CS 101 **Project 5** **Summer 2017**

Project 5 is similar to Project 1, except that now your program must discover a solution rather than just verify whether a given solution is correct.

Write a C++ program project5.cpp that behaves as described below. Your program will read from standard input and write to standard output.

First read integer N which denotes the number of dimensions of a multi-dimensional array.
Next read the dimensions d_0 d_1 … d_{N-1}, so the array will have dimensions d_0 by d_1 by … by d_{N-1}.
Next read the initial values of the array using array aggregate notation. The array elements will always include the value 0 and will always consist of distinct consecutive non-negative integers, but these integers might be arranged in any order. Example:

2 3 4 {{4,1,2,7},{11,8,3,9},{5,10,0,6}}	$\begin{bmatrix} 4 & 1 & 2 & 7 \\ 11 & 8 & 3 & 9 \\ 5 & 10 & 0 & 6 \end{bmatrix}$

Recall from Project 1 that each move relocates the 0 element to a new position, and is specified as a dimension and a distance. The dimension k will be in the range 0 to N–1, and the distance indicates how far the 0 element will travel (positive indicates increasing indexes, and negative indicates decreasing indexes). However, to prevent reaching outside the range of valid indexes 0 to d_k–1, the indexes along each dimension will wrap around in a circular fashion. As the value 0 travels the specified distance, it swaps places with each other value it encounters.

The goal is to find a sequence of moves that leads to the array configuration such that all the array elements will be in ascending order when written in aggregate notation (or in row-major order). The correct sequence of moves is not necessarily unique. Several examples follow; each example shows an input and then two of the many possible corresponding output sequences. Your final output should only show the move sequence, not the array contents. But you will likely find it helpful to also display the array contents while you are debugging your program.

2 2 2 {{3,2},{1,0}}	
1 1 0 1 1 -1 0 -1 1 1 0 1	0 -1 1 -1 0 1 1 1 0 -1 1 -1

2 3 4 {{4,1,2,7},{11,8,3,9},{5,10,0,6}}

1 1 0 -1 1 -1 0 1 1 -3 0 2 1 2 0 1 1 -1 0 -2 1 -1 0 3 1 1	1 1 0 -1 1 -1 0 1 1 -1 1 -1 1 -1 0 1 0 1 1 1 1 1 0 1 1 -1 0 -1 0 -1 1 -1 0 1 0 1 0 1 1 1

2 4 5 {{5,1,2,3,4},{10,6,7,8,9},{11,12,13,18,14},{15,0,16,17,19}}	
1 2 0 -1 1 -3 0 -2	1 1 1 1 0 -1 1 -1 1 -1 1 -1 0 -1 0 -1

3 3 3 3 {{{0,4,2},{3,7,5},{6,25,8}},{{9,10,11},{12,13,14},{15,16,17}},{{1,19,20},{18,22,23},{21,24,26}}}	
0 -1 1 2 2 1 0 1 1 -2 2 -1	0 -1 1 1 1 1 2 1 0 1 1 -1 1 -1 2 -1

Your program should behave as illustrated in the above examples. The program should be runnable by typing either of the following commands:

./a.out
./a.out < *inputfile* > *outputfile*

For the first line above, your program will read from the keyboard and write to the screen. The second line redirects standard input to come from the specified *inputfile*, and it redirects standard output to go to the specified *outputfile*, but your program is unaware of these files.

To earn full credit, your program should work correctly when the array has at most 3 dimensions. To earn extra credit, your program should work correctly for arbitrary number of dimensions.

There exist some input values for which no solution can exist, but your program will not be tested using such inputs, so it is not necessary for your program to recognize when no solution exists.

Hints:

- As shown in the previous examples, it is always possible to replace any move with distance $d \geq 2$ by d moves with distance exactly 1 each. You can experiment to find which way works better for your program's design.
- One possible approach is to explore all the possible move sequences in breadth-first order, by maintaining a queue of all the array configurations that can be reached. This guarantees you would eventually find a shortest sequence of moves, but your program would need to build a tree of the possible paths to explore, and it might run out of memory or waste a lot of time exploring paths that are not getting nearer to the goal.
- Another possible approach is to explore the possible move sequences in depth-first order. Either use recursion, or place all the array configurations you can reach into a stack. However you would need to make sure your program doesn't get caught in an infinite recursion, and again it's possible to waste a lot of time exploring paths that are not getting nearer to the goal.
- The best approach for solving larger problems efficiently might be to use a priority queue rather than either a FIFO queue or a stack. To compute a priority for a given array configuration, you could measure how far each array element currently is from its desired final location, and then take the sum of all these distances. If your program explores configurations in the order of smallest priority first, this will enable you to explore the paths soonest that are getting nearest to the goal. This approach is somewhat similar to Dijkstra's algorithm.

Please carefully read the following requirements:

- You are always permitted to use <iostream>, <string>, and any C libraries such as <cstdlib> and <cmath>. **For this project, you are also permitted to include any STL libraries such as <list>, <vector>, <stack>, <queue>, <deque>, <algorithm>, <numeric>, <utility>, … in your program.**
- You must do your own work. You must not borrow any code from any other person, book, website, or any other source. You also must not share your code with any other person, or

post it on any website. We run plagiarism detection software on every project. So if you violate these rules, you may receive an invitation to the dean's office to discuss the penalties for academic misconduct.

- Make sure your program runs properly on the cs-intro.ua.edu server, because this is where your program will be graded. In particular, make sure your program initializes the values of all variables when they are declared or allocated. Otherwise it might behave differently on Linux than it does on a PC or Mac.
- Your program will be compiled using this command: g++ project5.cpp –Wall –lm –std=c++11. Alternatively, if you split your program into multiple files, then it will instead be compiled using this command: g++ *.cpp –Wall –lm –std=c++11.
- Verify that all the necessary .h and .cpp files which are needed to compile your program are included in the same directory before you submit your program. There should be no extra subdirectories and no extra .h or .cpp files, otherwise your program might not compile the way you intended.
- Compress your project into a zip file that contains your C++ program source file. Right-click (or secondary click) on your project directory, and then (depending on your operating system) select either the Compress option or Send To → Compress from the popup menu. Finally upload your .zip file that contains your .cpp file for this project to Blackboard.
- If you violate the above requirements such that it breaks our grading script, your project will be assessed a significant point deduction, and extreme or multiple violations may cause the project to be considered ungradable.
- Every semester many students lose some points because they don't follow all the instructions. So please read and follow all the project specifications precisely to prevent losing points unnecessarily. If anything is unclear, please ask for clarification well before the project is due. Please pay particular attention to input and output formats.
- Submit your project on Blackboard by the due date (11:59pm Friday). There is a grace period of 24 hours (until 11:59pm Saturday). Projects submitted on Sunday will be assessed a late penalty of 5% per hour. No projects will be accepted after Sunday. Once it is graded, your project score will be posted on Blackboard and the results of the grading script will be sent to your Crimson email account.
- Double-check and triple-check your submission when you submit it. Errors discovered later cannot be fixed and resubmitted after the project is graded. Projects will not be re-graded unless an error is found in the grading script or in the input/output files that are used during grading.