TEMOO Cargo

Executive Summary

Axyl, Depanshu, and Kevin have been contracted by Temoo to develop an expedited shipping service for delivering cargo to major airports across Canada using cargo planes. Each plane departs from a central hub airport (e.g., Vancouver) and must return to this hub after completing deliveries.

The system must determine a routing strategy for all planes within a specified time period from the central hub airport, considering various constraints. New packages arrive at predetermined time throughout the day at this hub and must be delivered to their respective destinations before their deadline time

Constraints:

- All planes have the same fuel capacity and same weight capacity.
- A fixed number of planes is available for deliveries, which cannot be used after they've been used once.
- Packages vary in weight, arrival time and destination.
- Each package must be delivered by its deadline time

Optimization:

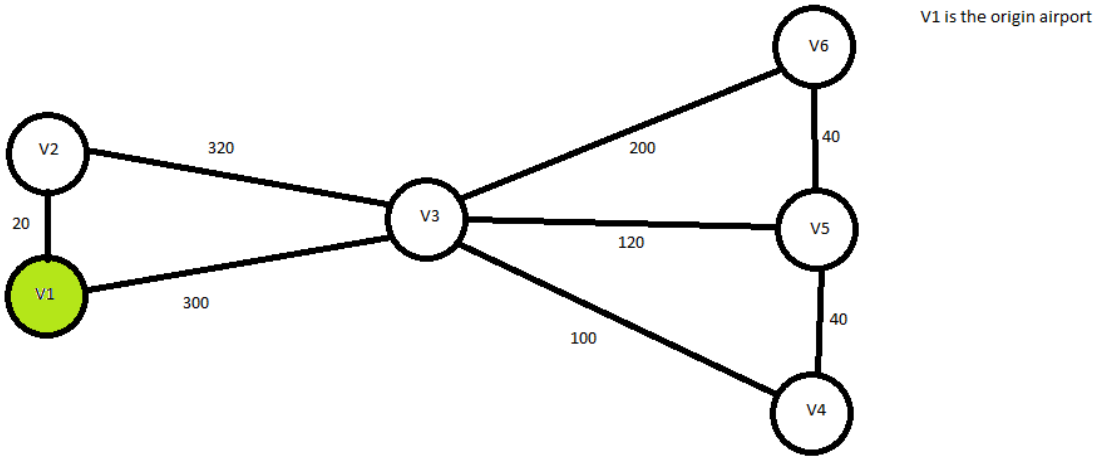
- Routes should be optimized to minimize operational costs, considering distance for delivery efficiency.

Examples

Our example follows the format where given graph G = (V, E) where our vertices V = {v1, v2, v2 …}, where each vertex is a unique airport, and edges E = {{v1,v2, d1}, {v2,v3, d2}},....., where each edge has the 2 connecting vertices as well as a distance; we construct a map of for our flights. We are also given flight constraints in the following format: Planes = {weight_capacity: value1, speed: value2, num_of_planes: value3};

Example problem:

V = {v1,v2,v3,v4, v5, v6}, E={{v1,v2, 20},{v1,v3, 300},{v2,v3, 320},{v3,v4, 100},{v3,v5,120},{v3,v6,200},{v4,v5,40},{v5,v6,40}}



With planes:

Planes = {weight_capacity: 25, speed: 100, num_of_planes: 2}

And packages

Packages = {{v6, 25, t1}, {v6, 25, t2}, {v5, 25, t3}, {v4,25, t4}}

An example of a solution ->

```
Unset
SOLUTION:
Plane 1:

Flight 1 (0:00 AM):
  Route: 1 → 3
  Packages: 25

Flight 2 (3:00 AM):
  Route: 3 → 6
  Packages: 25

Flight 3 (5:00 AM):
  Route: 6 → 3
  Packages: 0
```

```
Flight 4 (7:00 AM):
  Route: 3 → 1
  Packages: 0

Flight 5 (10:00 AM):
  Route: 1 → 3
  Packages: 25

Flight 6 (13:00 AM):
  Route: 3 → 6
  Packages: 25

Flight 7 (15:00 AM):
  Route: 6 → 3
  Packages: 0

Flight 8 (17:00 AM):
  Route: 3 → 1
  Packages: 0

Plane 2:

Flight 1 (0:00 AM):
  Route: 1 → 3
  Packages: 25

Flight 2 (3:00 AM):
  Route: 3 → 5
  Packages: 25

Flight 3 (4:12 AM):
  Route: 5 → 3
  Packages: 0

Flight 4 (5:24 AM):
  Route: 3 → 1
  Packages: 0

Flight 5 (8:24 AM):
  Route: 1 → 3
  Packages: 25

Flight 6 (9:24 AM):
```

```
  Route: 3 → 4
  Packages: 25

Flight 7 (10:24 AM):
  Route: 4 → 3
  Packages: 0

Flight 8 (13:24 AM):
  Route: 3 → 1
  Packages: 0
```

Input Format

The input would be in the following format in a single file.

Example of the distance matrix which we receive in the file:

```
Unset
# Distance matrix between cities (where -1 represents no direct route)
0 10 15 -1 30
10 0 35 20 25
15 35 0 25 40
-1 20 25 0 50
30 25 40 50 0
```

Example of Package Data which we receive in the file:

```
Unset
# Package details: <Package ID>, <Weight>, <Arrival Time (timestamp in HH:MM)>,
<Destination City>
P001 100 00:34 2
P002 150 12:34 3
P003 80 23:55 1
P004 200 22:45 4
P005 50 09:34 5
```

Example of Shipping Constraints which we receive in the file:

```
Unset
# Number of planes available for shipping
planes 5

# Weight capacity of each plane
weight_capacity 500
plane_speed 100
```

Valid Solutions & Output Format

Solution will be outputted to a solution.txt in a format analogous to this example:

```
Unset
SOLUTION(Numbers are random):
Plane 1:

Flight 1 (8:20 AM):
  Route:  0 -> 1
  Packages: 1

Flight 2 (9:30 AM):
  Route: 1 -> 3
  Packages: 4, 5

Plane 2:

Flight 1 (8:45 AM):
  Route: 0 → 3
  Packages: 2

Flight 2 (9:30 AM):
  Route: 0 -> 3
  Packages: 2, 3
```

Programming Language/Development Environment (Imperative)

Javascript / Node

Programming Language/Development Environment (Functional)

Haskell

Summary of the Method of Solving

We will simulate different node traversals calculating transit time for each package. If the amount of time required in transit is over 24 hours, then that set of node traversals is rejected, and the program will continue to simulate other possible branches.

The tool may use a graph search algorithm (Dijkstra's or A* for shortest path or minimum spanning tree) to explore possible flight paths:

1. Generate Possible Flight Paths:
    - Consider all reachable nodes given plane speed and fuel constraints.
    - Prioritize shortest routes while ensuring all packages reach their destinations.
2. Detect Dead Ends and Backtrack:
    - If a route violates constraints (e.g., overweight cargo, deadline missed), it is abandoned.
    - The algorithm backtracks to find an alternative route.

Testing

Testing will involve:

1. Unit Testing for core components (graph traversal, scheduling logic).
2. Integration Testing for the full system workflow using manually created test cases.
3. Edge Case Testing, such as:
    - No valid delivery route available.

- Planes with insufficient capacity for given cargo.
- High congestion scenarios where multiple packages arrive simultaneously.

Testing documentation and test plans are available in a separate document.