BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
HYDERABAD
CAMPUS,
Data Structures and Algorithms
CS F211 / IS F211
Homework Assignment – 8

1. Write a menu driven C program to implement a **queue** with 2 **stacks**. Your queue should have an enqueue() and a dequeue() function and it should be "first in first out" (FIFO). Your program must support the following operations.
    a. Insert an element (enqueue)
    b. Delete an element (dequeue) – print the element
    c. Print the elements of the stack (1. stack1, 2. stack2)
    d. Exit
2. Write a menu driven C program to implement a **stack** with 2 **queues**. Your stack should have a push and a pop function and it should be "last in first out" (LIFO). Your program must support the following operations.
    a. Insert an element (Push)
    b. Delete an element (Pop) – print the element
    c. Print the elements of the queue (1. queue1, 2. queue2)
    d. Exit
3. Write a menu driven C program to find the k^{th} largest element in a max-heap? Your program must support the following operations.
    a. Insert an element into the max heap
    b. Print the elements of the max heap
    c. Find the k^{th} largest element of the max heap (requires input k)
    d. Exit
4. Write a menu driven C program to remove duplicates from a given linked list. Your program must support the following operations. All the following operations except Exit must be implemented as functions.
    a. Insert an element at the beginning of the list
    b. Insert an element at the end of the list
    c. Remove duplicates from the list
    d. Print the list
    e. Exit

    Example:
    If the created list is: 2->4->1->3->4->2->6->5->7->5->3->NULL
    Then the list after removal of duplicates is 2->4->1->3->6->5->7->NULL
    (**Note:** You cannot use array)

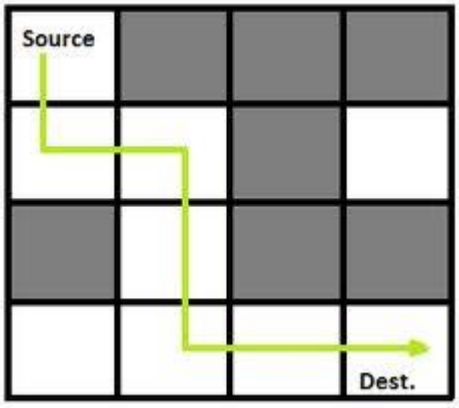
5. Write a C program to sort a given array elements based on heap sort algorithm.

6. Write a C program to do following operation in amortized O(1) time.

    a. enqueue in the queue
    b. dequeue in the queue
    c. find minimum in the queue
    d. exit

    You can use extra space apart from a queue data structure.

7. You have a NxN maze in the form of a matrix. This maze will have some cells as open and some cells will be closed (obstacles).



A Rat can move one cell at a time in one of the four directions with following priority:

First it tries to move to one cell right, if it can't move right because of obstacle or matrix bound, it tries to go Down, if it can't move Down also, it tries to move Left, lastly if it can't move in any of the other directions it tries to move Up. The rat tries to exhaust all these possibilities before backtracking to a path it has come from.
So the priority of directional movement is:

Right > Down > Left > Up

Initially, the Rat is at the top left position *i.e.* at matrix[0][0]

There's a way out at the bottom right which the Rat has to reach *i.e.* position matrix[N-1][N-1].
The rat starts to move one cell at a time from the beginning and its goal is to reach the bottom right position. If the rat finds a dead-end while trying to move and it has exhausted all its movement possibilities, it tries to backtrack upto the last position where it has some options open to move according to its priority order.
Write a program for two different 10x10 matrices given below that will print the complete movement of the rat in reaching the exit at bottom right, or will print "No path exists to exit" and show its movements in the matrix after trying all the movements. If there is a path, the program should print how many times the rat has moved in a cell during the journey. In the matrix below, elements with 1 show path open and elements 0 mean it is closed.
The output should also be in the form of a matrix that shows all the paths the rat has tried and the final taken. If a cell is in the path taken by the rat, you can increment it by one for

Note: <u>Do NOT use recursive calls in your program</u>. Create and use explicit stack(s). Apply backtracking method by using the stack(s).

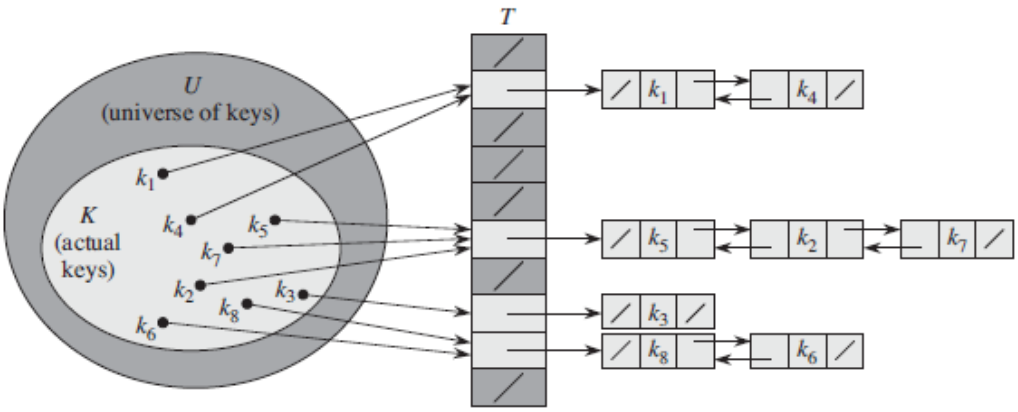
The matrices are:

1 1 1 1 1 1 1 0 0 0
1 0 1 1 0 0 0 0 0 0
1 1 0 0 1 1 0 0 0 0
1 0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 0 0 0 0
1 1 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 0 0 0
0 0 0 0 1 0 0 1 1 1
0 0 0 0 1 1 1 1 0 1
0 0 0 1 1 1 1 1 0 1

AND

1 1 1 1 1 1 1 0 0 0
1 0 1 1 0 0 0 0 0 0
1 1 0 0 1 1 0 0 0 0
1 0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 0 0 0 0
1 1 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 0 0 0
0 0 0 0 1 0 0 0 1 1
0 0 0 0 1 1 1 1 0 1
0 0 0 1 1 1 1 1 0 1

8. You will be given N keys (will be integers). How you store it decides the time complexity of various operation on it. Your requirement is to search fast so you have decided to use hash tables to store your data. You will be given a value P representing slots. Your hash function is h(key) = key%P which gives you the slot where the key should go. In case of collision you have to resolve it through chaining. For implementing it you will be required to use linked-list implementation as depicted in the diagram below.



Write a menu driven program to ask options from user.
a. Add key to hash table.
b. Search key into hash table
c. Delete key from hash table.
d. Print entire hash table
e. exit

9. You are working on a project of building a web-browser and you are working along with your team on component which is responsible for searching some pattern string **P** in web-page having Text string **T**. Obviously you want the search to happen fast. Hence you are using some smart method rather than brute force approach. This is what your team has decided to do and asked you to implement it :
   Let's say length of pattern P is **k**
   a. Calculate a numerical (hash) value for the pattern P
   b. To compute the hash value for k-length string s : $\sum ascii(si)$
      Ex : h(ABC) = 65 + 66 + 67
   c. For each k-character substring of text T, compare the numerical (hash) values (computed by calculating the hash value of substring) with hash value of pattern P. If both the values are same (might be the case of collision) then it compares the pattern P with the substring character by character otherwise it shifts to next substring of T to compare with P.
   d. If you compute hash value of first substring then you can compute hash value of next substrings in O(1). You can easily figure it out.

   Note : long long will be enough to store the hash value

10. In a village N people have bought lottery with different lottery number. The criteria of the winner is : Any lottery number will be broken into different contiguous subsequence (like substring) parts and if product for every digit of a contiguous subsequence are different then that person will win that lottery. There could be multiple winners. You have to read N lottery number (containing only digits) and find out the winners.
    Ex : if lottery number is 782 : then he is winner since different contiguous subsequence of 782 is 7, 8, 2, 78, 82, 782 and their product are 7, 8, 2, 56, 16, 112 and all are distinct.
    Note : You can assume that product of any subsequence cannot exceed 10000.

----------XXXXXXXXXXXXXXXXXXXXXXXXXX----------