The Name of the Binding

ICFP Programming Contest 2025

TASK DESCRIPTION

Contestants are tasked with constructing a map of a labyrinthine library by collecting information during repeated exploratory expeditions. This is version 1.2*—see the changelog on the task page.*

BACKGROUND

In which our characters, Adso & William, encounter the library at the Ædificium, and seek to uncover the secrets therein.

Adso of Milner and his master, William of Backus, have at last arrived at the famed monastery of St. Kleene, under the stewardship of the monks of the Holy Order of Evaluation, home of the world-renowned Library of Lambdas, where it is said that all of the world's functions may be found.

The library makes up the greatest part of the Ædificium: the large, imposing, fortress-like building casting a long shadow over the cloisters in the setting sun. It is closely guarded by the Abbot and his librarian, Alonzo of Curry, so William and Adso can only gain access to the library via skulduggery, sneaking into the Ædificium when the monks and nuns are busy at the International Conference on Functional Programming.

The Ædificium is huge, and only Alonzo and the Abbot know all of its chambers and passages. Some even say that a curse lies on the library, and that any who enter unbidden will never find their way out again. William, a learned scholar of great intuition and logic, dismisses such superstitions out of hand, steadfast in his determination to uncover the secrets of the library. He exclaims in the fashion of his master, Hilbert, "*Wir müssen wissen, wir werden wissen!*"

Adso, ever doubtful and unsure, inquires of his master, "How shall we know where to go without the guidance of the librarian? We do not even have a map!"

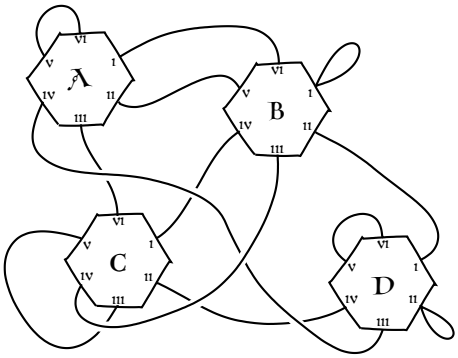
William, a devious glint behind his eyes, replies, "Well, my dear Adso, we shall have to make one."

THE ÆDIFICIUM

In which the structure of the Library is described, as well as the nature of the task before William & Adso.

The Ædificium is comprised of a number of hexagonal rooms, each side of which has a door, labelled with a number 0–5. Through each door lies a passage to another room. The rooms themselves are

also labelled, but the label is in a language that Adso cannot read. William understands the language, but with his failing eyesight, he can only discern the first two bits of the label. More than one room may have the same label. Passages may lead to the same room from which they started — or even the same door.



An example of a 4-room Ædificium.

To avoid getting lost, Adso & William must devise a *route plan* before entering the library. A route plan consists of a series of numbers 0–5, indicating the sequence of doors through which they plan to travel, starting from the initial room. The route plans always start from the same room.

As they travel through the library according to their route plan, Adso will record the 2-bit integers that William reads from the label of each room. Thus, after executing a route plan of length x, Adso will have a record consisting of $x + 1$ 2-bit integers.

The task before Adso & William is to construct a series of route plans such that they gain the information necessary to construct an accurate map of the library, in the form of an undirected graph, using as few expeditions into the library as possible.

PROTOCOL

In which the format of route plans and Adso's records are specified, as well as the representation of the map.

Queries are made to the Ædificium by making an HTTP request to the server at the URL:

`https://31pwr5t6ij.execute-api.eu-west-2.amazonaws.com/`

POST `/register` *registers a new team.*

Request Body:

```
{
  "name": string,
  "pl": string,
  "email": string
}
```

Response Body:

```
{
  "id": string
}
```

The `id` given in the response is secret and used to identify your team in future requests. Remember it but do not disclose it publicly!

POST `/select` *selects a problem to solve.*

Request Body:

```
{
  "id": string,
  "problemName": string,
}
```

Response Body:

```
{
  "problemName": string
}
```

The `id` should be exactly the same string produced by `/register`. The `problemName` may be any of the available problems, the full list of which is available on the leaderboard page. Each problem has a specified number of rooms,

but the exact layout of the map is randomly generated each time a problem is selected.

For testing purposes, a simple three-room labyrinth, small enough to be solvable by hand, may also be selected by using the problem name `"probatio"`.

Note if a problem is already selected, POSTing to `/select` will discard the old problem and generate a new one to solve.

POST `/explore` *explores the ædificium.*

Request Body:

```
{
  "id": string,
  "plans": [string],
}
```

Response Body:

```
{
  "results": [[int]],
  "queryCount": int
}
```

Only POST to `/explore` after selecting a problem with `/select`.

The `id` should be exactly the same string produced by `/register`. The `plans` field should consist of a list of route plans, each represented as a string of digits 0–5, specifying the numbers of each door to enter. For example, the string `"0325"` is the route plan that first enters door 0, then 3, then 2, then 5.

The route plans are limited in length — Adso must make it out of the library by sunrise. They can enter at most $18n$ doorways per night, where n is the number of rooms in the library.

The `results` field consists of a list of Adso's records, one for each route plan submitted. Each record is a list of integer values — the 2-bit integer values observed by William upon entering each room. The `queryCount` field contains the total number of expeditions made into the Ædificium so far. Remember, Adso and William want to make as few expeditions into the library as possible, so guesses made with a low `queryCount` will rank higher in the leaderboards.

Note: We allow multiple route plans to be submitted in a single HTTP request. This reduces our server costs, so to incentivise contestants batching their route plans, an additional one-point `queryCount` penalty applies per `/explore` request made.

POST `/guess` *submit a candidate map.*

Request Body:

```
{
  "id": string,
  "map": {
    "rooms": [int],
    "startingRoom": int,
    "connections": [
      {
        "from": {
          "room": int,
          "door": int
        },
        "to": {
          "room": int,
          "door": int
        }
      }
    ]
  }
}
```

Response Body:

```
{ "correct": boolean }
```

Only POST to `/guess` after selecting a problem with `/select`.

The `id` should be exactly the same string produced by `/register`. The `map` field contains a description of the layout of the library. The `rooms` field of this description is a list of

the 2-bit integer labels read by William, one per room. Rooms are identified by their index into this list. The `startingRoom` field specifies the index of the initial room. The `connections` field contains a list of objects that specify how each room is connected, where the `room` fields specify the index of a room and the `door` field specifies a door number, 0–5. Note that the graph constructed is undirected, so if one has already connected, say, `from` door 3 of room 5 `to` door 0 of room 2, there is no need to connect `from` door 0 of room 2 `to` door 3 of room 5 — this connection will already exist.

The field `correct` is `true` iff the submitted map is *equivalent* to the map generated when `/select` was invoked. By *equivalent*, we mean that they have the same number of rooms, and that they are indistinguishable by any route plan — if there is no route plan that demonstrates the difference between two maps of the same size, they are considered equivalent.

If a correct map is submitted, if the `queryCount` for your team is an improvement on your previous score for the currently selected problem, then your score is updated.

Regardless of if the submitted map is correct or not, *the problem will be deselected* and the library cleared when `/guess` is invoked. This means that, if your guess was incorrect, you will have to start again on a new map with `/select`.

SCORING

In which the criteria by which entries are evaluated are given, along with pointers to global scoreboards.

Grading of correct maps is based on the *efficiency* with which they were produced: Specifically, the amount of expeditions required to produce the graph.

Each team is ranked on the local leaderboard for each problem, with the team that required the fewest expeditions to correctly generate the map ranked first, and all teams that have not (yet) submitted a correct solution to that problem ranked equal last.

The global leaderboard is computed using the Borda count method, similar to last year's contest. For each problem, each team is given one point for every other team that places strictly worse in the rankings. The team that has the most points across all problems is ranked first in the global leaderboard. An important property of this system is that absolute scores for a problem do not matter — only the relative order.

The leaderboards are updated every few minutes throughout most of the contest, however they are temporarily frozen two hours before the end of the lightning round and also in the last two hours of the contest.

SUBMISSION

In which the method of participation and entry submission is described, as well as the dates during which this is available.

To be considered for prizes, your team must submit their code. This is accomplished by the following Google form. Include your `id` from `/register` and a URL at which your code can be accessed (for example, a Git repository). Don't make your code public until after the contest.

`https://forms.gle/M5SNTamHvoc6zWHq7`

This form closes 3 hours after the end of the contest. It is not necessary to make a separate submission for the lightning round, but please include a README file indicating which parts are from the first 24 hours.

Note that we also hand out a "jury prize", so being on top of the scoreboard is not required to win (but it is a recommended way of winning).