**РУСЕНСКИ УНИВЕРСИТЕТ “АНГЕЛ КЪНЧЕВ”**

**КУРСОВА РАБОТА**

**№1**

**по Изкуствен интелект**

Студент:

Факултетен номер:

Група:

Специалност:

Курс:

**Изготвил:**

**Дата: Проверил:**

# Задание.

Да се разположат върху шахматна дъска 8 царици така, че да не се “бият”.

Алгоритъм: търсене в дълбочина [*DFS*].

# Представяне на задачата като задача за търсене в ПС.

За да се поставят върху една шахматна дъска 8 царици така, че да не се „бият“ е необходимо да няма повече от една царица на ред, колона и по диагоналите.

За по-прегледно приемаме, че цариците са 4 (принципът е същият). С ‚-‘ са обозначени местата, които са забранени за поставянето на други царици, а с ‚\*‘ цариците.



|  |  |  |  |
| --- | --- | --- | --- |
| - | \* | - | - |
| - | - | - |  |
|  | - |  | - |
|  | - |  |  |



|  |  |  |  |
| --- | --- | --- | --- |
| - | \* | - | - |
| - | - | - | \* |
|  | - | - | - |
|  | - |  | - |



|  |  |  |  |
| --- | --- | --- | --- |
| - | \* | - | - |
| - | - | - | \* |
| \* | - | - | - |
| - | - |  | - |



|  |  |  |  |
| --- | --- | --- | --- |
| - | \* | - | - |
| - | - | - | \* |
| \* | - | - | - |
| - | - | \* | - |

# Програмна реализация.

## Псевдо-код на алгоритъма.

void FindSolutionDFS(брой\_царици N) {

добавяне\_на\_всички\_възможни\_решения\_от\_ниво\_0\_в\_опашката;

do {

вземи първият елемент от опашката;

*проверяване на всички наследници:*

Ако може да се постави „царица“ - маркиране

--провери дали е последната – Level == N

--печатай решението

--изход

--ако не е

--добави наследника в опашката

--демаркирай

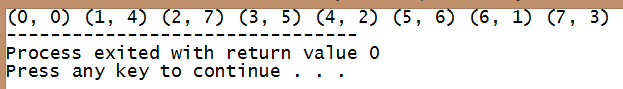
} while(true);

}

## Описание на разработениете модули.

* **Вид на функцията**: void FindSolutionDFS(int n)
* **Действие**: Функцията служи за намиране решението на „8-queens problem“, чрез *DFS* алгоритъма;
* **Параметри**: int n – брой царици;
* **Вид на функцията**: int main(void)
* **Действие**: Главна функция
* **Параметри**: няма;
* **Използвани функции**: FindSolutionDFS().

## Тестови примери.



## Код на програмата.

**#include <vector>**

**#include <queue>**

**#include <iostream>**

**#include <cstdlib> // abs()**

**using namespace std;**

**struct Position {**

**short row;**

**short column;**

**};**

**struct NBoard {**

**int N;**

**int LIdx;**

**vector<Position> Assign;**

**NBoard(const NBoard & a) {**

**N = a.N;**

**LIdx = a.LIdx;**

**Assign.resize(N);**

**copy(a.Assign.begin(), a.Assign.end(), Assign.begin());**

**}**

**NBoard(int n) {**

**N = n;**

**LIdx = -1;**

**Assign.resize(N);**

**}**

**NBoard & operator = (const NBoard & other) {**

**if (this != &other) {**

**if (N != other.N)**

**Assign.resize(other.N);**

**copy(other.Assign.begin(), other.Assign.end(), Assign.begin());**

**N = other.N;**

**LIdx = other.LIdx;**

**}**

**return \*this;**

**}**

**bool PlaceQueen(Position p) {**

**for (int i = 0; i <= LIdx; i++) {**

**if (p.column == Assign[i].column ||**

**abs(p.row - Assign[i].row) == abs(p.column - Assign[i].column))**

**return false;**

**}**

**LIdx++;**

**Assign[LIdx] = p;**

**return true;**

**}**

**void RemoveQueen() {**

**LIdx--;**

**}**

**int Level() {**

**return LIdx + 1;**

**}**

**};**

**void FindSolutionDFS(int n) {**

**NBoard curBoard(n);**

**queue<NBoard> boards;**

**Position pos;**

**for (int i = 0; i < n; i++){**

**pos.row = 0;**

**pos.column = i;**

**curBoard.PlaceQueen(pos);**

**boards.push(curBoard);**

**curBoard.RemoveQueen();**

**}**

**do {**

**curBoard = boards.front();**

**boards.pop();**

**pos.row = curBoard.Level();**

**for (int i = 0; i < n; i++) {**

**pos.column = i;**

**if (curBoard.PlaceQueen(pos)) {**

**if (curBoard.Level() == n) {**

**/\* Печатане на решението и прекратяване на търсенето на други \*/**

**for (vector<Position>::iterator it = curBoard.Assign.begin();**

**it != curBoard.Assign.end(); ++it) {**

**cout << "(" << it->row << ", " << it->column << ") ";**

**}**

**return;**

**}**

**else {**

**boards.push(curBoard);**

**}**

**curBoard.RemoveQueen();**

**}**

**}**

**} while(true);**

**}**

**int main() {**

**int numOfQueens = 8;**

**FindSolutionDFS(numOfQueens);**

**return 0;**

**}**

# Творческа задача.

Достатъчно е да се премахне операторът return във функцията FindSolutionDFS() и да се добави проверка дали са намерени всички решения и тогава да се прекрати търсенето. Псевдо-кодът на алгоритъма става:

void FindSolutionDFS(брой\_царици N) {

добавяне\_на\_всички\_възможни\_решения\_от\_ниво\_0\_в\_опашката;

do {

Намерени ли са всички решения

--изход

вземи първият елемент от опашката;

*проверяване на всички наследници:*

Ако може да се постави „царица“ - маркиране

--провери дали е последната – Level == N

--печатай текущото решението

--ако не е

--добави наследника в опашката

--демаркирай

} while(true);

}

Кодът на алгоритъма става:

**int FindSolutionDFS(int n) {**

**long NumModel = 0;**

**NBoard curBoard(n);**

**queue<NBoard> boards;**

**Position pos;**

**for (int i = 0; i < n; i++){**

**pos.row = 0;**

**pos.column = i;**

**curBoard.PlaceQueen(pos);**

**boards.push(curBoard);**

**curBoard.RemoveQueen();**

**}**

**do {**

**/\* намерени ли са всички решени¤ \*/**

**if (boards.size() == 0) {**

**break;**

**}**

**curBoard = boards.front();**

**boards.pop();**

**pos.row = curBoard.Level();**

**for (int i = 0; i < n; i++) {**

**pos.column = i;**

**if (curBoard.PlaceQueen(pos)) {**

**if (curBoard.Level() == n) {**

**/\* увеличаване на брояча и печатане на текущото намерено решение \*/**

**NumModel++;**

**for (vector<Position>::iterator it = curBoard.Assign.begin();**

**it != curBoard.Assign.end(); ++it) {**

**cout << "(" << it->row << ", " << it->column << ") ";**

**}**

**cout << endl;**

**}**

**else {**

**boards.push(curBoard);**

**}**

**curBoard.RemoveQueen();**

**}**

**}**

**} while(true);**

**return NumModel;**

**}**

Функцията е леко променена: сега връща броя на намерените решения.

