Tape Scheduling

Limits: 5 sec., 1024 MiB

Tape storage or magnetic tape data storage is a system that is used for storing digital information on magnetic tape using digital recording. With data volumes growing rapidly worldwide, tape storage is the most suitable system for long-term and large capacity storage.

When accessing data, the tape media needed to be loaded into a driver, and tens of seconds are required to rewind this tape to re-position the tape drive to the right position. When a tape drive receives multiple requests, the total access time often can be reduced by the use of a scheduler, which reorganizes the retrieval order of the tape requests.

A tape contains 4 band. Each band contains 52 wraps: 26 forward direction wraps and 26 backward direction wraps. The wraps are conveniently numbered from 0 to 207: the first band contains wraps 0 to 51, the second band contains wraps 52 to 103 and so on. Wraps with even numbers are forward direction wraps and wraps with odd numbers are backward direction wraps.

The length of a tape is 960 meters. A tape contains 33 anchors (the distance between adjacent anchors is 30 meters). The first anchor is located at the beginning of the tape on position 0, the second one is located on position 30, ..., the last one is located on position 960.

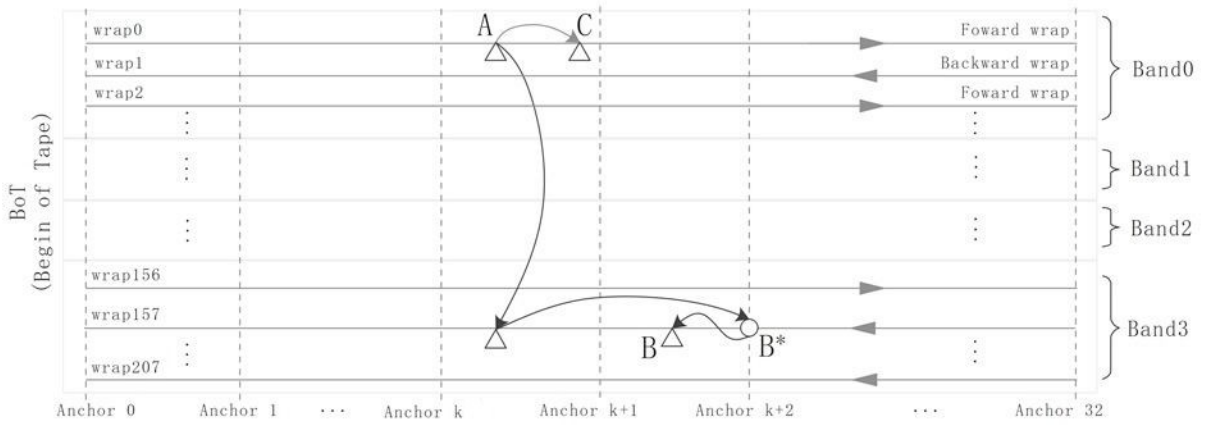
The position of each data unit on the tape can be represented by two numbers: a number of the wrap (integer number from 0 to 207) and a distance from the beginning of the tape to the position of the data unit (real number from 0 to 960). For example, $(47, 744.4)$ means that a data unit is on a wrap number 47 and is located on a position 744.4 meters from the start of the tape.

Time to move from position (w_1, d_1) to (w_2, d_2) is calculated as follows:

1. If w_1 and w_2 belong to different bands, the current band is changed. This takes 4 seconds.
2. If $w_1 \neq w_2$, the current wrap is changed. This takes 1 second.
3. Move the current position to the closest anchor such that the position w_2 is reachable from the anchor with respect to the direction of the current wrap. The velocity of moving is 8 m/s.
4. Slowly move to the position w_2. The velocity of slow moving is 2 m/s.

Step 3 may be omitted if there is no wrap change on step 2 ($w_1 = w_2$) and position d_2 is reachable from d_1 with respect to the direction of the current wrap.

If in order to perform moves on step 3 or step 4, a change of the direction of moving is required, there is an additional time penalty of 2.5 seconds. Please note that if both steps 3 and 4 require a change of the direction, the penalty of 2.5 seconds is added only once.



Consider few examples:

To move from A to B you need:

- 4 seconds to change the band.

- 1 second to change the wrap.
- $(B^* - A)/8$ seconds to move to the anchor position B^*.
- 2.5 seconds to change the direction.
- $(B^* - B)/2$ seconds to slowly move to the position B.

To move from A to C you don't need to change any bands or wraps as well as move to an anchor position. Instead you can just slowly move to the position C which takes $(C - A)/2$ seconds.

Your task is to create a scheduler for organizing a retrieval order of the tape requests. You are given a set of n request positions (w_i, d_i). You have to produce a permutation p of request indices (0-based) that minimizes the total time of request processing. For more details please refer to the Scoring section. Please note, that the starting position is $(0, 0)$ and the direction of moving is forward. After all of the requests are processed you have to return to the state $(0, 0)$ and the direction has to be changed to forward.

Input

The first line of the input contains a single integer n — the number of requests. The next n lines contain requests. Each line contains two numbers: w_i — integer number of the wrap and d_i — real distance from the beginning of the tape to the position of the request.

All real numbers in the input will be given in double-precision floating-point format with maximum precision available.

Output

In a single line print a space-separated permutation p_i — the order of executing requests from the input.

Constraints

$1 \le n \le 10000$,
$0 \le w_i \le 207$,
$0 < d_i < 960$,
$0 \le p_i < n$.

Samples

Input (*stdin*)	Output (*stdout*)

5
7 400.0
47 410.0
47 404.0
74 777.7
77 770.0
	3 4 1 2 0

Notes

The sample output provided above corresponds to the quickstart solution. It scores 201501 points with a total time to process requests of 247.1375 seconds. The sequence of data points for this output is the following:

1. (0, 0) — (74, 777.7): 112.6 seconds:
    - 4 seconds to change the band from 0 to 1,
    - 1 second to change the wrap from 0 to 74,
    - $(750 - 0)/8 = 93.75$ seconds to move from the position 0 to the anchor position 750,
    - $(777.7 - 750)/2 = 13.85$ seconds to slowly move from the position 750 to the position 777.7.
2. (74, 777.7) — (77, 770.0): 8.7875 seconds:
    - 1 second to change the wrap from 74 to 77,
    - $(780 - 777.7)/8 = 0.2875$ seconds to move from the position 777.7 to the anchor position 780,
    - $(780 - 770)/2 = 5$ seconds to slowly move from the position 780 to the position 770,
    - 2.5 seconds to change the direction.
3. (77, 770.0) — (47, 410.0): 53.75 seconds:
    - 4 seconds to change the band from 1 to 0,
    - 1 second to change the wrap from 77 to 47,
    - $(770 - 420)/8 = 43.75$ seconds to move from the position 770 to the anchor position 420,
    - $(420 - 410)/2 = 5$ seconds to slowly move from the position 420 to the position 10.
4. (47, 410.0) — (47, 404.0): 3 seconds:
    - $(410 - 404)/2 = 3$ seconds to slowly move from the position 410 to the position 404.
5. (47, 404.0)— (7, 400.0): 15.5 seconds:
    - 1 second to change the wrap from 47 to 7,
    - $(420 - 404)/8 = 2$ seconds to move from the position 404 to the anchor position 420,
    - $(420 - 400)/2 = 10$ seconds to slowly move from the position 420 to the position 400,
    - 2.5 seconds to change the direction.
6. (7, 400.0) — (0, 0): 53.5 seconds:
    - 1 second to change the wrap from 7 to 0,
    - $(400 - 0)/8 = 50$ seconds to move from the position 400 to the anchor position 0,
    - 2.5 seconds to change the direction.

Scoring

If the permutation you printed is invalid, e.g. if it doesn't contain each number 0 through $n - 1$ exactly once or has any other elements, your score for the test is 0 points. Otherwise let T be the total time it takes to access all of the data points in order of the permutation p (providing the initial and the final positions are $(0, 0)$ and the time to move between two data points is calculated using the algorithm described above). Your score for the test is calculated as

$$\left\lfloor \frac{n}{T+1} \cdot 10^7 \right\rfloor .$$

Your overall score is the sum of your scores over all test cases.

Submissions

- The execution time limit is 5 seconds per test case and the memory limit is 1024 mebibytes.
- The code size limit is 64 kibibytes.
- The compilation time limit is 1 minute.
- There are 50 provisional test cases. Your submissions will be evaluated on the provisional set during the submission phase.
- You can submit your code once per 10 minutes and you will get feedback with your score for each of the provisional tests.
- There will be 500 test cases in the final testing after the submission phase is over. The final results will be announced in one week.

Quickstart

Check the sample solution, which first accesses all of the requests on forward wraps in the ascending order of the distance from the beginning of the tape and then all of the requests on backward wraps in the descending order of the distance from the beginning of the tape.

The source code is available for some of the contest programming languages:

- Python

Tests

All test cases including provisional and final sets are generated by the generator. There are three types of tests:

- Random distribution tests,
- Gaussian distribution tests,
- Mixed random distribution and sequential access tests.

A test case is produced by providing the generator with a seed number. Seed number 1 corresponds to the sample case. You do NOT know seed numbers for provisional or final test sets, but you can use the generator for local testing by taking a few easy steps:

1. Check the generator source code
2. Save it to a file named `generator.java`
3. Compile the source code with Java

   ```
   javac generator.java
   ```

4. Generate a test case for some seed number

   ```
   java generator <seed>
   ```

   This will print the test case for `<seed>` to *standard output*.