CS 101 **Project 5** **Summer 2016**

Write a C++ program project5.cpp that operates as described below.

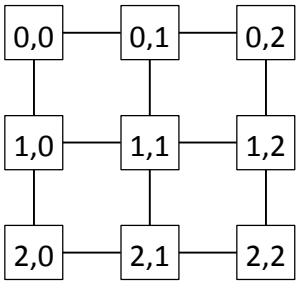
Your program should accept two command line arguments. Argument 1 is an input file name, and argument 2 is an output file name.

Your program will first construct an undirected graph G based on parameters specified in the input. The graph will represent a multi-dimensional space, and the first input line specifies the number of dimensions N. Your program is only required to handle the cases N=2 and N=3, but you can earn extra credit if your program also works for arbitrary N ≤ 12. The second input line specifies the lengths L_1 L_2 … L_N of the N dimensions. Graph G will have $L_1 \times L_2 \times \ldots \times L_N$ vertices, and each vertex can be considered as a tuple $X = (x_1, x_2, \ldots, x_N)$ where $0 \le x_j < L_j$ for each dimension $1 \le j \le N$. You may assume that the number of vertices will be at most 4096.

Next there will be one or more additional input lines, and each such input line specifies some offsets d_1 d_2 … d_N that will be used to add unweighted edges to the graph. These offsets $(d_1, d_2, \ldots, d_N)$ can be considered as a vector that specifies the pattern of edges that should be added. Mathematically, graph G will have an edge between vertex $V = (v_1, \ldots, v_N)$ and vertex $W = (w_1, \ldots, w_N)$ if there exists any permutation π of the dimensions (1, …, N) such that $v_j = w_j \pm d_{\pi(j)}$ for each dimension $1 \le j \le N$.

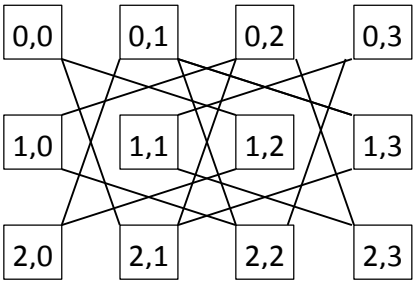
The graphs G that correspond to some example input files are shown below:

input0.txt
2
3 3
0 1



($v_1 = w_1 \pm 0$ and $v_2 = w_2 \pm 1$) or
($v_1 = w_1 \pm 1$ and $v_2 = w_2 \pm 0$)

input1.txt
2
3 4
1 2



($v_1 = w_1 \pm 1$ and $v_2 = w_2 \pm 2$) or
($v_1 = w_1 \pm 2$ and $v_2 = w_2 \pm 1$)

Hint: you can arrange the list of vertices linearly using *row-major order* or *column-major order*. Then you can use a one-dimensional array rather than a multi-dimensional array.

After your program reads the inputs and builds graph G, it should print a line that shows the number of vertices and the number of edges in graph G.

Next your program will use breadth-first search to compute the length of each shortest path (based on number of edges) from each root or start vertex V to each destination vertex W. Alternatively, you may use Dijkstra's algorithm to compute these shortest paths. Finally, the remaining output lines will show the number of shortest paths that have each finite length. Each of these output lines displays a distinct finite positive path distance (in ascending order), along with the number of vertex pairs (V, W) such that the shortest path from V to W has exactly that length (number of edges).

Here are the output files that correspond to the previously given input files. For example, the graph G constructed for input0.txt has 9 vertices and 12 edges. It also has 24 vertex pairs at distance 1, 28 vertex pairs at distance 2, 16 vertex pairs at distance 3, and 4 vertex pairs at distance 4.

output0.txt
9 12
1 24
2 28
3 16
4 4

output1.txt
12 14
1 28
2 40
3 38
4 20
5 6

Hints:
- Map vertices (x1, x2, …, xN) into the range from 0 to (L1 * L2 * ... * LN) - 1.
- Check all permutations more efficiently via sorting.
- Check positives and negative differences together using absolute value.

Please carefully read the following requirements:

- You must do your own work; you must not share any code. If you violate this rule, you may receive an invitation to the dean's office to discuss the penalties for academic misconduct.
- Make sure your program runs properly on cs-intro.ua.edu. Your program will be graded on that system.
- Submit your project in a zipfile that contains only your project5.cpp file. There should be no subdirectories or extra files.
- If you violate the 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.
- Many pairs of example input/output files and a test script are provided. Run the following commands to test your program. If any differences appear, then your project does not match the specifications. Points will be deducted for each incorrect line of output.

```
unzip *.zip
chmod u+x testscript.sh
./testscript.sh
```

- Alternatively, instead of running the test script, you can instead enter these commands.

```
g++ project5.cpp –Wall –lm –o project5
./project5 input0.txt temp0.txt
diff output0.txt temp0.txt
./project5 input1.txt temp1.txt
diff output1.txt temp1.txt
…
```

- 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.
- Double-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 used during grading.