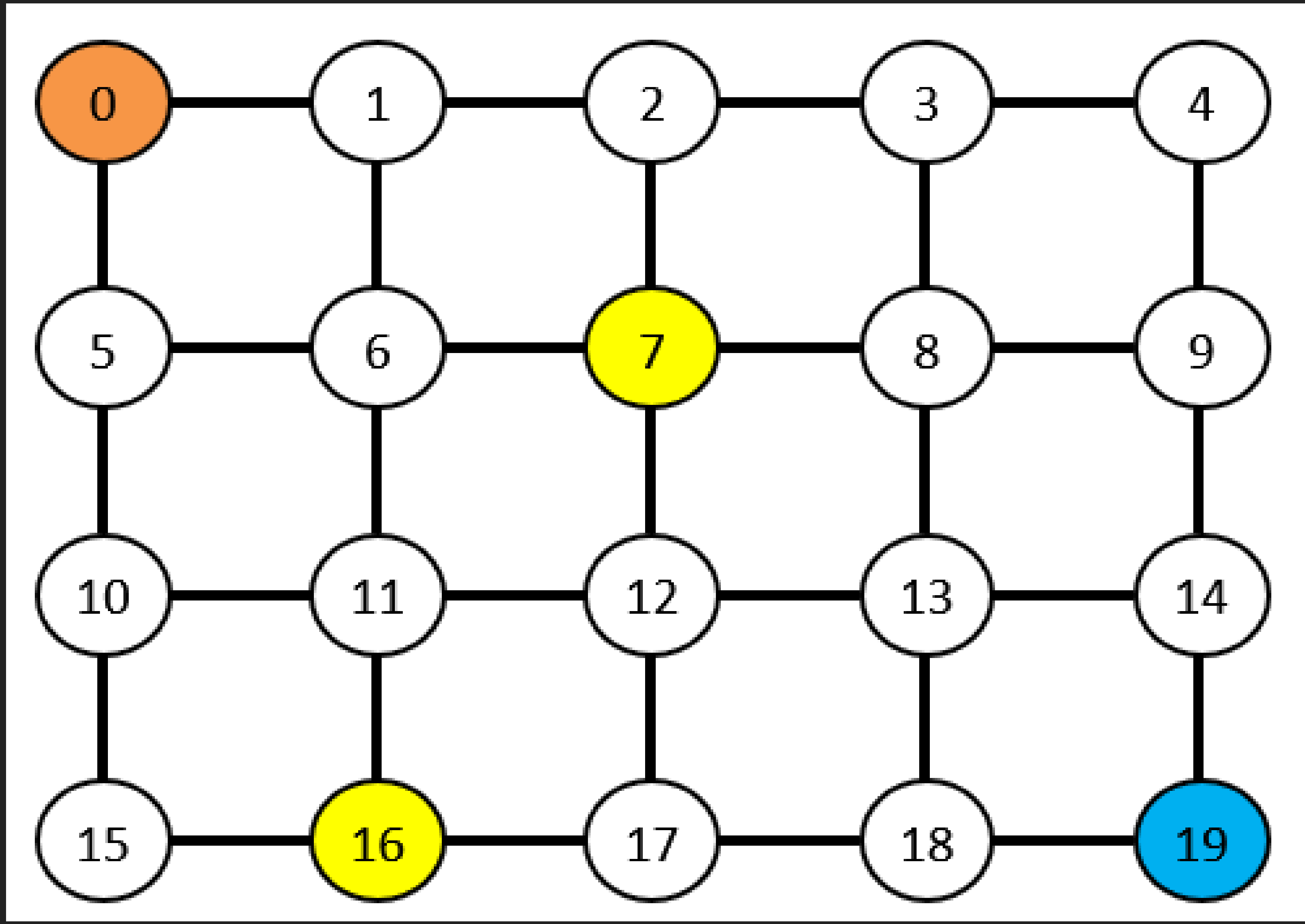
No.	Node	Heuristic
1	S	1
2	A	2,24
3	B	4,12
4	C	5,1
5	G	0

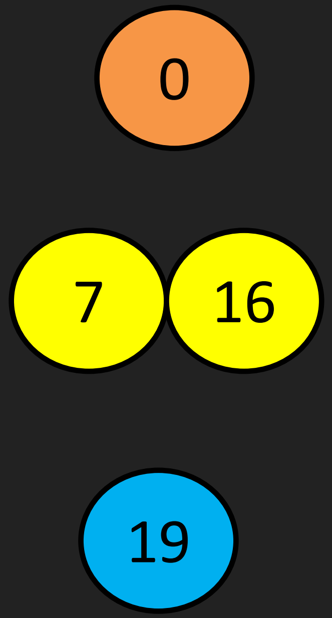
- The Heuristic List of Each Node

Assumption


연세대학교
박물관
연세암병원
SK엔크린
이화여대사대부속
초등학교
이화여자
대학교
자연사박물관
신촌역
이룸타워
신촌럭키아파트
머노까머나
한림돈가
다이소
그레인
바운더리
고냉지
비밀
비아37
형제갈비
신촌동
주민센터
엔트라리움
대현문화공원
대신초등학교
프레퍼스
현대백화점
신촌자이엘라
신촌우체국
이대역
CGV
신촌예스
뚜레쥬르
고팡커피
파리바게뜨
대흥동
주민센터
신촌그랑자이
이마트
신촌역
창천중학교
도시피자
원픽뮤직
투썸플레이스

Assumption





-> Start node

-> Transfer node

-> End node

Path1: 0 -> 7 -> 16 -> 19
Path2: 0 -> 16 -> 7 -> 19

Which path is the optimal solution?

A* algorithm code

f(x): a* algorithm

```python
# A* Algorithm
def a_star(start_node, end_node):
    # Ensure proper formatting
    start_node = int(start_node)
    end_node = int(end_node)

    # Signal to end search when target is reached
    target_reached = False
    # The A* path
    current_path = []
    # List of all checked nodes
    passed = [start_node]
    # Intermediate node checked through the loop
    latest_node = start_node
    while not target_reached:
        # Placeholder variable to find the minimum value
        min_score = float("inf")
        best_node = -1
        for (a,b) in all_paths:
            # Finding relevant paths that do not include checked nodes
            if a == latest_node and b not in passed:
                middle_node = b
                g = dijkstar(latest_node, middle_node)
                h = heuristic(middle_node, end_node)
                # The A* score
                f = g + h
                # Recording the node with the best A* score
                if f < min_score:
                    min_score = f
                    best_node = b
        # If stuck on a dead-end, return to previous node
        if best_node == -1:
            latest_node = passed[-2]
            current_path.pop(-1)
            continue
        # Add the new intermediate node to the path
        current_path.append((latest_node, best_node))
        passed.append(best_node)
        latest_node = best_node
        # If the target has been found, end the search
        if best_node == end_node:
            target_reached = True

    return current_path
```

g(x):Dijkstar algorithm

```python
# Dijkstar Algorithm, used in A* as g()
def dijkstar(start_node, end_node):
    # Ensure proper formatting
    start_node = int(start_node)
    end_node = int(end_node)

    # Skip if the source and destination are the same
    if start_node == end_node:
        return 0

    # Initial vertex distances are set to float("inf")inity
    dijk_vertices = {
        x: float("inf") for x in all_edges if x != start_node
    }
    # Updating known direct path costs
    for (a,b) in all_paths:
        if a == start_node:
            dijk_vertices[b] = all_path_costs[(a,b)]

    # List that contains unchecked nodes
    not_checked = list(dijk_vertices.keys())
    # The loop will continue until all nodes have been checked
    while not_checked:
        # Selection of unchecked node with the lowest cost
        min_node = -1
        min_val = float("inf")
        for node in not_checked.copy():
            node_val = dijk_vertices[node]
            if node_val < min_val:
                min_val = node_val
                min_node = node
        # Updating vertex distances whenever a lower value is found
        for (a,b) in all_paths:
            if a == min_node and b != start_node:
                target_val = dijk_vertices[b]
                new_val = dijk_vertices[a] + all_path_costs[(a,b)]
                if new_val < target_val:
                    dijk_vertices[b] = new_val
        # Take off the unchecked list
        not_checked.remove(min_node)

    return dijk_vertices[end_node]
```

h(x):Euclidean distance

```python
# Heuristic Algorithm, used in A* as h()
# A simple Euclidean distance calculator
def heuristic(start_node, end_node):
    # Ensure proper formatting
    start_node = int(start_node)
    end_node = int(end_node)

    # Skip if the source and destination are the same
    if start_node == end_node:
        return 0

    # Obtain the coordinates for the given nodes
    start_coord = all_coords[start_node]
    end_coord = all_coords[end_node]
    # Obtain Euclidian distances
    result = (start_coord[0] - end_coord[0]) ** 2 + ₩
             (start_coord[1] - end_coord[1]) ** 2
    result = sqrt(result)
    return result
```

- $f(x) = g(x) + h(x)$

Result

Comparing cost of possible path

```python
# apply A star algorithm
import itertools


permuts = list(itertools.permutations([16,7],2))
# print(permuts)

initial=0
final=19

best_path = []
temp_path = []
best_permut = tuple()

best_cost = float("inf")
for _permut in permuts:
    total_cost = 0
    for i, n in enumerate(_permut):
        if i == 0:
            _path = a_star(initial, n)
            total_cost += len(_path)
            temp_path.append(_path)
        else:
            _path = a_star(_permut[i-1], n)
            total_cost += len(_path)
            temp_path.append(_path)

    _path = a_star(n,final)
    total_cost += len(_path)
    temp_path.append(_path)

    if total_cost < best_cost:
        best_cost = total_cost
        best_path = temp_path
        best_permut = _permut

    temp_path = []


print(f'best path: {initial} -> {best_permut} -> {final}')
print(f'best path_detial: {best_path}')
print(f'best cost: {best_cost}')
```

Result

```
best path: 0 -> (7, 16) -> 19
best path_detial: [[(0, 1), (1, 2), (2, 7)], [(7, 12), (12, 17), (17, 16)], [(16, 17), (17, 18), (18, 19)]]
best cost: 9
```

Path1: 0 -> 7 -> 16 -> 19
Path2: 0 -> 16 -> 7 -> 19

Path1 is the shortest path!

Conclusion

1) Calculation of shortest path by applying the A* algorithm is done with the purpose of finding the shortest path from vehicle location into several restaurants

2) In real life problem, we should consider lots of constraints such as traffic conditions, food freshness, weather conditions, etc.

3) If we had more objects and transfer nodes, we might need more efficient model to solve np-hard problem.

Thank you for listening!