Breadcrumbs

Desperate to gain an edge before next year's bread-eating contest, you have sent your spy, Amelia, to infiltrate a rival city. After months of careful observation, Amelia has finally uncovered their secret techniques. Any slip up will immediately reveal her as a spy, and so she must be very careful when sending information back to you.

Direct communication is impossible. Instead, she will transmit a message by walking through the city and leaving behind a trail of breadcrumbs. You must reconstruct the hidden message by observing her path.

The city has N locations, numbered from 1 to N, and M streets that each connect two different locations. You can travel along the streets in either direction, and it is possible to travel between any two locations using the streets.

Amelia's walk is a sequence of locations $V_1, V_2, \ldots, V_L$ such that for every $1 \leq i < L$, the locations V_i and V_{i+1} are connected by a street. The walk must start at **location 1** (so $V_1 = 1$) and may revisit the same location or street multiple times. Using this, you must recover a binary message consisting of **exactly 1 000 bits**.

Your task is to implement both sides of this scheme:

- **Encoding:** Given the map of the city and a sequence of 1 000 bits, produce a walk of any length that starts at location 1.
- **Decoding:** Given the same map and the walk that you produced, recover the 1 000 bits.

You are scored based on the length of the encoding walk, where shorter walks receive higher scores. See the scoring section for details.

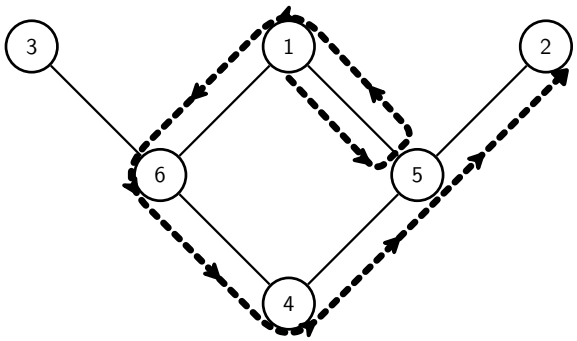

Figure 1: An example city and a walk

Figure 1 shows a city with $N = 6$ locations and streets between $(1, 5)$, $(1, 6)$, $(2, 5)$, $(3, 6)$, $(4, 5)$, and $(4, 6)$.

The walk shown goes through $1 \to 5 \to 1 \to 6 \to 4 \to 5 \to 2$. This is a valid walk of length 7, starting at location 1.

This walk might encode a binary message of length 1 000 (e.g. $[1, 0, 1, 1, 0, 0, \ldots]$). The decoder, given this walk and the city, must output the same 1 000 bits.

Implementation Details

In this task, **do not read from standard input nor write to standard output.** Do not interact with any files. Do not implement a `main` function. Instead, begin your program by including the header file `crumbs.h` (`#include "crumbs.h"`) and interact with it as described below.

Functions

You must implement the functions `init`, `encode` and `decode`:

```
void init(int N, int M, std::vector<int> A, std::vector<int> B);
```

- N is the number of locations and M is the number of streets.
- Vectors A and B each have size M. For $0 \leq j < M$, street j connects locations `A[j]` and `B[j]`. Locations are numbered from 1 to N. It is possible to travel between any pair of locations using the streets and no two streets connect the same pair of locations.
- This function is called once for each instance of your program. See the Judging section and the Experimentation section for more details.

```
std::vector<int> encode(std::vector<int> bits);
```

- `bits` has size 1 000 and each element is 0 or 1.
- You must return a walk starting at location 1: a `std::vector<int>` of location indices $V_1, \ldots, V_L$ with $V_1 = 1$, where each consecutive pair (V_i, V_{i+1}) is connected by a street. The length of the walk is L, the length of the vector. Note that the vector should start from index 0, so that index 0 has V_1 and index $L - 1$ has V_L.
- This function is called 50 times per test case.
- **Your score for the test case depends on the maximum length of the walk you return. See the scoring section for details.**

```
std::vector<int> decode(std::vector<int> walk);
```

- `walk` is a sequence of locations forming a valid walk starting at location 1 (as produced by your encoder).
- You must return a `std::vector<int>` of 1 000 integers (each 0 or 1) that is the same as the bits that were passed into `encode` to produce this walk.
- This function is called 50 times per test case.

If you violate any of the conditions listed above, your program will be judged as incorrect and you will receive 0% of the points for the test case.

Judging

The judge will start two separate instances of your program: an *encoder* and a *decoder*.

- Firstly, the judge will call `init` once on the encoder.
- Secondly, the judge chooses 50 binary messages and calls `encode` 50 times on the encoder.
- Thirdly, the judge will call `init` once on the decoder. **The exact same parameters used for the first step will be provided to this function.**
- Lastly, the judge will call `decode` 50 times on the decoder, once for each walk produced by the encoder, in an arbitrary order.

The encoder and decoder cannot communicate in any way beyond sending a walk. In particular, any information saved to static or global variables in the encoder is not available in the decoder.

The time spent by your solution will be measured as the sum of the time spent by the two instances.

Subtasks and Constraints

For all subtasks:

- $3 \leq N \leq 1\,000$.
- $2 \leq M \leq 4\,000$.
- $1 \leq A[i] < B[i] \leq N$ for all i. No two streets connect the same pair of locations.
- It is possible to travel between any two locations using a sequence of streets.

There is one test case per subtask. Additional constraints for each subtask are given below.

Subtask	Points	Additional constraints	J (judge)	T (threshold)
1	8	$N = 3$ and the city forms a complete graph.[1]	1,001	1,001
2	8	$N = 5$ and the city forms a complete graph.	501	501
3	8	$N = 4$ and the city forms a complete graph.	632	1,001
4	8	$N = 10$, locations 1 to 5 form a complete graph, and there is a street $(i, i-5)$ for all $6 \leq i \leq 10$. There are no other streets.	481	501
5	8	$N = 3$ and the city forms a line, with streets between locations i and $i+1$ for all i.	1,999	2,001
6	8	$N = 5$ and the city forms a line, with streets between locations i and $i+1$ for all i.	1,263	1,391
7	8	$N = 9$ and the city forms a 3×3 grid.[2]	668	831
8	8	$N = 25$ and the city forms a 5×5 grid.[3]	560	711
9	12	$N = 10$, $M = 15$, and the city layout is randomly generated.[4]	591	821
10	12	$N = 100$, $M = 800$, and the city layout is randomly generated.	247	291
11	12	$N = 1\,000$, $M = 4\,000$, and the city layout is randomly generated.	315	391

For all subtasks, the city layout is available for download from the contest site. They will be in the same format as specified by the sample grader.

Scoring

If your solution produces an incorrect decoding (wrong bits), or violates any of the conditions in the Implementation Details section, your score will be 0.

Otherwise, let W be the maximum length of the walk that your encoder produces over all calls for that test case, and let J be the longest length of the walk the judges' solution can produce for the same test case over all possible encoding messages. Additionally, each subtask has a threshold T (see the table above). Your percentage score for that test case is:

- **100%** if $W \leq J$,
- $\mathbf{25 + \frac{T-W}{T-J} \times 75\%}$ if $J < W \leq T$,
- **10%** if $T < W \leq 10000$,
- **0%** otherwise.

[1] In a complete graph, there is a street between every pair of locations

[2] If we rearrange the city into 3 rows and 3 columns where the locations are numbered 1 to N when going left to right and top to bottom, there is a street between two locations if and only if they are adjacent horizontally or vertically.

[3] Similarly defined as the previous subtask.

[4] All cities that satisfy the given constraints are equally likely to be generated.

There is one test case per subtask. Your score for that subtask is the score of that test case multiplied by the number of points for that subtask.

Experimentation

In order to experiment with your code on your own machine, first download the provided files `crumbs.cpp`, `crumbs.h` and `grader.cpp`.

You should modify `crumbs.cpp`, which contains a basic example implementation.

Compile your solution with:

```
g++ -std=c++20 -O2 -Wall crumbs.cpp grader.cpp -o crumbs
```

This will create an executable `crumbs` which you can run with `./crumbs`. If you have trouble compiling, please send a message in the Communication section of the contest website.

The provided grader reads from standard input in the following format:

- The first line of input contains N and M.
- The next M lines of input describe the streets. The ith such line contains two integers A and B, representing a street between locations A and B.

The grader will first call `init` and will then decide 50 different messages. For each message, it will run `encode` to decide a walk, then immediately call `decode`. It will return an error if the returned message is not the same as the original. If all the messages pass, it will return the length of the longest walk returned by `encode`.

The sample grader chooses the bits randomly. **The judge grader may not do this**. Also note that the sample grader runs `encode` and `decode` from the same instance of your program.

Sample Grader Input & Sample Session

The sample grader is supplied the following input:

```
6 6
1 5
1 6
2 5
3 6
4 5
4 6
```

This corresponds to the diagram on the first page. One possible interaction is as follows:

The grader first calls `init(6, 6, [1, 1, 2, 3, 4, 4], [5, 6, 5, 6, 5, 6])`. This corresponds to the supplied input.

The grader then decides on some message composing of 1 000 bits.

- The grader calls `encode([1, 0, 1, 1, 0, 0, ...])` with these 1 000 bits.
- The student returns `[1, 5, 1, 6, 4, 5, 2]`, a walk of length 7 starting at location 1.
- The grader calls `decode([1, 5, 1, 6, 4, 5, 2])` with that walk.
- The student returns `[1, 0, 1, 1, 0, 0, ...]`, the same 1 000 bits. This is correct.

The sample grader makes 49 more calls to `encode` and `decode` with different messages.

The sample grader prints the length of the longest walk generated by `encode`.