GIT RECAP
• Check status since last commit:

```bash
$ git status
```

• Stage changes/add new files:

```bash
$ git add file_name
```

• Record changes (advisable for new units of code):

```bash
$ git commit -m "Relevant message here"
```

• Push to remote repository (so we can see your code):

```bash
$ git push origin master
```

• Lists all your commits:

```bash
$ git log
```
SIMULATION COMPONENTS
SERVICE AREAS & ROUTE PLANNING
SERVICE AREAS

- We need an abstract representation of roads layout and bin locations for the different areas.
- We need to model the roads between different locations and the time required to travel these.
- We need to account for the fact that some streets only allow one way traffic.
EXAMPLE

Leith Walk, Edinburgh; 20 bin locations. (bing.com)
GRAPH REPRESENTATION

- In mathematical terms such a collection of bin locations interconnected with street segments can be represented through a graph.

- A graph $G = (V,E)$ comprises a set of vertices $V$ that represent objects (bin locations/depot) and $E$ edges that connect different pairs of vertices (links/street segments).

- Graphs can be *directed* or *undirected*. 
UNDIRECTED GRAPHS

- Edges have no orientation, i.e. they are unordered pairs of vertices. That is there is a symmetry relation between nodes and thus \((a,b) = (b,a)\).
DIRECTED GRAPHS

- Edges have a direction associated with them and they are called *arcs* or directed edges.
- Formally, they are *ordered* pairs of vertices, i.e. \((a,b) \neq (b,a)\) if \(a \neq b\).
GRAPH REPRESENTATION IN YOUR SIMULATORS

- For our simulations we will consider *directed* graph representations of the service network.
- This will increase complexity, but is more realistic.
BACK TO THE EXAMPLE

This area...
CORRESPONDING GRAPH

...can be represented by

We numbered vertices & added node '0' for the depot.
WEIGHTED GRAPH

- We also need to model the distances between bin locations.
- We will use a *weighted graph* representation, where a number (weight) is associated to each arc.
- In our case weights will represent the average travel duration between two locations (vertices) in one direction, expressed in minutes.
WEIGHTED GRAPH

For our example, this may be
**INPUT**

- For each area, graph representation of bin locations and distances between them will be given in the input script (file) in *matrix* form.

- We will consider the lorry depot as location 0. For a service area with $N$ locations, an $N \times N$ matrix will be specified.

- The *roadsLayout* keyword will precede the matrix.

- Where there is no arc in the graph between two vertices we will use a -1 value in the matrix.
**FOR THE PREVIOUS EXAMPLE**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>9</td>
<td>-1</td>
<td>8</td>
<td>10</td>
<td>-1</td>
<td>...</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>0</td>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>...</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>...</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>...</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>...</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>4</td>
<td>0</td>
<td>...</td>
<td>-1</td>
<td>-1</td>
<td>...</td>
</tr>
<tr>
<td>19</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>...</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>...</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

*Note that the matrix is not symmetric.*
ROUTE PLANNING

- In each area the lorry is schedule at fixed intervals.
- The occupancy thresholds are used to decide which bins need to be visited.
- There are lorry weight and volume constraints that you may account for at the start when planning.
- Equally, you may decide on the fly, i.e. when lorry capacity exceeded.
- Naturally there are efficiency implications here. You need to chose the approach and explain why you did that.
- This is not something to argue for/against. The purpose is for you to think critically about different approaches.
ROUTE PLANNING

- Your goal is to compute shortest routes that service all bins exceeding occupancy thresholds at the minimum cost in terms of time.
- Remember all routes are circular, i.e. they begin and end at the depot.
- Some bins along a route may not need service.
- Thus it may be appropriate to work with an equivalent graph where vertices that do not require to be visited are isolated and equivalent arc weights are introduced.
- Sometimes it may be more efficient to travel multiple times through the same location, even if the route previously serviced bins that required that.
THE (MORE) CHALLENGING PART

- How to partition the service areas and find (almost) optimal routes that visit all vertices that require so with minimum cost?
- This is entirely up to you, but I will discuss some useful aspects next.
- You must justify your choice in the final report and comment appropriately the simulator code.
- You may wish to implement more than one algorithm.
USEFUL TERMINOLOGY

- A *walk* is a sequence of arcs connecting a sequence of vertices in a graph.
- A *directed path* is a walk that does not include any vertex twice, with all arcs in the same direction.
- A *cycle* is a path that starts & ends at the same vertex.
directed paths / cycle
USEFUL TERMINOLOGY

- A *trail* is a walk that does not include any arc twice.
- A trail may include a vertex twice, as long as it comes and leaves on different arcs.
- A *circuit* is a trail that starts & ends at the same vertex.
trail / circuit (tour)
There may be multiple paths that connect two vertices in a directed graph.

In a *weighted* graph the shortest path between two vertices is that for which the sum of the arc costs (weights) is the smallest.
SHORTEST PATHS

- There are several algorithms you can use to find the shortest paths on a given service network.

- A non-exhaustive list includes
  - Dijkstra's algorithm (single source),
  - Floyd-Warshall algorithm (all pairs),
  - Bellman-Ford algorithm (single source).

- Each of these have different complexities, which depend on the number of vertices and/or arcs.

- The size and structure of the graph will impact on the execution time.
FLOYD–WARSHALL ALGORITHM

- A single execution finds the lengths of the shortest paths between all pairs of vertices.
- The standard version does not record the sequence of vertices on each shortest path.
- The reason for this is the memory cost associated with large graphs.
- We will see however that paths can be reconstructed with simple modifications, without storing the end-to-end vertex sequences.
FLOYD–WARSHALL ALGORITHM

- Complexity is $O(N^3)$, where $N$ is the number of vertices in the graph.

  The core idea:

- Consider $d_{i,j,k}$ to be the shortest path from $i$ to $j$ obtained using intermediary vertices only from a set \{1,2,...,k\}.

- Next, find $d_{i,j,k+1}$ (i.e. with nodes in \{1,2,...,k+1\}).
  
  - This could be $d_{i,j,k+1} = d_{i,j,k}$ or
  
  - A path from vertex $i$ to $k+1$ concatenated with a path from vertex $k+1$ to $j$. 
FLOYD–WARSHALL ALGORITHM

- Then we can compute all the shortest paths recursively as

\[ d_{i,j,k+1} = \min(d_{i,j,k}, d_{i,k+1,k} + d_{k+1,j,k}). \]

- Initialise \( d_{i,j,0} = w_{i,j} \) (i.e. start from arc costs).

- Remember that in your case the absence of an arc between vertices is represented as a -1 value, so you will need to pay attention when you compute the minimum.
EXAMPLE

First let's increase vertex indexes by one, since we were starting at 0.
EXAMPLE (CONT'D)

$k = 0$

$k = 1$

$k = 2$

$k = 3$

$k = 4$

$k = 5$
EXAMPLE (CONT'D)

All shortest paths found at this step.
PSEUDOCODE

Denote \( \mathbf{d} \) the \( N \times N \) array of shortest path lengths.
Initialise all elements in \( \mathbf{d} \) with \( \infty \).

For \( i = 1 \) to \( N \)
  For \( j = 1 \) to \( N \)
    \( d[i][j] \leftarrow w[i][j] \) \quad // assign weights of existing arcs;
  
For \( k = 1 \) to \( N \)
  For \( i = 1 \) to \( N \)
    For \( j = 1 \) to \( N \)
      If \( d[i][j] > d[i][k] + d[k][j] \)
        \( d[i][j] \leftarrow d[i][k] + d[k][j] \)
      End If
FLOYD–WARSHALL ALGORITHM

- This will give you the lengths of the shortest paths between each pair of vertices, but not the entire path.

- You do not actually need to store all the paths, but you would want to be able to reconstruct them easily.

- The standard approach is to compute the shortest path tree for each node, i.e. the spanning trees rooted at each vertex and having the minimal distance to each other node.
**PSEUDOCODE**

Denote \texttt{d}, \texttt{nh} the $N \times N$ arrays of shortest path lengths and respectively the next hop of each vertex.

For $i = 1$ to $N$

\hspace{1em} For $j = 1$ to $N$

\hspace{2em} $d[i][j] \leftarrow w[i][j]$ \hspace{1em} // assign weights of existing arcs;

\hspace{2em} $nh[i][j] \leftarrow j$

For $k = 1$ to $N$

\hspace{1em} For $i = 1$ to $N$

\hspace{2em} For $j = 1$ to $N$

\hspace{3em} If $d[i][j] > d[i][k] + d[k][j]$

\hspace{4em} $d[i][j] \leftarrow d[i][k] + d[k][j]$

\hspace{4em} $nh[i][j] \leftarrow nh[j][k]$

End If
RECONSTRUCTING THE PATHS

To retrieve the sequence of vertices on the shortest path between nodes $i$ and $j$, simply run a routine like the following.

```plaintext
path ← i
While i != j
    i ← nh[i][j]
    append i to path
EndWhile
```
FINDING OPTIMAL ROUTES GIVEN A SET OF USER REQUIREMENTS

- Finding shortest paths between different bin locations is only one component of route planning.

- The problem you are trying to solve is a flavour of the Vehicle Routing Problem (VRP). This is a known *hard* problem.

- Simply put, an optimal solution may not be found in polynomial time and the complexity increases significantly with the number of vertices.
HEURISTIC ALGORITHMS

- Heuristics work well for finding solutions to hard problems in many cases.
- Solutions may not be always optimal, but good enough.
- Work relatively fast.
- When the number of vertices is small, a 'brute force' approach could be feasible.
- Guaranteed to find a solution (if there exists one), and this will be optimal.
CLARIFICATIONS

1. If returning to depot and having to immediately service other bins in the same schedule, should I check the occupancy status of all bins again?
   - No. This was not explicitly discussed in the handout, and can be debatable. For this exercise, check all bins at the beginning of a schedule, and plan according to their status even if performing multiple trips.

2. If visiting some bin locations on a path between two bins requiring service, should I output all that information?
   - No. This is useful to check your implementation is correct, but takes up memory.
CHOOSING ROUTE PLANNING ALGORITHMS

- You have complete freedom to choose what heuristic you implement, but
- make sure you document your choice and discuss its implication on system’s performance in your report.
- It is likely that you will need to compute shortest paths.
- Again, you can choose any algorithm for this task, e.g. Floyd-Warshall, Dijkstra, etc., but explain your choice.
- You can implement multiple solutions, as some may not work for any graph or will perform poorly.
- More about route planning next time.
ASSIGNMENT, PART 1

- Not mandatory, zero weighted (just for feedback)
- Short proposal document outlining planned simulator.
- You should be able to explain plans for:
  1. Handling command line arguments;
  2. Parsing and validation of input scripts;
  3. Generation, scheduling, and execution of events;
  4. Graph manipulation/route planning algorithms;
  5. Statistics collection;
  6. Experimentation support and results visualisation;
  7. Code testing.
ASSIGNMENT, PART 1

- No code will be checked.
- Have created 'doc' folder inside repository, copy 'proposal.pdf' inside, git push.

```sh
$ cd doc
$ git add proposal.pdf
$ git commit -m 'Added proposal document'
$ git push
```

- Deadline today, 7 Oct at 16:00.
QUESTIONS?