**Logic Boat**

**Problem:**
THE BOAT ONLY HOLDS TWO, BUT YOU CAN'T LEAVE THE GOAT WITH THE CABBAGE OR THE WOLF WITH THE GOAT.

**Solution:**
1. TAKE THE GOAT ACROSS.
2. RETURN ALONE.
3. TAKE THE CABBAGE ACROSS.
4. LEAVE THE WOLF;
   WHY DID YOU HAVE A WOLF?
Robert Tarjan, another Turing awardee.

Giving a talk at Brown
December 6
module type QUEUE =
  sig
  type 'a queue
  val empty: 'a queue
  val is_empty: 'a queue -> bool
  val enqueue: 'a queue * 'a -> 'a queue
  val peek: 'a queue -> 'a
  val dequeue: 'a queue -> 'a queue
  end
module type SET =
  sig
    type 'a set
    val empty: 'a set
    val insert: 'a set * 'a -> 'a set
    val containsP: 'a set * 'a -> bool
  end
Say $f$ is a procedure:
- input: a binding environment,
- output: true if the environment includes a binding for "password".
Could take a long time.

Now say you have a list of environments.
Want to see if any environment in the list contains "password".

Use map to apply $f$ to every environment in the list?
Wasteful.
Instead, stop after first occurrence.

\[
\text{successP: } ('a \to \text{bool}) \times 'a\text{ list} \to \text{bool}
\]

**Quiz: Write it.**

```ocaml
let rec successP = function
    _,[] -> false
| f, first::rest ->
    if f first then true else successP (f, rest)
```
Say $f$ is a procedure:
- input: a binding environment,
- output: true if the environment includes a binding for "password".
Could take a long time.

Now say you have a list of environments.
Want to see if any environment in the list contains "password".

Use map to apply $f$ to every environment in the list?
Wasteful.
Instead, stop after first occurrence.

```
successP: ('a -> bool) * 'a list -> bool
```

**Quiz:** Write it.

```
let rec successP = function
    _,[] -> false
  | f, first::rest ->       f first || successP (f, rest)
```
Say \( f \) is a procedure:
- input: a binding environment,
- output: value to which "password" is bound

Now say you have a list of environments. Want to find binding of "password" in any environment.

Stop after first occurrence.

```
let rec first_success = function
    _,[] -> None
  | f,a::rest ->
      match f a with
        Some b -> Some b
        | None -> first_success (f,rest)
```

Quiz: Write it.

type 'a option = Some of 'a | None
Graph

vertices

edges

Neighbors of a vertex

Represent by procedure

**input**: a vertex

**output**: its neighbors

function 1 → [2; 3; 4]
| 2 → [5; 6; 3] |
| 6 → [5] |
Define procedure `pathP` that takes a graph, an origin vertex, and a destination vertex, and returns true if there is a path in the graph from origin to destination:

- a vertex sequence $v_1, v_2, \ldots, v_k$ such that $v_1$ is origin, $v_k$ is destination, and $v_i$ is neighbor of $v_{i-1}$ for $i=2, 3, \ldots, k$