ICFP programming contest 2017
Lambda punter (1.3)

ICFP programming contest organisers

4th August 2017

1 Introduction

This year's task is to efficiently transport lambdas around the world by punt. A punt is a kind of flat-bottomed boat propelled by a long pole.

As the picture illustrates, punts are ideal for transporting lambdas. This particular punt happens to be in Cambridge, but they are also the preferred form of transport in Oxford, where ICFP will be held this year.

Five years ago ICFP contest participants alleviated the worldwide shortage of lambdas by optimising the lifting of lambdas from the lambda mines of Fife:

`https://icfpcontest2012.wordpress.com/`

With the continued popularity of functional programming and the increasing demand for lambdas in imperative as well as functional programming languages the bottleneck has now moved from the mines to the transport network. As a punter, your job is to set up punting routes to transport lambdas from the lambda mines to programmers around the world. You will be competing with other punters. May the best lambda punter win!

1.1 Refinements

As the contest progresses the specification will be refined. We will make at most *one* refinement during the first 24 hours, and at most *four* refinements during the contest overall. We will make *no* refinements

in the 12 hours before either the lightning or full deadline.

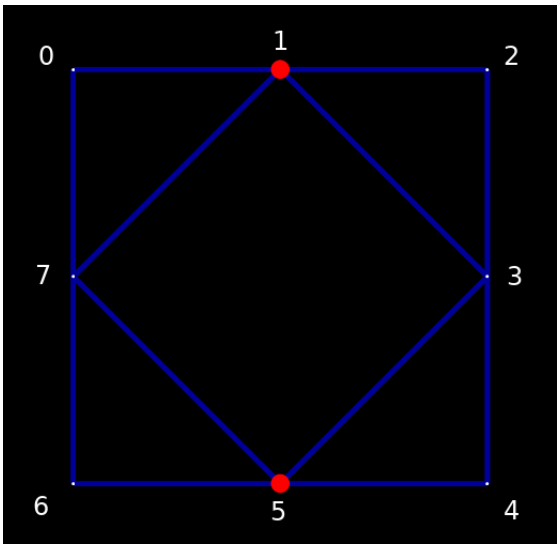
Details of any refinements will be posted in the following ways:

- via the contest web site `http://icfpcontest.org/`
- by email on the mailing list `icfp-contest2017@inf.ed.ac.uk`

2 Games

A lambda punter game is contested by n punters, for some $n \geq 2$, on a fixed map G. A map is an undirected graph given by a collection of sites (nodes) and rivers (edges) between them. Some sites are designated as mines. They are lambda producers. All sites are lambda consumers (lambdas make the world go round!). The goal is to build punting routes from the mines across as much of the map as possible.

Here is an example map.



Sites 1 and 5 are mines.

Punters take it in turns to claim a river or to pass. Once a river has been claimed by punter P, only punter P may use that river for transporting lambdas and no other punter can claim that river. The game ends when R moves have been made, where R is the total number of rivers on the map (not all rivers may have been claimed, as some punters may have passed).

We have provided a selection of sample maps, along with a simple visualiser. You can access these here:

`http://punter.inf.ed.ac.uk/graph-viewer/`

3 Scoring

Let the set of mines be $\{M_0, \ldots, M_{m-1}\}$ and the set of sites be $\{S_0, \ldots, S_{s-1}\}$. Each punter P is assigned a score at the end of the game, $score(P)$, computed as the sum of the scores for P at mine M, $score(P, M)$, for all mines M.

$$score(P) = score(P, M_0) + \cdots + score(P, M_{m-1})$$

The score for punter P at mine M, $score(P, M)$, is computed as the sum of the scores for P for the journey from M to each S, for every site S.

$$score(P, M) = score(P, M, S_0) + \cdots + score(P, M, S_{s-1})$$

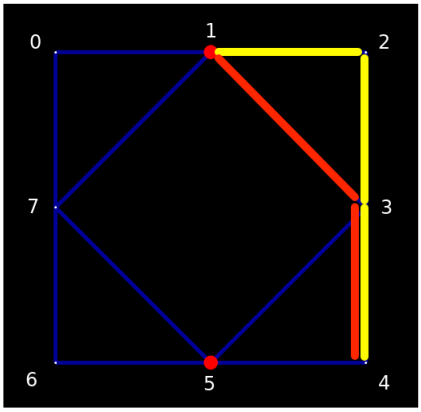
The score for punter P for the journey from M to S, $score(P, M, S)$, is

- 0 if there is no route from M to S along P's rivers; or
- $d \times d$, if there is a route along P's rivers, and d is the length of the shortest route between M and S along any rivers (whether or not they have been claimed by P or any other punter).

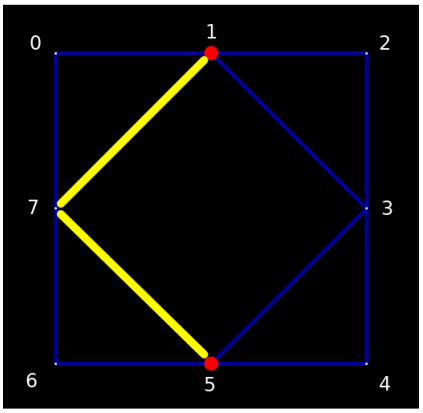
3.1 Examples

The following examples are based on the simple map in Section 2.

- If Alonso has rivers 1–2–3–4 (highlighted in yellow below), then Alonso's score for the journey from 1 to 4 is $2 \times 2 = 4$, as the shortest route (1–3–4, highlighted in red) has length 2.



- If Alonso has rivers 1–7–5 (highlighted in yellow below), then Alonso's total score is $(1+4)+(1+4)$ as he scores for the routes starting from each mine (1–7, 1–5, 5–7, 5–1).



3.2 Timeouts

Punters will be given a limited amount of time to respond to the server. The timeout timer begins immediately before the server sends a message to the punter and is stopped immediately after the server receives the response from the punter. Details are given in Section 4.

If a punter fails to move within the specified time, then they will be made to pass for that turn. The server will send a notification to a punter on each occasion that they time out.

3.3 Zombie punters

If a punter times out for **10** moves in a row then they become a zombie punter. The punter will be disconnected and all their remaining moves in the game will be forfeit.

4 The lambda punter protocol

The lambda punter protocol exchanges data using the JSON format. Lambda punter supports two modes: *online mode* and *offline mode*.

Online mode
Online mode supports concurrent punters running on different machines. Communication is via TCP/IP sockets.

Offline mode
Offline mode supports only one punter running at once. Thus the game state must be serialised between moves. The lambda punter server coordinates the punters, running each in turn and keeping hold of the game state for each punter alongside each message. Communication is via unix pipes.

After the contest has finished, the final evaluation will be performed exclusively in offline mode. This will allow us to ensure that every punter has access to equal resources and teams cannot gain an unfair advantage by buying up large amounts of cloud compute time. During the evaluation punters will have no external internet access.

4.1 Messages

Messages between the server and punter are encoded using the JSON format.

`http://www.json.org/`

Every message takes the form $n{:}json$, where n is a natural number encoded as a string of digits, and $json$ is a JSON string of exactly n bytes in size. The string representation of n must be no more than 9-digits in length, thus limiting the size of the $json$ string to 999999999 bytes (just under 1 GB). In the descriptions below we omit the n: prefix.

4.2 Online mode

We write $P \to S$ for a message sent from the punter to the server and $S \to P$ for a message sent from the server to the punter.

There are four phases to the protocol: handshake, setup, gameplay, and scoring.

0. **Handshake**

$$\begin{array}{ll} P \to S & \{\texttt{"me"} : name\} \\ S \to P & \{\texttt{"you"} : name\} \end{array}$$

$name$: String (punter name)

The punter initiates the conversation by supplying a name, e.g., {"me" : "Alonso"}. The server responds by repeating the name, e.g. {"you" : "Alonso"}. All subsequent interactions will be driven by the server.

1. **Setup**

(setup timeout timer begins here)
$S \to P$ {"punter" : p, "punters" : n, "map" : map}
$P \to S$ {"ready" : p}
(setup timeout timer ends here)

p : PunterId (punter id)
n : Nat (total number of punters)
map : Map (the map)

Pid = Nat
Map = {"sites" : [Site], "rivers" : [River], "mines" : [SiteId]}
Site = {"id" : SiteId}
River = {"source" : SiteId, "target" : SiteId}
SiteId = Nat

Once all punters are connected, the server broadcasts the initial game state to all of the punters. The initial game state consists of a unique punter id (p), the total number of punters (n), and the map (map). The punter ids are assigned sequentially from 0 up to $n - 1$. The map consists of a list of sites, a list of rivers, and a list of sites which are designated as mines. The map may also contain additional meta data (e.g., coordinates of sites, but any such meta data can be safely ignored). Each punter responds with a ready message containing their punter id.

2. **Gameplay**

$$
\begin{array}{ll}
 & \text{(gameplay timeout timer begins here)} \\
S \to P & \{\texttt{"move"} : \{\texttt{"moves"} : \mathit{moves}\}\} \\
P \to S & \mathit{move} \\
 & \text{(gameplay timeout timer ends here)}
\end{array}
$$

$$
\begin{array}{rl}
\mathit{moves} & : [\mathsf{Move}] \quad \text{(moves from previous turn)} \\
\mathit{move} & : \mathsf{Move} \quad (P\text{'s chosen move}) \\
\multicolumn{2}{l}{\mathsf{Move} = \{\texttt{"claim"} : \{\texttt{"punter"} : \mathsf{PunterId}, \texttt{"source"} : \mathsf{SiteId}, \texttt{"target"} : \mathsf{SiteId}\}\}} \\
\multicolumn{2}{l}{\quad\quad \mid \{\texttt{"pass"} : \{\texttt{"punter"} : \mathsf{PunterId}\}\}}
\end{array}
$$

In each turn each punter must make a single move. The server communicates with the punters in ascending order of punter id.

For each such interaction, punter P is prompted to move by the server, who sends a list of all moves made in the previous turn. This list will always contain one entry per punter. At the beginning of the game the previous move for each punter is initialised to a pass.

Having been prompted, the punter must now make a move: either claim a single river or pass. (For this communication, the `"punter"` field is technically redundant. The server will record when a punter is confused about their identity, but otherwise ignore this confusion.) If the punter makes an illegal claim the server will treat the move like a pass.

Gameplay continues until r moves have been made, where r is the total number of rivers on the map. (If some punters pass during the game then at the end of the game not all rivers will be claimed.)

3. **Scoring**

$$S \to P \quad \{\texttt{"stop"} : \{\texttt{"moves"} : \mathit{moves}, \texttt{"scores"} : \mathit{scores}\}\}$$

$$
\begin{array}{rl}
\mathit{moves} & : [\mathsf{Move}] \quad \text{(collection of moves)} \\
\mathit{scores} & : [\mathsf{Score}] \quad \text{(collection of scores)} \\
\multicolumn{2}{l}{\mathsf{Score} = \{\texttt{"punter"} : \mathsf{PunterId}, \texttt{"score"} : \mathsf{Nat}\}}
\end{array}
$$

The server notifies the punters that gameplay has ended by sending each in turn a `"stop"` message. This is accompanied by the final collection of moves as well as the final score for each punter. For convenience, P's last move and any other moves that have already been reported to P are converted into passes. This means that P can safely apply all of the reported moves to their internal game state without worrying about accidentally applying the same move twice.

4.3 Offline mode

In offline mode an encoding of the game state is passed in the `"ready"` message and then threaded through the gameplay messages. In addition, rather than having a single handshake, one is performed each time the punter binary is invoked.

1. **Setup**

$$\begin{array}{ll} P \to S & \{\texttt{"me"} : \mathit{name}\} \\ S \to P & \{\texttt{"you"} : \mathit{name}\} \\ & \text{(setup timeout timer begins here)} \\ S \to P & \{\texttt{"punter"} : p, \texttt{"punters"} : n, \texttt{"map"} : \mathit{map}\} \\ P \to S & \{\texttt{"ready"} : p, \texttt{"state"} : \mathit{state}\} \\ & \text{(setup timeout timer ends here)} \end{array}$$

$$\mathit{state} : \mathsf{GameState} \quad \text{(initial game state)}$$

   The variable state can be used to encode whatever game state the punter chooses using. At a minimum it should probably include an encoding of p, n, and map, otherwise it will be rather difficult to play the game!

2. **Gameplay**

$$\begin{array}{ll} P \to S & \{\texttt{"me"} : \mathit{name}\} \\ S \to P & \{\texttt{"you"} : \mathit{name}\} \\ & \text{(gameplay timeout timer begins here)} \\ S \to P & \{\texttt{"move"} : \{\texttt{"moves"} : \mathit{moves}\}, \texttt{"state"} : \mathit{state}\} \\ P \to S & \mathit{move} \uplus \{\texttt{"state"} : \mathit{state}'\} \\ & \text{(gameplay timeout timer ends here)} \end{array}$$

$$\begin{array}{rl} \mathit{state} : \mathsf{GameState} & \text{(game state before this move)} \\ \mathit{state}' : \mathsf{GameState} & \text{(game state after this move)} \end{array}$$

   The protocol for making a move starts with a handshake before the gameplay timer is started. Then the previous state is input by the punter alongside the move request and the updated state is output by the punter alongside the move. We write $\uplus$ for the binary disjoint union operator on JSON objects: move must be a JSON object that doesn't contain a `"state"` field and $\mathit{move} \uplus \{\texttt{"state"} : \mathit{state}'\}$ is move with an extra `"state"` field whose value is state'.

3. **Scoring**

$$\begin{array}{ll} P \to S & \{\texttt{"me"} : \mathit{name}\} \\ S \to P & \{\texttt{"you"} : \mathit{name}\} \\ S \to P & \{\texttt{"stop"} : \{\texttt{"moves"} : \mathit{moves}, \texttt{"scores"} : \mathit{scores}\}, \texttt{"state"} : \mathit{state}\} \end{array}$$

$$\mathit{state} : \mathsf{GameState} \quad \text{(game state after } P\text{'s final move)}$$

   The protocol for scoring starts with a handshake before the previous state is input alongside the stop message. There is no need for the punter to update the state again as this is the end of the game.

4.4 Timeouts

If a punter is too slow to respond then their move will be forfeit. In online mode, the server sends `"timeout"` message along with the length of the timeout for this phase.

$$S \to P \quad \{\texttt{"timeout"} : t\}$$

$$t : \mathsf{Float} \quad \text{(length of the timeout)}$$

In offline mode, the server kills the punter, and keeps track of the moves from the current turn so that they can be reported to the punter next turn. If a punter times out for several turns in a row then this list of moves accumulates. Whether in online or offline mode, if a punter times out **10** times then they become a zombie punter who always passes. The server makes no attempt to communicate with a punter once they have become a zombie.

In offline mode, the setup timeout is **10** seconds and the gameplay timeout is **1**. In online mode, timeouts may be more lax in order to accommodate human players.

5 Game servers

A collection of lambda punter servers will be made available for the duration of the contest in order to allow teams to test their implementations against one another while they develop their solutions.

You can find a status page detailing the active games at

`http://punter.inf.ed.ac.uk/status.html`

To connect to a game, open a connection to `punter.inf.ed.ac.uk:<port>`, where `<port>` is the port of the game you wish to connect to.

Additionally, if you wish to play a game yourself, you can find a web interface at

`http://punter.inf.ed.ac.uk/puntfx/`

6 Determining the winner

We will use the same procedure to determine the winner in both the lightning and full divisions.

The result for each game will be determined by ranking the punters by game score (the absolute game score does not matter). The winner of a game with n punters is awarded n points, the player who comes second $n - 1$ points, and so on. If two or more punters are ranked equally they all are awarded $n - k$ points where k is the number of punters having been ranked higher in the game.

For determining the overall winners there will be three rounds:

1. We will run each entry on a collection of small maps. Entries scoring below the median score will be eliminated.
2. We will then run each remaining entry on a number of larger maps. Again, entries scoring below the median score will be eliminated.
3. Finally, we will run each remaining entry on a number of fiendish maps. The entry with the largest score will be the winner.

The selection of punters in any given game will be randomised. In order to ensure that all punters play the same number of games in total in each round, some punters may be randomly drafted in to play extra games, but the results of the extra punters will not be taken into account in the final reckoning. We will announce the results of the first two elimination rounds in the weeks following the contest, and the overall winners at the International Conference on Functional Programming in Oxford.

You are free to create your own maps, and submit them to us if you wish. We may even use them to help judge the contest.

6.1 The judges' prize

The judges' prize will be picked by the judges. All entries in both the full and lightning divisions are eligible for the judges' prize.

7 Submission

In order to register your entry go here:

`http://punter.inf.ed.ac.uk:9000/register/`

Your submission must be a single `.tar.gz` file, named

`icfp-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.tar.gz`

where

`XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX`

is the 36-character registration code, obtained from registering through the web form.

To submit your entry, share it via Google Docs or Google Drive with one of the following accounts:

- `icfpcontest2017@gmail.com` for full submissions.
- `icfpcontest2017lightning@gmail.com` for lightning division submissions.

You can do this via the Google Docs web interface (`http://docs.google.com/`) by uploading the file, selecting the uploaded file then clicking **Share**. You may submit in either or both divisions, as you wish.

The virtual machine available at `https://icfpcontest2017.github.io/vm/` with 4 GB of ram and 2 CPU cores will be used for the evaluation.

The `.tar.gz` file will be unpacked into a unique user's home directory, and should contain (at least) the following:

- An executable file `./install`, which is executed exactly once.
- An executable file `./punter`, which may be generated by `./install`, and will be executed on each test map. It must comply with the offline protocol. The `./punter` executable will not have any access to the network. It should not access the filesystem either, except `stdin`, `stdout`, and `stderr`.
- A file `./PACKAGES` listing the names of any non-standard packages required by either your `./install` or `./punter` executables, one per line.
- A directory `./src`, containing your source code. We will not try to compile this, but we will use it to help choose the judges' prize.
- A file `./README`, listing your team members and (optionally) describing how your entry works. We will use this to help choose the judges' prize.

Ideally `./install` should not need network access. If `./install` needs access to the network, then please explain why in the `./README` file.

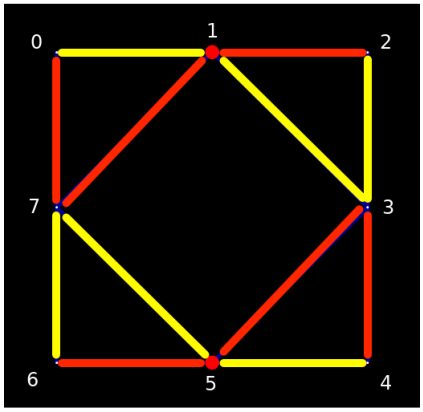
If `./punter` really needs access to the filesystem (e.g. because it uses a system that relies on creating files at runtime), then please explain why this is necessary in your `./README` file and we will decide whether to enable limited access to the file system for your entry.

Lightning contest deadline: 1200 UTC on Saturday 5th August
Full submission deadline: 1200 UTC on Monday 7th August

Good luck, and happy lambda punting.

A Sample play

Final game state after Alice (yellow) and Bob (red) claimed all rivers.



In the following, we write -> as shorthand for $P \to S$ and <- as shorthand for $S \to P$.

Punter 0:

```
-> {"me":"Alice"}
<- {"you":"Alice"}

<- {"punter":0,
    "punters":2,
    "map":{"sites":[{"id":4},{"id":1},{"id":3},{"id":6},{"id":5},{"id":0},{"id":7},{"id":2}],
           "rivers":[{"source":3,"target":4},{"source":0,"target":1},{"source":2,"target":3},
                     {"source":1,"target":3},{"source":5,"target":6},{"source":4,"target":5},
                     {"source":3,"target":5},{"source":6,"target":7},{"source":5,"target":7},
                     {"source":1,"target":7},{"source":0,"target":7},{"source":1,"target":2}],
           "mines":[1,5]}}
-> {"ready":0}

<- {"move":{"moves":[{"pass":{"punter":0}},{"pass":{"punter":1}}]}}
-> {"claim":{"punter":0,"source":0,"target":1}}

<- {"move":{"moves":[{"claim":{"punter":0,"source":0,"target":1}},{"claim":{"punter":1,"source":1,"target":2}}]}}
-> {"claim":{"punter":0,"source":2,"target":3}}

<- {"move":{"moves":[{"claim":{"punter":0,"source":2,"target":3}},{"claim":{"punter":1,"source":3,"target":4}}]}}
-> {"claim":{"punter":0,"source":4,"target":5}}

<- {"move":{"moves":[{"claim":{"punter":0,"source":4,"target":5}},{"claim":{"punter":1,"source":5,"target":6}}]}}
-> {"claim":{"punter":0,"source":6,"target":7}}

<- {"move":{"moves":[{"claim":{"punter":0,"source":6,"target":7}},{"claim":{"punter":1,"source":7,"target":0}}]}}
-> {"claim":{"punter":0,"source":1,"target":3}}

<- {"move":{"moves":[{"claim":{"punter":0,"source":1,"target":3}},{"claim":{"punter":1,"source":3,"target":5}}]}}
-> {"claim":{"punter":0,"source":5,"target":7}}

<- {"stop":{"moves":[{"claim":{"punter":0,"source":5,"target":7}},{"claim":{"punter":1,"source":7,"target":1}}],
            "scores":[{"punter":0,"score":6},{"punter":1,"score":6}]}}
```

Punter 1:

```
-> {"me":"Bob"}
<- {"you":"Bob"}

<- {"punter":1,
    "punters":2,
    "map":{"sites":[{"id":4},{"id":1},{"id":3},{"id":6},{"id":5},{"id":0},{"id":7},{"id":2}],
           "rivers":[{"source":3,"target":4},{"source":0,"target":1},{"source":2,"target":3},
                     {"source":1,"target":3},{"source":5,"target":6},{"source":4,"target":5},
                     {"source":3,"target":5},{"source":6,"target":7},{"source":5,"target":7},
                     {"source":1,"target":7},{"source":0,"target":7},{"source":1,"target":2}],
           "mines":[1,5]}}
-> {"ready":1}

<- {"move":{"moves":[{"claim":{"punter":0,"source":0,"target":1}},{"pass":{"punter":1}}]}}
-> {"claim":{"punter":1,"source":1,"target":2}}

<- {"move":{"moves":[{"claim":{"punter":0,"source":2,"target":3}},{"claim":{"punter":1,"source":1,"target":2}}]}}
-> {"claim":{"punter":1,"source":3,"target":4}}

<- {"move":{"moves":[{"claim":{"punter":0,"source":4,"target":5}},{"claim":{"punter":1,"source":3,"target":4}}]}}
-> {"claim":{"punter":1,"source":5,"target":6}}

<- {"move":{"moves":[{"claim":{"punter":0,"source":6,"target":7}},{"claim":{"punter":1,"source":5,"target":6}}]}}
-> {"claim":{"punter":1,"source":7,"target":0}}

<- {"move":{"moves":[{"claim":{"punter":0,"source":1,"target":3}},{"claim":{"punter":1,"source":7,"target":0}}]}}
-> {"claim":{"punter":1,"source":3,"target":5}}

<- {"move":{"moves":[{"claim":{"punter":0,"source":5,"target":7}},{"claim":{"punter":1,"source":3,"target":5}}]}}
-> {"claim":{"punter":1,"source":7,"target":1}}

<- {"stop":{"moves":[{"claim":{"punter":0,"source":5,"target":7}},{"claim":{"punter":1,"source":7,"target":1}}],
            "scores":[{"punter":0,"score":6},{"punter":1,"score":6}]}}
```

B Version history

- 1.0: Initial task description
- 1.1: Added link to registration form. Removed spurious reference to 2012 contest!
- 1.2: Changes made:
  - Updated protocol to include handshakes in offline mode and to specify more precisely where timeouts occur.
  - Clarified game outcome when multiple punters score equally.
  - Clarified treatment of illegal moves by the server.
  - Clarified maximum length of messages.
  - Clarified meaning of disjoint union operator.
- 1.3: Changes made:

- Fixed typo in the type of the contents of the `"timeout"` message: Float rather than GameState.
- Clarified which moves are sent along with a stop message.
- Clarified that the size header in a message is a string.
- Clarified that the VM will be used for the evaluation.