Aim:-Implement N-queen problem & sum of subsets using backtracking method.

Theory:- **Backtracking** is a general [algorithm](https://en.wikipedia.org/wiki/Algorithm) for finding all (or some) solutions to some [computational problems](https://en.wikipedia.org/wiki/Computational_problem), notably [constraint satisfaction problems](https://en.wikipedia.org/wiki/Constraint_satisfaction_problem), that incrementally builds candidates to the solutions, and abandons a candidate ("backtracks") as soon as it determines that the candidate cannot possibly be completed to a valid solution.[[1]](https://en.wikipedia.org/wiki/Backtracking#cite_note-1)[[2]](https://en.wikipedia.org/wiki/Backtracking#cite_note-2)

The classic textbook example of the use of backtracking is the [eight queens puzzle](https://en.wikipedia.org/wiki/Eight_queens_puzzle), that asks for all arrangements of eight [chess](https://en.wikipedia.org/wiki/Chess) [queens](https://en.wikipedia.org/wiki/Queen_(chess)) on a standard [chessboard](https://en.wikipedia.org/wiki/Chessboard) so that no queen attacks any other. In the common backtracking approach, the partial candidates are arrangements of *k* queens in the first *k* rows of the board, all in different rows and columns. Any partial solution that contains two mutually attacking queens can be abandoned.

The **eight queens puzzle** is the problem of placing eight [chess](https://en.wikipedia.org/wiki/Chess) [queens](https://en.wikipedia.org/wiki/Queen_(chess)) on an 8×8 chessboard so that no two queens threaten each other; thus, a solution requires that no two queens share the same row, column, or diagonal. The eight queens puzzle is an example of the more general ***n* queens problem** of placing *n* non-attacking queens on an *n*×*n* chessboard, for which solutions exist for all natural numbers *n* with the exception of *n*=2 and *n*=3

Program for N-queen.

#include<stdio.h>

#include<stdbool.h>

int N;

void printSolution(int board[N][N])

{

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

printf(" %d ", board[i][j]);

printf("\n");

}

}

int isSafe(int board[N][N], int row, int col)

{

int i, j;

/\* Check this row on left side \*/

for (i = 0; i < col; i++)

if (board[row][i])

return 0;

/\* Check upper diagonal on left side \*/

for (i=row, j=col; i>=0 && j>=0; i--, j--)

if (board[i][j])

return 0;

/\* Check lower diagonal on left side \*/

for (i=row, j=col; j>=0 && i<N; i++, j--)

if (board[i][j])

return 0;

return 1;

}

int solveNQUtil(int board[N][N], int col)

{

/\* base case: If all queens are placed

then return true \*/

if (col >= N)

return 1;

/\* Consider this column and try placing

this queen in all rows one by one \*/

for (int i = 0; i < N; i++)

{

/\* Check if the queen can be placed on

board[i][col] \*/

if ( isSafe(board, i, col) )

{

/\* Place this queen in board[i][col] \*/

board[i][col] = 1;

/\* recur to place rest of the queens \*/

if ( solveNQUtil(board, col + 1) )

return 1;

/\* If placing queen in board[i][col]

doesn't lead to a solution, then

remove queen from board[i][col] \*/

board[i][col] = 0; // BACKTRACK

}

}

/\* If the queen cannot be placed in any row in

this colum col then return false \*/

return 0;

}

int solveNQ()

{

int n;

printf("enter board size\n");

scanf("%d",&n);

N=n;int i=0,j=0;

int board[N][N];

for( i=0;i<N;i++)

for(j=0;j<N;j++)

board[i][j]=0;

if ( solveNQUtil(board, 0) == 0 )

{

printf("Solution does not exist");

return 0;

}

printSolution(board);

return 1;

}

int main()

{

solveNQ();

return 0;

}

Output:-

enter board size

6

 0  0  0  1  0  0

 1  0  0  0  0  0

 0  0  0  0  1  0

 0  1  0  0  0  0

 0  0  0  0  0  1

 0  0  1  0  0  0

Theory:- Subset sum problem is to find subset of elements that are selected from a given set whose sum adds up to a given number K. We are considering the set contains non-negative values. It is assumed that the input set is unique (no duplicates are presented).

Program for Sum of subsets.

#include<stdio.h>

void printSubset(int subset[],int a[],int size)

{

for(int i=0;i<size;i++)

{

if(subset[i]==1)

printf("%d\t",a[i]);

}

printf("\n");

}

void getSubset1(int subset[],int i,int a[],int size,int sum,int currentSum,int remainingSum)

{

if(currentSum==sum)

{

printSubset(subset,a,size);

}

if((i+1)!=size && currentSum<sum)

{

if(currentSum+remainingSum>=sum)

{

i++;remainingSum-=a[i];

subset[i]=1;

currentSum+=a[i];

getSubset1(subset,i,a,size,sum,currentSum,remainingSum);

subset[i]=0;

currentSum-=a[i];

getSubset1(subset,i,a,size,sum,currentSum,remainingSum);

}

}

}

int main()

{

int a[]={5, 7, 10, 12, 15, 18, 20};

int size=7;

int sum=35;

int subset[5]={0};

getSubset1(subset,-1,a,size,sum,0,87);

return 0;

}

Output:-

5       10      20

5       12      18

7       10      18

15      20

Conclusion:-

Backtracking algorithm is used for the problems which have multiple solutions and to reach the solution multiple path is to be worked and in the process accepts the paths which leads to solution while discards which don't.