NOTE: **less than the usual number of late days**.

The purpose of this assignment is to give you practice with graphs and with backtrack searches.

Beneath the ancient city of "Otog" there exists a magical labyrinth, which is made up of a number of chambers with passages between them. This cavern is controlled by an evil sorcerer, who has given the passages the strange property that after a person has left a passage and entered into a chamber, the passage just followed may simply disappear, or another passage may possibly appear in its place. Thus, it is not possible to guess the destination of a passage leaving a room unless it has been followed from the current chamber previously.

Whenever the sorcerer catches a person within his city, he magically transports his victim to one of the chambers below the city. Any adventurer managing to find a way out of the maze beneath the city is allowed to depart. If the person is extremely unlucky, the initial transportation may be to a chamber from which there is no path leading to the surface. Wandering through the maze without a map is a dangerous proposition, since once you enter a room, you may not be able to get out. Those who do not find a way out, either due to a lack of skill or due to bad luck, eventually become part of the grotesque adornment of the chambers and passages.

You have just been caught and, along with your trusty laptop, have been teleported into the maze. You look in the dust in the chamber and see a data CD, with the markings "*maze graph*", and you realize that this is the key to your survival. You must write a program which will use the data on the CD and find a the path to the surface, if one exists (note that the program can return to its last decision point, while you, the adventurer, cannot necessarily directly, or possibly ever, return to the same point).

The program you are to write is to accept the following command line input

- **optional** dump switch, "-d" — determines whether the original maze is printed or not. Note, it will not be used to test level 1 listed below.
- file name — file name of maze and starting room information described below.

There are several "levels" to the program. To get points relative to a level, you must get the previous level working correctly.

1. (50 pts max) Using the stored data after the maze has been input, and not simply echoing the input data, print out each chamber number and the numbers of the chambers which are adjacent to it, using the tabular form shown in the sample output. Dashes are used in the output to represent no passage in the specified direction. Note: table is simply tab aligned (tabs output between items). Note: the nodes need not be printed in the same order as input
2. (55 pts max) Everything from the previous level **except** the program is to use the *dump switch* described above to determine whether to print the maze or not. The program prints the maze if the switch is present, and doesn't if it is not present.

3. (75 pts max) Everything from previous level. **Additionally**, in graphs **without** the potential for circular paths, the program is to correctly determine whether a path to the surface exists and either report its existence or report that there is no path to the surface.

4. (90 pts max) Everything from the previous level **Additionally**, the program is to print a path from the starting position to the outside (if any exist) in the form shown in the sample in the sample output. (Note, it is acceptable for the program to print multiple paths from the starting position to the outside [if any exist]. If multiple paths are printed, do not print duplicate paths. To be different the paths must have a different number of nodes, different nodes, or the nodes must be in a different order.)

5. (100 pts max) Everything from previous level. **Additionally** the program must also be able to correctly handle graphs with, or with the potential for, circular paths.

The program is only to print non-circular paths (good cave adventurers know to drop things in chambers before they leave so they know if they have been there or not). That is, do not print paths that include cycles. You are only to print the successful paths, not all the abortive attempts. Note: for debugging it might be useful to print out when you enter a room and when you “back out” of a room – that way you can follow the progress of your path searching algorithm to insure it is working correctly.

There may be **any number** of chambers in the labyrinth (although if you wish you may assume that there are no more than 1,000 chambers). There will be at most 4 passages leading out of each chamber (corresponding to north, east, south, and west). Each chamber may have any number of passages leading into it.

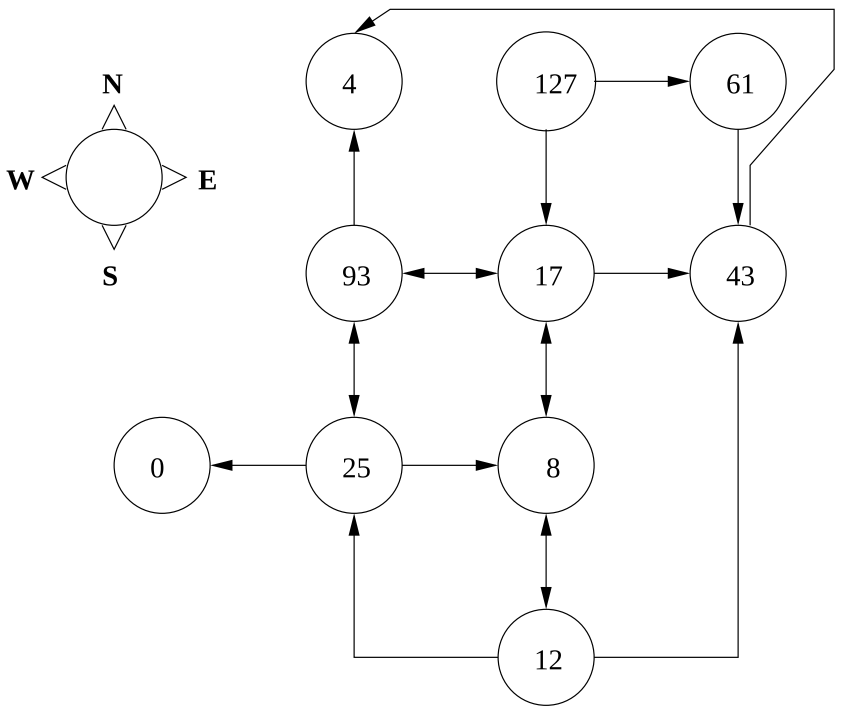
For the purposes of the program, the chambers are simply represented by numbers. The number zero (0) is used to represent the outside of the maze. The other numbers are not necessarily sequential, and have no significance other than identification.

The input, which is to be read from the text file provided on the command line, is as follows:

- an integer — which specifies the number of chambers which will be actually used.
- one integer per chamber, which are the chambers’ names/numbers.
- one line per chamber, each line containing 5 integers. The first is the chamber number. The second is the number of the adjacent chamber in the northerly direction. The third is the number of the adjacent chamber in the easterly direction. The fourth is the number of the adjacent chamber in the southerly direction. The fifth is the number of the adjacent chamber in the westerly direction. The lines may come in any order, not necessarily the same order as the list of chamber numbers. A value of -1 indicates there is no adjacent chamber in that direction. A value of 0 indicates that the outside world can be reached in that direction.
- The last piece of data is a single integer, which provides the chamber number into which the person has just been teleported.

You may assume that all input will be good. The "zero" chamber (outside) will **not** be listed as a room in the input (although it can be at the end of a passage leading from the chamber), nor will it included in the count of rooms. You can choose whether or not to print it out as a separate chamber on the dump of the maze (the dump below does print it).

The design and implementation of this program is up to you. You should use data abstractions where appropriate. You need not try to find the most efficient way to solve the problem, but it should be reasonably efficient (you will be penalized for a cpu time exceeded message which occurs if you use more than 20 seconds). You will be graded to some extent on the design/structure of your program and the appropriateness of the data structures you use as well as correctness. You must also generate your own test data. The simple sample set shown below is provided in the course public directory, simply to insure that you accept the correct form of the input. The following is the maze that this data represents.



which would be represented by the following input data. Note: this graph **does** contain the potential for cyclic paths.

```
9
4 93 25 127 17 8 61 43 12
4 -1 -1 -1 -1
93 4 17 25 -1
127 -1 61 17 -1
61 -1 -1 43 -1
17 -1 43 8 93
43 4 -1 -1 -1
25 93 8 -1 0
8 17 -1 12 -1
12 8 43 -1 25
127
```

Using the above input the fully functional sample solution produces the following output (three runs are shown, one with and one without the maze dump, and one without the dump which prints all the paths rather than just one). Note: if you print multiple paths, the print order of the paths is not important (although the order of the steps within each path obviously is).

```
blj(mithrandir): a.out -d sample.in
original maze

room    north   east    south   west

0       --      --      --      --
127     --      61      17      --
12      8       43      --      25
93      4       17      25      --
4       --      --      --      --
43      4       --      --      --
61      --      --      43      --
25      93      8       --      0
8       17      --      12      --
17      --      43      8       93


starting in room 127
path found
   in 127 go south
   in 17 go south
   in 8 go south
   in 12 go west
   in 25 go west
   outside

blj(mithrandir): a.out sample.in


starting in room 127
path found
   in 127 go south
   in 17 go south
   in 8 go south
   in 12 go west
   in 25 go west
   outside


blj(mithrandir): allPaths sample.in


starting in room 127
path found
   in 127 go south
   in 17 go south
   in 8 go south
   in 12 go west
   in 25 go west
   outside

path found
   in 127 go south
   in 17 go west
   in 93 go south
   in 25 go west
   outside

blj(mithrandir):
```