**УНИВЕРЗИТЕТ “ГОЦЕ ДЕЛЧЕВ” – ШТИП  
ФАКУЛТЕТ ЗА ИНФОРМАТИКА**

**A circular logo with a building in the middle

Description automatically generated**

**СЕМИНАРСКА РАБОТА ПО НАПРЕДНИ АЛГОРИТМИ**

**ТЕМА “Backtracking – пребарување на сите можни**

**решенија на даден проблем”**

**Ментор: Изработил:**

**Проф.Д-р Наташа Стојковиќ Мартин Ристов 102620**

**Содржина**

[**Вовед** 3](#_Toc157798029)

[**Backtracking – Пристап на брутална сила** 4](#_Toc157798030)

[**Branch and Bound – Гранење и ограничување** 5](#_Toc157798031)

[**N – Queens** 6](#_Toc157798032)

[**Bounding function – Гранична функција** 10](#_Toc157798033)

[**N – Knights** 13](#_Toc157798034)

[**Заклучок** 15](#_Toc157798035)

# **Вовед**

Backtracking е моќен алгоритамско-програмски метод кој се користи за решавање на проблеми со пребарување на сите можни решенија. Оваа стратегија на извршување претставува варијација на методот на груба сила, но со посебен фокус на "повратно следење" или "повратен преглед".

Принципот на работа на backtracking е едноставен, но моќен. Алгоритмот почнува со избор на едно можно решение или стартна точка и потоа следи секвенца на одлуки за да го провери секое можно продолжение или избор. Ако се стигне до точка каде што не постои можно продолжение или се утврди дека тековното продолжение не доведува до целното решение, алгоритмот се враќа наназад (backtracks) и пробува со други алтернативи.

Овој метод е често користен во ситуации каде што не може да се примени прост пресметков метод, бидејќи просто генерирање на сите можни комбинации или пребарување на сите можни патеки може да биде прекуточно. Backtracking дозволува ефикасно пребарување на решенија, избегнување на непотребни решенија и претставува ефикасна стратегија за решавање на проблеми што се моделираат со дрвја на одлука, игри, генетски алгоритми и многу други области.

# **Backtracking – Пристап на брутална сила**

Backtracking, познат и како пристап на брутална сила, претставува алгоритамско решение за проблеми каде што е потребно да се истражат сите можни комбинации или варијации за да се најде решение. Овој метод е особено корисен во ситуации кога не постои ефикасен начин за директно пресметување на решението, туку треба систематски да се испробаат сите можни опции.

Во backtracking, алгоритмот започнува со првична одлука или избор и потоа секвентно се пробуваат сите можни патеки, следејќи ја функцијата на одлука. Ако се стигне до точка каде што се утврдува дека тековната линија на извршување не доведува до решение, алгоритмот "се враќа назад" (backtrack) и пробува со други можности.

Овој пристап е познат по својата снага во решавањето на проблеми како генерирање на комбинаторни низи, решавање на задачи во игри, пребарување на патеки во графови и многу други сценарија. Иако може да биде временски застапен, backtracking често дава точни и оптимални решенија во ситуации каде што други методи не се погодни.

Пример со DFS(depth-first-search)

Проблемот кај Backtracking е што содржи некое ограничување и потоа ги додаваме сите решенија кој го задоволуваат тој услов!   
Поставуваме едно ограничување :  
- девојката не може да седи во средина помеѓу двете момчиња.

3 студенти : М1, М2, Д1 -> n=3 3! = 3\*2\*1 = 6 -> различни решенија

Варијации:

A screenshot of a cell phone

Description automatically generated

A white circles with green and red x and red ticks

Description automatically generated

Сега откако ќе направиме проверка , сите точни решенија се : 1, 3, 5, 6 .

# **Branch and Bound – Гранење и ограничување**

Во методот на "Гранење и ограничување", просторот на можни решенија се разгранува и се истражува во длабочина. Секоја гранка или стабло во дрвото на просторот на состојби претставува еден можен пат кон решение. За да се подобри ефикасноста, алгоритмот го користи концептот на "ограничување" за да се изостри претрагата, така што се исфрлаат или изоставаат делови од просторот на решенија кои не би довеле до оптимално решение.

Во "Гранење и ограничување", се применува и стратегијата на backtracking за пребарување на просторот на состојби, но со фокус на ограничување на обемот на пребарување и со намерата да се најде оптималното решение.

Ќе го разгледаме истиот пример од претходно погоре.

A black background with white circles

Description automatically generatedВо Branch-and-Bound се генерираат мудро самите нивоа кој што никогаш не лажат.

# **N – Queens**

Проблемот со N-Queens(кралици) е класичен проблем во компјутерските науки и теоријата на проблемите. Целта на проблемот е да се постават N шаховски кралици на N×N шаховска табла, така што ниту една кралица не може да нападне друга кралица. Во конкретен случај, тоа значи дека нема две кралици на ист ред, колона или било која од дијагоналите на таблата.

За решавање на проблемот со N-Queens, често се користи backtracking. Основната идеја е да се постават кралиците една по една на таблата и да се проверуваат условите за безбедност. Ако сите кралици се поставени без прекршување на правилата, се добива валидно решение.

Еден можен начин за решавање со backtracking е:

1. **Постави кралица:**

Започни со поставување на кралица на првиот ред и првата колона.

Продолжи со следната колона, проверувајќи дали е безбедно да се постави кралицата.

1. **Продолжи рекурзивно:**

Ако е безбедно, продолжи рекурзивно за следниот ред.

Ако не е безбедно, врати се наназад (backtrack) и обиди се со различна колона за ист ред.

1. **Заврши:**

Кога ќе се постават сите N кралици и секоја од нив е на безбедна позиција, тоа е валидно решение.

Доколку не може да се најде безбедна позиција за некоја кралица, алгоритмот се враќа наназад (backtrack) за да проба различни опции.

Оваа техника на backtracking ги истражува сите можни комбинации на поставување кралици, но со ефикасни проверки за безбедност. Проблемот со N-Queens е добар пример за употребата на backtracking во решавањето на комбинаторни проблеми.

Пример: 4 кралици -> Q1, Q2, Q3, Q4

Матрици:

1)

|  |  |  |  |
| --- | --- | --- | --- |
| Q1 |  |  |  |
|  | Q2 |  |  |
|  |  | Q3 |  |
|  |  |  | Q4 |

2)

|  |  |  |  |
| --- | --- | --- | --- |
| Q1 |  |  |  |
|  | Q2 |  |  |
|  |  |  | Q3 |
|  |  | Q4 |  |

3)

|  |  |  |  |
| --- | --- | --- | --- |
| Q1 |  |  |  |
|  |  | Q2 |  |
|  | Q3 |  |  |
|  |  |  | Q4 |

4)

|  |  |  |  |
| --- | --- | --- | --- |
| Q1 |  |  |  |
|  |  | Q2 |  |
|  |  |  | Q3 |
|  | Q4 |  |  |

5)

|  |  |  |  |
| --- | --- | --- | --- |
| Q1 |  |  |  |
|  |  |  | Q2 |
|  | Q3 |  |  |
|  |  | Q4 |  |

6)

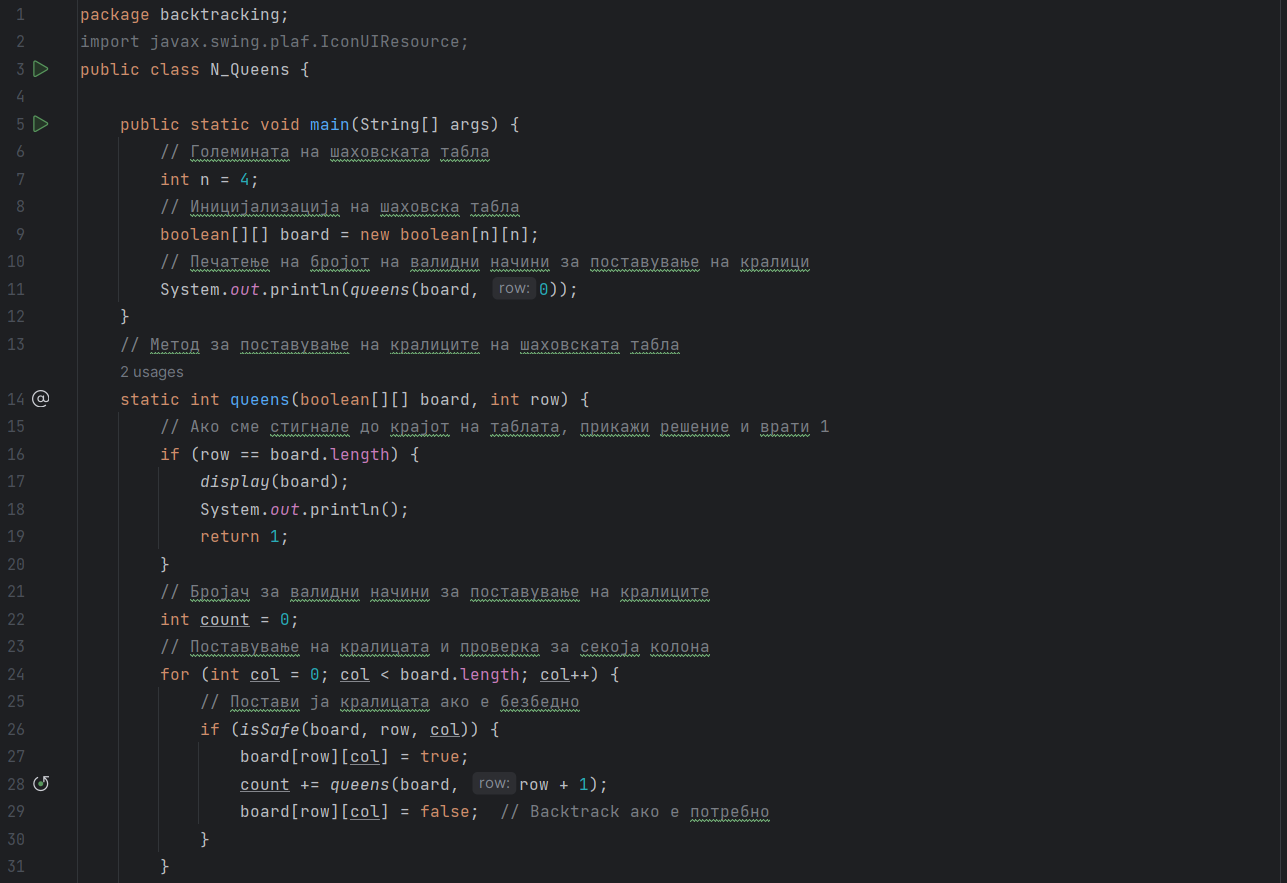
|  |  |  |  |
| --- | --- | --- | --- |
| Q1 |  |  |  |
|  |  |  | Q2 |
|  |  | Q3 |  |
|  | Q4 |  |  |

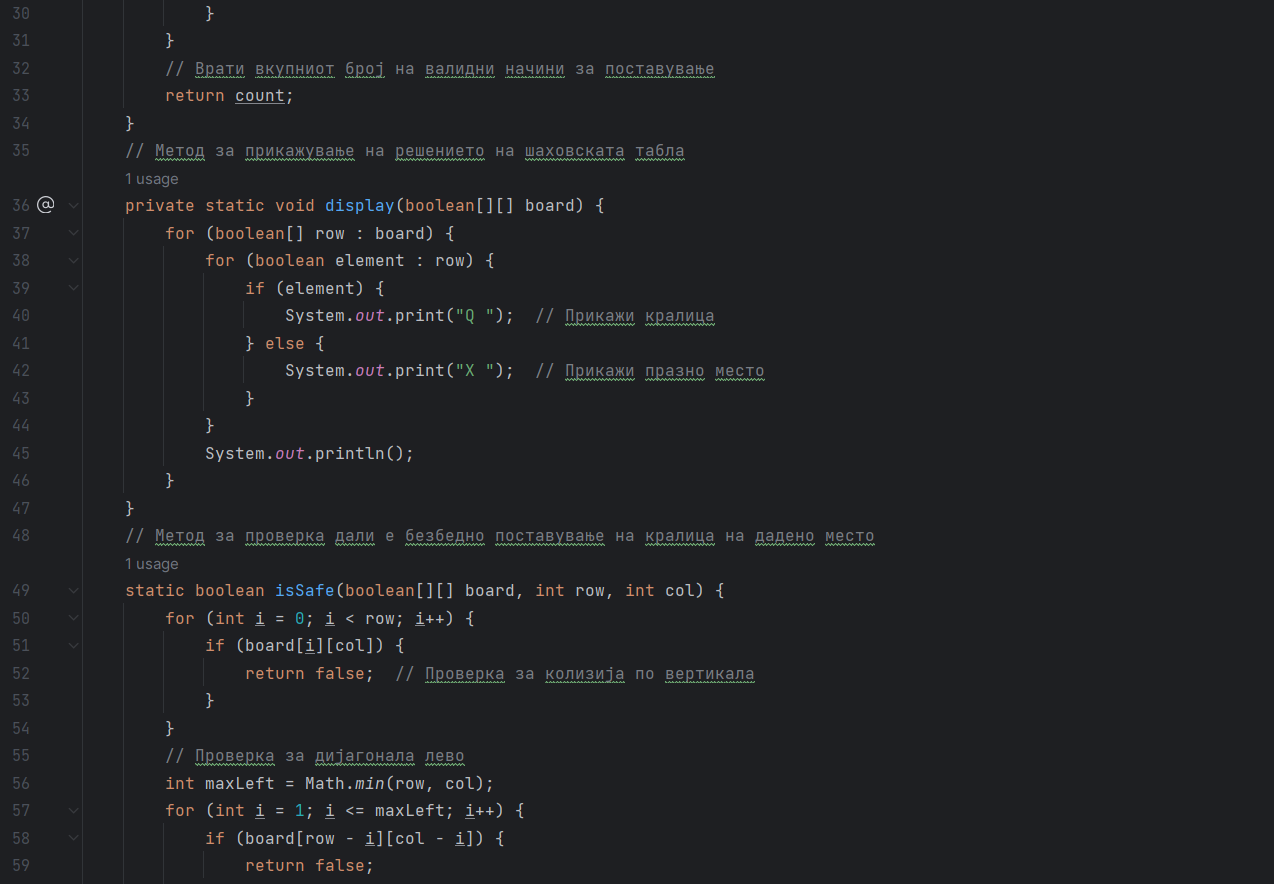
A black background with white dots

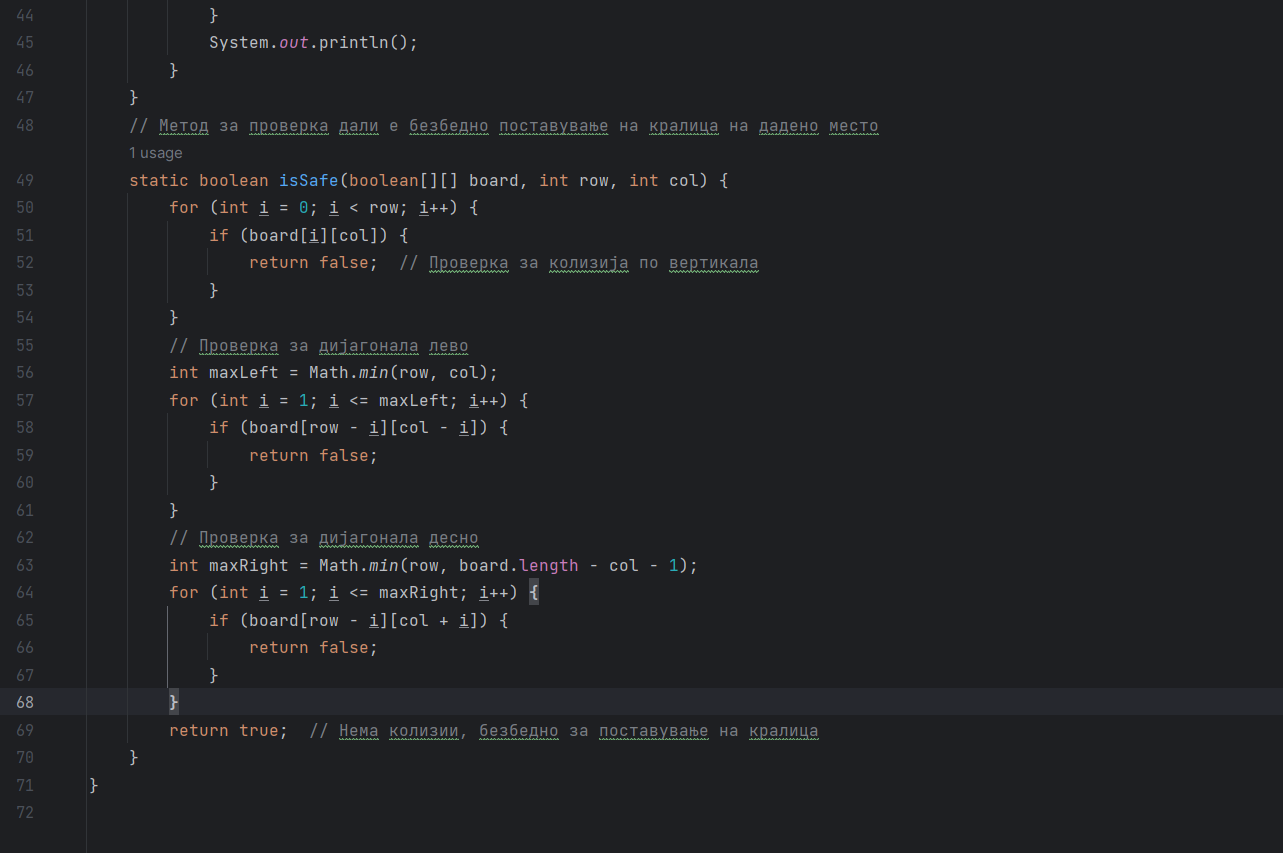
Description automatically generated

1 + 4 + (4\*3) + (4\*3\*2) + (4\*3\*2\*1) = 65 вкупно јазли

Примерот решен со програмски код во јава :







# **Bounding function – Гранична функција**

Bounding function или функција за ограничување во контекстот на N-Queens проблемот се користи како стратегија за ограничување на пребарувањето на решенија и побрзо пристигнување до оптималното решение. Овој пристап е често вграден во методи како Branch and Bound.

Во контекстот на N-Queens, bounding function би била функција која предвидува колку добри или лоши можат да бидат различните стапки во изградбата на решението. Основната цел е да се искористи информацијата за да се пресмета горна граница на решенијата и на тој начин да се исфрлаат делови од дрвото на пребарување кои не би довеле до оптимално решение.

Пример за bounding function во N-Queens проблемот може да биде бројот на веќе поставени кралици кои не се во конфликт. Доколку алгоритмот постави кралица на безбедна позиција, bounding function може да пресмета колку кралици веќе се поставени и да додели тежина на конкретното решение. Ако bounding function открие дека и сите преостанати кралици можат да бидат безбедно поставени, тогаш алгоритмот може да продолжи со следната стапка. Ако, пак, bounding function открие дека дури и со безбедната поставка на преостанатите кралици нема можност за оптимално решение, тогаш може да се прекине пребарувањето во тој дел од дрвото на пребарување.

Bounding function игра важна улога во значително намалување на бројот на потребни проверки и обезбедува поефикасно пребарување на оптималното решение во проблемот со N-Queens. Обично кај оваа функција тие можат да бидат во комфликт ако се поклопат во редица, колона и дијагонално.

Пример:

A group of white circles on a black background

Description automatically generated

Матрици : Q1, Q2, Q3, Q4

|  |  |  |  |
| --- | --- | --- | --- |
| Q1 |  |  |  |
|  | Q2 | Q2 | Q2 |
|  | Q3 |  | Q3 |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Q1 |  |  |  |
|  |  |  | Q2 |
|  | Q3 | Q3 |  |
|  |  | Q4 |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Q1 |  |  |
| Q2 |  | Q2 | Q2 |
| Q3 |  |  |  |
|  |  | Q4 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 4 | 1 | 3 |

Точно решение

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Q1 |  |
| Q2 |  |  |  |
|  |  |  | Q3 |
|  | Q4 |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 1 | 4 | 2 |

Друго решение би добиле ако направиме пресликување(mirroring) на претходното решение.

# **N – Knights**

Проблемот со N-Коњи, слично како и со N-Кралици, е класичен проблем во теоријата на игри и компјутерските науки. Целта е да се постават N коњи на N×N шаховска табла, така што ниту еден коњ не може да нападне друг. Во овој случај, напад на коњ се смета кога два коња се наоѓаат на иста позиција на таблата или кога еден коњ може да се помести на позиција каде што може да нападне друг.

Решавањето на проблемот со N-Коњи може да биде изведено со backtracking метода. Пример за backtracking алгоритам за овој проблем би бил:

**Постави коњ:**

Започни со поставување на коњ на првиот ред и првата колона.

Продолжи со следната колона, проверувајќи дали е безбедно да се постави коњот.

**Продолжи рекурзивно:**

Ако е безбедно, продолжи рекурзивно за следниот ред и колона.

Ако не е безбедно, врати се наназад (backtrack) и обиди се со различна колона за ист ред.

**Заврши:**

Кога ќе се постават сите N коњи и секој од нив е на безбедна позиција, тоа е валидно решение.

Доколку не може да се најде безбедна позиција за некој коњ, алгоритмот се враќа наназад (backtrack) за да проба различни опции.

Bounding function во контекстот на N-Коњи може да биде слична на bounding function за N-Кралици, користејќи информации за бројот на коњи кои веќе се поставени и пресметувајќи горна граница за решенијата. Ова помага во исфрлање на делови од просторот на пребарување кои не би довеле до оптимално решение.

Проблемот решен со програмски код во јава:  
  
A screenshot of a computer program

Description automatically generated

A screenshot of a computer screen

Description automatically generated

# **Заклучок**

Backtracking претставува моќна алгоритамска техника во областа на компјутерските науки. Овој метод се користи за решавање на проблеми каде што потребно е да се истражат сите можни комбинации или состојби за да се најде оптимално решение. Основната идеја зад backtracking е рекурзивно пребарување на дрво на состојби, каде што секој јазол претставува можна одлука или стапка кон решение.

Примената на backtracking е широка и се наоѓа во различни области како што се комбинаторика, игри, графови, и оптимизација. Примерите на проблеми кои можат да се решат со backtracking вклучуваат проблеми со N-Кралици, N-Коњи, Sudoku, и други.

Основниот процес на backtracking вклучува првичен избор на решение, рекурзивно истражување на секоја можна одлука, проверка на услови за валидност, и ако се достигне крај на еден пат, враќање назад (backtrack) и пробување со други можности. Ова овозможува ефикасно пребарување на големи простори на решенија и наоѓање на точни или оптимални резултати.

Bounding functions, како дополнителен елемент, се користат за да се ограничи обемот на пребарувањето, што резултира во побрзо наоѓање на решенија. Backtracking, со својата флексибилност и ефикасност, останува одличен алат за решавање на проблеми во различни домени.