**CSE2012-LAB**

**NAME** : O G RAGAVI

**REG NO**: 20BCE1988

**EX N0:** 8

**TITLE:** N-QUEENS PROBLEM

1.BRUTE FORCE C++ PROGRAM TO FIND SOLUTION FOR **n=5**:

CODE:

#include<iostream>

using namespace std;

#include<vector>

#include<stack>

void print\_board(vector<vector<int> > board)

{

int n,i,j;

n = board.size();

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

{

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

cout<<board[i][j]<<" ";

cout<<endl;

}

cout<<"Completed"<<endl;

}

bool not\_attacked(vector<vector<int> > &board,int row,int col)

{

int n = board.size(),i,j;

// check if there is a queen in the same row

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

{

if((j!=col)&&(board[row][j]==1))

return 0;

}

// Check if there is a queen in same column

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

{

if((i!=row)&&(board[i][col]==1))

return 0;

}

// both row and column increase

for(i=row+1,j=col+1;i<n&&j<n;i++,j++)

{

if(board[i][j]==1)

return 0;

}

// both row and column decrease

for(i=row-1,j=col-1;i>=0&&j>=0;i--,j--)

{

if(board[i][j]==1)

return 0;

}

// row increase and column decrease

for(i=row+1,j=col-1;i<n&&j>=0;i++,j--)

{

if(board[i][j]==1)

return 0;

}

// row decrease and column increase

for(i=row-1,j=col+1;i>=0&&j<n;i--,j++)

{

if(board[i][j]==1)

return 0;

}

return 1;

}

void n\_queens(vector<vector<int> > &board, int n, int curr\_row, stack<int> &pos)

{

int i;

// if all n-queens are placed then print board and return

// since we only print only one solution

if(curr\_row==n)

{

print\_board(board);

return ;

}

// if stack do not have an entry for current row then start

// from first column

if(pos.size()<=curr\_row)

i = 0;

// otherwise backtrack

else

{

// try column next to previous try

i = pos.top()+1;

// remove queen from previous column

board[curr\_row][pos.top()] = 0;

// remove previous entry in stack

pos.pop();

}

for (;i<n;i++)

{

// if ith column is valid

if (not\_attacked(board,curr\_row,i))

{

// store i in stack for current row

pos.push(i);

// mark in the board

board[curr\_row][i] = 1;

// try to place queen in next row

n\_queens(board,n,curr\_row+1,pos);

// return

return;

}

}

// since queen cannot be placed in current row

// backtrack to previous row

n\_queens(board,n,curr\_row-1,pos);

}

int main()

{

int n,i,j;

cin>>n;

vector<int> row(n,0);

vector<vector<int> > board(n, row);

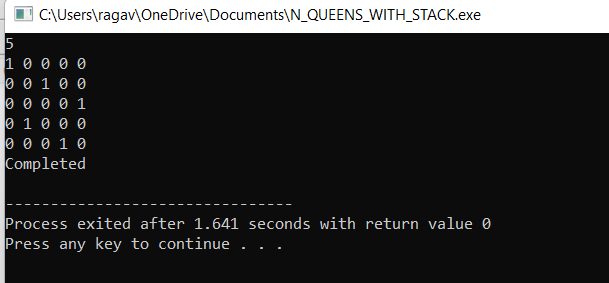
// stack to keep track of column positions taken for a row

stack<int> pos;

n\_queens(board,n,0,pos);

}

OUTPUT:



2.BRUTE FORCE C++ PROGRAM TO FIND SOLUTION FOR **n=6:**

**CODE:**

#include<iostream>

using namespace std;

#include<vector>

#include<stack>

void print\_board(vector<vector<int> > board)

{

int n,i,j;

n = board.size();

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

{

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

cout<<board[i][j]<<" ";

cout<<endl;

}

cout<<"Completed"<<endl;

}

bool not\_attacked(vector<vector<int> > &board,int row,int col)

{

int n = board.size(),i,j;

// check if there is a queen in the same row

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

{

if((j!=col)&&(board[row][j]==1))

return 0;

}

// Check if there is a queen in same column

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

{

if((i!=row)&&(board[i][col]==1))

return 0;

}

// both row and column increase

for(i=row+1,j=col+1;i<n&&j<n;i++,j++)

{

if(board[i][j]==1)

return 0;

}

// both row and column decrease

for(i=row-1,j=col-1;i>=0&&j>=0;i--,j--)

{

if(board[i][j]==1)

return 0;

}

// row increase and column decrease

for(i=row+1,j=col-1;i<n&&j>=0;i++,j--)

{

if(board[i][j]==1)

return 0;

}

// row decrease and column increase

for(i=row-1,j=col+1;i>=0&&j<n;i--,j++)

{

if(board[i][j]==1)

return 0;

}

return 1;

}

void n\_queens(vector<vector<int> > &board, int n, int curr\_row, stack<int> &pos)

{

int i;

// if all n-queens are placed then print board and return

// since we only print only one solution

if(curr\_row==n)

{

print\_board(board);

return ;

}

// if stack do not have an entry for current row then start

// from first column

if(pos.size()<=curr\_row)

i = 0;

// otherwise backtrack

else

{

// try column next to previous try

i = pos.top()+1;

// remove queen from previous column

board[curr\_row][pos.top()] = 0;

// remove previous entry in stack

pos.pop();

}

for (;i<n;i++)

{

// if ith column is valid

if (not\_attacked(board,curr\_row,i))

{

// store i in stack for current row

pos.push(i);

// mark in the board

board[curr\_row][i] = 1;

// try to place queen in next row

n\_queens(board,n,curr\_row+1,pos);

// return

return;

}

}

// since queen cannot be placed in current row

// backtrack to previous row

n\_queens(board,n,curr\_row-1,pos);

}

int main()

{

int n,i,j;

cin>>n;

vector<int> row(n,0);

vector<vector<int> > board(n, row);

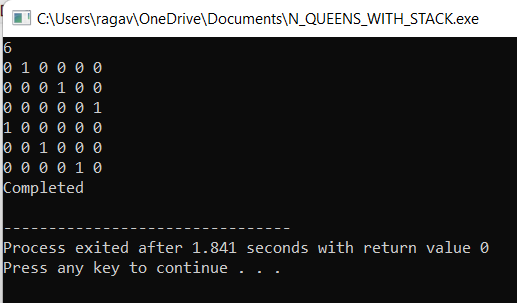
// stack to keep track of column positions taken for a row

stack<int> pos;

n\_queens(board,n,0,pos);

}

OUTPUT:



3. BRUTE FORCE C++ PROGRAM TO FIND SOLUTION FOR **n=8:**

CODE:

using namespace std;

#include<vector>

#include<stack>

void print\_board(vector<vector<int> > board)

{

int n,i,j;

n = board.size();

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

{

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

cout<<board[i][j]<<" ";

cout<<endl;

}

cout<<"Completed"<<endl;

}

bool not\_attacked(vector<vector<int> > &board,int row,int col)

{

int n = board.size(),i,j;

// check if there is a queen in the same row

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

{

if((j!=col)&&(board[row][j]==1))

return 0;

}

// Check if there is a queen in same column

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

{

if((i!=row)&&(board[i][col]==1))

return 0;

}

// both row and column increase

for(i=row+1,j=col+1;i<n&&j<n;i++,j++)

{

if(board[i][j]==1)

return 0;

}

// both row and column decrease

for(i=row-1,j=col-1;i>=0&&j>=0;i--,j--)

{

if(board[i][j]==1)

return 0;

}

// row increase and column decrease

for(i=row+1,j=col-1;i<n&&j>=0;i++,j--)

{

if(board[i][j]==1)

return 0;

}

// row decrease and column increase

for(i=row-1,j=col+1;i>=0&&j<n;i--,j++)

{

if(board[i][j]==1)

return 0;

}

return 1;

}

void n\_queens(vector<vector<int> > &board, int n, int curr\_row, stack<int> &pos)

{

int i;

// if all n-queens are placed then print board and return

// since we only print only one solution

if(curr\_row==n)

{

print\_board(board);

return ;

}

// if stack do not have an entry for current row then start

// from first column

if(pos.size()<=curr\_row)

i = 0;

// otherwise backtrack

else

{

// try column next to previous try

i = pos.top()+1;

// remove queen from previous column

board[curr\_row][pos.top()] = 0;

// remove previous entry in stack

pos.pop();

}

for (;i<n;i++)

{

// if ith column is valid

if (not\_attacked(board,curr\_row,i))

{

// store i in stack for current row

pos.push(i);

// mark in the board

board[curr\_row][i] = 1;

// try to place queen in next row

n\_queens(board,n,curr\_row+1,pos);

// return

return;

}

}

// since queen cannot be placed in current row

// backtrack to previous row

n\_queens(board,n,curr\_row-1,pos);

}

int main()

{

int n,i,j;

cin>>n;

vector<int> row(n,0);

vector<vector<int> > board(n, row);

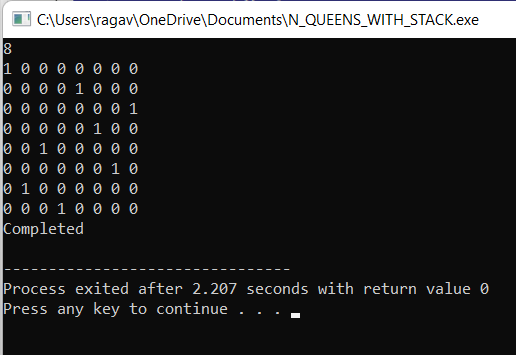
// stack to keep track of column positions taken for a row

stack<int> pos;

n\_queens(board,n,0,pos);

}

OUTPUT:



4. BRUTE FORCE C++ PROGRAM TO FIND SOLUTION FOR **n=9:**

**CODE:**

using namespace std;

#include<vector>

#include<stack>

void print\_board(vector<vector<int> > board)

{

int n,i,j;

n = board.size();

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

{

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

cout<<board[i][j]<<" ";

cout<<endl;

}

cout<<"Completed"<<endl;

}

bool not\_attacked(vector<vector<int> > &board,int row,int col)

{

int n = board.size(),i,j;

// check if there is a queen in the same row

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

{

if((j!=col)&&(board[row][j]==1))

return 0;

}

// Check if there is a queen in same column

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

{

if((i!=row)&&(board[i][col]==1))

return 0;

}

// both row and column increase

for(i=row+1,j=col+1;i<n&&j<n;i++,j++)

{

if(board[i][j]==1)

return 0;

}

// both row and column decrease

for(i=row-1,j=col-1;i>=0&&j>=0;i--,j--)

{

if(board[i][j]==1)

return 0;

}

// row increase and column decrease

for(i=row+1,j=col-1;i<n&&j>=0;i++,j--)

{

if(board[i][j]==1)

return 0;

}

// row decrease and column increase

for(i=row-1,j=col+1;i>=0&&j<n;i--,j++)

{

if(board[i][j]==1)

return 0;

}

return 1;

}

void n\_queens(vector<vector<int> > &board, int n, int curr\_row, stack<int> &pos)

{

int i;

// if all n-queens are placed then print board and return

// since we only print only one solution

if(curr\_row==n)

{

print\_board(board);

return ;

}

// if stack do not have an entry for current row then start

// from first column

if(pos.size()<=curr\_row)

i = 0;

// otherwise backtrack

else

{

// try column next to previous try

i = pos.top()+1;

// remove queen from previous column

board[curr\_row][pos.top()] = 0;

// remove previous entry in stack

pos.pop();

}

for (;i<n;i++)

{

// if ith column is valid

if (not\_attacked(board,curr\_row,i))

{

// store i in stack for current row

pos.push(i);

// mark in the board

board[curr\_row][i] = 1;

// try to place queen in next row

n\_queens(board,n,curr\_row+1,pos);

// return

return;

}

}

// since queen cannot be placed in current row

// backtrack to previous row

n\_queens(board,n,curr\_row-1,pos);

}

int main()

{

int n,i,j;

cin>>n;

vector<int> row(n,0);

vector<vector<int> > board(n, row);

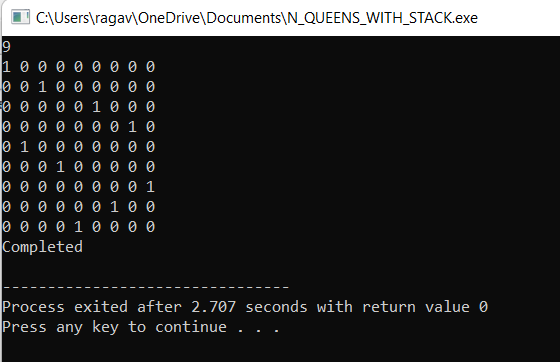
// stack to keep track of column positions taken for a row

stack<int> pos;

n\_queens(board,n,0,pos);

}

OUTPUT:



6.GREEDY C++ PROGRAM FOR N-QUEENS(n=8):