Expand All Collapse All

Discrete distribution

How can I determine the number of command-line arguments?

The argument to `main()` is an array `args[]`. So, you can use the expression `args.length` to access its length.

How should I format the output?

The assignment specification says to separate the *m* integers by *whitespace*, which could be space or newline characters (or any combination of the two). Our sample executions print 25 integers per line, separated by space characters.

Instead of picking a random integer between 0 and $S_n - 1$, could I pick a random real number between 0 and S_n?

Yes, that would work too. Technically, it should be a real number between 0 (inclusive) and S_n (exclusive).

Any way to speed up the computation when *n* is large?

Yes, though you are not required to do so on this assignment. One effective strategy is to pre-compute the cumulative sums $S_i = a_1 + a_2 + \ldots + a_i$; generate a random integer r between 0 and $S_n - 1$; then use *binary search* to find the unique index i between 1 and n such that $S_{i-1} \leq r < S_i$. Here is an even faster algorithm.

Thue–Morse weave

Can I use strings concatenation and string operations?

No. You must use arrays.

How should I construct the Thue–Morse sequence of length *n*?

One strategy is to apply the definition and successively build the Thue–Morse sequences of lengths 1, 2, 4, 8, 16, and so forth, doubling the length of the sequence in each step. When *n* is a power of 2,

- Create a boolean array of length *n*.
- Initialize the first bit in the array—the one at index 0— to `false`.
- Copy the first bit in the array into the next position, replacing `false` with `true` in the copy.
- Copy the first 2 bits in the array into the next 2 positions, replacing `false` with `true` and `true` with `false` in the copy.
- Copy the first 4 bits in the array into the next 4 positions, replacing `false` with `true` and `true` with `false` in the copy.
- . . .

If *n* is not a power of 2, construct the Thue–Morse sequence for the smallest power of 2 larger than *n*.

An alternative strategy is fill an array of length *n* one bit a time by applying the following remarkable propery:

$$\text{thue}[i] = \begin{cases} 0 & \text{if } i = 0 \\ \text{thue}[i \,/\, 2] & \text{if } i \text{ is even} \\ 1 - \text{thue}[i - 1] & \text{if } i \text{ is odd} \end{cases}$$

How should I print the *n*-by-*n* Thue–Morse weave?

First, build the Thue–Morse sequence of length *n* (or possibly the next largest power of 2). Use a double nested loop to determine whether bits *i* and *j* in the sequence are equal (and what to print).

Can I use a two-dimensional array?

No. For this problem, a two-dimensional array is unnecessary.

Birthday problem

Any advice on which arrays to define and use?

Our reference solution defines two arrays:

- A boolean array of length *n* (to keep track of which birthdays have been encountered so far in a single experiment).
- An integer array of length *n* + 2 (to keep track of the number of times that exactly *i* people entered the room across all experiments). Note that at most *n* + 1 people will enter the room before encountering a pair that shares a birthday.

How should I format the table of results?

Row *i* should contain the count *i*, the number of times that exactly *i* people enter the room, and the percentage of times that *i* or fewer people enter the room, with each number separated by whitespace. The sample output in the assignment specification uses tab (`\t`)

characters to separate the numbers in a row. In Lecture 5, you'll learn about `printf()`, which makes it easier to format and align numeric output.

▼ **As a function of *n*, how many people must enter a room until two share a birthday?**

Asymptotically, the answer is $\sqrt{\pi n / 2}$.

▼ **How do the results change if some birthdays are more common than others?**

The probability of two matching birthdays only increases if the distribution of birthdays is non-uniform. For example, in the northern hemisphere, there are more births in the months of August and September; in the U.S., births are much rarer on holidays (because of C-sections and induced labor).

Minesweeper

▼ **Any advice on which arrays to define and use?**

Our reference solution defines two arrays:

- A boolean 2D array to specify which cells contain mines.
- An integer 2D array to count the number of neighboring mines.

▼ **Any advice on how to decompose my program into separate steps?**

Here are the four things you need to do:

- Read the command-line arguments.
- Place the k mines uniformly among the mn cells.
- Count the number of neighboring mines.
- Print the results.

We recommend organzing your program accordingly.

▼ **My program has many corner cases for computing the number of neighbors for cells on the boundary. Any tips for simplifying my code?**

One effective strategy is to declare an (m+2)-by-(n+2) array, using the interior m-by-n cells for the Minesweeper board. That way, each Minesweeper cell has 8 neighbors that you can access without going out of bounds.

▼ **My program has 8 cases for identifying the 8 neighbors of a cell. Any tips for simplifying my code?**

Consider using a double nested loop to iterate over 9 cells whose row- and column-indices are both within 1 of the given site.

▼ **How do I determine in which *k* of the *mn* cells to place mines?**

Here are two possible approaches:

- Choose k cells (arbitrarily) to contain mines. Then, use the shuffling algorithm from lecture (adapted to 2D arrays) to rearrange the k mines uniformly at random.
- Pick a row i and column j, each uniformly at random; if cell (i, j) does not yet contain a mine, put one there. Repeat until you have placed k mines. If $k \leq m\ n\ /\ 2$, then this algorithm will complete quickly; otherwise, use the same idea, but pick the sites that do *not* contain mines.