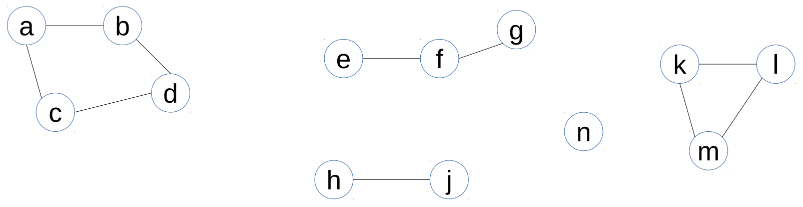

Figure 1: Example, villages and roads between them.

1 [10p]

Suppose we have a graph (V, R) with vertices V representing villages and edges $R \subseteq V \times V$ representing the roads between them, similar to the one in Figure 1 above. The road relation is *symmetric*, that is $\forall v_1, v_2 \in V \ ((v_1, v_2) \in R \Leftrightarrow (v_2, v_1) \in R)$.

1. [5 p] In the example above in Figure 1, we have the vertices $V = \{a, b, c, d, e, f, g, h, j, k, l, m, n\}$.
   Give the edge relation R for the example:

   $$R = \begin{array}{l}\{\ (a,b),(b,a),(a,c),(c,a),(b,d),(d,b),(c,d),(d,c),\\ \quad (e,f),(f,e),(f,g),(g,f),\\ \quad (h,j),(j,h),\\ \quad (k,l),(l,k),(k,m),(m,k),(l,m),(m,l)\}\end{array}$$

   Here, of course, the key is that the relation be symmetric, as per the definition above.

2. [5 p] Define, for any such graph (V, R) (not just for the example above!), the set D of dead-ends, i.e. the set of all villages one cannot leave (i.e. go to different village) at all by road, or by at most one road. In the example, this set would be $D = \{e, g, h, j, n\}$.

   $D = \{v \in V : \#R(v) \leq 1\}$

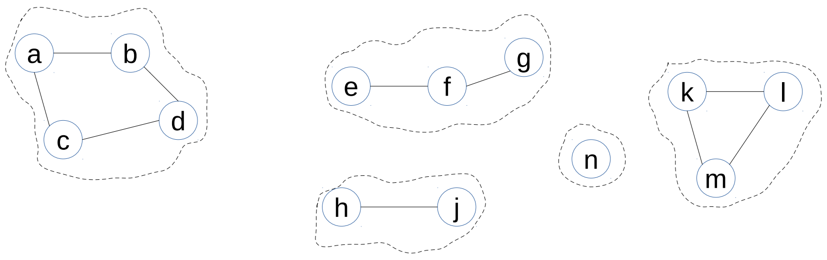

Figure 2: Grouping villages into islands

2 [10 p]

Suppose the villages and roads in (V, R) from task 1 are located on a set of islands, and each island corresponds exactly to the villages that can be reached from one another by road (i.e. all the villages on an island are connected by roads, not necessarily directly), as shown for our example in Figure 2.

Define for any such (V, R) the set A of islands, that is a set of sets of villages that can reach each other by road. In the example, we would have

$$A = \{\{a, b, c, d\}, \{e, f, g\}, \{h, j\}, \{k, l, m\}, \{n\}\}.$$

$$A = \{R[\{v\}] : v \in V\}$$

also, just for example,

$$\{R^+(v) \cup \{v\} : v \in V\}$$

The key here was to compute, for every vertex v, the set of vertices that can be reached from it, by iterating R starting from v (and to not forget to include the vertex itself, which is important if one uses the transitive closure R^+ of R, and the vertex is isolated like the village n in the example). The easiest way to do this is arguably a closure, but there are of course other options. Doing this for every vertex will typically compute the same set multiple times, because (due to symmetry), every vertex within an "island" can reach every other on the same island. However, adding the same set multiple times will not change the result, so there is no problem.

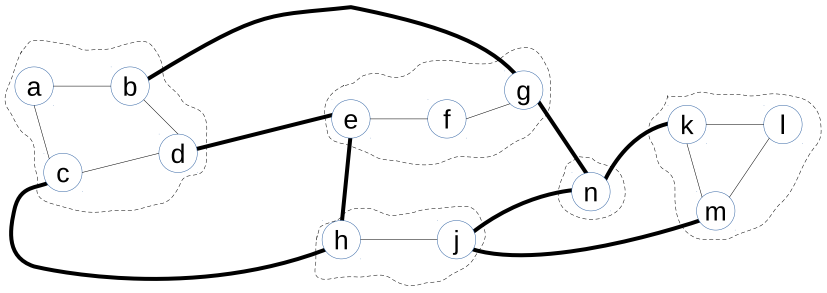

Figure 3: Our example villages and roads on the archilepago, with ferry connections between islands drawn in bolder lines.

3 [10 p]

Suppose that in addition to the villages in V and the symmetric relation of the roads connecting them R from task 1 we also have another *symmetric* relation $F \subseteq V \times V$ of ferry connections connecting villages on different islands, as in Figure 3. We assume that $F \cap R = \emptyset$, that is that there is no ferry connecting villages that are also connected by a road.

Define the set $Ports$ of ports, that is villages that have at least one ferry connection. In the example, this would be $Ports = \{b, c, d, e, g, h, j, k, m, n\}$.

$Ports =$ $F(V)$

also

$\text{dom } F$ or $\text{rng } F$ or $\{v \in V : F(v) \neq \emptyset\}$ and many others

4 [35 p]

Suppose the villages V, the roads $R \subseteq V \times V$, and the ferry connections $F \subseteq V \times V$, as in task 3. The combined connections between all the villages, using roads and ferries, is the relation $E = R \cup F$. (Recall that R and F are disjoint, so $R \cap F = \emptyset$.)

We want to define a function $P : V \times V \longrightarrow \mathcal{P}(V^*)$ such that $P(v_1, v_2)$ returns all cycle-free paths from v_1 to v_2 in E, that is all paths such that every vertex (village) occurs at most once in it. If $v_1 = v_2$, it returns a set containing only the empty path from v_1 to itself, that is $P(v_1, v_1) = \{v_1\}$.

We define P using a helper function $P' : V^* \times V \times V \longrightarrow \mathcal{P}(V^*)$ that builds the path as it searches for its destination:

$$P : V \times V \longrightarrow \mathcal{P}(V^*)$$

$$v_1, v_2 \mapsto P'(\varepsilon, v_1, v_2)$$

1. [25 p] Define P'. You might find it useful to talk about the set of all vertices in a sequence of vertices (that is, in a path) – if $p \in V^*$, then you can use $\mathrm{set}(p)$ to describe the set of all vertices that occur in p.

$$P' : V^* \times V \times V \longrightarrow \mathcal{P}(V^*)$$

$$p, v, w \mapsto \begin{cases} \{pv\} & \text{for } v = w \\ \bigcup\limits_{q \in E(v) \setminus \mathrm{set}(pv)} P'(pv, q, w) & \text{for } v \neq w \end{cases}$$

In this graph algorithm problem, the keys are (a) to figure out when to produce an output path (here: when v and w are equal) and terminate, (b) which direction(s) to continue the recursion into (here: $E(v)$), and (c) which directions to NOT go into (in this case: all the vertices seen up to this point, i.e. $\mathrm{set}(pv)$). Notice that in the above solution, (b) and (c) are combined, by ensuring that for any call $P'(p, v, w)$, it is always true that $v \notin \mathrm{set}(p)$ – which means we do not need to specially test for that case.

As is always the case with problems like this one, there are many ways in which to solve them. A common alternative, which separates dealing with (b) and (c) into distinct cases, went something like this:

$$p, v, w \mapsto \begin{cases} \{pv\} & \text{for } v = w \\ \bigcup_{q \in E(v)} P'(pv, q, w) & \text{for } v \neq w, v \notin \mathrm{set}(p) \\ \emptyset & \text{for } v \neq w, v \in \mathrm{set}(p) \end{cases}$$

Here, it is not always true that $v \notin \mathrm{set}(p)$, so we need to test this explicitly and catch a vertex that has already been visited. This in done in the last case, where we simply return an empty set of paths. In the first solution, P' would not be called on such a vertex, but the result is the same.

2. [10 p] Define the set $\Pi \subseteq V^*$ of all non-cyclic paths in (V, E), as computed by P:

$$\Pi = \bigcup_{v_1, v_2 \in V} P(v_1, v_2)$$

Other options include $P(V \times V)$ and things like $\{p : \exists v_1, v_2 \in V \ (p \in P(v_1, v_2))\}$ etc.

5 [20 p]

Suppose we have a function $d : E \longrightarrow \mathbb{R}^+$ from the combined connections to the positive real numbers, signifying for each $(v_1, v_2) \in E$ the time it takes to travel from v_1 to v_2 (by either road or ferry, depending on whether $(v_1, v_2) \in R$ or $(v_1, v_2) \in F$). For any $(v_1, v_2), (v_2, v_1) \in E$, it is NOT guaranteed that $d(v_1, v_2) = d(v_2, v_1)$.

Given a path $v_0 v_1 v_2 ... v_n$ of length n in E, we want to measure it regarding the total time it takes and also the number of ferry connections in the path. The function $M : \Pi \longrightarrow \mathbb{N} \times \mathbb{R}_0^+$ computes for each non-cyclic path $p \in \Pi$ (the set of all non-cyclic paths from the previous task) a pair (n, r), where n is the number of ferry connections and r the sum of the travel time of all the connections, by road or ferry, in the path.

For every $v \in V$, applying M to the empty path v results in $(0, 0)$, i.e. $\forall v \in V \ (M(v) = (0, 0))$.

Define M.

$M : \Pi \longrightarrow \mathbb{N} \times \mathbb{R}_0^+$

$$p \mapsto \begin{cases} (0,0) & \text{for } p = v \\ (n+1, r + d(v_1, v_2)) & \text{for } p = v_1 v_2 q \wedge (v_1, v_2) \in F \wedge M(v_2 q) = (n, r) \\ (n, r + d(v_1, v_2)) & \text{for } p = v_1 v_2 q \wedge (v_1, v_2) \in R \wedge M(v_2 q) = (n, r) \end{cases}$$

Hint: you might find it useful to distinguish cases where p has the form v (i.e. it is an empty path), from cases where it has the form $v_1 v_2 q$, i.e. a path with at least two vertices (i.e. of length at least 1).

This is a problem involving recursion (or otherwise iterating) over strings/finite sequences, in this case sequences representing paths. The first point here, and one that most answers got right, is the realization that one needs to distinguish between “ferry” and “road” edges connecting vertices in the path. The second is that while we need to look at the first two vertices (so we can figure out what kind of edge connects them), we must only “peel off” the first vertex, and keep recurring on the path beginning with the second. In other words, in the above formulation, the recursive call needs to be $M(v_2 q)$ rather than just $M(q)$.

Some solutions tried to get away with a non-recursive way of computing the two components of the result, which is okay if it is done correctly. Similarly, some solutions used position-wise addition on pairs, which is fine (some of you had asked me about this during the exam), e.g. like so:

$$p \mapsto \begin{cases} (0,0) & \text{for } p = v \\ (1, d(v_1, v_2)) + M(v_2 q) & \text{for } p = v_1 v_2 q \wedge (v_1, v_2) \in F \\ (0, d(v_1, v_2)) + M(v_2 q) & \text{for } p = v_1 v_2 q \wedge (v_1, v_2) \in R \end{cases}$$

There was considerable variation in the way people expressed manipulations of strings, ranging from the solution above to little bits of Clojure code involving `drop` and indexing into strings, like $p[0]$ and $p[1]$. I accepted all of those (if they were otherwise correct), even when they seemed to confuse strings and sets (assuming, again, that the meaning was otherwise clear and correct).

6 [15 p]

With $P : V \times V \longrightarrow \mathcal{P}(V^*)$ from task 4 we can compute the set $P(v_1, v_2)$ of all non-cyclic paths between two villages v_1 and v_2, and with $M : \Pi \longrightarrow \mathbb{N} \times \mathbb{R}_0^+$ from task 5 we can measure each path in terms of the number of ferry connections involved and its total time.

We are interested in the set of "best" paths, which we define as follows:

Suppose $M(p_1) = (n_1, r_1)$ and $M(p_2) = (n_2, r_2)$.
Path p_1 is "better" than p_2. written as $p_1 \prec p_2$, iff $(n_1 < n_2) \vee (n_1 = n_2 \wedge r_1 < r_2)$.

Using the function P defined in task 4, define the function $\hat{P} : V \times V \rightarrow \mathcal{P}(V^*)$ such that $\hat{P}(v_1, v_2)$ returns the set of best paths between v_1 and v_2.

Note:

1. $\forall v_1, v_2 \in V \ (\hat{P}(v_1, v_2) \subseteq P(v_1, v_2))$
2. $\forall v_1, v_2 \in V \ \forall p, q \in \hat{P}(v_1, v_2) \ (M(p) = M(q))$

$$\hat{P} : V \times V \longrightarrow \mathcal{P}(V^*)$$

$$v_1, v_2 \mapsto \{p \in P(v_1, v_2) : \neg \exists q \in P(v_1, v_2) \ (q \prec p)\}$$

A key to this problem was the realization that different paths might have the same M score, so one cannot make it a condition that $\forall q \in P(v_1, v_2)(p \prec q)$ – for one thing, this will never be true, since $p \in P(v_1, v_2)$ and $p \not\prec p$, but it is not much better to require that $\forall q \in P(v_1, v_2) \setminus \{p\}(p \prec q)$, since there might be other paths with the same M score.