COMSATS University Islamabad (CUI)

Project Report

for

SUPPLY CHAIN MANAGEMENT (Optimized AI Searches)

(The main purpose is to achieve the optimized way of searching paths and finding nodes)

Version 1.0

By

Hasaan Ahmed SP22-BSE-017

Mona Khalil SP22-BSE-028

Submitted To:

Sir Waqas Ali

Bachelor of Science in Computer Science (2022-2025)

Contents

Executive Summary:

Our project aims to revolutionize supply chain management through an AI-driven system developed using Python. By employing advanced algorithms like Uniform Cost Search, Depth-First Search, Breadth-First Search, and A*, our solution optimizes decision-making processes, enhances inventory management, and ensures real-time stock updates. With a focus on reducing procurement lead times, improving operational efficiency, and boosting user experience, our software promises to elevate organizational productivity and adaptability in the face of market dynamics. Through comprehensive tools and technologies such as VS Code and Python 3.12.3, we provide a user-friendly solution tailored to meet the evolving needs of modern businesses.

SUPPLY CHAIN MANAGEMENT:

1. Introduction

In today's highly competitive commercial landscape, effective supply chain management is crucial for organizational success. This project aims to develop an AI-powered supply chain management system using Python. By leveraging AI algorithms such as Uniform Cost Search (UCS), Depth-First Search (DFS), Breadth-First Search (BFS), and A*, the system will enable real-time analysis and strategic planning. These algorithms will intelligently navigate the supply chain network to optimize decision-making processes, identify cost-effective routes, improve overall performance, and optimize inventory management. Our AI-driven solution will empower businesses to adapt swiftly to market fluctuations

2. Problem Statement

The current supply chain management system lacks real-time updates on stock availability, leading to dissatisfaction and inefficiency among employees. As a result, the procurement process is delayed and inconsistent, with staff spending an average of thirty minutes daily traversing the supply chain. This impacts job satisfaction and productivity. To address these issues, we aim to develop software that provides real-time stock updates, streamlines procurement, and reduces the time required to acquire supplies. Our goal is to enhance the working environment, increase operational efficiency, improve supply chain management, and save time and effort.

3. Problem Solution/Objectives of the Proposed System

Our software solution aims to enhance procurement procedures, implement real-time inventory tracking, and improve user experience to optimize supply chain management. We seek to reduce lead times for ordering goods, enhance operational effectiveness by providing precise stock level information, and prioritize flexibility, scalability, and usability to accommodate the company's changing needs. By leveraging data security protocols and collaboration tools, we aim to promote creativity, teamwork, and well-informed decision-making. Ultimately, our objective is to develop a comprehensive solution that boosts productivity, reduces expenses, and generates long-term value for the company.

4. CODE:

4.1 CSV FILE:

supply_chain_data.csv

```python
import csv
import time

class Node:
    def __init__(self, data):
        self.data = data
        self.children = []
        self.type = data.get('Type', None)  # Adjusted for 'Type' attribute

    def add_child(self, child):
        self.children.append(child)

    def __repr__(self, level=0):
        ret = "\t"*level+repr(self.data)+"\n"
        for child in self.children:
            ret += child.__repr__(level+1)
        return ret

data = {}
with open('./supply_chain_data.csv', 'r') as file:
    reader = csv.DictReader(file)
    for row in reader:
        data[row['SKU']] = row

root = Node(data['SKU0'])
for key, value in data.items():
    if key == 'SKU0':
        continue
    node = Node(value)
    root.add_child(node)
```

```python
# Search Node Through SKU
def search_node(root):
    sku = input("Enter SKU: ")
    if sku not in data:
        print("Invalid SKU. Please try again.")
        return
    node = _search_node(root, sku)
    if node:
        print_node_info(node)
    else:
        print("Node not found.")

def _search_node(node, sku):
    if node.data['SKU'] == sku:
        return node
    for child in node.children:
        result = _search_node(child, sku)
        if result:
            return result
    return None

# Search using A* considering Availability being the heuristic
def search_node_a_star(root):
    sku = input("Enter SKU: ")
    if sku not in data:
        print("Invalid SKU. Please try again.")
        return
    open_list = [root]
    closed_list = []
    while open_list:
        current = min(open_list, key=lambda node: node.data['Availability'])
        open_list.remove(current)
        closed_list.append(current)
        if current.data['SKU'] == sku:
            print_node_info(current)
            return
```

```
            for child in current.children:
                if child not in closed_list:
                    open_list.append(child)
        print("Node not found.")

    # Search using BFS
    def search_node_bfs(root):
        sku = input("Enter SKU: ")
        if sku not in data:
            print("Invalid SKU. Please try again.")
            return
```

```
    queue = [root]
    while queue:
        current = queue.pop(0)
        if current.data['SKU'] == sku:
            print_node_info(current)
            return
        for child in current.children:
            queue.append(child)
    print("Node not found.")

# Search using DFS
def search_node_dfs(root):
    sku = input("Enter SKU: ")
    if sku not in data:
        print("Invalid SKU. Please try again.")
        return
    stack = [root]
    while stack:
        current = stack.pop()
        if current.data['SKU'] == sku:
            print_node_info(current)
            return
        for child in current.children:
            stack.append(child)
    print("Node not found.")

# Search using UCS considering Revenue generated being the cost also print costs being compared to show how its working
def search_node_ucs(root):
    sku = input("Enter SKU: ")
    if sku not in data:
        print("Invalid SKU. Please try again.")
        return
    queue = [(root, 0)]
    print("Costs being compared: ")
    while queue:
```

```
        current, cost = queue.pop(0)
        print(cost, current.data['Costs'])
        if current.data['SKU'] == sku:
            print_node_info(current)
            return
        for child in current.children:
            queue.append((child, cost + float(child.data['Costs'])))
    print("Node not found.")

# Search from one node to another node having cost as heuristic using A* algorithm and print PATH from start to goal
def search_path_cost(root):
    start_sku = input("Enter Start SKU: ")
    if start_sku not in data:
        print("Invalid Start SKU. Please try again.")
        return
    goal_sku = input("Enter Goal SKU: ")
    if goal_sku not in data:
        print("Invalid Goal SKU. Please try again.")
        return
    open_list = [(root, 0)]
    closed_list = []
    while open_list:
        current, cost = min(open_list, key=lambda x: x[1])
        open_list.remove((current, cost))
        closed_list.append((current, cost))
        if current.data['SKU'] == goal_sku:
            print("Path from start to goal: ")
            for node, _ in closed_list:
                print(node.data['SKU'])
            return
        for child in current.children:
            if child not in [x[0] for x in closed_list]:
                open_list.append((child, cost + float(child.data['Costs'])))
    print("Path not found.")
```

```
# Search using Best-First Search (BFS) with a specified heuristic function
def search_node_best_first(root, heuristic_fn):
    sku = input("Enter SKU: ")
    if sku not in data:
        print("Invalid SKU. Please try again.")
        return
    queue = [(root, heuristic_fn(root))]
    closed_set = set()
    while queue:
        queue.sort(key=lambda x: x[1])
        current, _ = queue.pop(0)
        if current.data['SKU'] == sku:
            print_node_info(current)
            return
        closed_set.add(current)
        for child in current.children:
            if child not in closed_set:
                queue.append((child, heuristic_fn(child)))
    print("Node not found.")

# Define a heuristic function for Best-First Search (BFS)
def availability_heuristic(node):
    # Example heuristic: prioritize nodes with higher availability
    return node.data['Availability']

# Function to print node information
def print_node_info(node):
    print("----------------------------------------")
    print("----------------------------------------")
    print("\nSKU:", node.data['SKU'])
    print("Product Type:", node.data['Product type'])
    print("Price:", node.data['Price'])
    print("Availability:", node.data['Availability'])
    print("Available Stock:", node.data['Stock levels'])
    print("Quantity Available:", node.data['Order quantities'])
    print("\n----------------------------------------")
```

```python
# Main menu function
def menu():
    while True:
        print("-------------------------------------------------")
        print("-------------------------------------------------\n")
        print("SUPPLY CHAIN MANAGEMENT SYSTEM")
        print("1. Search Node Through SKU (e.g SKU2, SKU45 etc)")
        print("2. Search Node Through A*")
        print("3. Search Node Through BFS")
        print("4. Search Node Through DFS")
        print("5. Search Node Through UCS")
        print("6. Best-First Search (BFS)")
        print("7. Finding path from one node to another node heuristic as the cost")
        print("0. Exit\n")
        print("-------------------------------------------------")
        print("-------------------------------------------------\n")

        try:
            choice = int(input("Enter your choice: "))
            if choice == 0:
                break
            elif choice == 1:
                search_node(root)
            elif choice == 2:
                search_node_a_star(root)
            elif choice == 3:
                search_node_bfs(root)
            elif choice == 4:
                search_node_dfs(root)
            elif choice == 5:
                search_node_ucs(root)
            elif choice == 6:
                search_node_best_first(root, availability_heuristic)
            elif choice == 7:
                search_path_cost(root)
```

```python
            else:
                print("Invalid choice. Please try again.")
        except Exception as e:
            print("An error occurred: ", e)


if __name__ == '__main__':
    menu()
```

4.2 OUTPUTS:

```
PS C:\Users\DELL\Downloads\Mid Term Lab\Mid Term Lab> & C:/Users/DELL/AppD
--------------------------------------------------
--------------------------------------------------

SUPPLY CHAIN MANAGEMENT SYSTEM
1. Search Node Through SKU (e.g SKU2, SKU45 etc)
2. Search Node Through A*
3. Search Node Through BFS
4. Search Node Through DFS
5. Search Node Through UCS
6. Best-First Search (BFS)
7. Finding path from one node to another node heuristic as the cost
0. Exit

--------------------------------------------------
--------------------------------------------------

Enter your choice:
```

```
SUPPLY CHAIN MANAGEMENT SYSTEM
1. Search Node Through SKU (e.g SKU2, SKU45 etc)
2. Search Node Through A*
3. Search Node Through BFS
4. Search Node Through DFS
5. Search Node Through UCS
6. Best-First Search (BFS)
7. Finding path from one node to another node heuristic as the cost
0. Exit

--------------------------------------------------
--------------------------------------------------

Enter your choice: 1
Enter SKU: SKU3
----------------------------------------
----------------------------------------

SKU: SKU3
Product Type: skincare
Price: 61.16334302
Availability: 68
Available Stock: 23
Quantity Available: 59

----------------------------------------
----------------------------------------
--------------------------------------------------
--------------------------------------------------
```

```
--------------------------------------------------
--------------------------------------------------

Enter your choice: 2
Enter SKU: SKU2
----------------------------------------
----------------------------------------

SKU: SKU2
Product Type: haircare
Price: 11.31968329
Availability: 34
Available Stock: 1
Quantity Available: 88

----------------------------------------
----------------------------------------
--------------------------------------------------
--------------------------------------------------

SUPPLY CHAIN MANAGEMENT SYSTEM
```

```
3. Search Node Through BFS
4. Search Node Through DFS
5. Search Node Through UCS
6. Best-First Search (BFS)
7. Finding path from one node to another node heuristic as the cost
0. Exit

------------------------------------------------
------------------------------------------------

Enter your choice: 3
Enter SKU: SKU4
----------------------------------------
----------------------------------------

SKU: SKU4
Product Type: skincare
Price: 4.805496036
Availability: 26
Available Stock: 5
Quantity Available: 56

----------------------------------------
```

```
   4. Search Node Through DFS
   5. Search Node Through UCS
   6. Best-First Search (BFS)
   7. Finding path from one node to another node heuris
   0. Exit

   ------------------------------------------------
   ------------------------------------------------

   Enter your choice: 4
   Enter SKU: SKU55
   ----------------------------------------
   ----------------------------------------

   SKU: SKU55
   Product Type: haircare
   Price: 79.85505834
   Availability: 16
   Available Stock: 97
   Quantity Available: 11

   ----------------------------------------
   ----------------------------------------
   ------------------------------------------------
   ------------------------------------------------
```

```
4. Search Node Through DFS
5. Search Node Through UCS
6. Best-First Search (BFS)
7. Finding path from one node to ano
0. Exit

----------------------------------------
----------------------------------------

Enter your choice: 5
Enter SKU: SKU9
Costs being compared:
0 187.7520755
503.0655791 503.0655791
141.9202818 141.9202818
254.7761592 254.7761592
923.4406317 923.4406317
235.4612367 235.4612367
134.3690969 134.3690969
802.0563118 802.0563118
505.5571342 505.5571342
995.9294615 995.9294615
----------------------------------------
----------------------------------------

SKU: SKU9
Product Type: skincare
Price: 64.01573294
Availability: 35
Available Stock: 14
Quantity Available: 83

----------------------------------------
```

```
4. Search Node Through DFS
5. Search Node Through UCS
6. Best-First Search (BFS)
7. Finding path from one node to another node heuristi
0. Exit

--------------------------------------------------
--------------------------------------------------

Enter your choice: 6
Enter SKU: SKU75
----------------------------------------
----------------------------------------

SKU: SKU75
Product Type: skincare
Price: 92.99688423
Availability: 29
Available Stock: 16
Quantity Available: 56

----------------------------------------
----------------------------------------
--------------------------------------------------
--------------------------------------------------
```

```
4. Search Node Through DFS
5. Search Node Through UCS
6. Best-First Search (BFS)
7. Finding path from one node to another node heuristic as the cost
0. Exit

------------------------------------------------
------------------------------------------------

Enter your choice: 7
Enter Start SKU: SKU2
Enter Goal SKU: SKU6
Path from start to goal:
SKU0
SKU88
SKU58
SKU28
SKU11
SKU15
SKU6
------------------------------------------------
------------------------------------------------
```

5. Working of Algorithms:

1. **A Search Algorithm*:**
    - User inputs SKU to search.
    - Checks if SKU exists, if not, informs the user.
    - Initializes open and closed lists.
    - Selects nodes based on minimum availability.
    - If goal found, prints node information and exits.
    - If not, expands nodes and continues.
    - Informs if SKU not found.

2. **Breadth-First Search (BFS)**:
    - User inputs SKU to search.
    - Checks if SKU exists, if not, informs the user.
    - Initializes queue with root node.
    - Explores nodes level by level.
    - If goal found, prints node information and exits.
    - If not, continues exploring.
    - Informs if SKU not found.

3. **Depth-First Search (DFS)**:
    - User inputs SKU to search.
    - Checks if SKU exists, if not, informs the user.
    - Initializes stack with root node.
    - Explores nodes deeply before siblings.
    - If goal found, prints node information and exits.
    - If not, continues exploring.
    - Informs if SKU not found.

4. **Uniform Cost Search (UCS)**:
    - User inputs SKU to search.
    - Checks if SKU exists, if not, informs the user.
    - Initializes priority queue with root node and cost.
    - Explores nodes with lowest cost.
    - If goal found, prints node information and exits.
    - If not, continues exploring.
    - Informs if SKU not found.

5. **Finding Path with Heuristic Cost**:
    - User inputs start and goal SKUs.
    - Checks if both exist, if not, informs the user.
    - Initializes open and closed lists.
    - Explores nodes based on lowest combined cost and heuristic.
    - If goal found, prints path and exits.
    - If not, continues exploring.
    - Informs if path not found.

6. Tools and Technologies

Table 1: Tools and Technologies for Proposed Project

Tools And Technologies	**Tools**	**Version**	**Rationale**
	VS CODE	1.88	IDE
	Technology	**Version**	**Rationale**
	Python	3.12.3	AI Searches Algorithms
Dataset	**Name**	**Rationale**	
	Supply Chain CSV	Supply Chain Example dataset