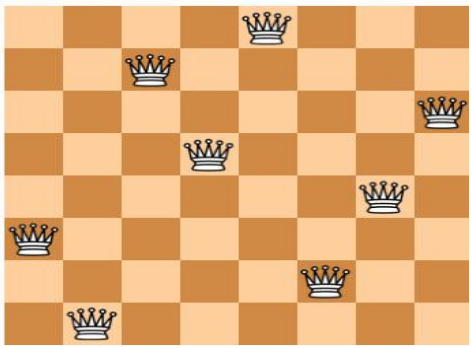
Week 6:
Write a program to find the non-attacking positions of Queens in a given chess board using backtracking

Problem:
Place N queens on an N x N chessboard so that no two queens attack each other. that is no two queens share the same row, column, or diagonal.
One possible solution when n=8



Algorithm:

- Algorithm place will check whether queen can be placed at k[th] row and i[th] column

```
Algorithm Place(k, i)
// Returns true if a queen can be placed in kth row and
// ith column. Otherwise it returns false. x[ ] is a
// global array whose first (k − 1) values have been set.
// Abs(r) returns the absolute value of r.
{
     for j := 1 to k − 1 do
          if ((x[j] = i) // Two in the same column
               or (Abs(x[j] − i) = Abs(j − k)))
                    // or in the same diagonal
               then return false;
     return true;
}
```

- Algorithm Nqueens place queens in safe positions using backtracking

```
Algorithm NQueens(k, n)
// Using backtracking, this procedure prints all
// possible placements of n queens on an n × n
// chessboard so that they are nonattacking.
{
     for i := 1 to n do
     {
          if Place(k, i) then
          {
               x[k] := i;
               if (k = n) then write (x[1 : n]);
               else NQueens(k + 1, n);
          }
     }
}
```

Sample Implementation:

```
#include<stdio.h>
```

```c
#include<conio.h>
#include<math.h>
int x[30],count=0;
int place(int pos)
{
    int i;
    for (i=1;i<pos;i++) {
        if((x[i]==x[pos])||((abs(x[i]-x[pos])==abs(i-pos))))
           return 0;
    }
    return 1;
}
void printboard(int n)
{
    int i,j;
    count++;
    printf("\n\nsolution%d:\n",count);
    for (i=1;i<=n;i++)
    {
        for (j=1;j<=n;j++)
        {
            if(x[i]==j)
                printf("Q%d\t",i); else
                printf("*\t");
        }
        printf("\n");
    }
}
void queen(int n)
{
    int k=1;
    x[k]=0;
    while(k!=0)
    {
        x[k]=x[k]+1;
        while((x[k]<=n)&&!place(k))
           x[k]++;
        if(x[k]<=n)
        {
            if(k==n)
                printboard(n);
            else
            {
                k++;
                x[k]=0;
            }
        } else
           k--;
    }
}
```

```
void main()
{
    int i,n;
    printf("enter the number of queens..:");
    scanf("%d",&n);
    queen(n);
    printf("\npossible solutions..: %d",count);
    getch();
}
```

Output:

```
enter the number of queens..:4

solution1:
*       Q1      *       *

*       *       *       Q2

Q3      *       *       *

*       *       Q4      *


solution2:
*       *       Q1      *

Q2      *       *       *

*       *       *       Q3

*       Q4      *       *

possible solutions..: 2
```

Week 7:

Find a subset of a given set S = {S_1, S_2, ..., S_n} of n positive integers, whose sum is equal to a given positive integer d. For example, if S= {1, 2, 5, 6, 8} and d = 9 there are two solutions {1, 2, 6} and {1, 8}. A suitable message is to be displayed if the given problem instance doesn't have a solution.

```
class sos {

int m;
        int w[];
        int x[];
        public sos() {
                w = new int[40];
                x = new int[40];
        }

        public void sos1(int s, int k, int r) {
                int i;
                x[k] = 1;
                if (s + w[k] == m) {
                        for (i = 0; i <= k; i++)
                                System.out.print(x[i] + "\t");
                        System.out.println();
                        System.out.print(" elements of set are \n");
                        for (i = 0; i <= k; i++)

                                if (x[i] == 1)

                                        System.out.print(w[i] + "\t");

                        System.out.println();

                } else if ((s + w[k] + w[k + 1]) <= m)
                        sos1(s + w[k], k + 1, r - w[k]);
                if ((s + r - w[k] >= m) && (s + w[k + 1] <= m)) {
                        x[k] = 0;
                        sos1(s, k + 1, r - w[k]);
                }
        }
}
```

```
class sosdemo {
        public static void main(String args[]) throws IOException {
                BufferedReader Bobj = new BufferedReader(new
InputStreamReader(System.in));
                int i, r = 0;
                sos o = new sos();
                System.out.println(" enter the number of elements of set:
");
                int n = Integer.parseInt(Bobj.readLine());
                System.out.print("\n enter the elements: ");
                for (i = 0; i < n; i++) {
                        o.w[i] = Integer.parseInt(Bobj.readLine());
                        r = r + o.w[i];
                }

                System.out.print("\n enter the sum to be computed: ");
                o.m = Integer.parseInt(Bobj.readLine());
                System.out.print(" \n subset whose sum is " + o.m + " are
as follows: ");
                o.sos1(0, 0, r);
        }
}




/*
    output

enter the number of elements of set: 4
enter the elements:
11 13 24 7
enter the sum to be computed: 31
subset whose sum is 31 are as follows:
  0   0   1   1
elements of set are
24   7
Process Exit...

*/
```