CMP 210 – Data Structures and Algorithms
Spring 2021
Assignment – 02

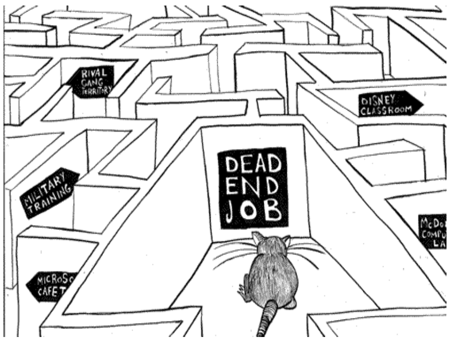
Issue Date: **Wednesday – May 12, 2021**
Submission Deadline: **Sunday – May 23, 2021 (Till 12:00 am)**

Instructions!

1. You are required to do this assignment on your own. Absolutely **NO** collaboration is allowed, if you face any difficulty feel free to discuss with me.
2. Cheating will result in a **ZERO** for the assignment. (Finding solutions online is cheating, copying someone else's solution is cheating). Also, do not hand your work over to another student to read/copy. If you allow anyone to copy your work, in part or in whole, you are liable as well.
3. Hard **DEADLINE** of this assignment is **Sunday, May 23, 2021**. No late submissions will be accepted after due date and time so manage emergencies beforehand.

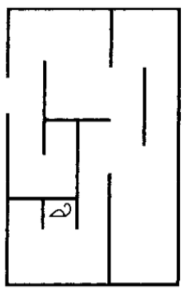
Rat in a Maze: [30 Marks]

Consider a problem where a trapped rat tries to find its way to exit in a maze. The rat hopes to escape from the maze by systematically trying all the route. If it reaches a dead end, it retraces its steps to the last position and begins at least one more untried path. For each position, the rat can move one of four directions i.e. forward, backward, left and right. The task is to check if there exists any path so that the rat can reach the destination or not, if it exists then mark the path for the rat.



You are required to implement a simple version of this problem in which you have to find the success path from given source to destination. The maze will be represented as form of a two dimensional binary matrix. Where,

1. 0's represent the open path whereas 1's represents barriers/brick walls. (as shown in figure below).
2. The legal moves are left, right, top and down from a given cell (Obviously, you are not allowed to move off the maze).
3. You are required to read maze from a file and stores your output to another file.



```
11111111111
10000010001
10100010101
c0100000101
10111110101
10101000101
10001010001
11111010001
101m1010001
10000010001
11111111111
```

(a) (b)

(a) A maze with a mouse. (b) Two dimensional character array representation of the given mouse and maze scenario. Where 'm' represents the mouse and 'c' represents the exit end.

Input:
A file named **in.txt** will contain the input matrix which represents the Maze. First line of the file shows the order of the matrix. Second and third lines show the source and destination respectively. The maze matrix is given from fourth line onwards.

Output:
The output of your program should be displayed on console and should be stored in a text file **out.txt** as well. The first, second and third line should be same as input file. At fourth line the success path should be printed and after that the maze matrix should be printed such that the success path should be shown with asterisk.

Sample Input File Format:

```
8
0 0
7 7
m 0 0 1 1 0 0 0
0 1 0 0 1 0 0 0
0 0 1 0 0 0 1 0
0 0 1 1 1 1 1 0
0 0 0 0 1 0 0 0
0 0 0 0 1 1 1 0
0 1 1 0 0 1 0 0
0 1 1 1 0 1 0 c
```

Sample Output File Format:

```
8
0 0
7 7
(0,0) (0,1) (0,2) (1,2) (1,3) (2,3) (2,4) (2,5) (1,5) (1,6) (1,7) (2,7) (3,7) (4,7) (5,7) (6,7) (7,7)

* * * 1 1 0 0 0
0 1 * * 1 * * *
0 0 1 * * * 1 *
0 0 1 1 1 1 1 *
0 0 0 0 1 0 0 *
0 0 0 0 1 1 1 *
0 1 1 0 0 1 0 *
0 1 1 1 0 1 0 *
```

Vigenère Cipher [30 Marks]

The *Vigenère Cipher* is a simple alphabet substitution Cipher. It uses a matrix, termed a *tabula recta*, that contains 26 different lines. Each line of the matrix is a complete alphabet but the alphabet is shifted right by one each level so that at level 2 (the second line) the alphabet was shifted twice.

The *Vigenère Cipher* uses a *keyword* to encrypt the text. There is no constraint on the length of the *keyword*, the keyword repeats itself until it matches the length of the text to encrypt. Let's take an example of the encipher principle: suppose that the text to be encrypted is "ATTACKED" and the keyword is "LEMON":
Now, consider all the characters of the original text one by one:
character 1: A
character 2: T
character 3: T
character 4: A
character 5: C
character 6: K
character 7: E
character 8: D

As the length of the keyword is 5, we will have to map the characters in keyword onto the string to encipher as follows: L E M O N L E M
This means that the character 'A' is assigned the 'L' of the keyword. The 'T' is assigned the second character of the key, and so on.
So the actual association becomes:

$$\begin{tikzcd}
A \arrow[d] & T \arrow[d] & T \arrow[d] & A \arrow[d] & C \arrow[d] & K \arrow[d] & E \arrow[d] & D \arrow[d] \\
L & E & M & O & N & L & E & M
\end{tikzcd}$$

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

The *Vigenère square* also known as the *tabula recta*, to be used for encryption and decryption.

Now, all we have to do to encipher is to find the crossing point between the line associated with a given character, and this character in *Vigenère square*. Let's follow up with our example:
For example, the first letter of the text, *A*, is paired with *L*, the first letter of the keyword. Therefore, row L and column A of the *Vigenère square* are used, namely L. Similarly, for the second letter of the text, the second letter of the key is used. The letter at row E and column T is X. The rest of the plaintext is enciphered in a similar fashion:

character 1: 'A', Consider the row 'L' and column 'A' => 'A' enciphered into 'L'
character 2: 'T', Consider the row 'E' and column 'T' => 'T' enciphered into 'X'
character 3: 'T', Consider the row 'M' and column 'T' => 'T' enciphered into 'F'
character 4: 'A', Consider the row 'O' and column 'A' => 'A' enciphered into 'O'
character 5: 'C', Consider the row 'N' and column 'C' => 'C' enciphered into 'P'
character 6: 'K', Consider the row 'L' and column 'K' => 'K' enciphered into 'V'
character 7: 'E', Consider the row 'E' and column 'E' => 'K' enciphered into 'I'
character 8: 'D', Consider the row 'M' and column 'D' => 'D' enciphered into 'P'

So the text "ATTACKED" is enciphered as "LXFOPVIQ" with the key LEMON.

To Decipher, decryption is performed by going to the row in the table corresponding to the key, finding the position of the ciphertext letter in that row and then using the column's label as the text. For example, in row 'L' (from LEMON), the ciphertext L appears in column 'A', which is the first text letter. Next, in row 'E' (from LEMON), the ciphertext 'X' is located in column T. Thus T is the second text letter. Going through the same way the actual association becomes:

$$\begin{tikzcd}
L \arrow[d] & X \arrow[d] & F \arrow[d] & O \arrow[d] & P \arrow[d] & V \arrow[d] & I \arrow[d] & P \arrow[d] \\
L & E & M & O & N & L & E & M
\end{tikzcd}$$

So in our example:

following row 'L', the column that gives a 'L' has a 'A' as top first character.
following row 'E', the column that gives a 'X' has a 'T' as top first character.
following row 'M', the column that gives a 'F' has a 'T' as top first character.
following row 'O', the column that gives a 'O' has a 'A' as top first character.
following row 'N', the column that gives a 'P' has a 'C' as top first character.
following row 'L', the column that gives a 'V' has a 'K' as top first character.
following row 'E', the column that gives a 'I' has a 'E' as top first character.
following row 'M', the column that gives a 'P' has a 'D' as top first character.

Hence, the decipher decodes the original text. Vigenère cipher was used to encrypt/decrypt secret messages during American Civil War. Your task is to write two functions: a *cipher* function that should receive a key and a message and return the ciphered text, a *decipher* function that should receive a cipher message and a key and return the original deciphered text. Implement a driver program that calls cipher and decipher functions to show their working.

Sample Run:

```
Enter the secret message: The internet never retreats.
Enter the cipher key: rose
Output: kvw zbliibwx bwzvf vvhjirhk.

Enter the cipher message: kvaw wk r ususv dskwruw.
Enter the cipher key: rose
Output: THIS IS A CODED MESSAGE.

Enter the secret message: I've been there but I didn't do it
Enter the cipher key: penguin
Output: x'ik jrtr zbmet oan v hvjh'g hb cb
```

XOR Linked List: [20 Marks]

An ordinary Doubly Linked List requires space for two address fields to store the addresses of previous and next nodes. A memory efficient version of Doubly Linked List can be created using only one space for address field with every node. This memory efficient Doubly Linked List is called XOR Linked List as the list uses bitwise XOR operation to save space for one address.

Instead of storing two address fields, XOR linked list compresses the same information into one address field by storing the bitwise XOR of the address for previous and the address for next in one field. Thus, in the XOR linked list, instead of storing actual memory addresses, every node stores the XOR of addresses of previous and next nodes.

```
// Node structure of a memory efficient doubly linked list
class Node
{
    public:
      int data;
      Node* npx; /* XOR of next and previous node */
};
```

Here, we store bitwise XOR of addresses of next and previous nodes with every node and we call it npx, which is the only address member we have with every node. When we insert a new node at the beginning, npx of new node will always be XOR of NULL and current head. And npx of the current head must be changed to XOR of new node and node next to the current head. For a better understanding you may visit wikipedia. Your task is to implement the following functions for XOR linked list.

```
void insertAT Head(int Val);
void insertAtTail(int Val);
int RemoveAt Head();
int RemoveAtTail();
bool search(int key);
void print();
```

Good Luck!

Grading policy:

1. To get full credit, your programs should be in running condition with correct output while handling all boundary cases and checks. There should not be any type of run time errors in your program.
2. Partial credit will be given to partially completed but in running condition programs.
3. ZERO will be given to copied programs (in part or whole). This includes copy solution from internet as well.
4. ZERO will be given if you allowed anyone to copy your solution in part or whole.
5. Partial credit will be given to programs producing correct outputs for most of the cases.
6. This assignment will be graded along a viva session. Your grades will also depend upon the performance in your viva voice.