Space explorer

Victor Cionca

Intro

With the discovery of the worm hole generator space exploration became a reality. Like most misfits, you managed to scrape together enough to buy the hyper-space engine and build a rickety space ship that would take you away from a crowded and smelly Earth and into the vast emptiness of space. You were tired of the rat-race, tired of being chased around by advertisers trying to sell you something, tired of feeling guilty for not recycling properly, not eating properly, watching too much TV. But there was something else that was pulling you out there. You'd heard about it in the underground chat groups and forums, there were stories going around. Of course no humans had seen it yet but some of the extra-terrestrials running around had. The famous particle mixer, able to turn dirt into gold, into diamonds, food, water. Morally speaking, it was the Holy Grail that could save Earth. And, of course, someone would be ready to pay good money for it.

There was no map for finding the particle mixer, but the extra-terrestrial merchants were selling these tiny devices that told you the distance to it. Did they actually work? Nobody knew, noone had come back to give evidence. But what if they did?

You'd made up your mind. You charged your hyper-space engine, got into your ship and ready to start hopping from solar-system to solar-system in search for the particle mixer. You would hop to another solar-system, check the distance to the particle mixer, then hop again, trying to get closer and closer, until, who knows, maybe you'd actually find it. Of course there was the risk of running out of hyper-fuel and being stuck somewhere in space. At least it would be quiet, you could finally finish your book or listen to Willie Nelson in peace, without ads.

Details

You are given a main file (`space_explorer.c`) that generates the universe and then runs the game. You have to implement the `space_hop` function that jumps from a current planet to another planet. The goal is to land on the planet where the particle mixer is found within a fixed number of hops.

You should implement the `space_hop` function in the provided `space_solution.c` file.

The `space_hop` function

The function is declared as:

```
ShipAction space_hop(unsigned int crt_planet,
                     unsigned int *connections,
                     int num_connections,
                     double distance_from_mixer,
                     void *ship_state);
```

The parameters are:

- `crt_planet` unsigned integer representing the ID of the planet where you landed; there is an initial planet where the exploration starts, this is selected randomly from the universe

- `connections` array of unsigned integers; each element is the ID of a planet that can be reached directly from the planet you are on (`crt_planet`); the next parameter, `num_connections` represents the number of planet IDs in the array
- `distance_from_mixer` indicates how far the current planet is from the mixer
- `ship_state` is an *opaque* data structure where you can store state, see below; *opaque* refers to the main program not knowing (or caring) about the contents of the structure; the main program will not attempt to open this structure, only pass the reference back to your code.

The function returns a `ShipAction` value, where `ShipAction` is the following structure:

```
typedef struct ship_action{
    unsigned int next_planet;
    void *ship_state;
} ShipAction;
```

With the `ShipAction` return value your function indicates

- the ID of the next planet to jump to `next_planet`
- the state of your function, which will be passed back to the function at the next call, in the `ship_state` parameter.

The first time the function is called the `ship_state` is passed as `NULL`. You are then able to generate your own data structure to store state over multiple invocations of the function. You return the address to the `ship_state` through the `ShipAction` return value, and the next time the function is called the `ship_state` is passed back into the function.

Jumping to planets

The objective is to find the planet that hosts the particle mixer (also termed "treasure planet" in the code). You hop to a different planet by setting the value of the `next_planet` field in the `ShipAction` return value from the `space_hop` function.

The ID of the next planet to jump to can be

- a valid planet ID
- you can also jump to a random planet by using the value `RAND_PLANET`

Typically you will use the ID of one of the planets that you already know. This does not need to be a direct connection of the current planet. For example you can come back to a planet that you visited earlier, or create a list of planets that you plan to visit, and keep exploring them.

The random planet (`shipaction.next_planet = RAND_PLANET`) can provide an advantage in the exploration if used carefully.

Solution quality

There are many possible solutions. The quality of a solution will be measured in the number of hops used to find the particle mixer planet. The number of hops is limited to the number of planets in the universe.

The goal is to find the particle mixer in the least number of hops.

Compiling and running

When compiling you need to

- include both source files, `space_explorer.c` and `space_solution.c`, and others if employed
- use the mathematical operations library, which can be included using the `-lm` flag.

If using `gcc` you can compile the code with: `gcc space_explorer.c space_solution.c -lm -o space_explorer`

Competition

This assignment will also run as a programming competition. A competition portal will be open where you will be able to upload your function and have it evaluated.

The competition portal will have a leaderboard with the scores of the submissions, i.e. the number of hops used to find the particle mixer.

On the competition portal you will have an unlimited number of attempts, until the competition ends.

Important notes about the competition submission

The universe generated by the program is random, in number of planets, position, connections, as well as in the start planet and the position of the particle mixer planet. The random seed used to evaluate your program will be changed, so it is not recommended that you hard-code values in your program that might accelerate your search. For example, you could find the IDs of all planets and then hard-code them in a list in your code. That would immediately fail once the random seed is modified.

On the competition server your solution will be run in a sand-boxed environment. Your solution should not rely on functions for output (printf), input, file access, as well as randomization, because these functions will be disabled in the sand box.

Grading

Assignment is worth 35 points of the final mark.

- code compiles 2
- ship state
  - not used 0
  - used, must be meaningful state 2
- particle mixer is found 4
- solution types
  - completely random exploration (next planet is always `RAND_PLANET`) 1
  - generic graph search, building graph as you go along 6
  - directed search, minimising distance 6
- solution qualities
  - prevent re-visiting the same planet 5
  - backtracking to avoid dead-ends 7
  - efficient use of random jumps 4
- top 5 places in the competition will get up to 5 points: 1st place 5 points, 2nd place 4 points, etc.

The assignments will be evaluated through face-to-face interviews.

Academic integrity

- plagiarism is a serious offense
- do not share code; your submissions will be processed by similarity checking software and those suspected of plagiarism may be submitted to the board of academic infringements
- if you use ChatGPT (or similar), please indicate which parts of your code were generated and add the prompt that you used.