Търсене с връщане назад

Изготвил:
гл.ас. д-р Нора Ангелова

Търсене с връщане назад

Общо название на клас от алгоритми за търсене на решение (решения) при ограничени условия.

При него всяко от решенията се строи стъпка по стъпка.

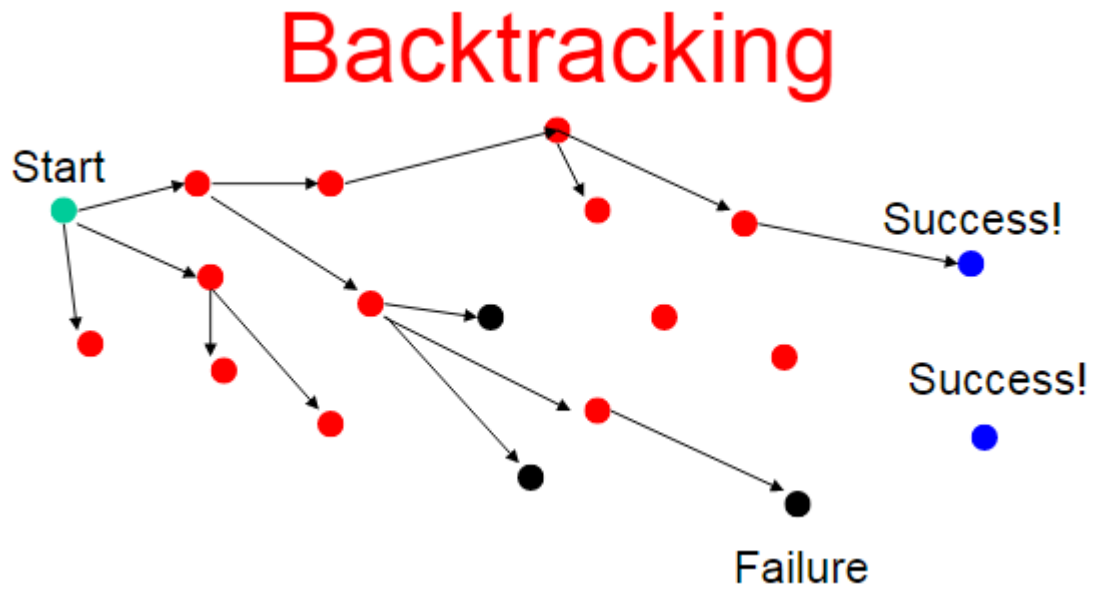
Частта от едно решение, построена до даден момент, се нарича **частично решение**.

Търсене с връщане назад

Решение:

1. Конструира се едно **частично решение**.
2. Проверява се дали частичното решение е търсеното (общо).
3. Ако **решението отговаря на условията**, то се запазва или извежда. Процесът на търсене или завършва, или продължава по същата схема, докато бъдат генерирани всички възможни решения.
4. Ако **частичното решение не е общо** се прави опит текущото частично решение да се продължи (според условието на задачата).
5. Ако на някоя стъпка се окаже **невъзможно ново продължение**, се извършва връщане назад към предишното частично решение и се прави нов опит то да се продължи по друг начин, различен от предишния.

Търсене с връщане назад



- Множество от състояния и действия(път до ново състояние).
- От едно състояние се виждат само състоянията до които има път.

Търсене с връщане назад

Играта СУДОКУ

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Търсене с връщане назад

Играта СУДОКУ

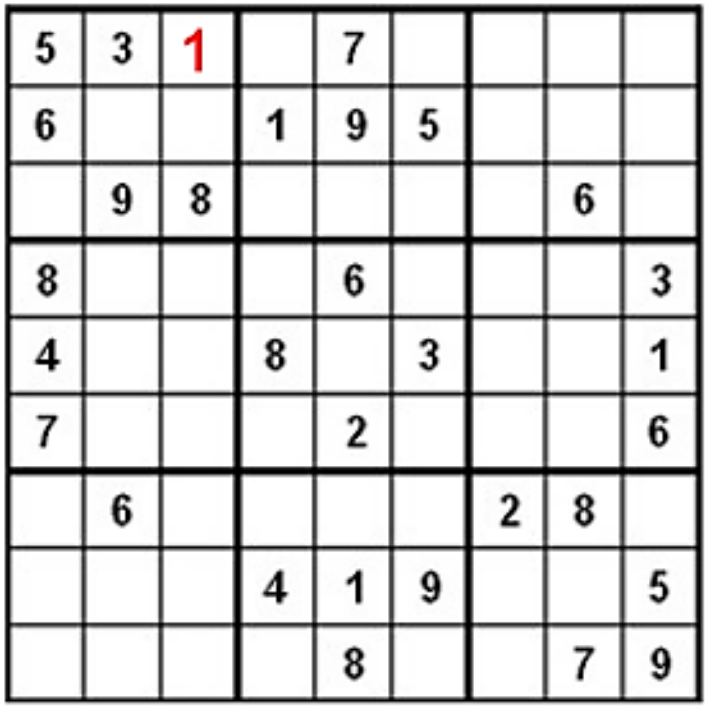
- Опитват се всички възможности до намирането на решение.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Търсене с връщане назад

Играта СУДОКУ

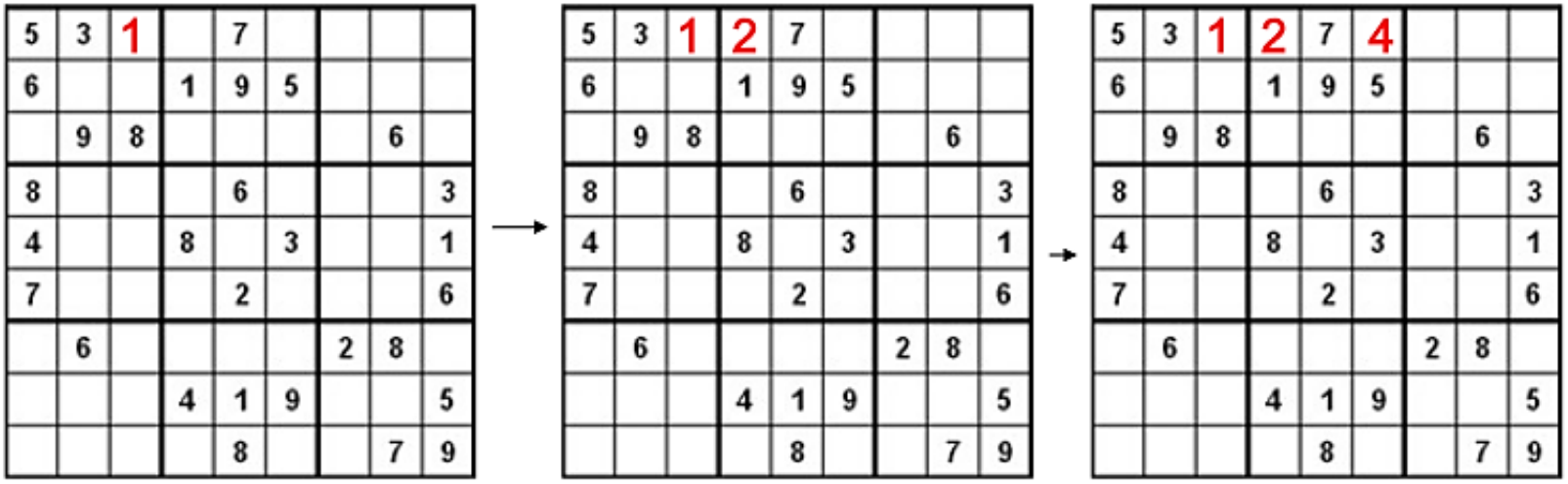
- Ако сме запълнили всички клетки, решението е намерено.
- В противен случай се търси празна клетка.
- Стартира се цикъл с всички възможни стойности от 1 до 9.
- Поставя се стойност и се проверява дали тя е валидна.
- *Продължава се с решението на същата задача.*

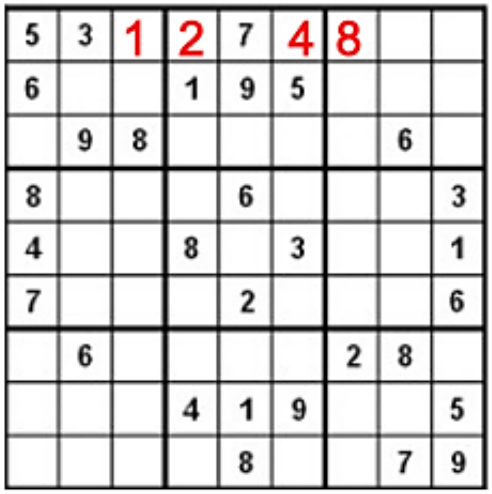


5	3	1		7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Търсене с връщане назад

Solving Sudoku – Later Steps





uh oh!

Търсене с връщане назад

Играта СУДОКУ

- Достигнато е невалидно частично решение, което не може да се продължи.
- Извършва се връщане назад към предишното частично решение (предишната клетка).
- *Алгоритъмът е запомнил, с кои числа е направен опит за достигане на решение.*

5	3	1	2	7	4	8	9	
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Търсене с връщане назад

Играта СУДОКУ

- Опитва се с нова стойност за текущата клетка.

Търсене с връщане назад

Предимства и недостатъци Играта СУДОКУ

- Лесен за реализация алгоритъм.
- Бавен заради проверката на всички възможности и липсата на оптимизация.

Търсене с връщане назад

Играта СУДОКУ
Решение

```
If at a solution, report success

for(every possible choice from current node) {
   Make that choice and take one step along path.
   Use recursion to solve the problem for the new node.
   If the recursive call succeeds, report the success to
   the next high level.
   Back out of the current choice to restore the state
   at the beginning of the loop.
}

Report failure
```

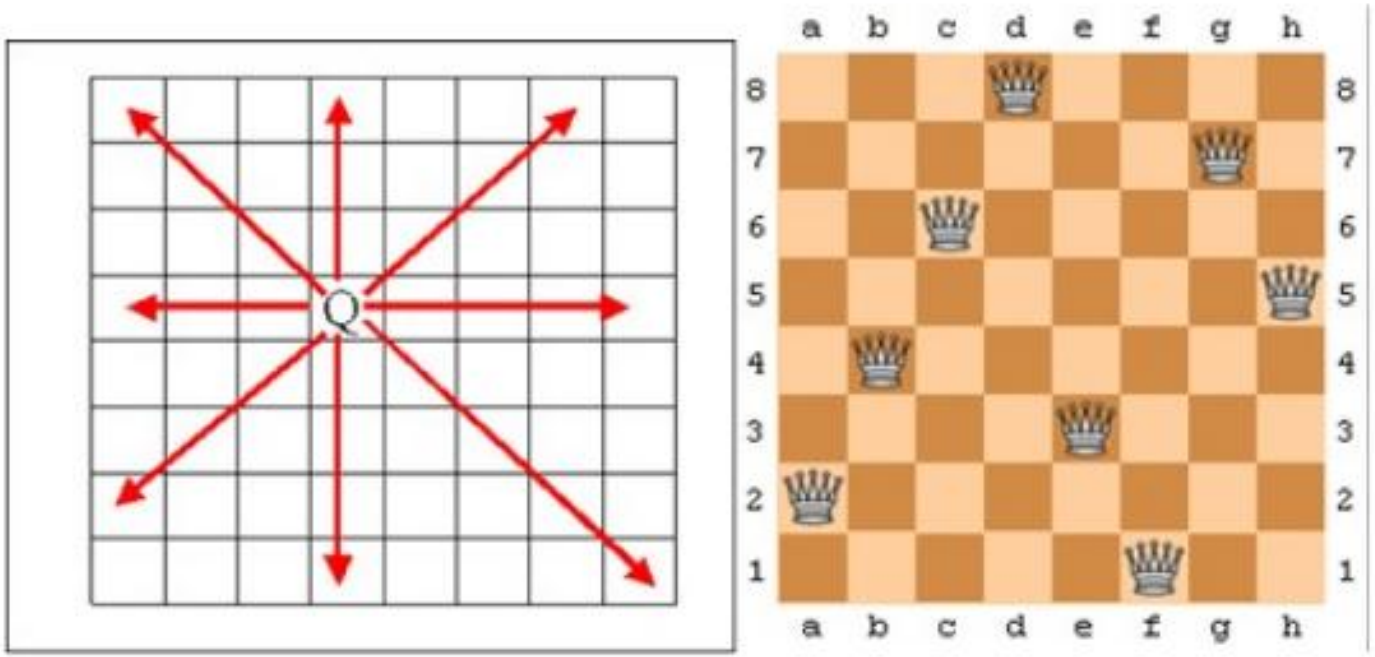
Търсене с връщане назад

Множество от задачи:

- Намиране на път до определена цел.
- Намиране на всички пътища.
- Намиране на най-добър път.

Търсене с връщане назад

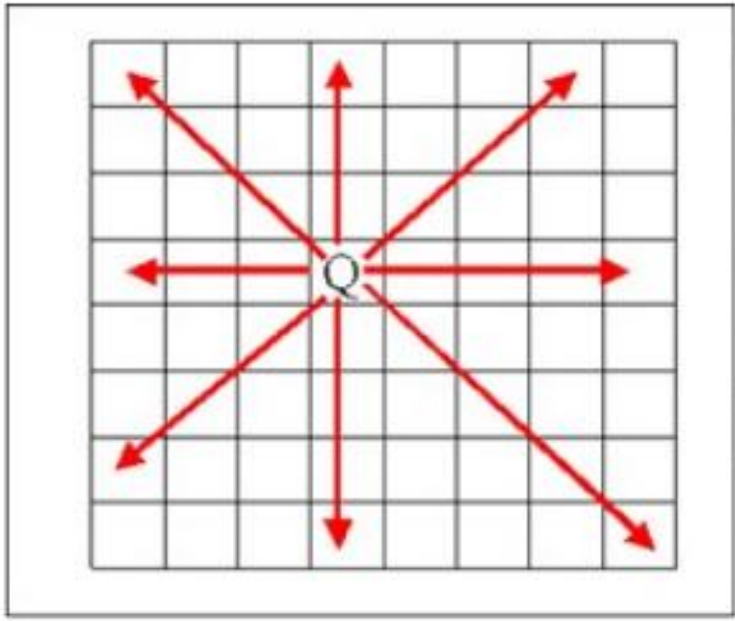
Задача за 8-те царици.



Търсене с връщане назад

Задача за 8-те царици.

- Дадени са N царици, които трябва да се поставят върху шахматна дъска с размери N x N.
- Колко са възможностите за поставяне на цариците?
- 8 x 8
  Има 64 квадтчета и 8 царици, които трябва да поставим в тях.



$$\frac{64 * 63 * 62 * 61 * 60 * 59 * 58 * 57}{8!} =$$

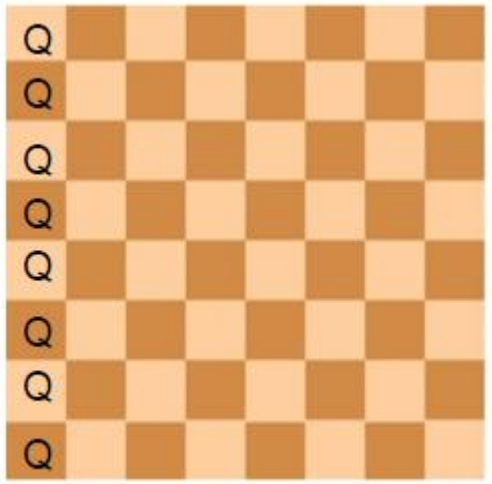
$$= \frac{78,462,\ 987,\ 637,\ 760}{8!} = 4,426,165,368$$

Търсене с връщане назад

Задача за 8-те царици.

Оптимизация

- Колко царици могат да стоят в една колона от шахматната дъска?
- Можем да изключим определени случаи.
- Колко са възможните варианти за поставяне на цариците?



$$8 * 8 * 8 * 8 * 8 * 8 * 8 * 8 = 16{,}777{,}216$$

Търсене с връщане назад

Задача за 8-те царици.

Решение

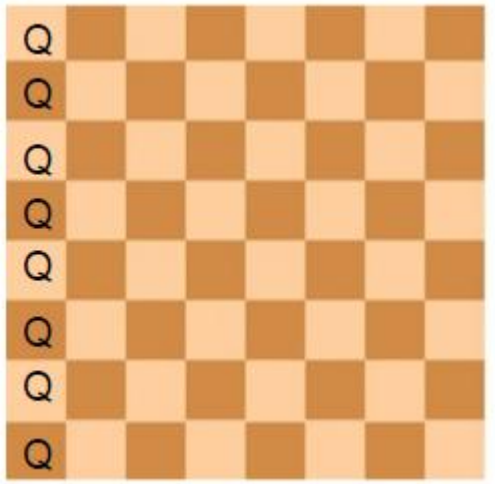
- Итерираме през редовете за всяка една от колоните.

```
for(int c0 = 0; c0 < 8; c0++){
        board[c0][0] = 'q';
         for(int c1 = 0; c1 < 8; c1++){
               board[c1][1] = 'q';
               for(int c2 = 0; c2 < 8; c2++){
                     board[c2][2] = 'q';
                     // a little later
                     for(int c7 = 0; c7 < 8; c7++){
                           board[c7][7] = 'q';
                           if( queensAreSafe(board) )
                                 printSolution(board);
                           board[c7][7] = ' '; //pick up queen
                     }
                     board[c6][6] = ' '; // pick up queen
```

Търсене с връщане назад

Задача за 8-те царици.

- Колко са циклите при N царици?
- Колко е N?

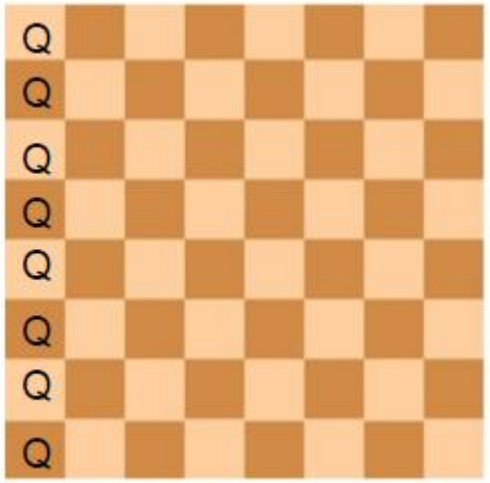


Търсене с връщане назад

Задача за 8-те царици. Решение

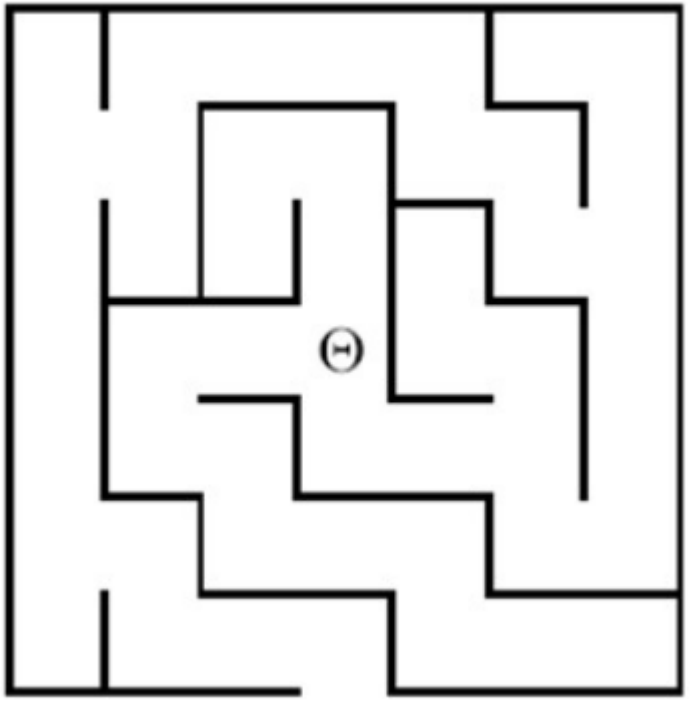
- Използва се същия шаблон за решението.

```
If at a solution, report success

for(every possible choice from current node) {
   Make that choice and take one step along path.
   Use recursion to solve the problem for the new node.
   If the recursive call succeeds, report the success to
   the next high level.
   Back out of the current choice to restore the state
   at the beginning of the loop.
}

Report failure
```



Търсене с връщане назад

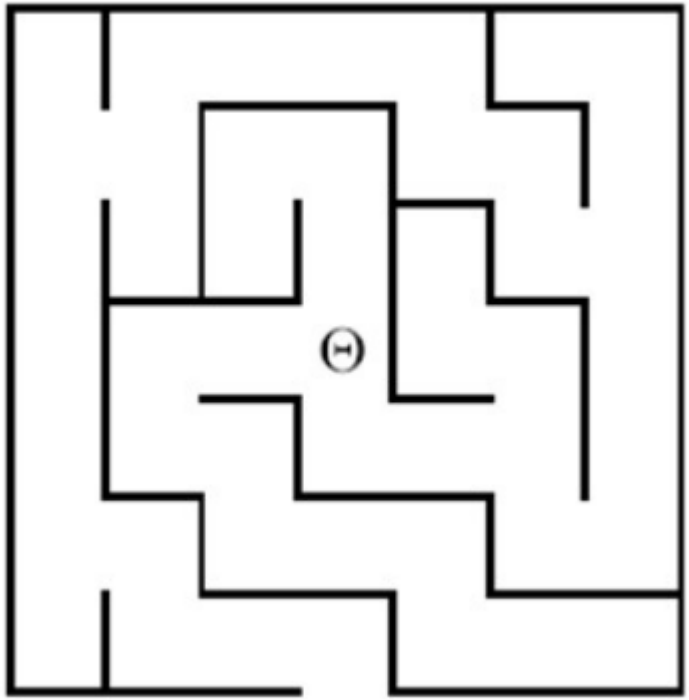
ЛАБИРИНТ



Търсене с връщане назад

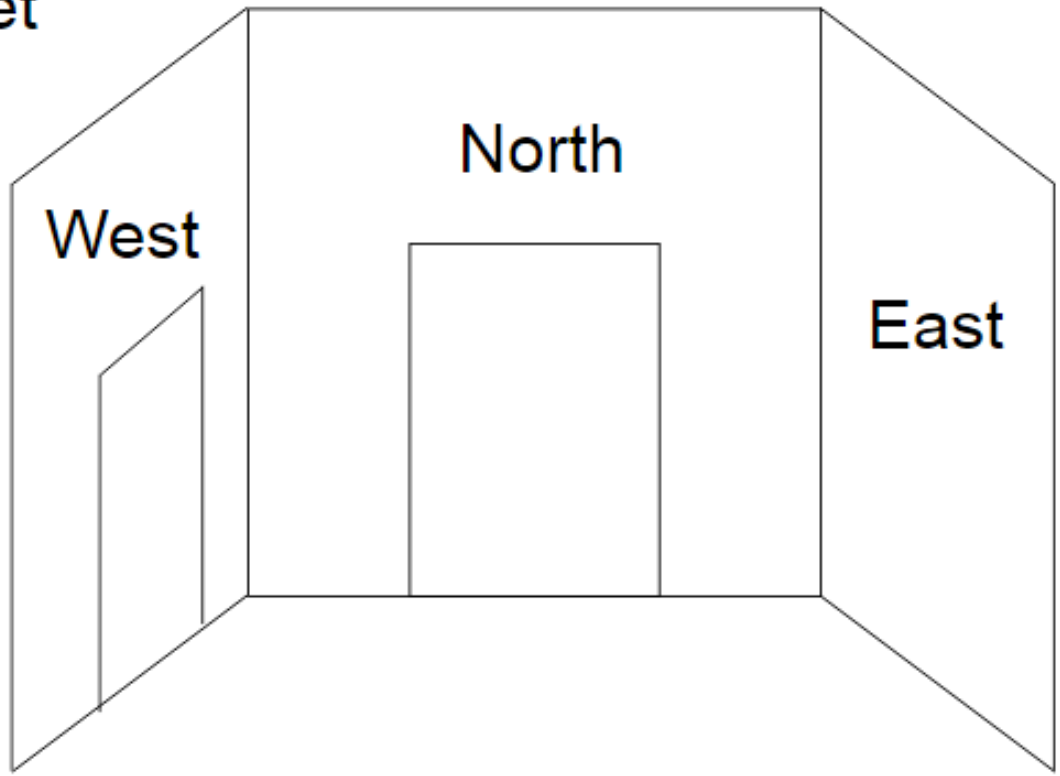
ЛАБИРИНТ

- Търси път докато не намери изхода.
- Ако няма възможен път се сигнализира.



Търсене с връщане назад

Which way do I go to get out?

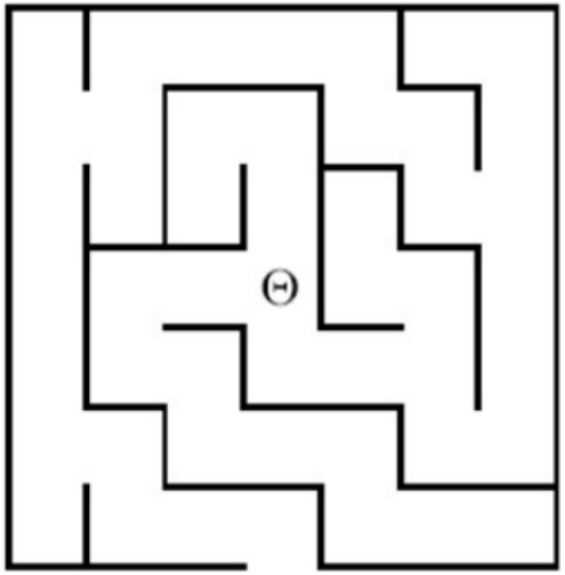


Behind me, to the South is a door leading South

Търсене с връщане назад

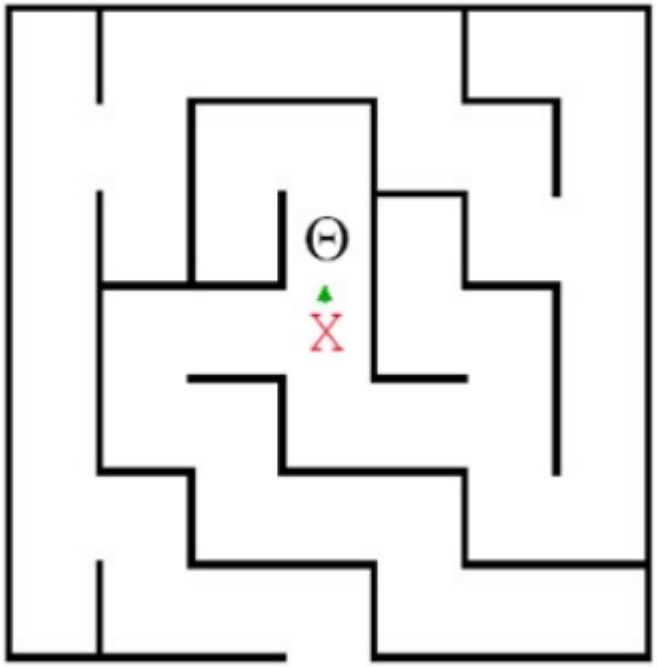
ЛАБИРИНТ
Решение

```
If the current square is outside, return TRUE to indicate
that a solution has been found.

Mark the current square.
for (each of the four compass directions) {
    if (this direction is not blocked by a wall and
        the current square is not marked) {
      Move one step in the indicated direction from the
current square.
      Try to solve the maze from there by making a recursive
call.
      If this call shows the maze to be solvable, return
TRUE to indicate that fact.
    }
}
Unmark the current square.
Return FALSE to indicate that none of the four directions
led to a solution.
```



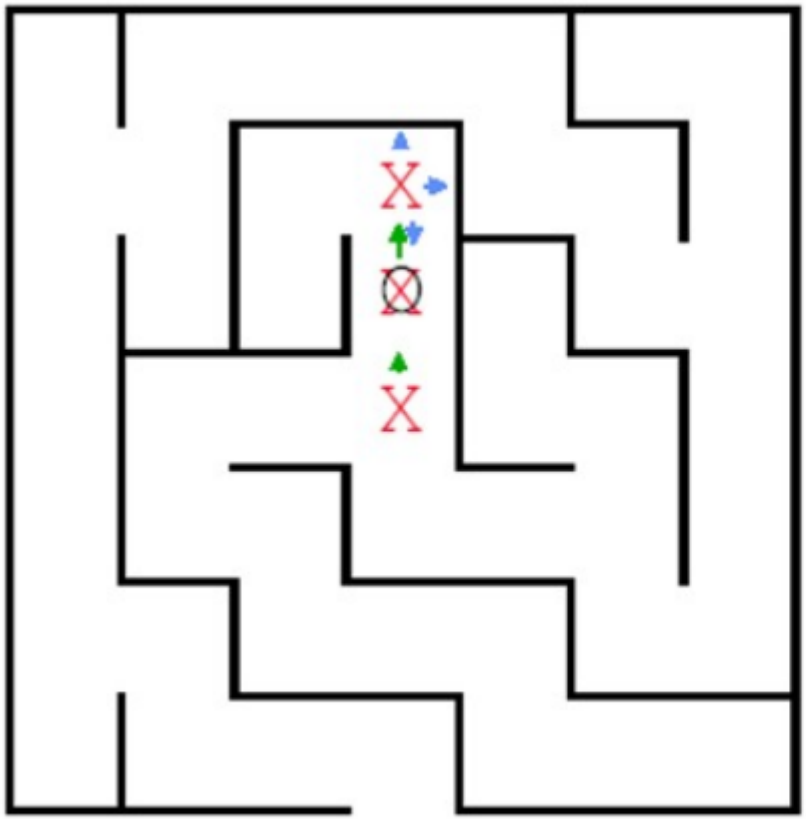
Търсене с връщане назад

ЛАБИРИНТ



Търсене с връщане назад

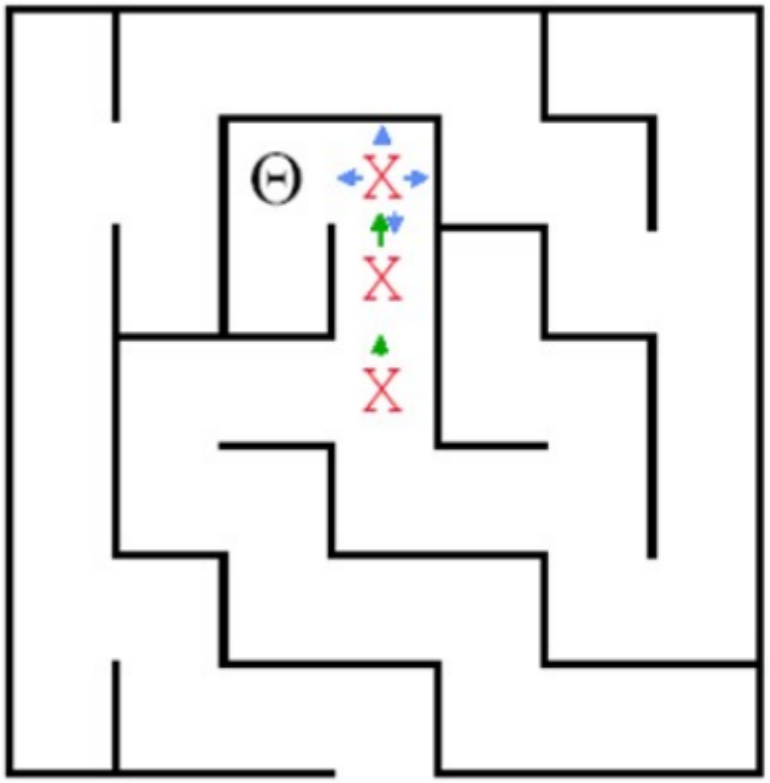
ЛАБИРИНТ



Here we have moved North again, but there is a wall to the North . East is also blocked, so we try South. That call discovers that the square is marked, so it just returns.

Търсене с връщане назад

ЛАБИРИНТ

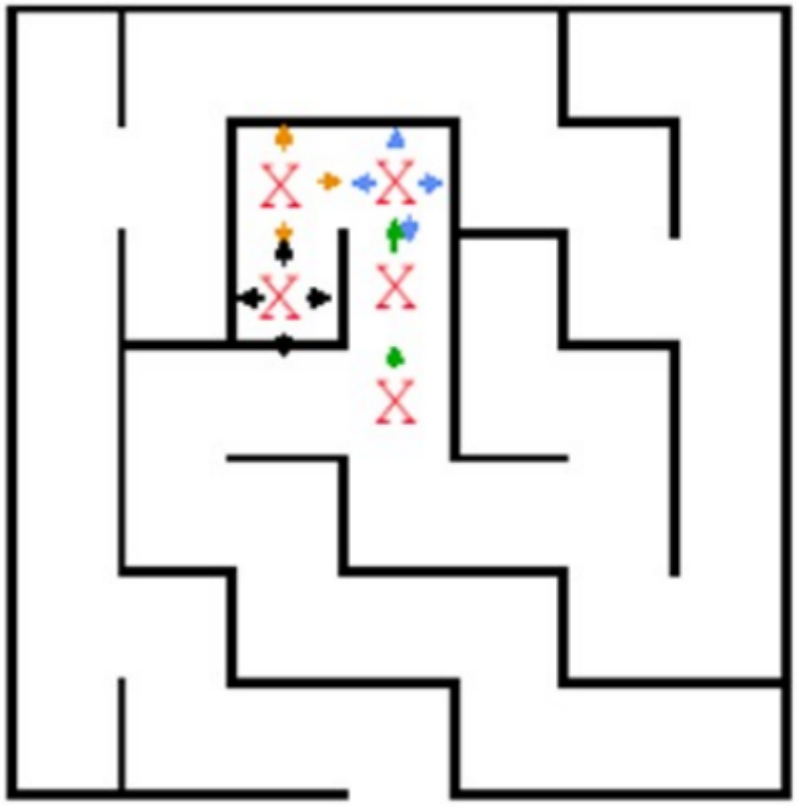


So the next move we can make is West.

Where is this leading?

Търсене с връщане назад

ЛАБИРИНТ



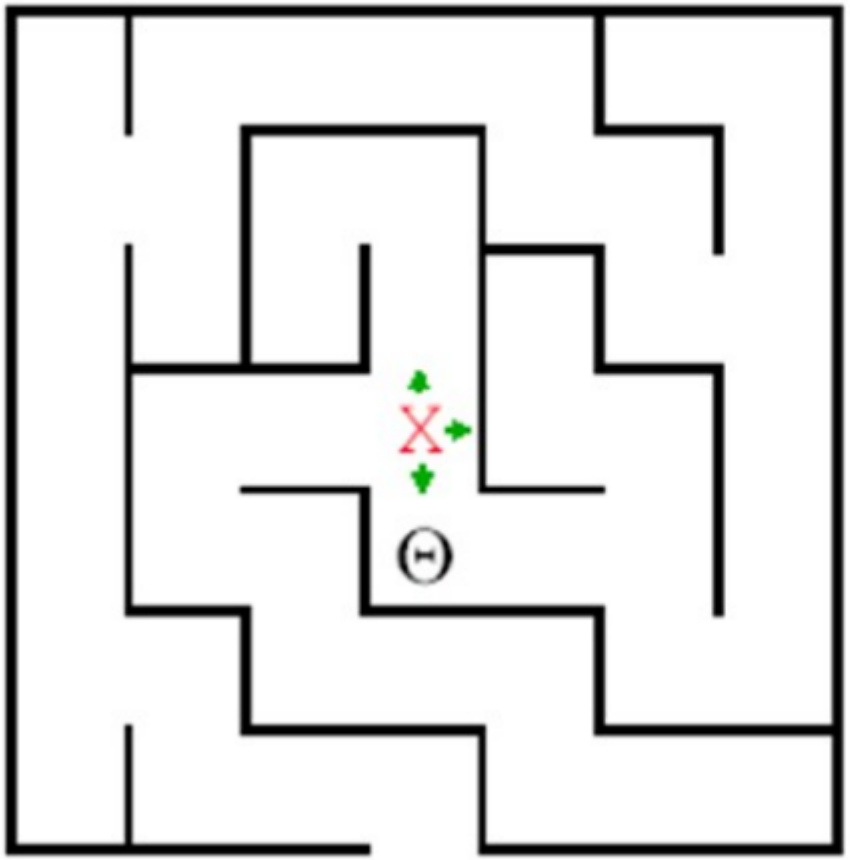
This path reaches a dead end.

Time to backtrack!

Remember the program stack!

Търсене с връщане назад

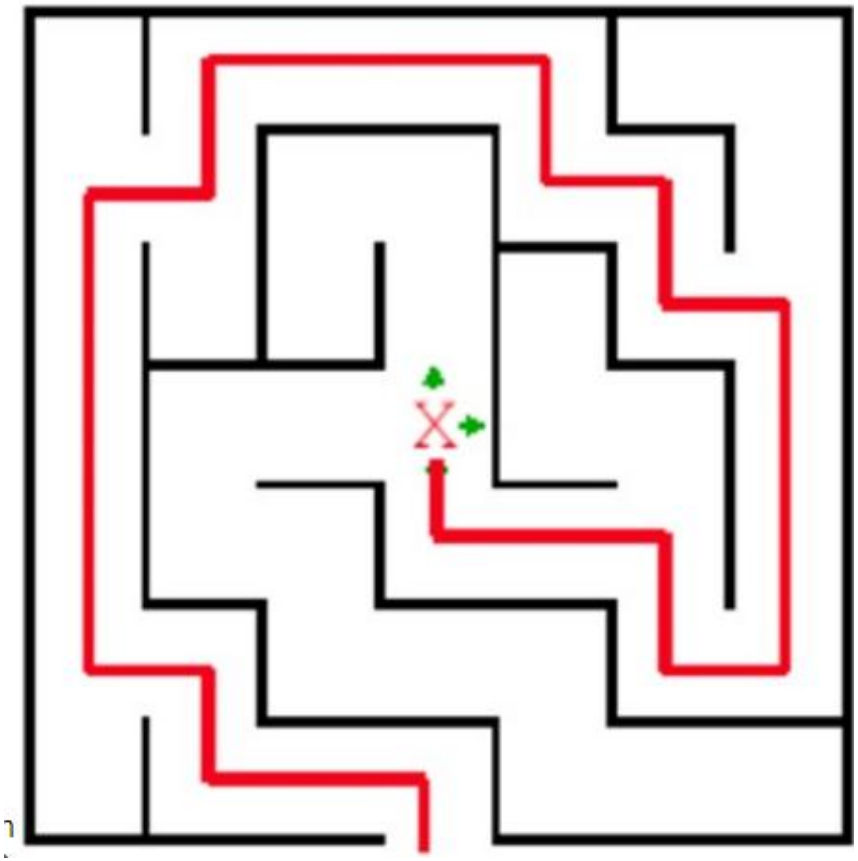
ЛАБИРИНТ



And now we try
South

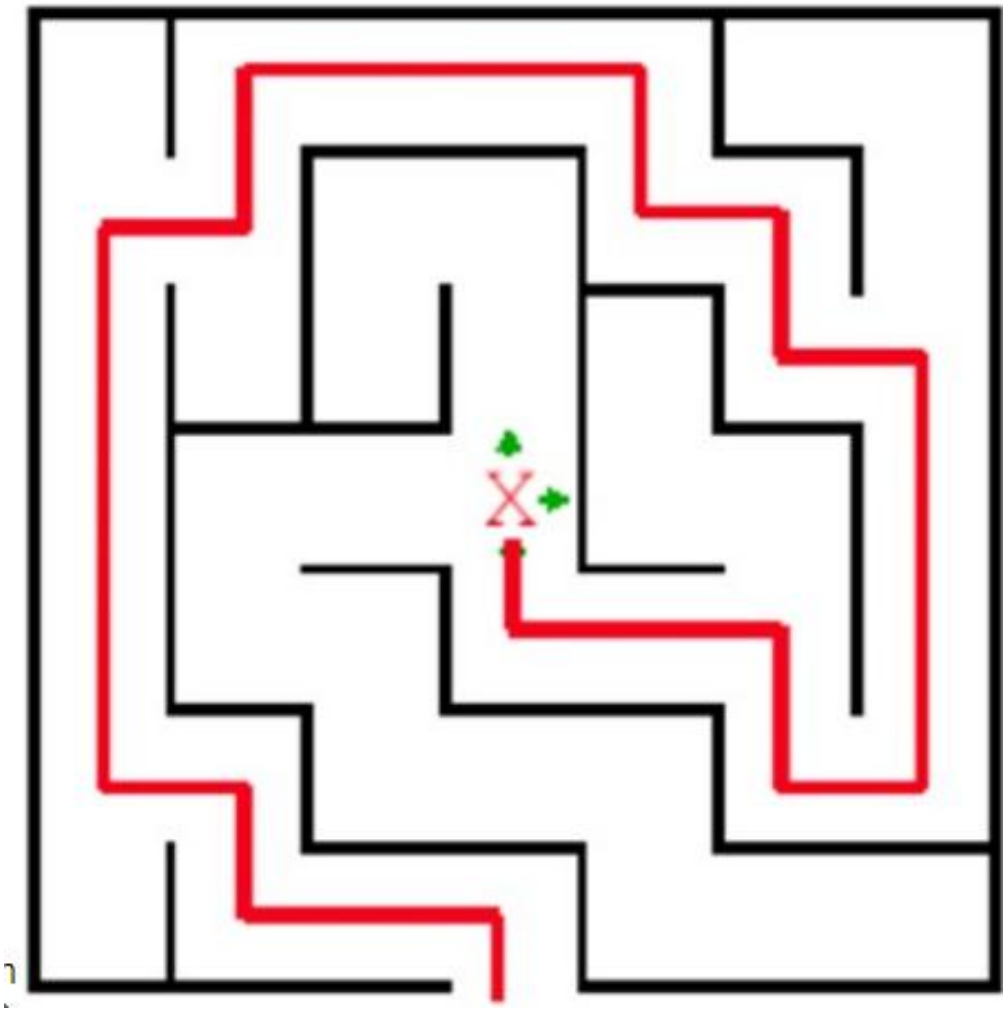
Търсене с връщане назад

ЛАБИРИНТ



Algorithm for Maze

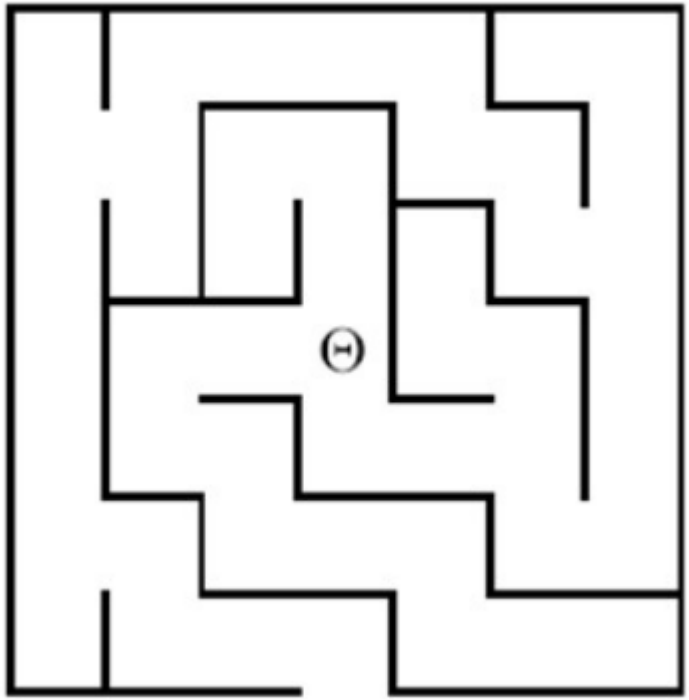
- One Path Found



Търсене с връщане назад

ЛАБИРИНТ

Лабиринт е представен с булева квадратна матрица 8 x 8.
Клетка се приема за проходима, ако елементът в съответната позиция е истина и за непроходима в противен случай.
Да се напише програма, която намира всички пътища от съседни в хоризонтално и вертикално направление проходими клетки на лабиринта, които започват в горния му ляв ъгъл и завършват в долния му десен ъгъл.



Търсене с връщане назад

```
#include <iostream>
using namespace std;

// глобален масив, съдържащ лабиринта
bool labyrinth[8][8] = {
    1, 0, 1, 1, 1, 1, 1, 1,
    1, 0, 1, 0, 0, 0, 0, 1,
    1, 1, 1, 0, 1, 1, 0, 1,
    0, 0, 0, 1, 1, 1, 0, 1,
    1, 1, 1, 1, 1, 1, 0, 1,
    1, 1, 1, 1, 1, 0, 0, 1,
    1, 1, 1, 1, 1, 0, 1, 1,
    1, 1, 1, 1, 1, 0, 1, 1
};
int pathCount = 1;
// извеждане на пътя, записан в масива way,
// като way[2*i] е абсцисата на i-тата клетка,
// а way[2*i+1] е ординатата й

void printWay(int *way, int n) {
    cout << "#" << pathCount;

    for (int i = 0; i < n-1; i++) {
        cout << "(" << way[2*i] << ","
             << way[2*i+1] << ")->";
    }
    cout << "(" << way[2*(n-1)] << ","
         << way[2*n-1] << ")" << endl;
    pathCount++;
    cout << endl;
}
```

Търсене с връщане назад

```
// Рекурсивна процедура, намираща всички пътища от
// клетка (x,y) до клетка (7,7) crrway е текущо изминатият път, а l е дължината му.
void way(int x, int y, int *crrWay, int l) {
    crrWay[2*l] = x;
    crrWay[2*l+1] = y;

    // Клетката е извън лабиринта.
    if (x < 0 || y < 0 || x > 7 || y > 7) {
        return;
    }

    // Намерен е път.
    if(x == 7 && y == 7) {
        printWay(crrWay,l+1);
        return;
    }

    // Клетката е непроходима.
    if (!labyrinth[x][y]) {
        return;
    }

    // Клетката е проходима. С цел предотвратяване на зацикляне, тази клетка се маркира като непроходима.
    labyrinth[x][y] = 0;

    // Търсене на всички пътища от четирите съседни на (x, y) клетки до клетка (7, 7)
    way(x+1, y, crrWay, l+1);
    way(x, y+1, crrWay, l+1);
    way(x-1, y, crrWay, l+1);
    way(x, y-1, crrWay, l+1);

    // "връщане назад"
    labyrinth[x][y] = 1;
}
```