Documentation: Prolog Campus Delivery System

12.01.2024

Introduction

The Prolog Campus Delivery System is designed to facilitate the delivery of various items across a university campus. This system, implemented in Prolog, manages delivery assignments by calculating the best routes and matching deliveries with available personnel based on their capacities and work hours.

What the Code Does

1. **Campus Layout**: The code defines a model of the campus, including various nodes (locations) and edges (paths between locations) with associated travel times.
2. **Delivery Personnel**: It maintains a roster of delivery personnel, each with specific attributes like capacity, work hours, and current location.
3. **Objects for Delivery**: Objects (items to be delivered) are defined with attributes like weight, pickup and drop-off locations, urgency, and assigned delivery person (if any).
4. **Dijkstra's Algorithm**: To find the shortest path for deliveries, the code employs Dijkstra's algorithm, a famous algorithm for finding the shortest paths between nodes in a graph.
5. **Matching Deliveries**: The system checks for available delivery personnel based on the object's weight and required delivery time, ensuring the selected person can complete the delivery within their work hours.
6. **Assigning and Delivering**: Functions are available to assign delivery tasks to personnel and to perform the delivery, updating the system's state accordingly.

How to Run the Code

1. **Setup Prolog Environment**: Ensure a Prolog interpreter (like SWI-Prolog) is installed.
2. **Load the Program**: Start the Prolog environment and load the program file. In linux you can use **swipl <filename>** and then call the necessary functions to execute required actions.
3. **Initialization**: Run **assert_all.** to initialize all predefined objects and personnel into the Prolog database. If you want to assert your own objects, you can use assert(<yourAssertion>). In console or you can write them to the source code and then use assert_all to assert them.
4. **Execute Functions**: Use Prolog queries to assign deliveries, check available personnel, and perform deliveries. For example, **available_delivery_person(ObjectID, DeliveryPersonID, TotalTime)** checks for available personnel for a given object. More on the function info later on the document.

How to Use the Program

This section details the steps for using the Prolog Campus Delivery System, including adding new objects and personnel, checking availability, and assigning deliveries.

1. **Asserting a New Object**:
    - To add a new object to the system, use the **object/6** predicate.
    - Syntax: **object(ID, Weight, Pickup, DropOff, Urgency, DeliveryPersonID).**
    - Example: **assert(object(o24, medium, cafeteria, library, high, none)).**
        - This adds an object with ID **o24**, medium weight, to be picked up from **cafeteria** and dropped off at **library**, with high urgency and no assigned delivery person.
2. **Asserting a New Delivery Person**:
    - To add a new delivery person, use the **delivery_person/5** predicate.
    - Syntax: **delivery_person(ID, Capacity, WorkHours, CurrentJob, Location).**
    - Example: **assert(delivery_person(dp11, large, [8, 12, 16], none, admin_office)).**
        - This adds a delivery person with ID **dp11**, large capacity, available at hours **8, 12, 16**, currently not assigned to any job (**none**), and located at the **admin_office**.
3. **Checking Availability for an Object**:
    - To check which delivery persons are available for a specific object, use the **available_delivery_person/3** predicate.
    - Syntax: **available_delivery_person(ObjectID, DeliveryPersonID, TotalTime).**
    - Example: **available_delivery_person(o24, DeliveryPersonID, TotalTime).**
        - This query will list available delivery persons for object **o24** along with the total time needed for delivery.
4. **Checking Delivery Status of an Object**:
    - To check the delivery status of an object, use the **check_delivery_status/3** predicate.
    - Syntax: **check_delivery_status(ObjectID, DeliveryPersonID, _).**
    - Example: **check_delivery_status(o24, DeliveryPersonID, _).**
        - This checks if object **o24** is already being delivered, and if so, by which delivery person.

5. **Assigning a Person to an Object**:
    - To assign a delivery task to a person, use the **assign_delivery_person/3** predicate.
    - Syntax: **assign_delivery_person(ObjectID, DeliveryPersonID, TotalTime).**
    - Example: **assign_delivery_person(o24, dp11, TotalTime).**
        - This assigns the delivery person **dp11** to the object **o24** and calculates the total time needed for the delivery.

Parameter Range Sets for Functions

- **node(Name)**: Defines a campus location. **Name** is an atom representing the location.
- **edge(Place1, Place2, Time)**: Defines a path between two nodes with a travel time. **Place1** and **Place2** are atoms, **Time** is an integer.
- **delivery_person(ID, Capacity, WorkHours, CurrentJob, Location)**: Represents a delivery person. **ID** is an atom, **Capacity** is one of **[large, medium, small]**, **WorkHours** is a list of integers, **CurrentJob** is the current delivery task or **none**, and **Location** is a node.
- **object(ID, Weight, Pickup, DropOff, Urgency, DeliveryPersonID)**: Represents an object to be delivered. **ID** is an atom, **Weight** is one of **[heavy, medium, light]**, **Pickup** and **DropOff** are nodes, **Urgency** is an atom, and **DeliveryPersonID** is the assigned personnel or **none**.
- **dijkstra(Start, End, Time, Path)**: Calculates the shortest path. **Start** and **End** are nodes, **Time** is the total travel time, and **Path** is the list of nodes in the path.
- **available_delivery_person(ObjectID, DeliveryPersonID, TotalTime)**: Checks available personnel for an object. **ObjectID** is the object's ID, **DeliveryPersonID** is the available personnel, and **TotalTime** is the time required for delivery.
- **assign_delivery_person(ObjectID, DeliveryPersonID, TotalTime)**: Assigns a delivery task. **ObjectID** is the object's ID, **DeliveryPersonID** is the personnel's ID, and **TotalTime** is the delivery time.
- **deliver(ObjectID)**: Performs the delivery of an object. **ObjectID** is the object's ID.

Conclusion

I tried to write an efficient system to manage deliveries in a campus. By leveraging Dijkstra's algorithm, the system ensures optimal routing, while its matching and assignment logic guarantees that deliveries are conducted effectively within the constraints of delivery personnel capacities and working hours.

Salih Karagöllü

210104004069