題目所付的 Inputs 為 sample Inputs，機考的 input 會有不同，請就題目的描述來實作

CCU CSIE

Data Structure Course

Computer-Based Test 2 pool

Teacher: Yu-Ling Hsueh

TA: Nick, Kile, Ruby

1. Topological sort

Given a directed graph G, where each node represents an activity labeled from 1 to n, your program should output the topological order. When there are multiple nodes with in-degree = 0, you must choose the node with the label that has the smallest number first.

The first line of the input file is n representing the number of vertices. The vertex number starts from 1 and ends at n, that is, V={1, 2, . . . , n}. The second line to the last line in the input file represent the adjacency matrix of a graph.

Test Case

Please test your program with Inputs 1-3, and then check the answer with Outputs 1-3. Note: each number is separated by a space.

Listing 1: Topological sort

```
Input1:
5
0 1 1 0 0
0 0 0 1 1
0 0 0 0 1
0 0 0 0 1
0 0 0 0 0

Output1:
1 2 3 4 5

Input2:
6
0 0 0 0 0 1
```

```
0 0 1 0 1 0
1 0 0 1 0 1
0 0 0 0 0 0
1 0 0 0 0 0
0 0 0 0 0 0

Output2:
2 3 4 5 1 6

Input3:
9
0 1 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 1
0 0 0 0 1 0 0 0 1
0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0

Output3:
1 2 4 6 5 7 9 8
```

2. Compute in-degree and out-degree

Given a directed graph G, your program should output the in-degree and out-degree for each node. The first line of the input file is n representing the number of vertices. The vertex number starts from 1 and ends at n, that is, V={1, 2, . . . , n}. The second line to the last line in the input file represent the adjacency matrix of a graph. For the output, each line should consist of the node number followed by the number of in-degree and out-degree, respectively.

Test Case

Please test your program with Inputs 1-3, and then check the answer with Outputs 1-3. Note: each number is separated by a space.

Listing 2 : Compute in-degree and out-degree

```
Input1:
5
0 1 1 0 0
0 0 0 1 1
0 0 0 0 1
0 0 0 0 1
0 0 0 0 0

Output1:
1 0 2
2 1 2
3 1 1
4 1 1
5 3 0

Input2:
6
0 0 0 0 0 1
0 0 1 0 1 0
1 0 0 1 0 1
0 1 0 0 0 0
1 0 0 0 0 0
0 0 0 0 0 0

Output2:
1 2 1
2 1 2
3 1 3
4 1 1
5 1 1
6 2 0

Input3:
5
1 1 1 0 1
0 0 0 1 0
0 1 1 0 0
0 0 1 0 1
```

```
1 0 0 1 1

Output3:
1 2 4
2 2 1
3 3 2
4 2 2
5 3 3
```

3. Transitive closure

Given a directed graph G, output the transitive closure in the form of an adjacency matrix. The first line of the input file is n representing the number of vertices. The vertex number starts from 1 and ends at n, that is, V={1, 2, . . . , n}. The second line to the last line in the input file represent the adjacency matrix of a graph.

Test Case

Please test your program with Inputs 1-3, and then check the answers with Outputs 1-3.

Note: each number is separated by a space.

Listing 3 : Transitive closure

```
Input1:
4
0 0 1 0
0 0 1 0
0 0 0 1
0 0 0 0

Output1:
0 0 1 1
0 0 1 1
0 0 0 1
0 0 0 0

Input2:
5
0 0 0 0 0
```

```
0 0 1 0 0
1 0 0 0 1
0 0 1 0 1
0 0 0 0 0

Output2:
0 0 0 0 0
1 0 1 0 1
1 0 0 0 1
1 0 1 0 1
0 0 0 0 0

Input3:
5
0 1 0 0 0
0 0 1 1 0
0 1 0 0 0
0 0 0 0 0
0 0 1 0 0

Output3:
0 1 1 1 0
0 1 1 1 0
0 1 1 1 0
0 0 0 0 0
0 1 1 1 0
```

4. Quick sort

Implement the quick sort algorithm and print out the intermediate sorted result for each pass. Your program should read a standard input file which stores a set of unsorted numbers, and use an array to store the numbers. You must use the first number in the unsorted array as the pivot and show the intermediate result for each pass.

Test Case

Please test your program with Inputs 1-3, and then check the answers with Outputs 1-3.

Note: each number is separated by a space.

Listing 4 : Quick sort

```
Input1:
6 7 2 9 4 5

Output1:
6 5 2 9 4 7
6 5 2 4 9 7
4 5 2 6 9 7
4 2 5 6 9 7
2 4 5 6 9 7
2 4 5 6 7 9

Input2:
8 6 2 4 5 1 9 3

Output2:
8 6 2 4 5 1 3 9
3 6 2 4 5 1 8 9
3 1 2 4 5 6 8 9
2 1 3 4 5 6 8 9
1 2 3 4 5 6 8 9

Input3:
26 5 37 1 61 11 59 15 48 19

Output3:
26 5 19 1 61 11 59 15 48 37
26 5 19 1 15 11 59 61 48 37
11 5 19 1 15 26 59 61 48 37
11 5 1 19 15 26 59 61 48 37
1 5 11 19 15 26 59 61 48 37
1 5 11 15 19 26 59 61 48 37
1 5 11 15 19 26 59 37 48 61
1 5 11 15 19 26 48 37 59 61
1 5 11 15 19 26 37 48 59 61
```

5. Kruskal's algorithm

Given an undirected graph G, find the minimum cost spanning tree using the Kruskal's algorithm. Please using the Kruskal's algorithm to find the minimum cost spanning tree, and print out each edge of the minimum cost spanning tree **according the processing order**. The first line of the input file is n representing the number of vertices. The vertex number starts from 1 and ends at n, that is, V={1, 2, . . . , n}. The second line to the last line in the input file represent the cost adjacency matrix of a graph. For the output, each line should consist of the starting node and the ending node for each edge with its corresponding cost. **The node with the smaller number should be the starting node.**

Test Case

Please test your program with Inputs 1-3, and then check the answer with Outputs 1-3. Note: each number is separated by a space.

Listing 5: Kruskal's algorithm

```
Input1:
5
0 5 10 0 0
5 0 0 7 3
10 0 0 0 0
0 7 0 0 9
0 3 0 9 0

Output1:
2 5 3
1 2 5
2 4 7
1 3 10

Input2:
6
0 6 7 0 9 8
6 0 2 1 5 0
7 2 0 3 0 4
0 1 3 0 0 0
9 5 0 0 0 0
8 0 4 0 0 0

```

```
Output2:
2 4 1
2 3 2
3 6 4
2 5 5
1 2 6

Input3:
5
0 1 0 5 9
1 0 2 0 8
0 2 0 4 3
5 0 4 0 0
9 8 3 0 0

Output3:
1 2 1
2 3 2
3 5 3
3 4 4
```

6. A breadth-first-search question (a strange lift)

The strange lift can stop at every floor as you want, and there is a number K_i ($0 \le K_i \le N$) on every floor. The lift has just two buttons: up and down. When you at floor i and if you press the button "UP", you will go up K_i floors. That is, you will go to the the (i+K_i)-th floor. If you press the button "DOWN", you will go down K_i floors. That is, you will go to the (i-K_i)-th floor. The lift can't go up high than N, and can't go down lower than 1. For example, there is a building with 5 floors, and K_1 = 3, K_2 = 3, K_3 = 1, K_4 = 2, K_5 = 5. Beginning from the 1st floor, you can press the button "UP", and you'll go up to the 4th floor, but if you press the button "DOWN", the lift can't do it, because it can't go down to the -2th floor. Given two parameters A and B, where A represents the floor you are located and B is the floor you want to go, how many times at least he has to press the button "UP" or "DOWN"?
The input contains two lines. The first line contains three integers N, A, B, where N is total of floors in the building (N $\ge$ 1, 1 $\le$ A, B $\le$ N), A is the floor you are located, and B is your destination; the second line consists of N integers K_1, K_2,…,K_N.

For each case of the input, the least times you have to press the button when you on floor A, and you want to go to floor B. Print the least total number of UP and DOWN to reach the target floor. If you can not reach floor B, print ”-1”.

Test Case

Please test your program with Inputs 1-3, and then check the answer with Outputs 1-3. Note: each number is separated by a space.

Listing 6 : A breadth -first-search question (a strange lift)

```
Input1:
5 1 5
3 3 1 2 5

Output1:
3

Input2:
10 1 9
4 7 5 2 2 4 6 4 1 7

Output2:
6

Input3:
6 1 6
3 1 3 1 2 3

Output3:
3
```

7. Dijkstra algorithm

Given a directed graph G, find the shortest path using the Dijkstra algorithm. Your program should read a standard input file. The first line of the input file is n representing the number of vertices. The vertex number starts from 1 and ends at n, that is, V={1, 2, . . . , n}. The second line to the n+1 line of the input file represent an adjacency matrix. The last line consists of two variables as the source and destination nodes. If there are multiple vertices with the same distance from the source, the vertex with the smallest vertex number is selected first. You must keep the path, where the nodes were originally selected to cause the updates. Finally, print every vertex in the shortest path and separate each vertex by a space. If there is no path between two specified vertices, print -1.

Test Case

Please test your program with Inputs 1-3, and then check the answer with Outputs 1-3. Note: each number is separated by a space.

Listing 7: Dijkstra algorithm

```
Input1:
7
0 3 6 9 0 0 0
0 0 2 0 4 0 0
0 0 0 0 0 7 0
0 0 0 0 0 8 0
0 0 1 0 0 0 0
0 0 0 0 0 0 0
0 0 0 5 0 0 0
2 6

Output1:
2 3 6

Input2:
6
0 0 0 0 0 8
0 0 2 0 5 0
7 0 0 3 0 4
0 1 0 0 0 0
9 0 0 0 0 0
0 0 0 0 0 0
3 2

Output2:
3 4 2

Input3:
```

```
7
0 6 5 5 0 0 0
0 0 0 0 1 0 0
0 2 0 0 1 0 0
0 0 2 0 0 1 0
0 0 0 0 0 0 3
0 0 0 0 0 0 3
0 0 0 0 0 0 0
1 7

Output3:
1 3 5 7
```

8. Prim's algorithm

Given an undirected graph G, find the minimum cost spanning tree using the Prim's algorithm, and print out each edge of the minimum cost spanning tree according the processing order. The vertex number starts from 1 and ends at n, that is, V={1,2,3,. . . , n }. The first line of the input file is n representing the number of vertices. The second line to the last line in the input file represent the adjacency matrix of a graph. For the output, each line should consist of the step number, the starting node and the ending node for each edge with its corresponding cost. The node with the smaller number should be the starting node.

Test Case

Please test your program with Inputs 1-3, and then check the answer with Outputs 1-3. Note: each number is separated by a space.

Listing 8 : Prim's algorithm

```
Input1:
5
0 2 5 0 0
2 0 6 4 0
5 6 0 9 3
0 4 9 0 0
0 0 3 0 0

Output1:
1 1 2 2
```

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

Input2:
5
0 5 8 2 4
5 0 0 0 0
8 0 0 0 1
2 0 0 0 3
4 0 1 3 0

Output2:
1 3 5 1
2 4 5 3
3 1 4 2
4 1 2 5

Input3:
6
0 8 0 2 3 6
8 0 0 0 1 0
0 0 0 5 0 4
2 0 5 0 0 0
3 1 0 0 0 0
6 0 4 0 0 0

Output3:
1 2 5 1
2 1 5 3
3 1 4 2
4 3 4 5
5 3 6 4
```

9. Bubble sort

Given a standard input file which stores a set of unsorted integers, and print out the intermediate sorted result for each pass during the bubble sort process. Your program should sort these integers in ascending order.

Test Case

Please test your program with Inputs 1-3, and then check the answer with Outputs 1-3. Note: each number is separated by a space.

Listing 9 : Bubble sort

```
Input1:
1 43 6 79 50 2

Output1:
1 6 43 50 2 79
1 6 43 2 50 79
1 6 2 43 50 79
1 2 6 43 50 79
1 2 6 43 50 79

Input2:
5 88 7 66 52 54 56 31 33 2

Output2:
5 7 66 52 54 56 31 33 2 88
5 7 52 54 56 31 33 2 66 88
5 7 52 54 31 33 2 56 66 88
5 7 52 31 33 2 54 56 66 88
5 7 31 33 2 52 54 56 66 88
5 7 31 2 33 52 54 56 66 88
5 7 2 31 33 52 54 56 66 88
5 2 7 31 33 52 54 56 66 88
2 5 7 31 33 52 54 56 66 88

Input3:
22 34 3 32 82 55 89 50 37 5 64 35 9 70

Output3:
22 3 32 34 55 82 50 37 5 64 35 9 70 89
3 22 32 34 55 50 37 5 64 35 9 70 82 89
3 22 32 34 50 37 5 55 35 9 64 70 82 89
3 22 32 34 37 5 50 35 9 55 64 70 82 89
```

```
3 22 32 34 5 37 35 9 50 55 64 70 82 89
3 22 32 5 34 35 9 37 50 55 64 70 82 89
3 22 5 32 34 9 35 37 50 55 64 70 82 89
3 5 22 32 9 34 35 37 50 55 64 70 82 89
3 5 22 9 32 34 35 37 50 55 64 70 82 89
3 5 9 22 32 34 35 37 50 55 64 70 82 89
3 5 9 22 32 34 35 37 50 55 64 70 82 89
3 5 9 22 32 34 35 37 50 55 64 70 82 89
3 5 9 22 32 34 35 37 50 55 64 70 82 89
```

10. Articulation point detection

Given an undirected graph G, find the articulation points in G. The vertex number starts from 1 and ends at n, that is, V={1,2,3,. . . , n }. The first line of the input file is n representing the number of vertices. The second line to the last line in the input file represent the adjacency matrix of a graph. Please use **depth-first search** for the purpose of converting the graph into spanning tree. **The node with the smaller number should be the starting node.** The vertex number starts from 1 and ends at n, that is, V={1, 2, . . . , n}. When there are multiple paths during traversal, you must choose the node with the smallest number first. For the output, print the articulation points ordered by the vertex number in ascending order.

Test Case

Please test your program with Inputs 1-3, and then check the answer with Outputs 1-3. Note: each number is separated by a space.

Listing 10 : Articulation point detection

```
Input1:
10
0100000000
1011000000
0100100000
0100110000
0011000000
0001001100
0000010100
0000011011
0000000100
```

```
0000000100

Output1:
2 4 6 8

Input2:
7
0010000
0010100
1101000
0010000
0100011
0000100
0000100

Output2:
2 3 5

Input3:
10
0100000000
1001000000
0000110100
0100001001
0010000100
0010001110
0001010011
0010110000
0000011000
0001001000

Output3:
2 4 6 7

```