Künstliche Intelligenz

Vorlesung 2: Suchverfahren
Uninformierte Suche

Starke und schwache KI-Hypothese

Schwache KI-Hypothese

Maschinen (Computer, Roboter,. . .) können agieren, alsob sie intelligent wären

Starke KI-Hypothese

Maschinen (Computer, Roboter,. . .) können wirklich denken und simulieren nicht nur das Denken.

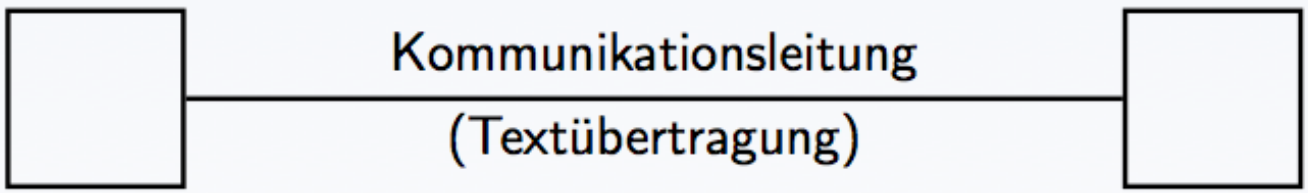
In der KI-Forschung

- Schwache KI-Hypothese wird als gegeben hingenommen
- Starke KI-Hypothese: Pragmatische Sichtweise: irrelevant, hauptsache das System funktioniert

Turing Test

Wiederholung

Test zum Nachweis der starken KI-Hypothese, vorgeschlagen von Alan Turing:



menschlicher Fragesteller

Gegenstelle (Mensch oder Computer)

- Mensch stellt schriftliche Fragen an Computer / Mensch
- Begrenzte Zeit
- Test ist bestanden, wenn Fragesteller nicht unterscheiden kann, ob Gegenstelle Mensch oder Computer ist

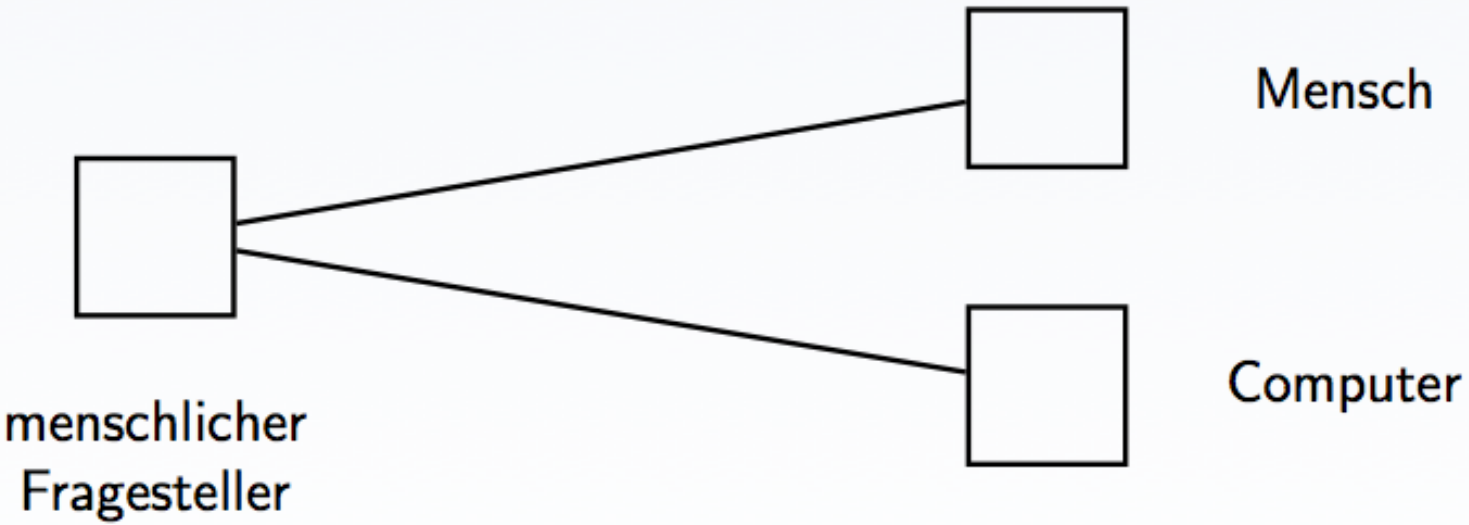
Turing Test

Wiederholung

- Kritik: Nicht objektiv, da der Test von den F ahigkeiten des Fragestellers abhängt
- Auch z.B. Wissen des Fragestellers über Fähigkeiten eines Computers
- Abhilfe: Test mit mehreren Personen wiederholen
- Totaler Turing-Test Unterschied zum Turing-Test: Zusätzlich Videoübertragung und Objekterkennung

Loebner Preis

- Jährlich ausgetragener Wettbewerb (seit 1991)
- Chatbots-Test ähnlich zum Turingtest
- Gewinner: Chatbot, der die Jury am meisten überzeugt hat (Bronze-Medaille)



Loebner Preis

Einmalige Preise:

- Silbermedaille (25.000 USD): Erster Chatbot, der die Jury überzeugend täuscht
- Goldmedaille (100.000 USD): wie Silbermedaille aber zusätzlich mit audio-visueller Kommunikation

Weder Silber noch Gold wurden bisher vergeben.

Turing-Test: Pro / Contra

- Pro: Halbwegs einsichtiges Kriterium für *Intelligenz*
- Contra: System als riesige Datenbank mit vorgefertigten Antworten. Ist das System intelligent?

ELIZA

- Von J. Weizenbaum entwickeltes Programm, das als Softbot einen Psychotherapeuten simuliert
- Konnte manche Menschen täuschen

Techniken:

- Vorgefertigte Phrasen, falls das System nichts versteht
  Erzählen Sie mir aus Ihrer Jugend
- Mustererkennung: in der Eingabe wird nach Schlüsselwörtern gesucht: *Erzählen Sie mir mehr über xyz*

Chinesischer Raum

Gedankenexperiment von John Searle als Gegenargument zur starken KI-Hypothese:

- Jemand, der kein Chinesisch versteht, sitzt in einen Raum
- Im Raum:
  - Stapel mit Chinesischen Zetteln
  - Handbuch (in Muttersprache) mit Regeln wie aus Chinesichen Zetteln neue Chinesische Zettel erzeugt werden können
- Ein Chinesicher Zettel wird durch Schlitz reingereicht
- Person erzeugt neue Zettel auf Stapel und gibt einen Zettel nach außen

Chinesischer Raum

Fragen

- Versteht die Person Chinesisch?
- Versteht das Gesamtsystem etwas?

J. Searle:

Kein Teil des System versteht irgendetwas

Das Prothesenexperiment

Annahme:

Neuronen können künstlich nachgebaut werden (elektronische Neuronen)

Experiment:

Ersetze einzelne Neuronen durch elektronische Neuronen

Frage

Ab welcher Anzahl verwandelt sich das Prothesen-Gehirn in einen Computer, der nichts versteht?

Folgerungen:

- Entweder: Starke KI-Hypothese gilt und nichts ändert sich
- Oder: Es gibt etwas, das noch unbekannt ist (Geist,...)

KI-Paradigmen

Zwei wesentliche Paradigmen

Physikalisches Symbolsystem

- explizites Programmieren
- verwenden von Logiken, Schlussregeln, Inferenzverfahren
- Stärken: Ziehen von Schlüssen, strategische Spiele, Planen,. . .

Lernverfahren

- insbesondere durch künstliche neuronale Netze
- Stärken: Bildererkennung, Mustverarbeitung, verrauschte Daten, maschinelles Lernen, adaptive Systeme

Komplexes KI-System benötigt i.A. alle Paradigmen

Wissensrepräsentationssysteme

Wissensrepräsentationshypothese (Brian Smith)

Die Verarbeitung von Wissen läßt sich trennen in:

- Repräsentation von Wissen, wobei dieses Wissen eine Entsprechung in der realen Welt hat
- Inferenzmechanismus, der Schlüsse daraus zieht.

$\Rightarrow$ Basis für Programme, deren innere Struktur als Modellierung von Fakten, Wissen, Beziehungen und als Operationen, Simulationen verstanden werden kann.

Repräsentations- und Inferenz-Systeme

Komponenten:

1. Formale Sprache: Festlegung der gültigen syntaktischen Formen (Wissensbasis, Anfragen)
2. Semantik: Bedeutung der Sätze der formalen Sprache (i.A. modular aufgebaut)
3. Inferenz-Prozedur (operationale Semantik) Wie kann man Schlüsse ziehen?
   Diese Inferenzen müssen korrekt bzgl. der Semantik sein.

Implementierung:

- Parser für die formale Sprache
- Implementierung der Inferenzprozedur.

Einführung: Suchalgorithmen

- Der einfache Reflexagent kann nur die aktuelle Wahrnehmung auf eine Aktion übertragen.
- Der zielbasierte Agent kann Aktionsfolgen planen, um die Ziele des Agenten erreichen.
- Wir werden nun einen Typ zielorientierter Agenten, den problemlösenden Agent untersuchen.
  - Was ist ein Problem und was ist seine Lösung?
  - Wie finde ich eine Lösung?
  - Suchalgorithmen BFS, DFS, DLS, IDS, BiS

Bemerkung

Zuerst werden wir uninformierte Suchalgorithmen untersuchen, d.h. Algorithmen, die keine problemspezifischen Informationen ausnutzen (und eine atomare Repräsentation verwenden).

Suchverfahren

Suchverfahren

Beispiele

- Spiele: Suche nach dem besten Zug
- Logik: Suche nach einer Herleitung einer Aussage
- Agenten: Suche nach der optimalen nächsten Aktion
- Planen: Suche nach einer Folge von Aktionen eines intelligenten Agenten.
- Optimierung: Suche eines Maximums einer mehrstelligen Funktion auf den reellen Zahlen

WIE SUCHT MAN NACH EINER LÖSUNG

$\longrightarrow$ SUCHSTRATEGIE!!!

WIE SUCHT MAN NACH EINER LÖSUNG

$\longrightarrow$ SUCHSTRATEGIE!!!

Inferenz

Schach

- Verzweigungsfaktor $b = 30$, Tiefe $d = 50$
- $30^{50} = 7.2 \cdot 10^{73}$ Blattknoten
- Anzahlt Inferenzschritte = $\sum_{d=0}^{50} 30^d = \frac{1-30^{51}}{1-30} = 7.4 \cdot 10^{73}$

Beispiel: Schach - Inferenzkomplexität

- 10000 Computer
- je einer Milliarde Inferenzen pro Sekunde
- Parallelisierung ohne Verluste
- Rechenzeit:

$$\frac{7.4 \cdot 10^{73} \text{ Inferenzen}}{10000 \cdot 10^{9} \text{ Inferenzen/sec}} = 7.4 \cdot 10^{60} \text{ sec} = 2.3 \cdot 10^{53} \text{ Jahre},$$

- 10^{43} mal Alter des Universums.

SCHLUSSFOLGERUNG

Suchproblem

Definition

Ein Suchproblem wird definiert durch folgende Größen

- *Zustand: Beschreibung des Zustands der Welt in dem sich ein Suchagent befindet.*
- *Startzustand: der Initialzustand in dem der Agent gestartet wird.*
- *Zielzustand: erreicht der Agent einen Zielzustand, so terminiert er und gibt (falls gewünscht) eine Lösung aus.*
- *Aktionen: Alle erlaubten Aktionen des Agenten.*
- *Lösung: Der Pfad im Suchbaum vom Startzustand zum Zielzustand.*
- *Kostenfunktion: ordnet jeder Aktion einen Kostenwert zu. Wird benötigt, um kostenoptimale Lösungen zufinden.*
- *Zustandsraum: Menge aller Zustände.*

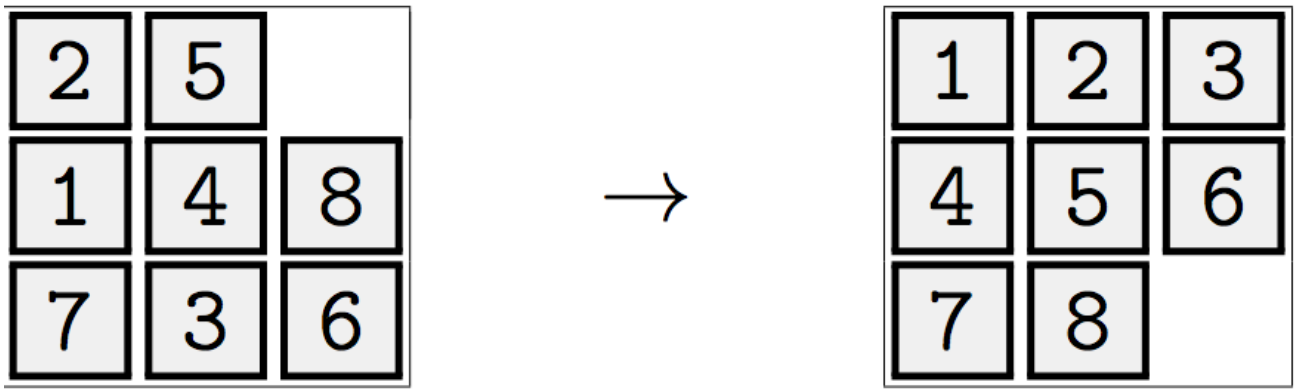
Beispiele

- Schach
  - Startzustand/Anfangssituation: Eine Stellung im Spiel
  - Aktionen/Nachfolgerfunktion: Mögliche Züge
  - Zielsituation/Zielzustand: Wenn man gewonnen hat
  - Gesucht: Zug der zum Gewinn führt
- Deduktionssystem
  - Startzustand/Anfangssituation: Zu beweisende Aussage A, Menge von Axiomen und bewiesenen Sätzen
  - Aktionen/Nachfolgerfunktion: Deduktionsregeln
  - Zielzustand/Zielsituation: Beweis
  - Gesucht: Beweis für A
- Planen
  - Startzustand/Anfangssituation: Formale Beschreibung des interessierenden Bereichs z.B. Fahrplan, Start- und Zielort,
  - Aktionen/Nachfolgerfunktion: Zugverbindungen usw.
  - Gesucht: Plan, der Reise ermöglicht

Uninformierte Suche

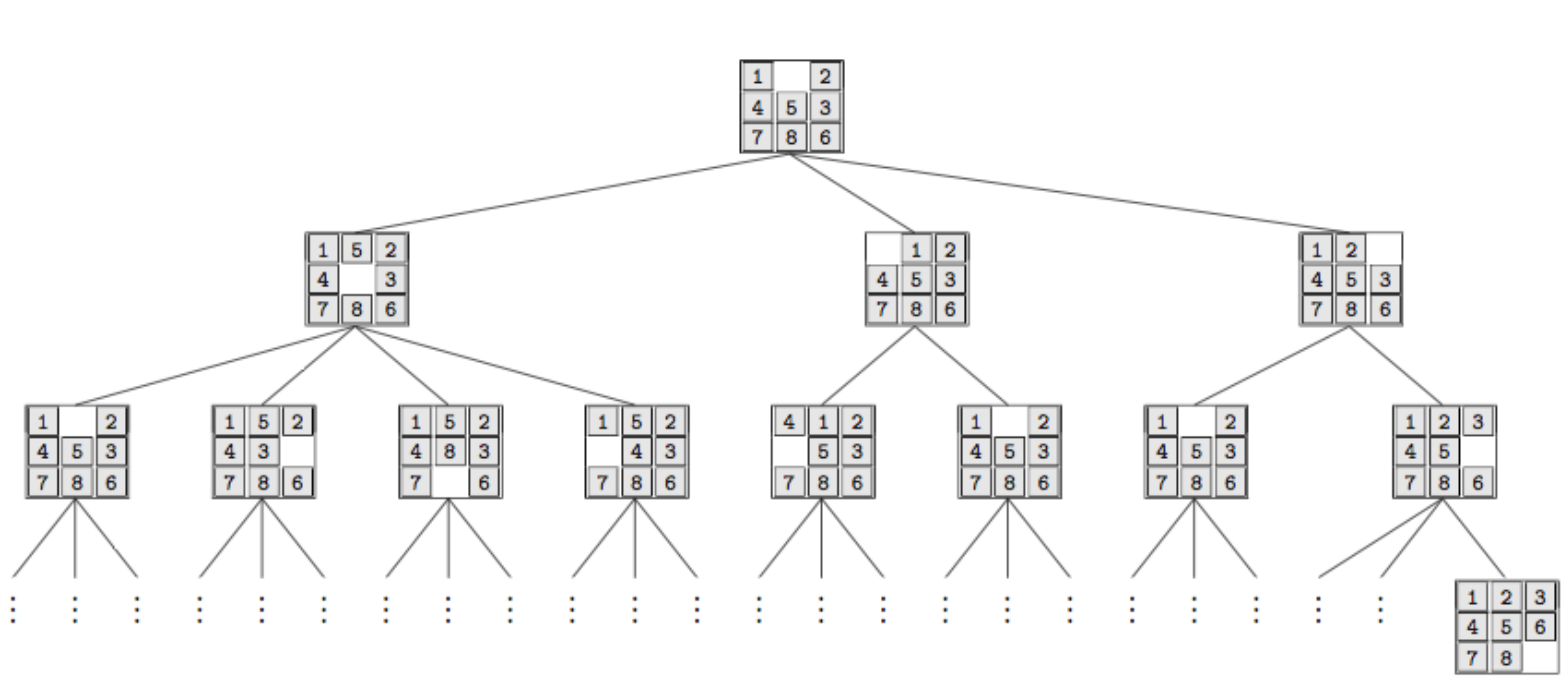
Wie funktioniert uninformierte Suche, d.h. das blinde Durchprobieren aller Möglichkeiten?

Beispiel: 8-Puzzle



Mögliche Start- und Zielzustände des 8-Puzzle.

8-Puzzle: Suchbaum



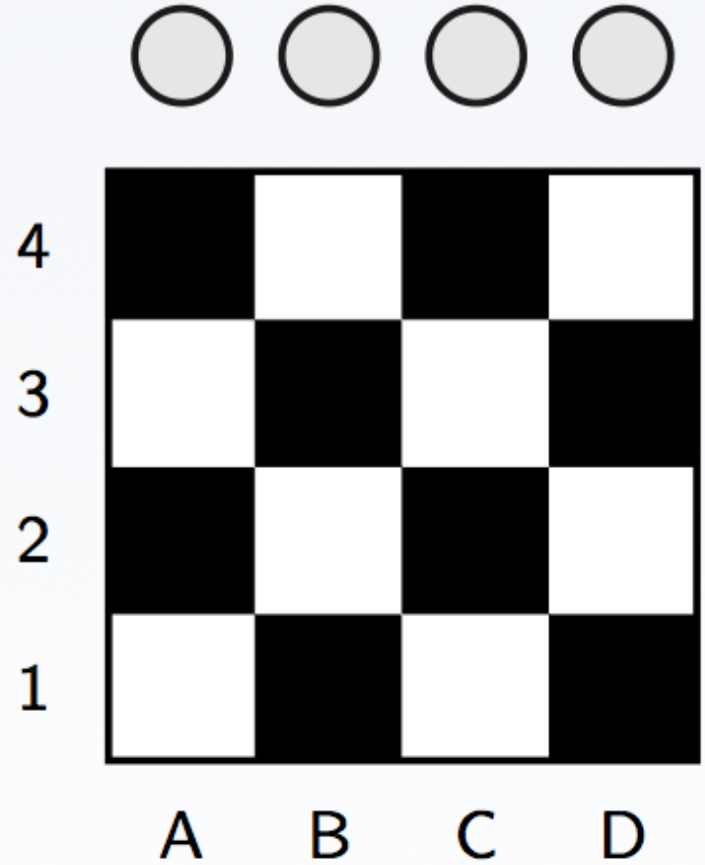
Wie entstehen Suchbäume?

Ausgehend von einem Zustand s führt eine Aktion a_1 in einen neuen Zustand s'. Es gilt also $s' = a_1(s)$. Eine andere Aktion kann in einen anderen Zustand s'' führen, das heißt $s'' = a_2(s)$. Durch die rekursive Anwendung aller möglichen Aktionen auf alle Zustände, beginnend mit dem Startzustand, entsteht der Suchbaum.

8-Puzzle

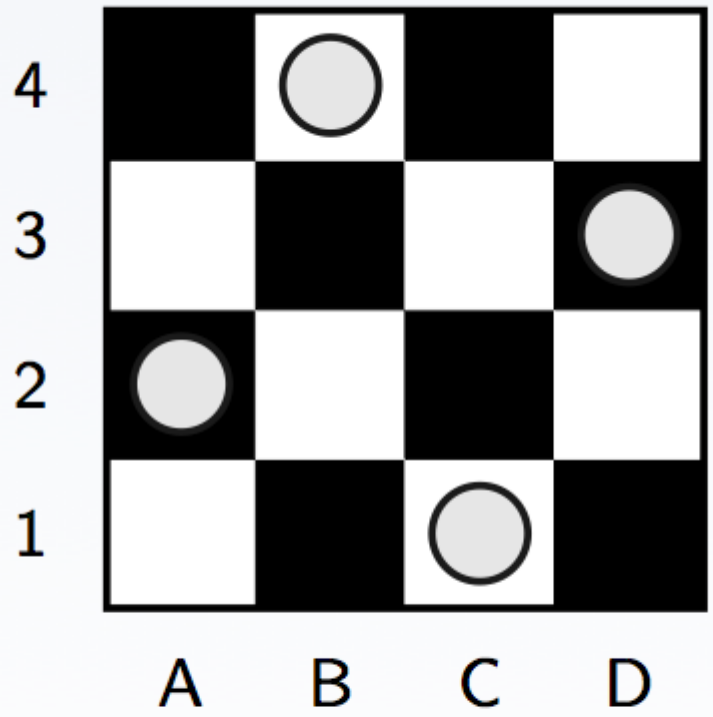
- Zustand: 3×3 Matrix S mit den Werten $1, 2, 3, 4, 5, 6, 7, 8$ (je einmal) und einem leeren Feld.
- Startzustand: Ein beliebiger Zustand.
- Zielzustand: Ein beliebiger Zustand.
- Aktionen: Bewegung des leeren Feldes S_{ij} nach links (falls $j \neq 1$), rechts (falls $j \neq 3$), oben (falls $i \neq 1$), unten (falls $i \neq 3$).
- Kostenfunktion: Die konstante Funktion 1, da alle Aktionen gleich aufwändig sind.
- Zustandsraum: Der Zustandsraum zerfällt in Bereiche, die gegenseitig nicht erreichbar sind. Daher gibt es nicht lösbare 8-Puzzle-Probleme.

Beispiel: n Damen



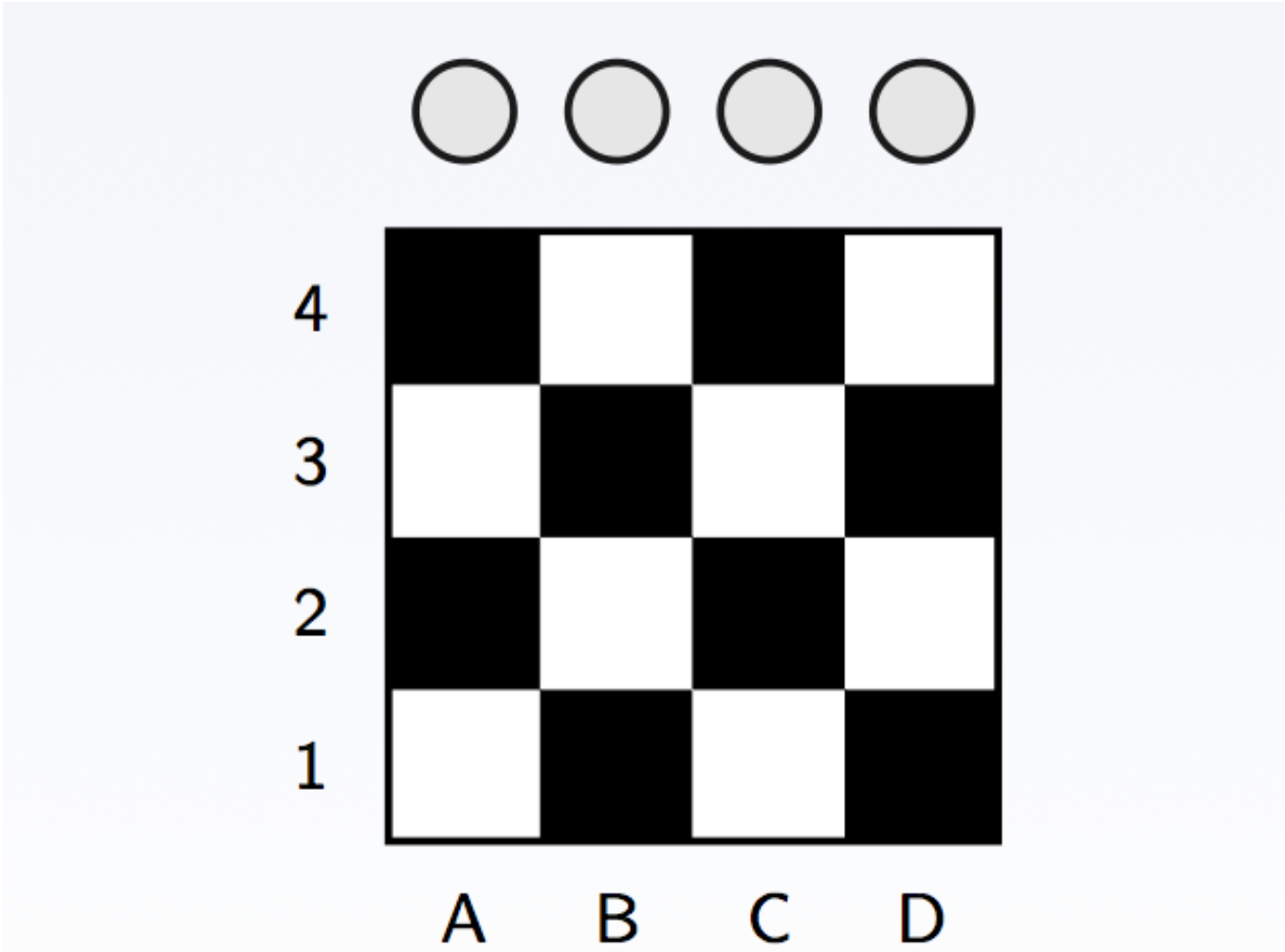
Platziere auf $n \times n$ Schachbrett n Damen,
so dass keine die andere bedroht

Beispiel: n Damen, mögliche Lösung



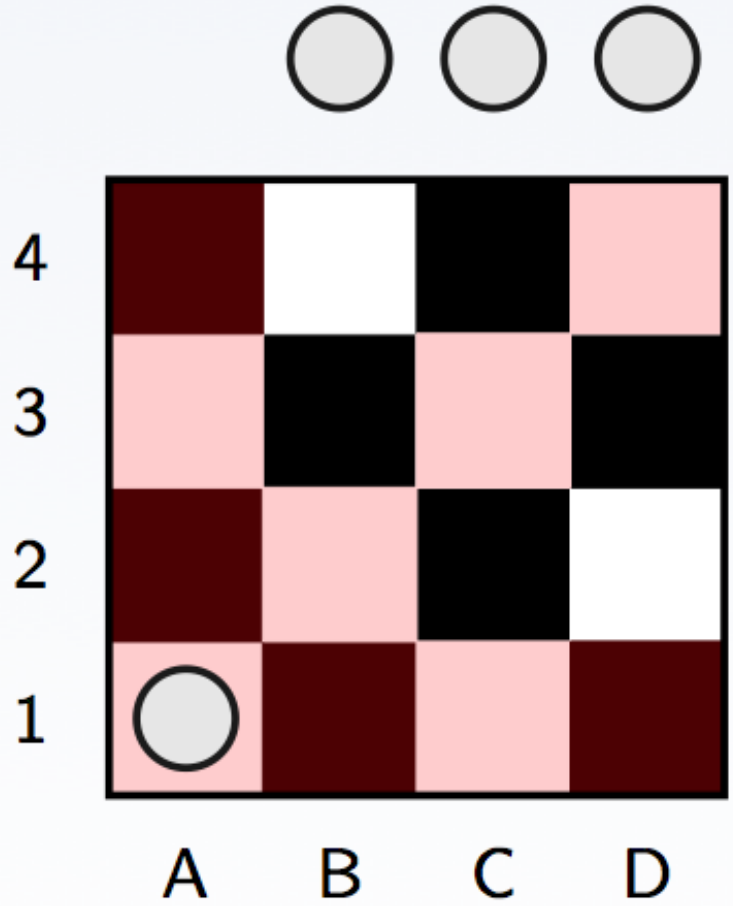
Platziere auf $n \times n$ Schachbrett n Damen,
so dass keine die andere bedroht

Beispiel: n Damen, mögliche Suche



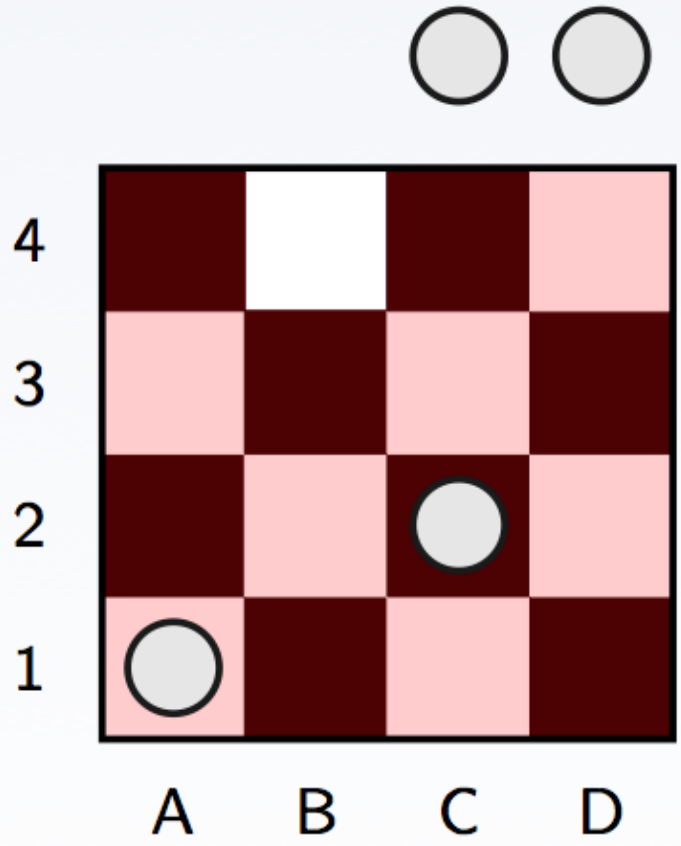
Platziere die Damen zeilenweise nacheinander, und backtracke bei Konflikt.

Beispiel: n Damen, mögliche Suche



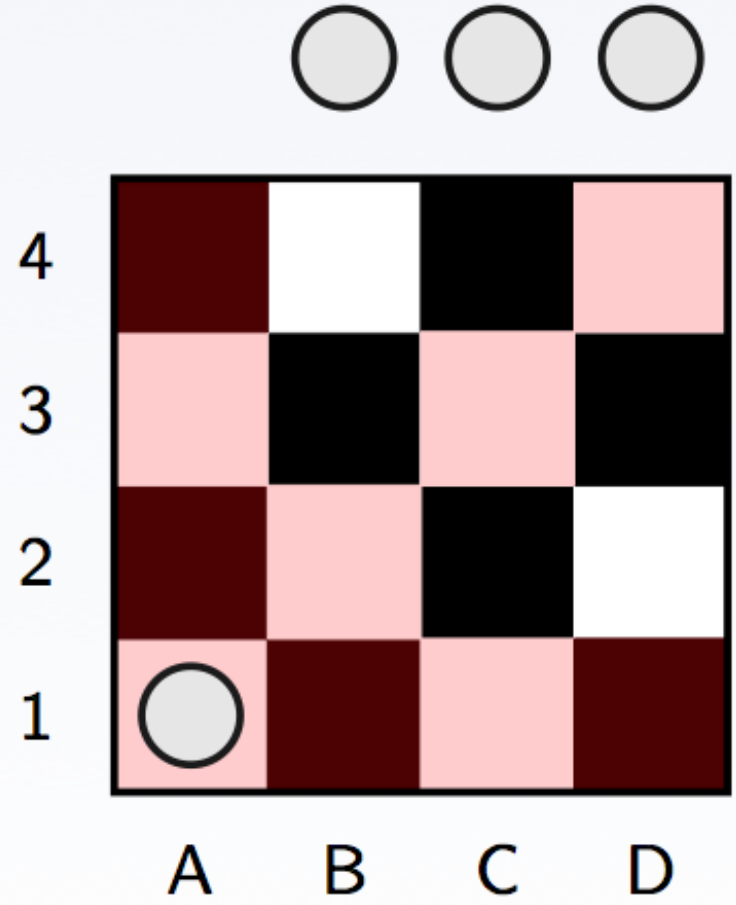
Platziere die Damen zeilenweise nacheinander,
und backtracke bei Konflikt.

Beispiel: n Damen, mögliche Suche



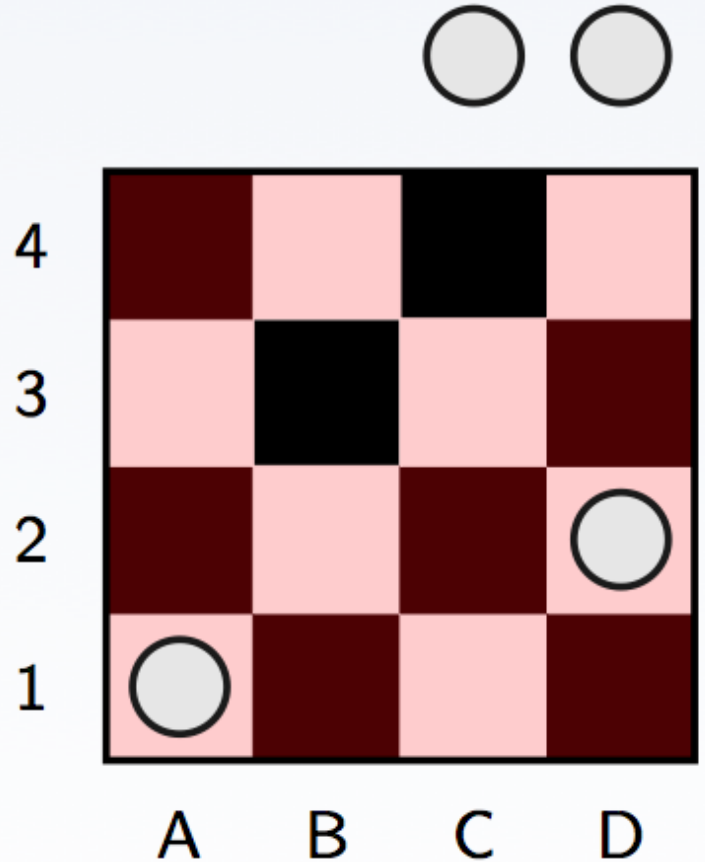
Platziere die Damen zeilenweise nacheinander,
und backtracke bei Konflikt.

Beispiel: n Damen, mögliche Suche



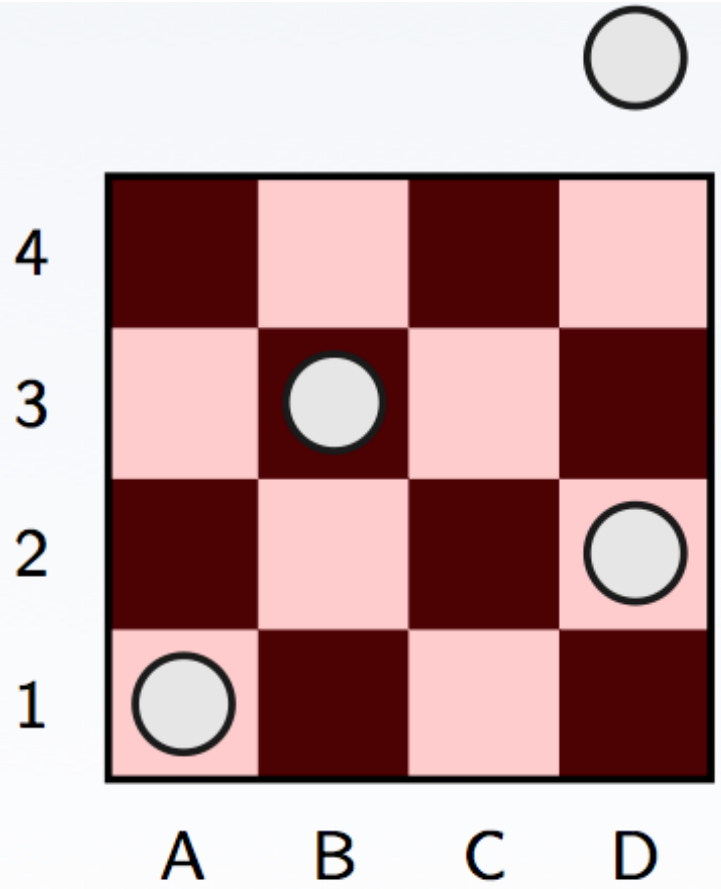
Platziere die Damen zeilenweise nacheinander, und backtracke bei Konflikt.

Beispiel: n Damen, mögliche Suche



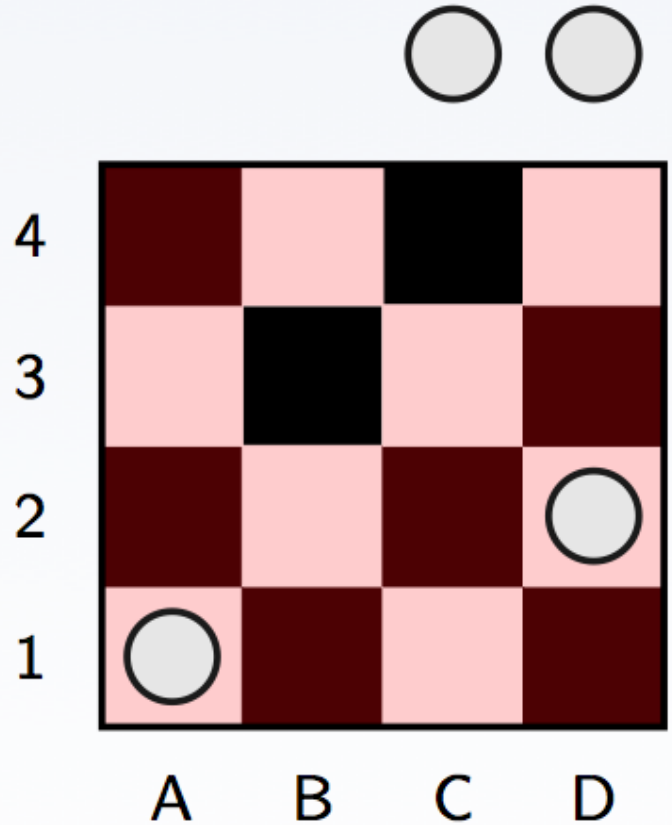
Platziere die Damen zeilenweise nacheinander,
und backtracke bei Konflikt.

Beispiel: n Damen, mögliche Suche



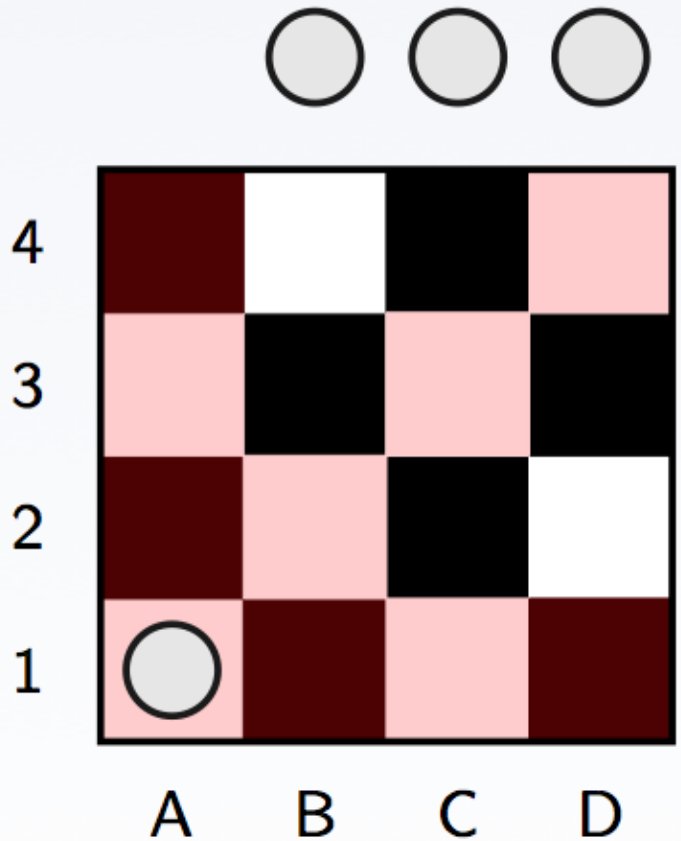
Platziere die Damen zeilenweise nacheinander,
und backtracke bei Konflikt.

Beispiel: n Damen, mögliche Suche



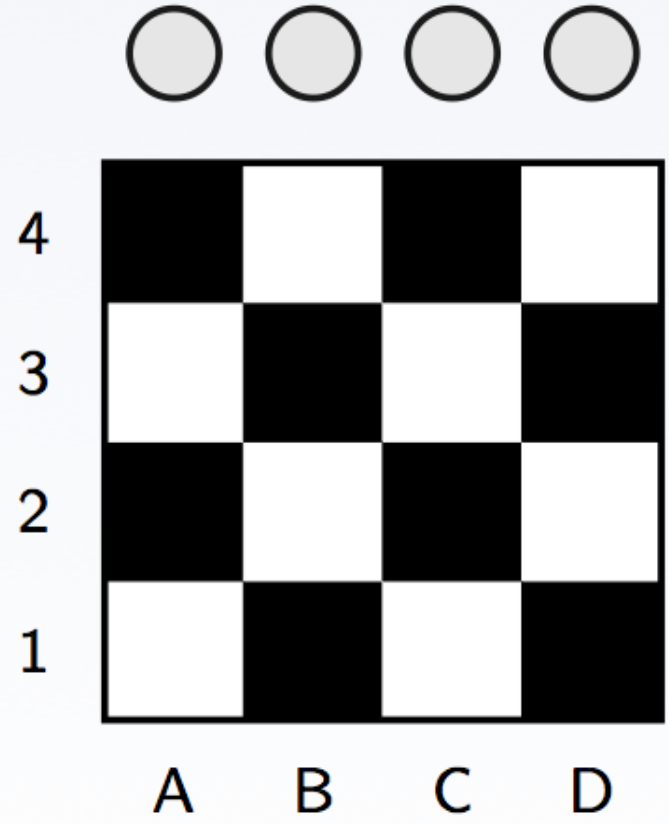
Platziere die Damen zeilenweise nacheinander,
und backtracke bei Konflikt.

Beispiel: n Damen, mögliche Suche



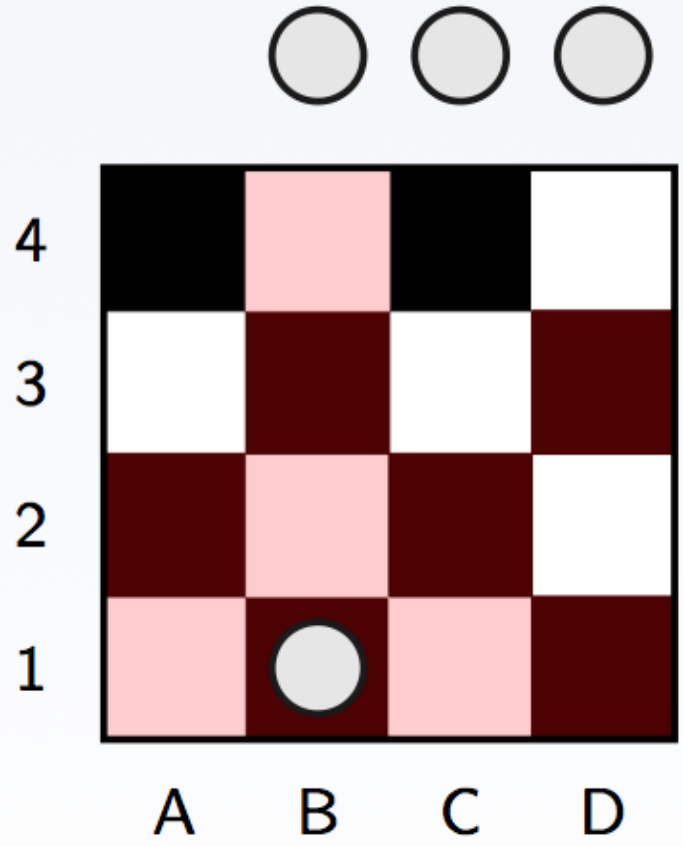
Platziere die Damen zeilenweise nacheinander,
und backtracke bei Konflikt.

Beispiel: n Damen, mögliche Suche



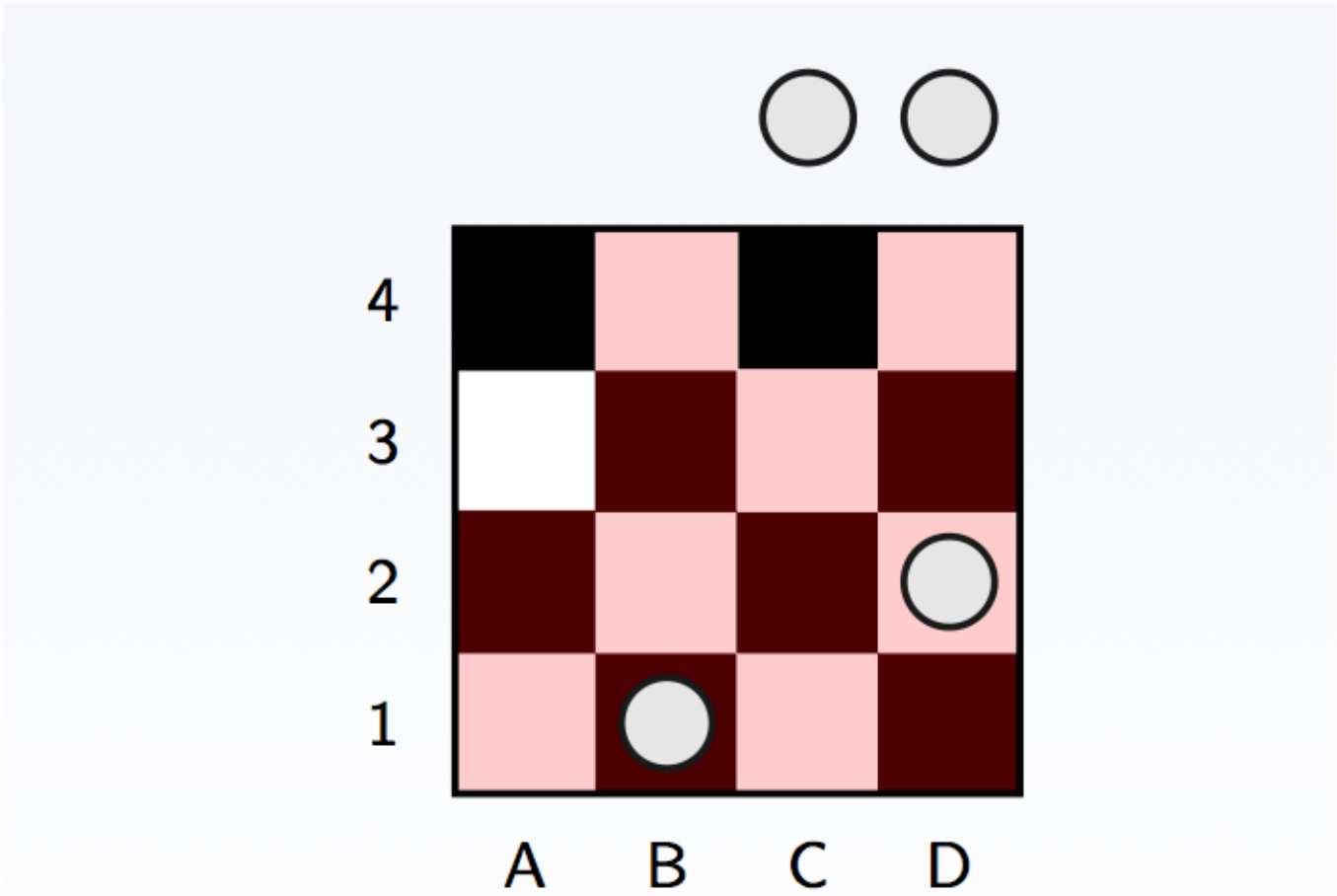
Platziere die Damen zeilenweise nacheinander,
und backtracke bei Konflikt.

Beispiel: n Damen, mögliche Suche



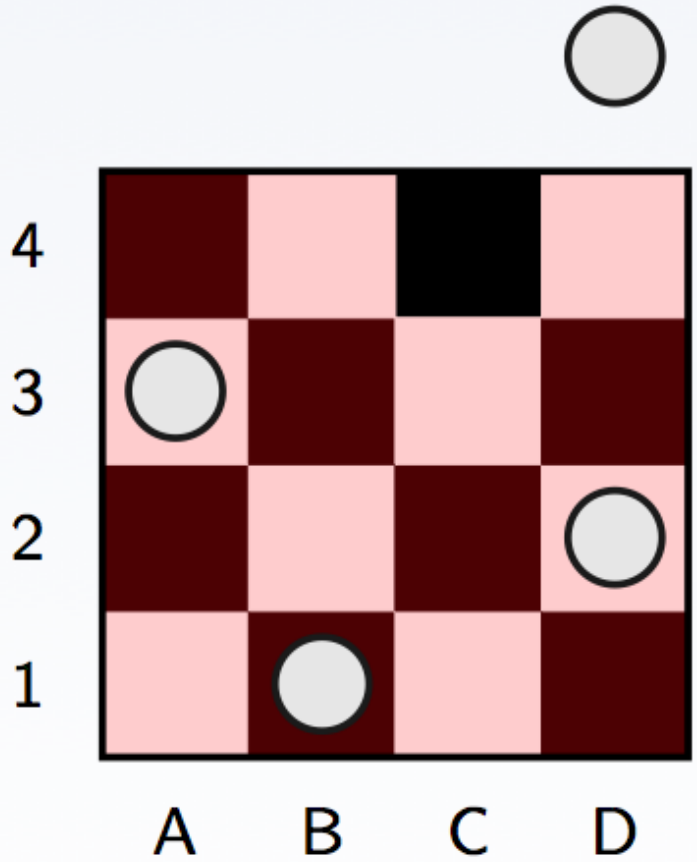
Platziere die Damen zeilenweise nacheinander,
und backtracke bei Konflikt.

Beispiel: n Damen, mögliche Suche



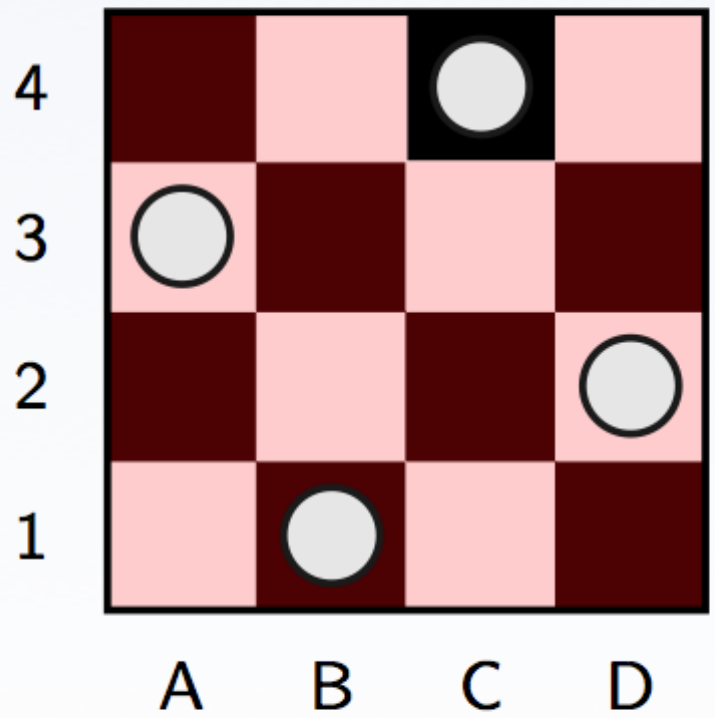
Platziere die Damen zeilenweise nacheinander,
und backtracke bei Konflikt.

Beispiel: n Damen, mögliche Suche



Platziere die Damen zeilenweise nacheinander,
und backtracke bei Konflikt.

Beispiel: n Damen, mögliche Suche



Platziere die Damen zeilenweise nacheinander, und backtracke bei Konflikt.

- Intelligente Agenten sollen ihr Leistungsmaß maximieren. Dies kann vereinfacht werden, wenn der Agent ein Ziel annehmen kann und darauf abzielt, es zu erfüllen.

Beispiel:Ein Modell und ein möglicher Lösungsweg

- ein Agent ist in der Stadt Arad
- Der Leistungsmaß ist von vielen Faktoren beeinflußt (Verbesserung des Nutzen, Sehenswürdigkeiten besichtigen, Nachtleben genießen, usw), die Entscheidungen schwer macht
- Nehmen wir nun an, der Agent hat am nächsten Tag eine nicht erstattungsfähige Fahrkarte, um von Bukarest aus zu fliegen
- Zielformulierung - nach Bukarest kommen - vereinfacht das Entscheidungsproblem des Agenten erheblich

Beispiel:Ein Modell und ein möglicher Lösungsweg

- Ein Ziel ist eine Reihe von Zuständen, in denen das Ziel erfüllt ist, aber wir müssen auch andere Zustnde und Handlungen akzeptieren, um uns von einem Zustand zum anderen fort zu bewegen
- Problemformulierung
  - Zustände entsprechen Großstädten
  - Aktionen könnten dem Fahren von einer Großstadt in einer andere entsprechen
- Wie finde ich einen Weg nach Bukarest findet, wenn drei Landstraßen von Arad ausgehen, eine nach Sibiu, eine nach Timisoara und eine nach Zerind?
- Wir nehmen an, dass der Agent eine Karte von Rumänien besitzt und kann mögliche Reisen erkunden (suchen), die beste auswählen und dann die Aktionen ausführen.

Problem solving

- Zielformulierung: Was wollen wir erreichen
- Problemformulierung: Wie?
- Aufgabe lösen: Wie finde ich die beste Sequenz von Handlungen, um mein Ziel zu erreichen?
- Lösung angeben: Falls ich die Handlungen weiß, was mache ich dann?

PROBLEM SOLVING

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
              state, some description of the current world state
              goal, a goal, initially null
              problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

Gut formulierte Probleme

- initialer Zustand: `in(Arad)`
- eine Beschreibung möglicher Handlungen und den entsprechenden, resultierenden Zustände
  - `SUCCESSOR-FN(in(Arad))=[go(Sibiu),in(Sibiu)]`
  - definiert implizit den Zustandsraum (Menge aller erreichbarer Zustände)
  - ein Pfad: ist eine Folge von Zustände verbunden durch Handlungen als Kanten.
- Zieltest: eine Funktion, die bestimmt, ob ein Zustand ein Zielzustand ist oder nicht `in(Bucharest)`
- Pfadkosten: eine Funktion, die numerische Werte jedem Pfad zuordnet (gibt die Effizienz an)

- Eine Lösung ist eine Reihenfolge von Handlungen, die aus einem initialen Zustand zu einem Zielzustand führen
- Eine optimale Lösung ist eine Lösung mit minimalem Kosten.

Probleme

- Toy examples
- Real problems

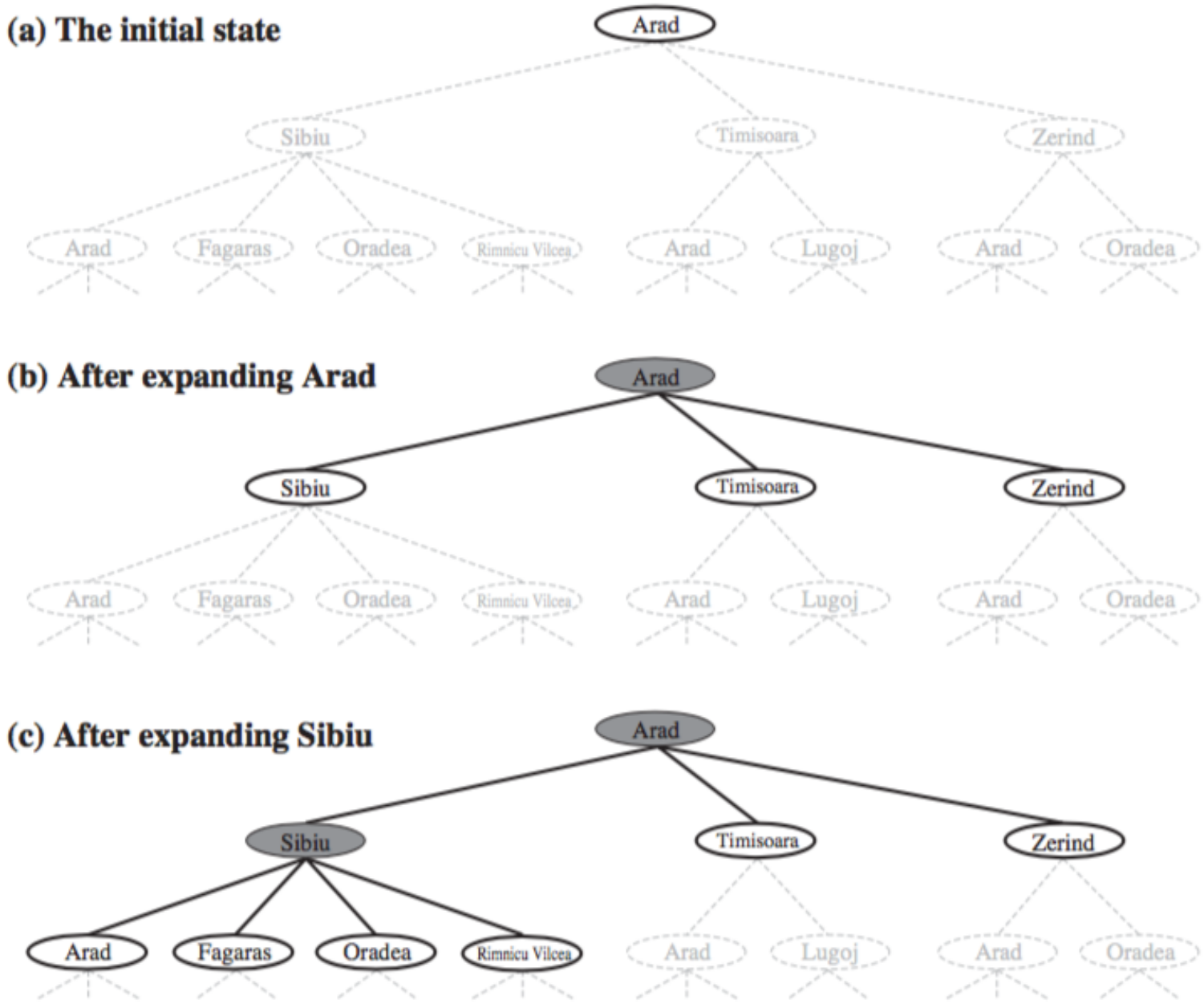
Real problems

- Route finding problems
- Touring problems: TSP, NP hart
- Product assembly problem: Roboter, Proteindesign aus einer Aminosäuresequenz

Wie wird eine Lösung gefunden?

- State space search!!!
- Starte in initialen Knoten (root)
- Teste ob der initiale Knoten ein Zielknoten ist
- Falls der aktuelle Zustand kein Zielzustand ist, expandiere den Zustand und erzeuge neue Zustände
- Selektiere einen neuen Zustand mit Hilfe einer Suchstrategie

PROBLEM SOLVING



PROBLEM SOLVING

function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

PROBLEM SOLVING

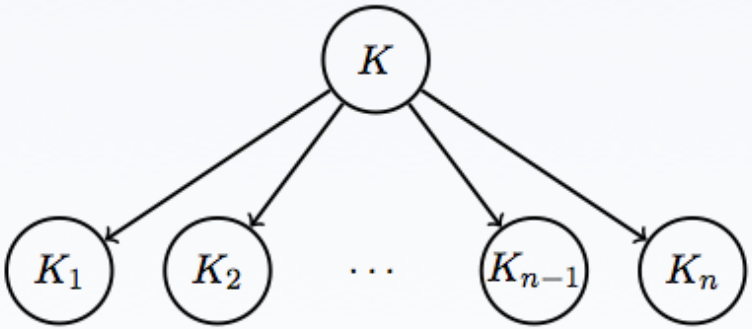
function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

Suchraum/Suchgraph

- Suche nach Lösung = Suche in einem gerichten Graphen
  $\rightarrow$ Der Graph ist nicht explizit gegeben!
- Suchgraph (Suchraum) gegeben durch:
  - Knoten: Situation, Zustände
  - Kanten: implizit als Nachfolger-Funktion N
  - Anfangssituation
  - Zieltest: Entscheidbarer Test, ob Knoten Zielknoten

Eigenschaften

- Verzweigungrate des Knotens K (branching factor):
  Anzahl der direkten Nachfolger von K, also $|N(K)|$.



$$N(K) = \{K_1, \ldots, K_n\}$$
$$|N(K)| = n$$

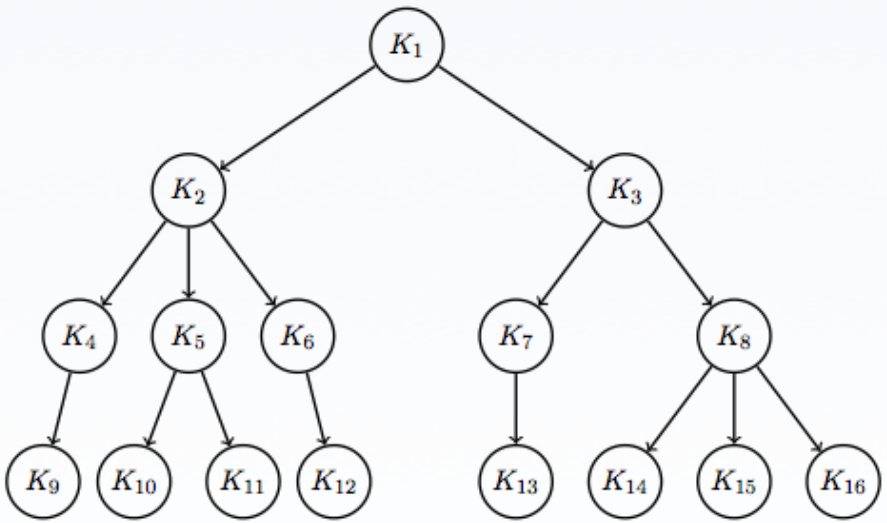
- Mittlere Verzweigunsrate des Suchraumes:
  Durchschnittliche Verzweigungrate aller Knoten.

Eigenschaften

- Größe des Suchraumes ab Knoten K in Tiefe d:
  Anzahl Knoten, die von K aus in d Schritten erreichbar sind

  D.h. $|\overline{N}^d(K)|$, wobei

$$\overline{N}^1(M) = \bigcup\{N(L) \mid L \in M\} \text{ und } \overline{N}^i(K) = \overline{N}(\overline{N}^{i-1}(K)).$$

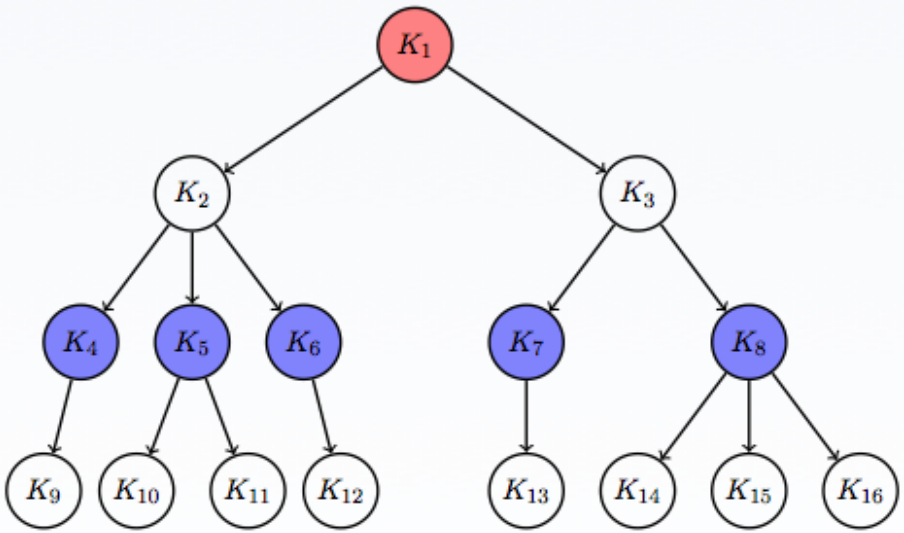


Eigenschaften

- Größe des Suchraumes ab Knoten K in Tiefe d:
  Anzahl Knoten, die von K aus in d Schritten erreichbar sind

  D.h. $|\overline{N}^d(K)|$, wobei

$$\overline{N}^1(M) = \bigcup\{N(L) \mid L \in M\} \text{ und } \overline{N}^i(K) = \overline{N}(\overline{N}^{i-1}(K)).$$



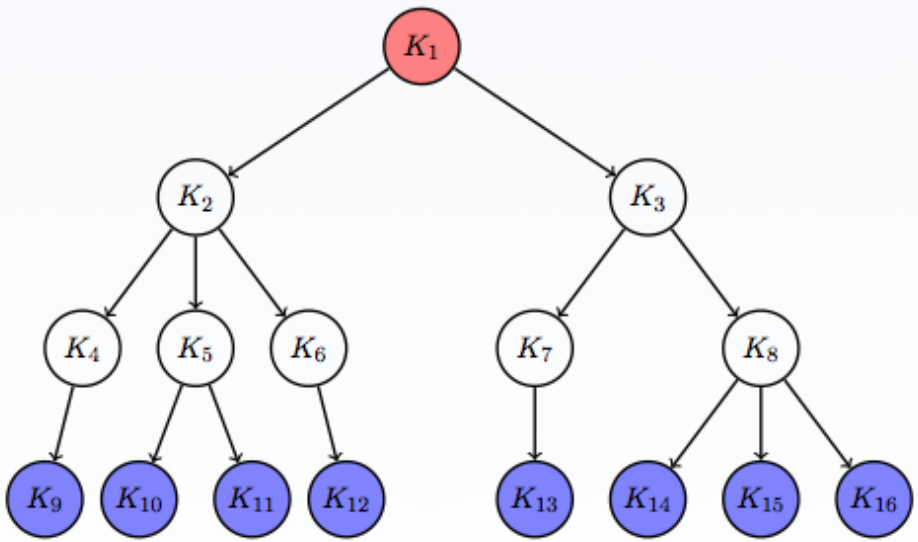
- Größe des Suchraums ab K_1 in Tiefe $2 = 5$

Eigenschaften

- Größe des Suchraumes ab Knoten K in Tiefe d:
  Anzahl Knoten, die von K aus in d Schritten erreichbar sind

  D.h. $|\overline{N}^d(K)|$, wobei

$$\overline{N}^1(M) = \bigcup\{N(L) \mid L \in M\} \text{ und } \overline{N}^i(K) = \overline{N}(\overline{N}^{i-1}(K)).$$



- Größe des Suchraums ab K_1 in Tiefe $2 = 5$
- Größe des Suchraums ab K_1 in Tiefe $3 = 8$

Eigenschaften

Eine Suchstragie ist vollständig, wenn sie einen Zielknoten nach endlichen vielen Schritten findet, falls dieser existiert.

Kombinatorische Explosion

- Üblicherweise: mittlere Verzweigungrate > 1
- $\Rightarrow$ Suche ist expontiell in der Tiefe des Suchraums
- das nennt man: kombinatorische Explosion
- Die meisten Suchprobleme sind NP-hart (NP-vollständig)

Knoten vs. Zustand

Zustandsraum ist vom Suchgraphen verschieden!

- Ein Knoten besteht aus
  - aktueller Zustand
  - eine Verbindung zum Elternknoten
  - eine Handlug, die zum aktuellen Zustand geführt hat
  - Pfadkosten vom root zum Knoten
  - Tiefe (Anzahl der schritte vom root)

Blind Search

- Blind Search = Nicht-informierte Suche
- Nur der Suchgraph ist (implizit) gegeben
- keine anderen Informationen (z.B. Heuristik)

Eingabe:

- Menge der initialen Knoten
- Menge der Zielknoten, bzw. eindeutige Festlegung der Eigenschaften der Zielknoten
- Nachfolgerfunktion N

Ausgabe:

Pfad zum Zielknoten (falls dieser existiert)

Nicht-informierte Suche, allgemein

Algorithmus Nicht-informierte Suche

Datenstrukturen:L = Menge von Knoten, markiert Weg dorthin
Eingabe: Setze L := Menge der initialen Knoten mit leerem Weg
Algorithmus:

1. Wenn L leer ist, dann breche ab.
2. Wähle einen beliebigen Knoten K aus L.
3. Wenn K ein Zielknoten ist, dann gebe aus: Zielknoten und Weg dorthin (d.h. Weg im Graphen dorthin)
4. Wenn K kein Zielknoten, dann nehme Menge $N(K)$ der direkten Nachfolger von K und verändere L folgendermaßen:
   $L := (L \cup N(K)) \setminus \{K\}$ (Wege entsprechend anpassen)
   Mache weiter mit Schritt 1.


Nicht-informierte Suche, allgemein

Algorithmus Nicht-informierte Suche

Datenstrukturen:L = Menge von Knoten, markiert Weg dorthin
Eingabe: Setze L := Menge der initialen Knoten mit leerem Weg
Algorithmus:

1. Wenn L leer ist, dann breche ab.
2. Wähle einen beliebigen Knoten K aus L.
3. Wenn K ein Zielknoten ist, dann gebe aus: Zielknoten und Weg dorthin (d.h. Weg im Graphen dorthin)
4. Wenn K kein Zielknoten, dann nehme Menge $N(K)$ der direkten Nachfolger von K und verändere L folgendermaßen:
   $L := (L \cup N(K)) \setminus \{K\}$ (Wege entsprechend anpassen)
   Mache weiter mit Schritt 1.

Keine Strategie!, nichtdeterministisch

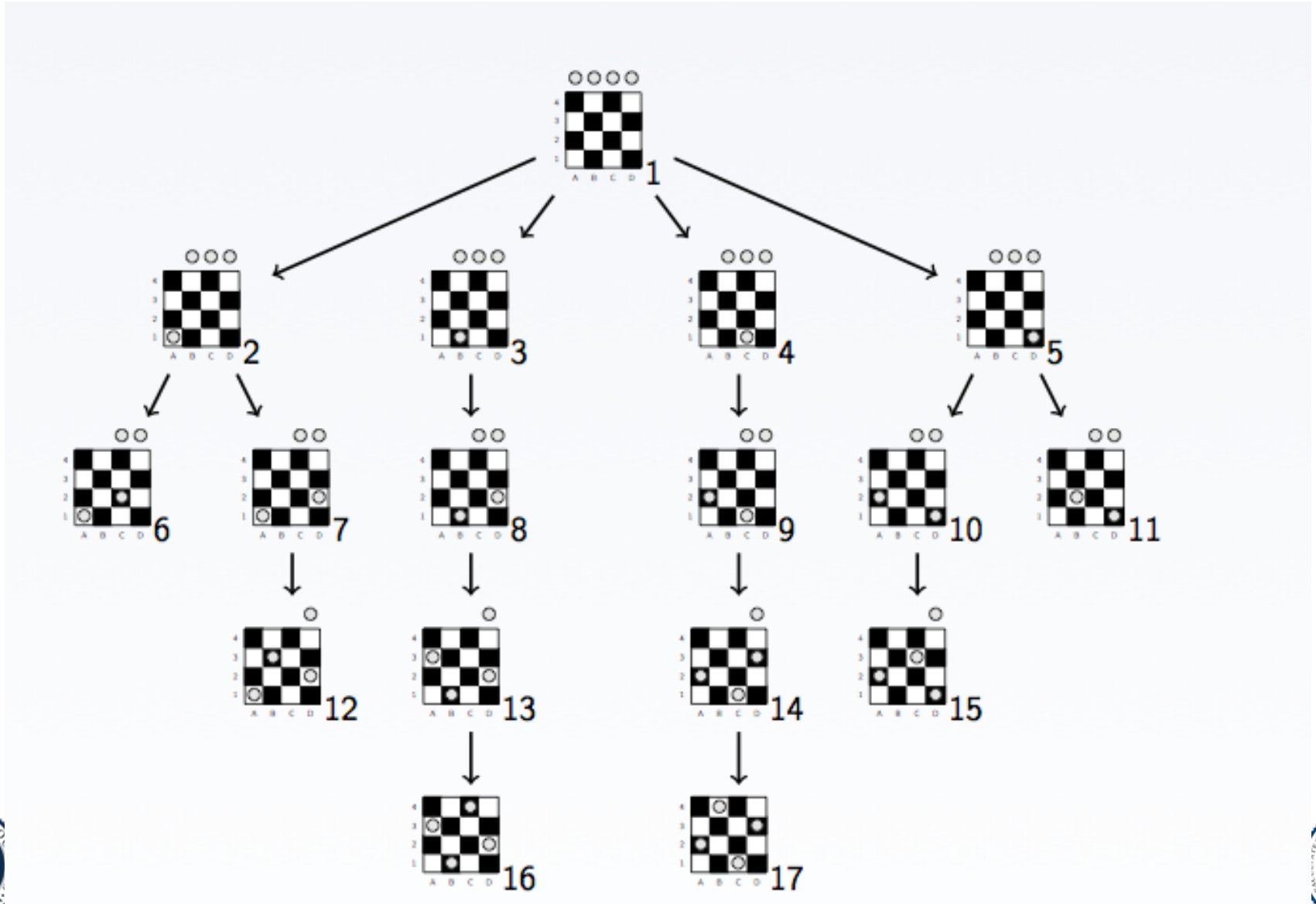

Tiefensuche

Algorithmus Tiefensuche

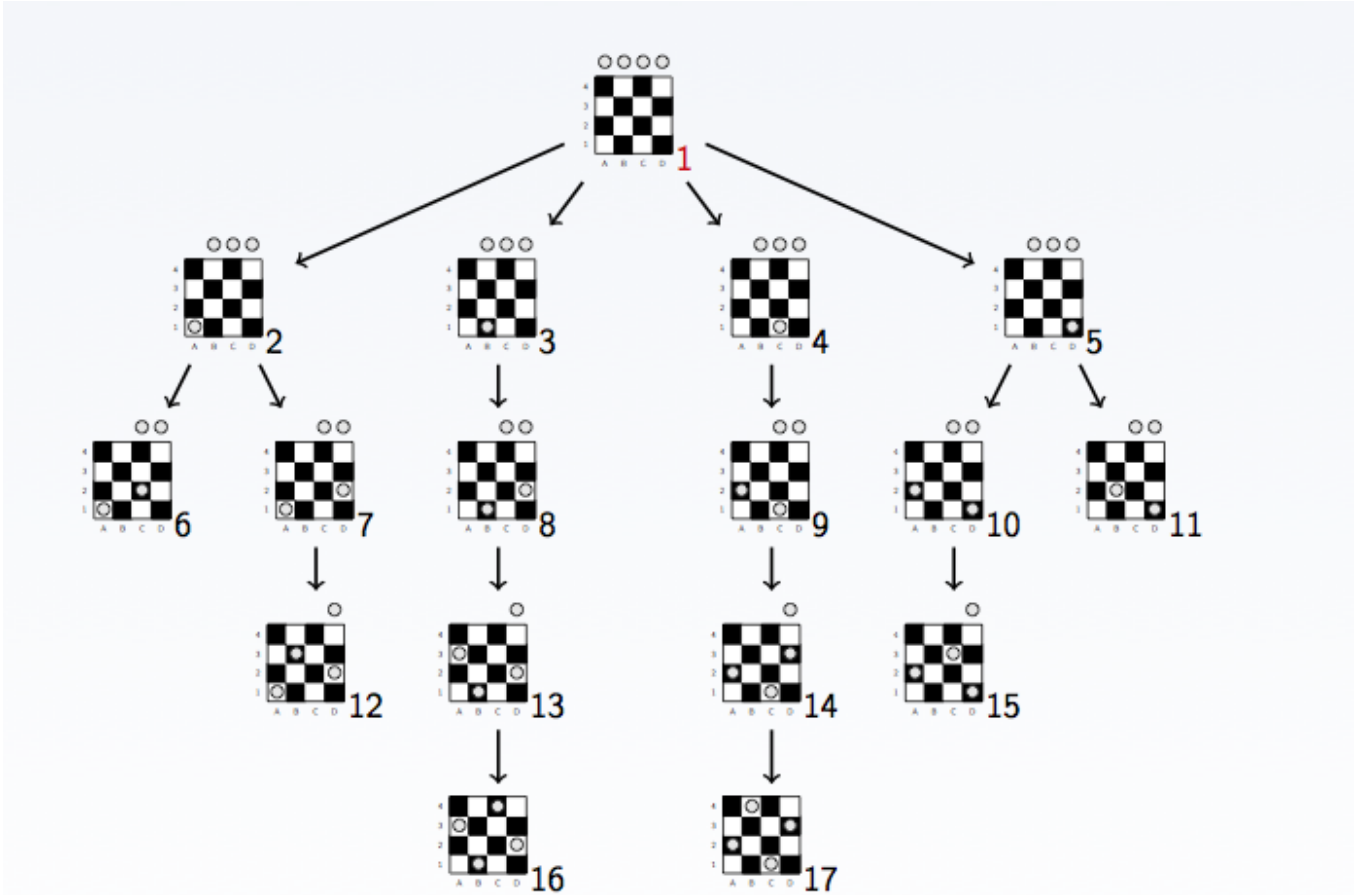
Datenstrukturen: L = Liste (Stack) von Knoten, markiert Weg dorthin
Eingabe: Füge die initialen Knoten in die Liste L ein.
Algorithmus:

1. Wenn L die leere Liste ist, dann breche ab.
2. Wähle ersten Knoten K aus L, sei R die Restliste.
3. Wenn K ein Zielknoten ist, dann gebe aus: Zielknoten und Weg dorthin (d.h. Weg im Graphen dorthin)
4. Wenn K kein Zielknoten, dann sei $N(K)$ die (geordnete) Liste der direkten Nachfolger von K, mit dem Weg dorthin markiert
   $L := N(K) + +R$. (wobei $++$ Listen zusammenhängt)
   Mache weiter mit 1.


Beispiel Tiefensuche



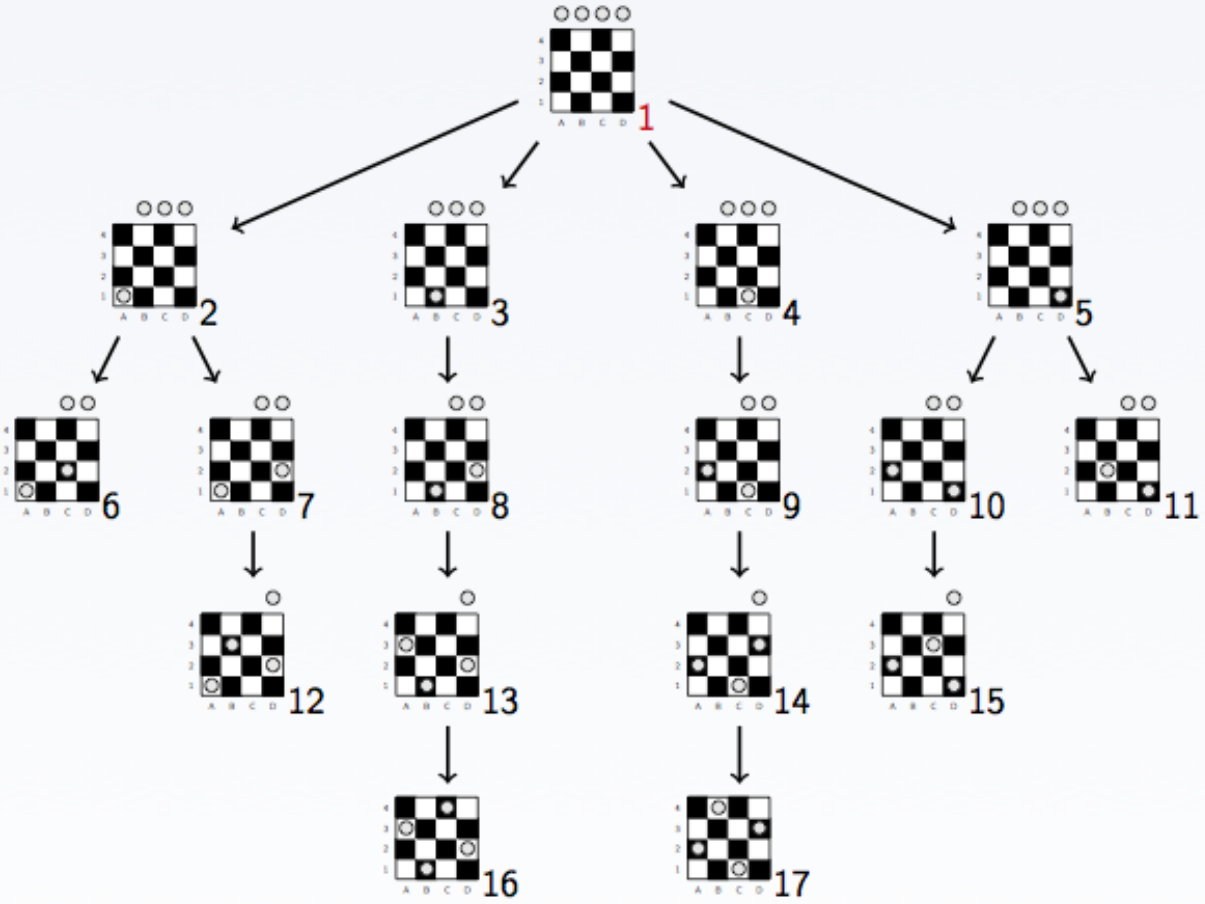
Beispiel Tiefensuche



Am Anfang:

$L := [(1, [])]$

Beispiel Tiefensuche



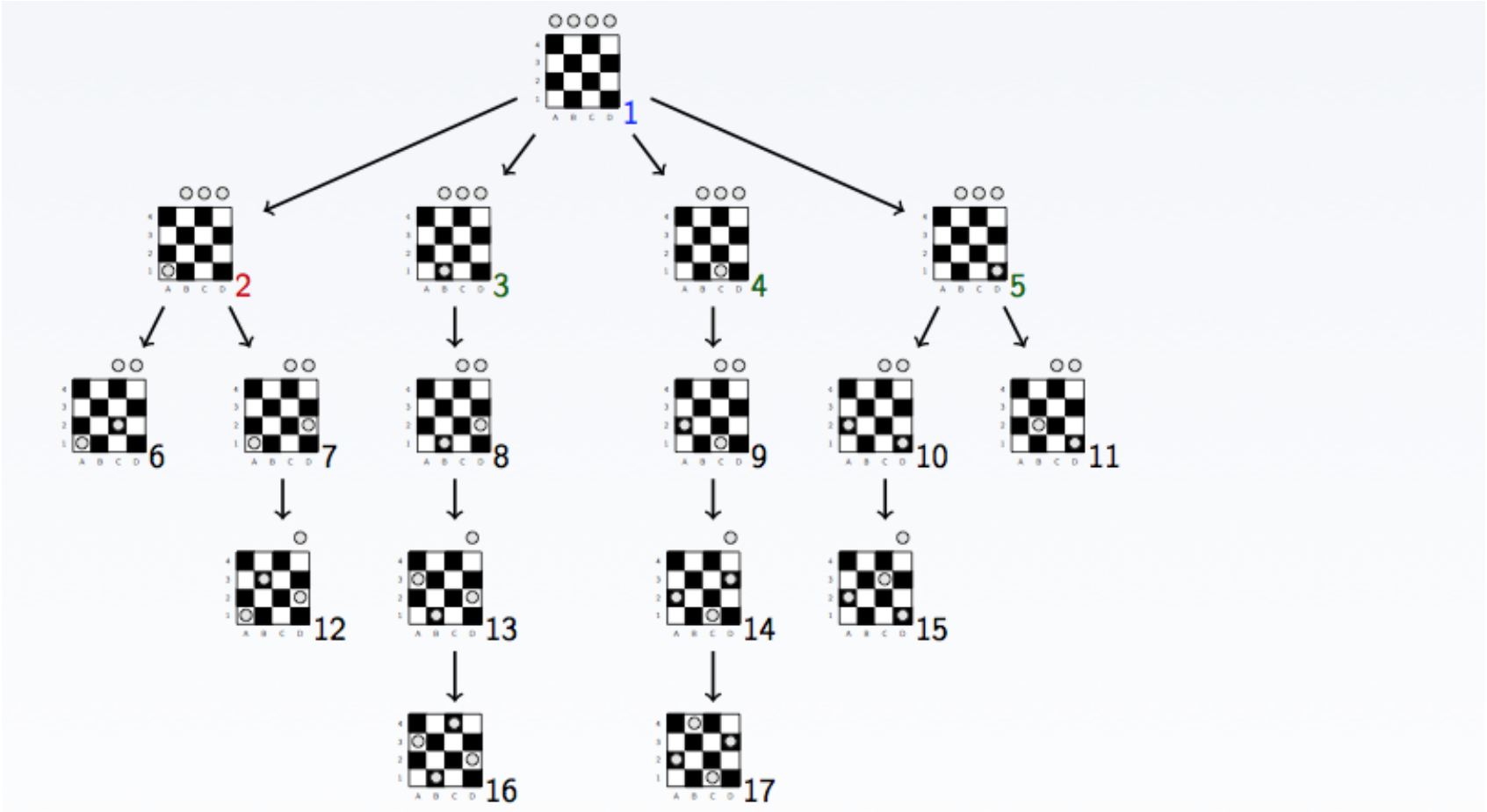
1. Knoten

$K := (1, [])$ $R := []$

$NF(K) = [2, 3, 4, 5]$

$L := [(2, [1]), (3, [1]), (4, [1]), (5, [1])] ++ R = [(2, [1]), (3, [1]), (4, [1]), (5, [1])]$

Beispiel Tiefensuche



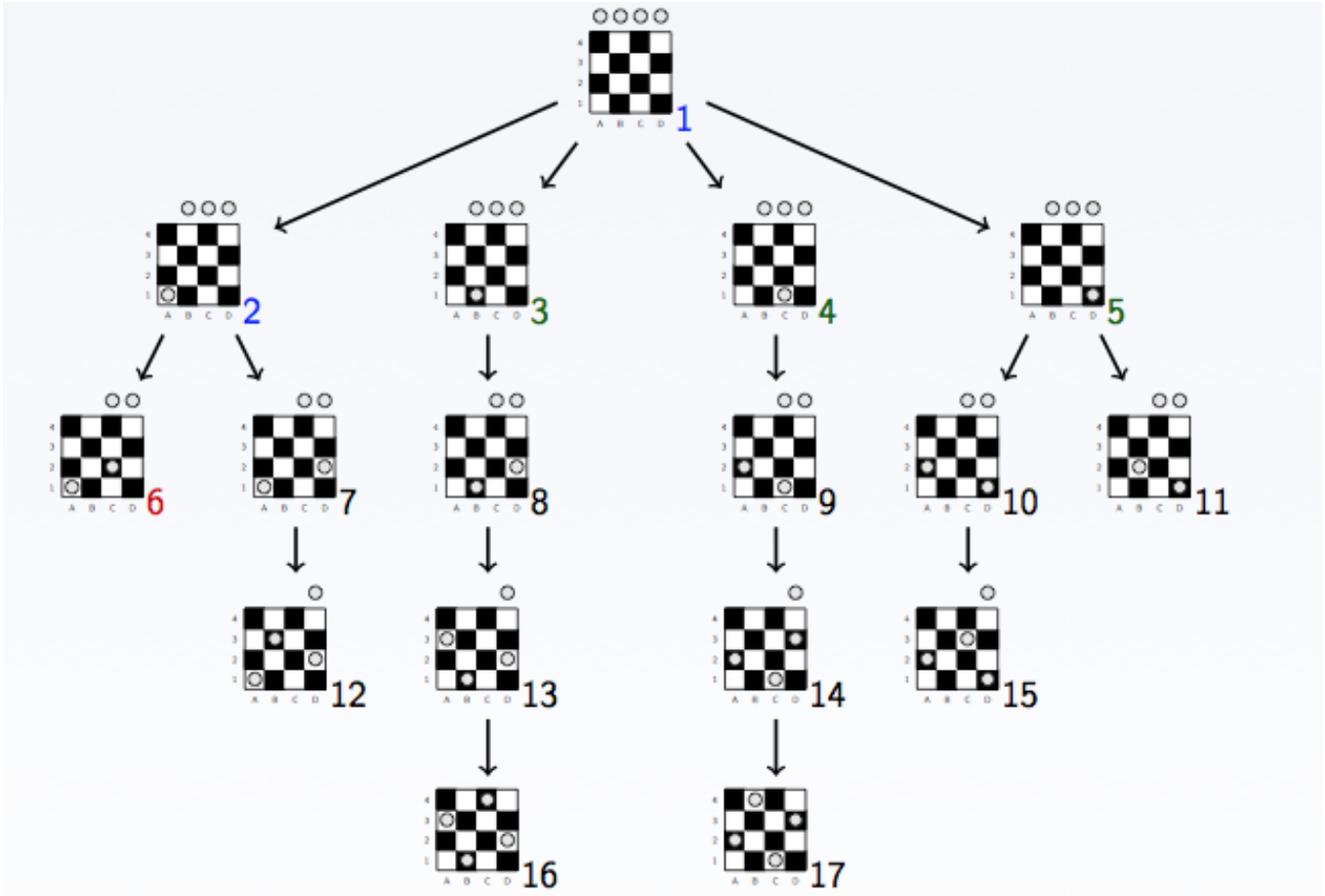
2. Knoten

$K := (2, [1])$ $R := [(3, [1]), (4, [1]), (5, [1])]$

$NF(2) = [6, 7]$

$L := [(6, [1, 2]), (7, [1, 2])]$++$R = [(6, [1, 2]), (7, [1, 2]), (3, 1), (4, 1), (5, 1)]$

Beispiel Tiefensuche



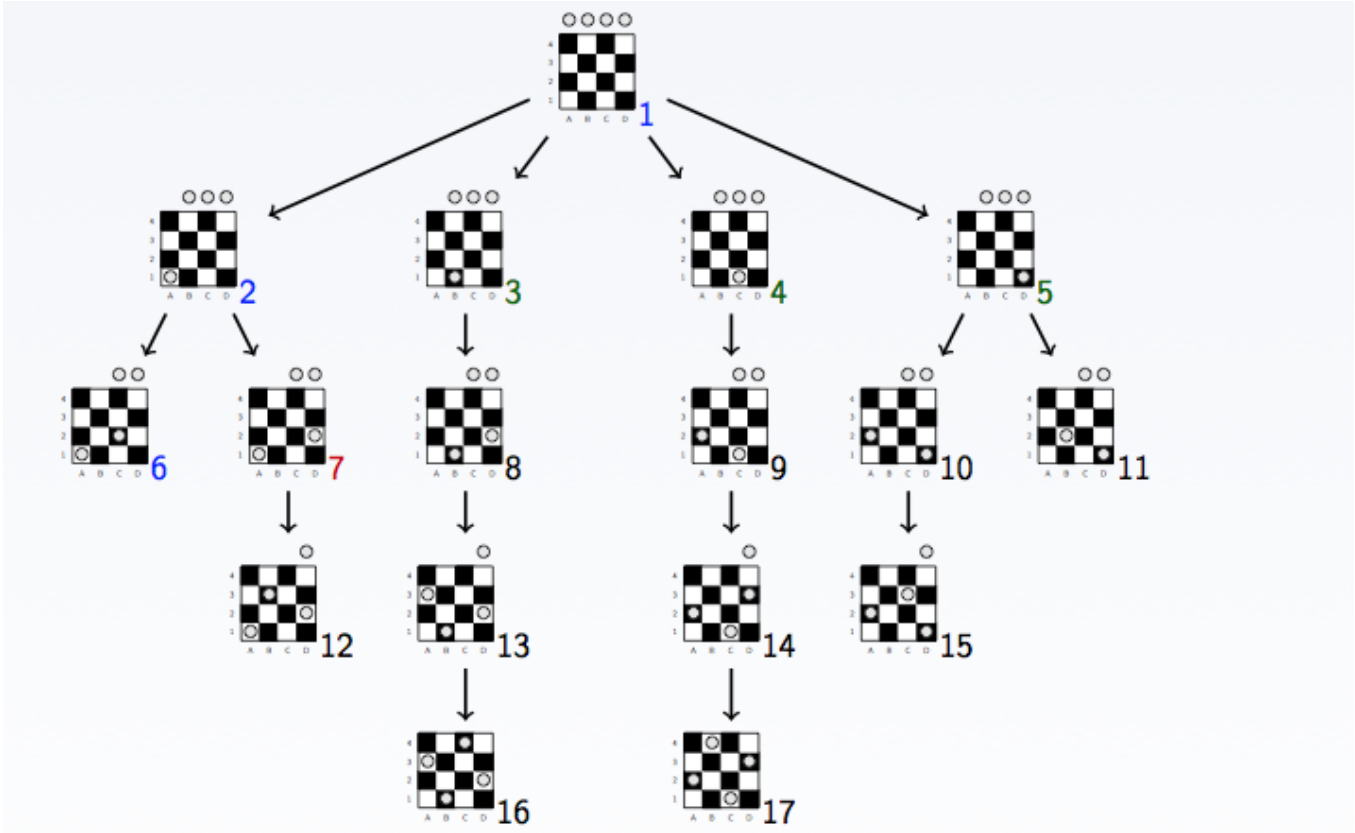
3. Knoten

$K := (6, [1, 2])$ $R := [(7, [1, 2]), (3, [1]), (4, [1]), (5, [1])]$

$NF(6) = []$

$L := []{+}{+}R = [(7, [1, 2]), (3, [1]), (4, [1]), (5, [1])]$

Beispiel Tiefensuche



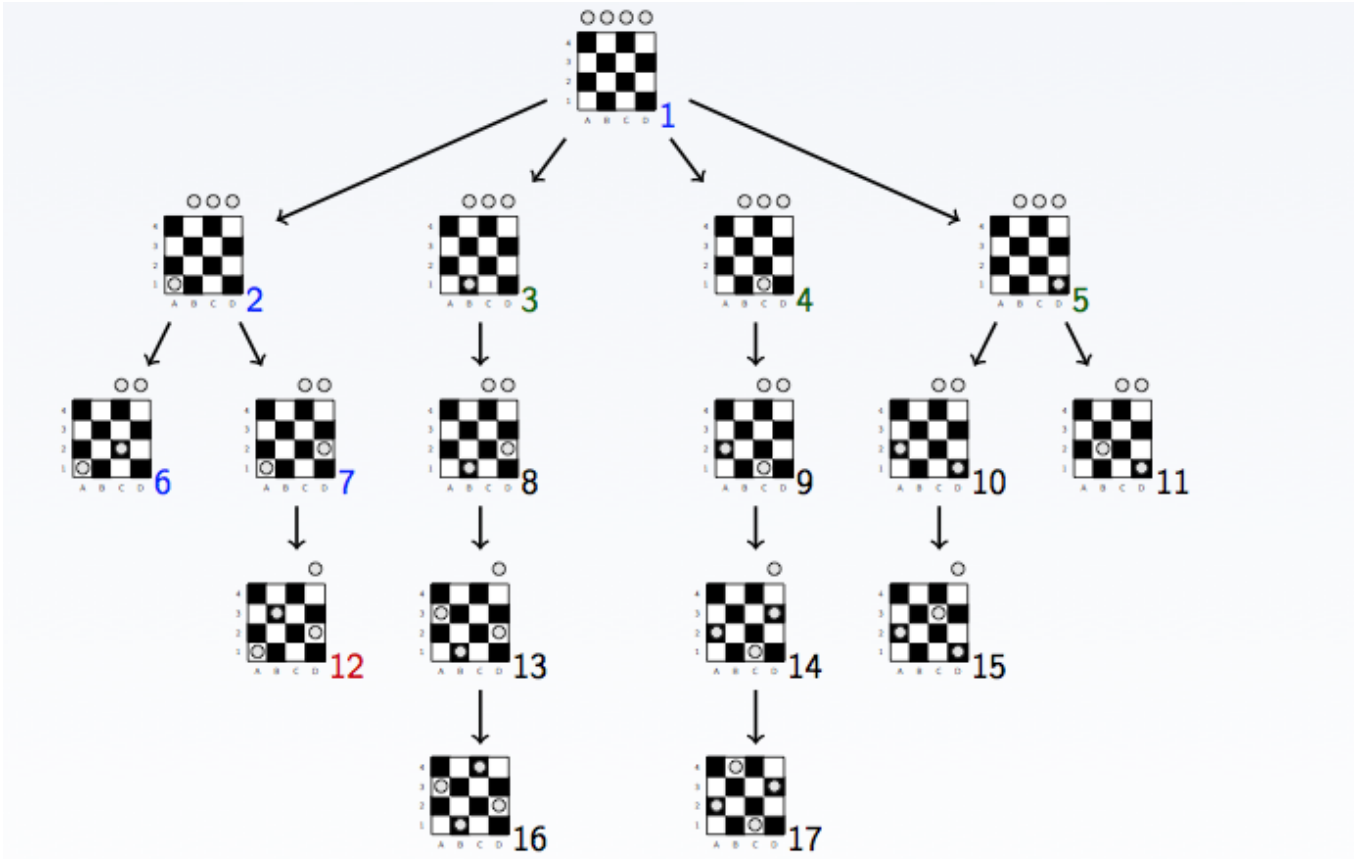
4. Knoten

$K := (7, [1, 2])$ $R := [(3, [1]), (4, [1]), (5, [1])]$

$NF(7) = [12]$

$L := [(12, [1, 2, 7])] ++ R = [(12, [1, 2, 7]), (3, [1]), (4, [1]), (5, [1])]$

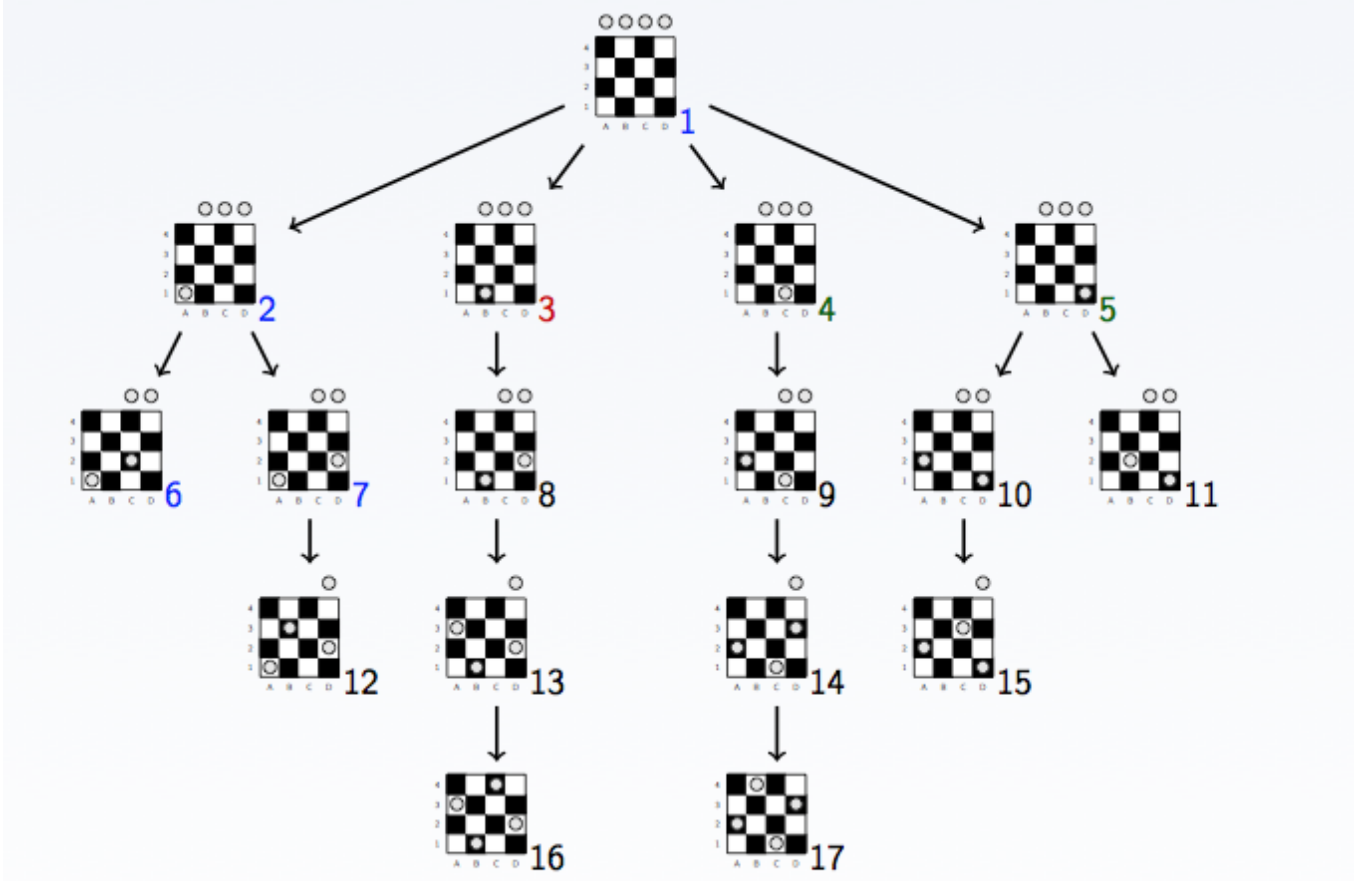
Beispiel Tiefensuche



5. Knoten

$K := (12, [1, 2, 7]) \quad R := [(3, [1]), (4, [1]), (5, [1])]$
$NF(12) = []$
$L := []{+}{+}R = [(3, [1]), (4, [1]), (5, [1])]$

Beispiel Tiefensuche



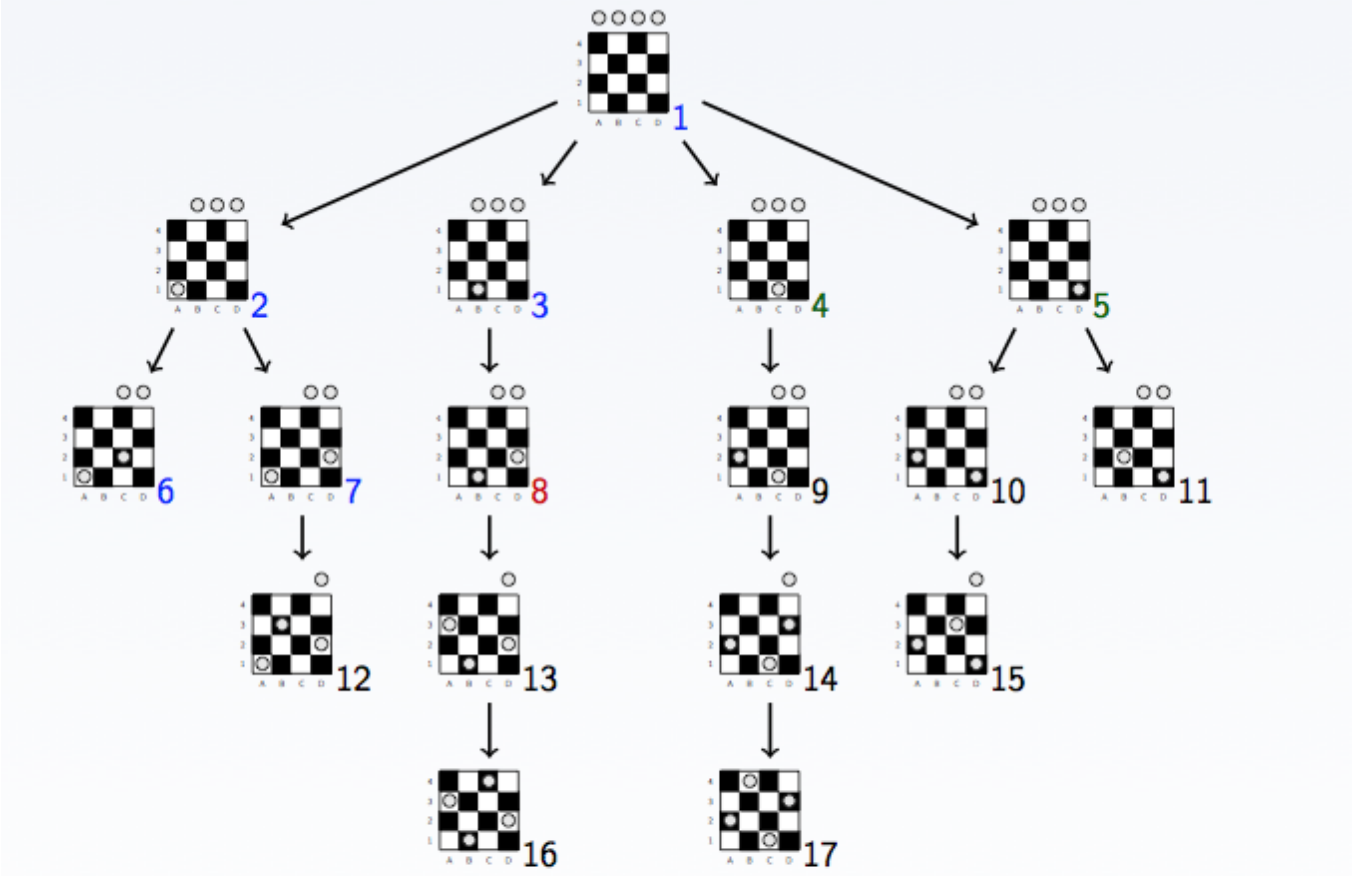
6. Knoten

$K := (3, [1]) \quad R := [(4, [1]), (5, [1])]$

$NF(3) = [8]$

$L := [(8, [1, 3])] ++ R = [(8, [1, 3]), (4, [1]), (5, [1])]$

Beispiel Tiefensuche



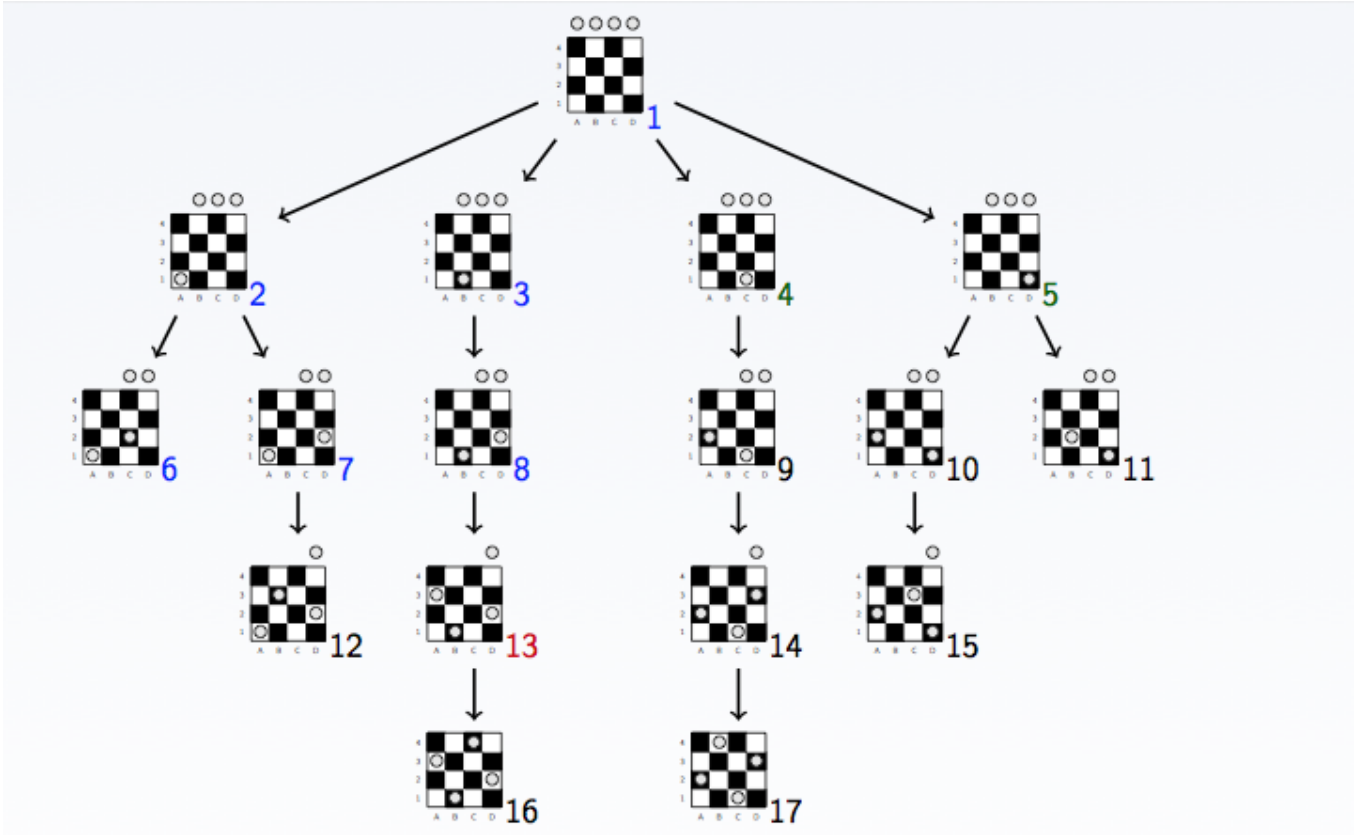
7. Knoten

$K := (8, [1, 3])$ $R := [(4, [1]), (5, [1])]$

$NF(8) = [13]$

$L := [(13, [1, 3, 8])] ++ R = [(13, [1, 3, 8]), (4, [1]), (5, [1])]$

Beispiel Tiefensuche



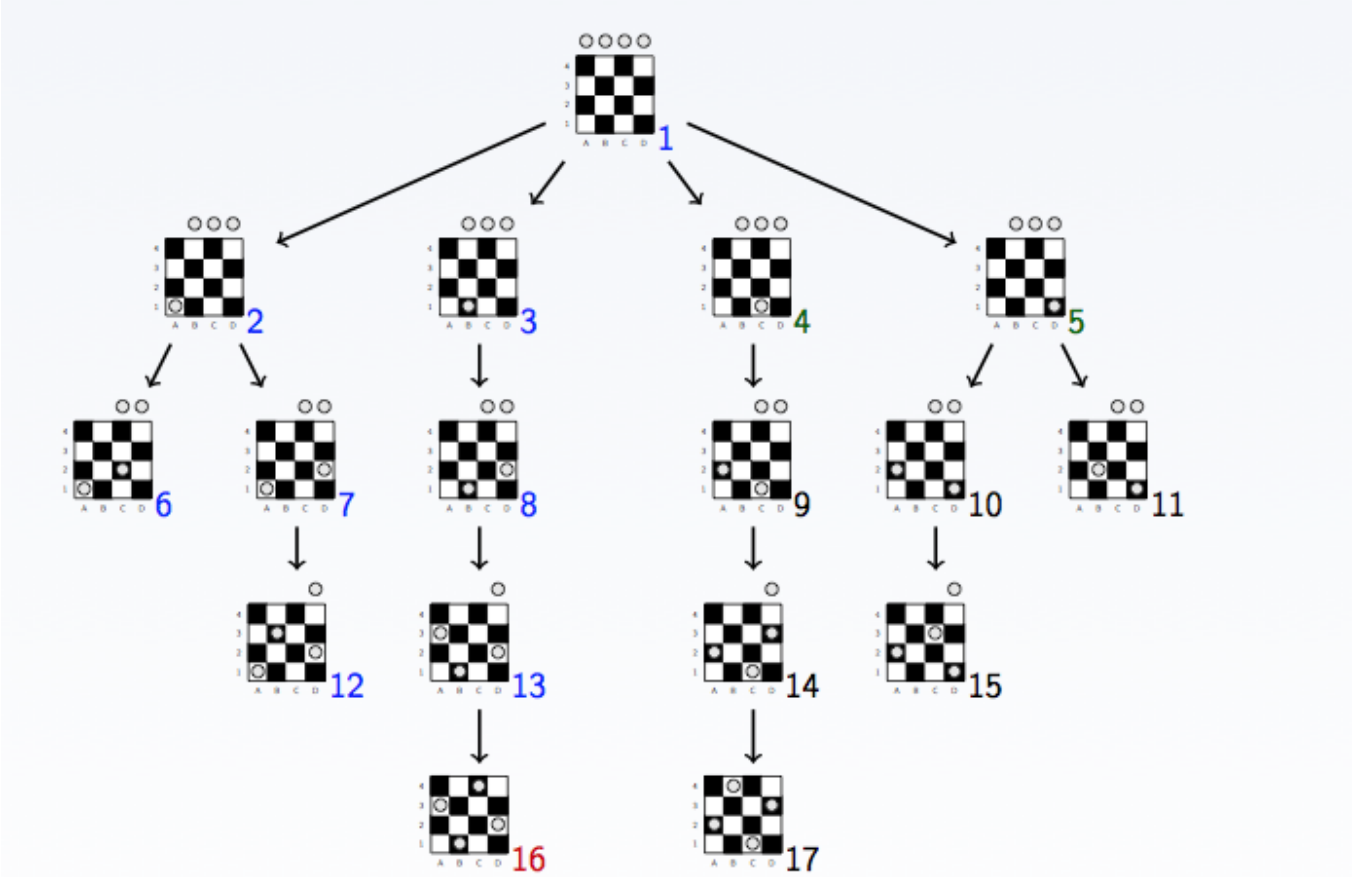
8. Knoten

$K := (13, [1, 3, 8])$ $R := [(4, [1]), (5, [1])]$

$NF(13) = [16]$

$L := [(16, [1, 3, 8, 13])] ++ R = [(16, [1, 3, 8, 13]), (4, [1]), (5, [1])]$

Beispiel Tiefensuche



9. Knoten

$K := (16, [1, 3, 8, 13])$ $R := [(4, [1]), (5, [1])]$

$Ziel(16) ==$ `True` $\Rightarrow$ gebe 1,3,8,13,16 aus

Eigenschaften der Tiefensuche

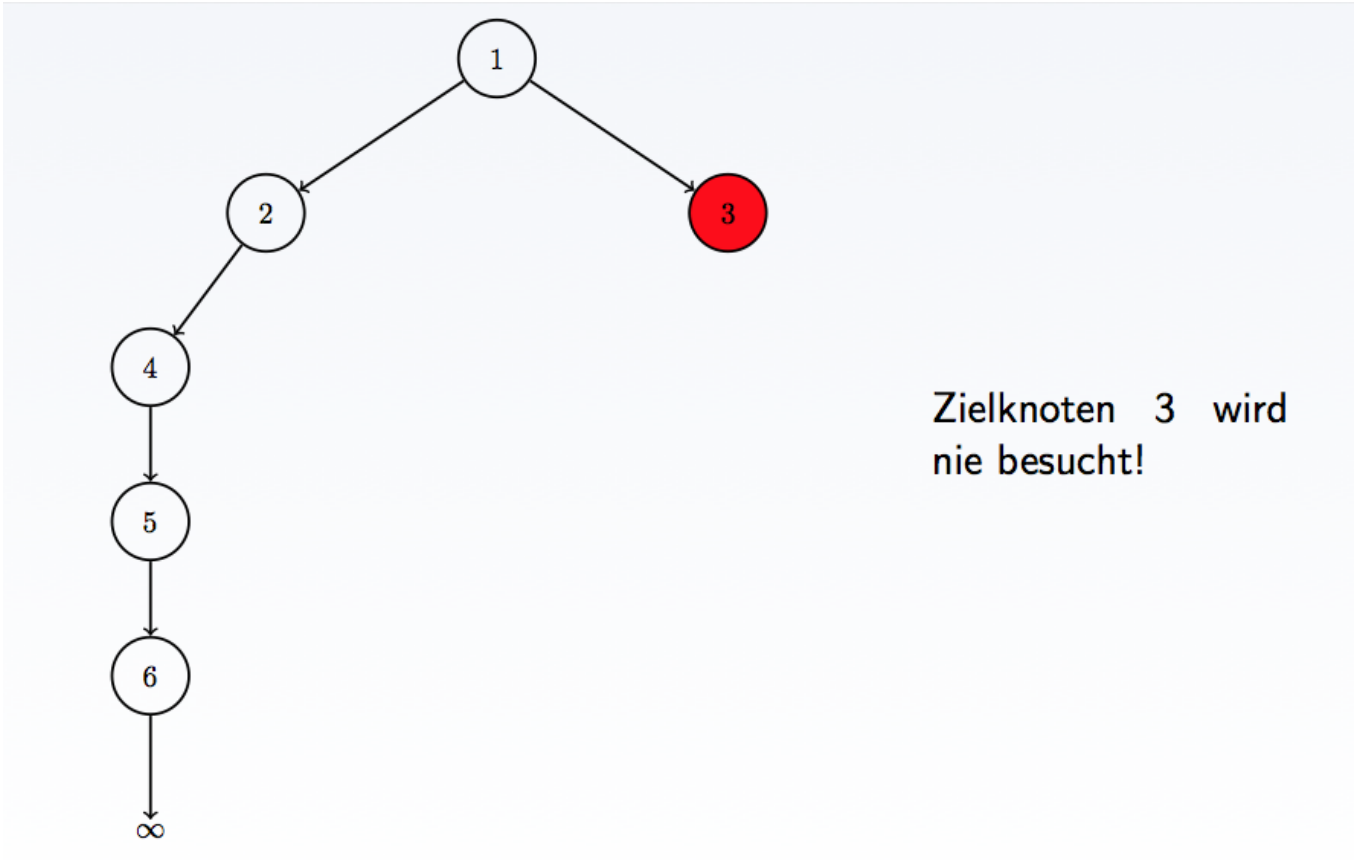
Komplexität (worst-case) bei fester Verzweigungrate $c > 1$:

- Platz: linear in der Tiefe
- Zeit: exponentiell in der Tiefe des Zielknotens

Vollständigkeit:

- Nicht vollständig, wenn der Suchgraph unendlich groß ist

Unvollständigkeit der Tiefensuche



Varianten der Tiefensuche

Tiefensuche mit Tiefenbeschränkung k

- Wenn Tiefe k überschritten, setze $NF(K) = \emptyset$
- Findet Zielknoten, die maximal in Tiefe k liegen

Varianten der Tiefensuche

Tiefensuche mit Sharing

- Merke bereits besuchte Knoten, um Knoten nicht doppelt zu besuchen (bei zyklischen Suchgraphen!)
- Speichern der besuchten Knoten: Hashtabelle
- Platz: Anzahl der besuchten Knoten (wegen Speicher fu r schon untersuchte Knoten)
- Zeit: $n \times \log(n)$ mit n = Anzahl der untersuchten Knoten.
- Pragmatische Verbesserung: Nur maximal l viele Knoten speichern (damit der Platz beschränkt ist)

Varianten beim Backtracking

Vorgestelltes Verfahren

- Chronologisches Backtracking

Varianten

- Dynamic Backtracking
- Dependency-directed Backtracking

Abkürzungen, wenn sichergestellt ist, dass kein Zielknoten übersehen wird.

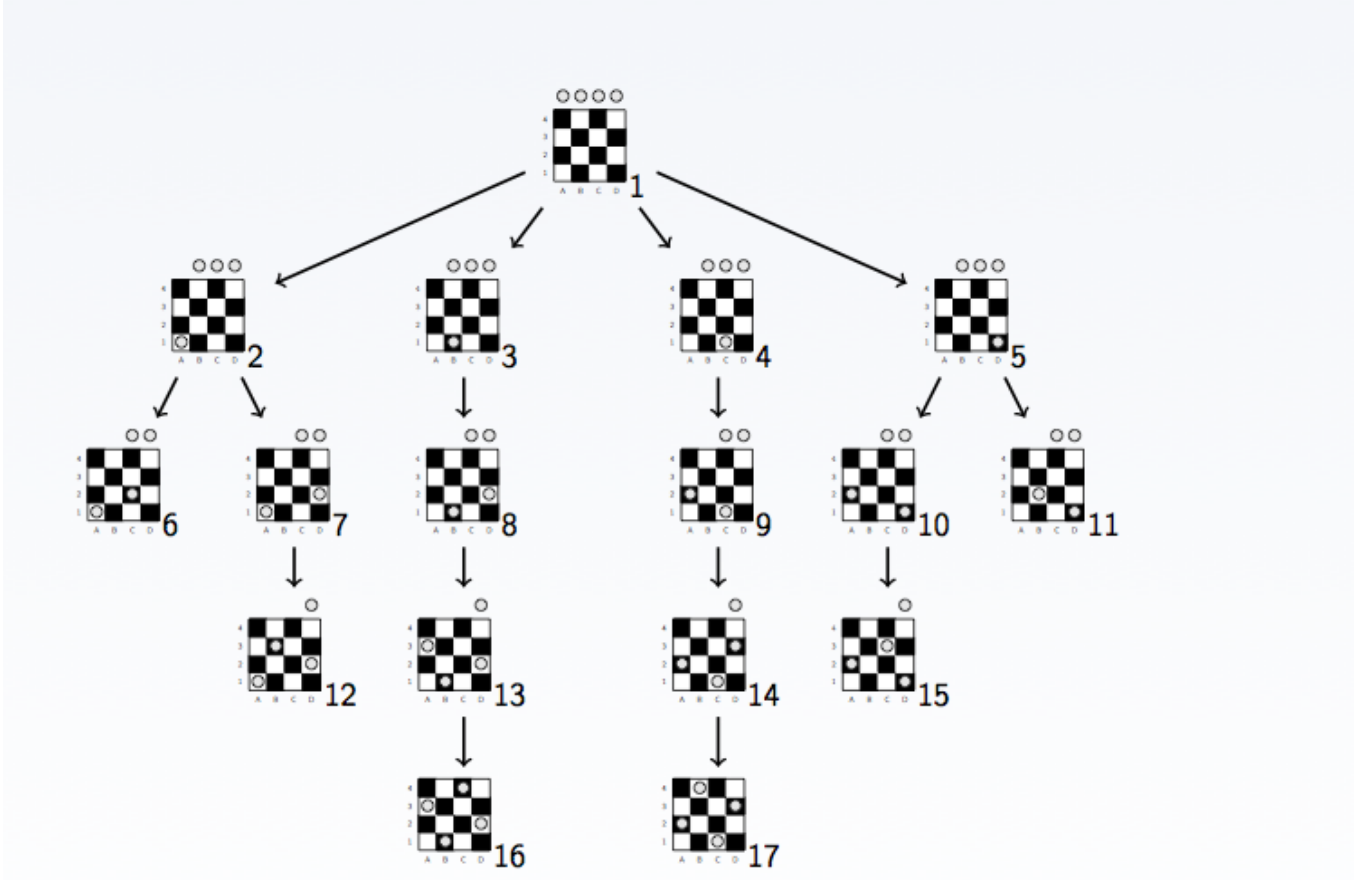
Breitensuche

Algorithmus Breitensuche

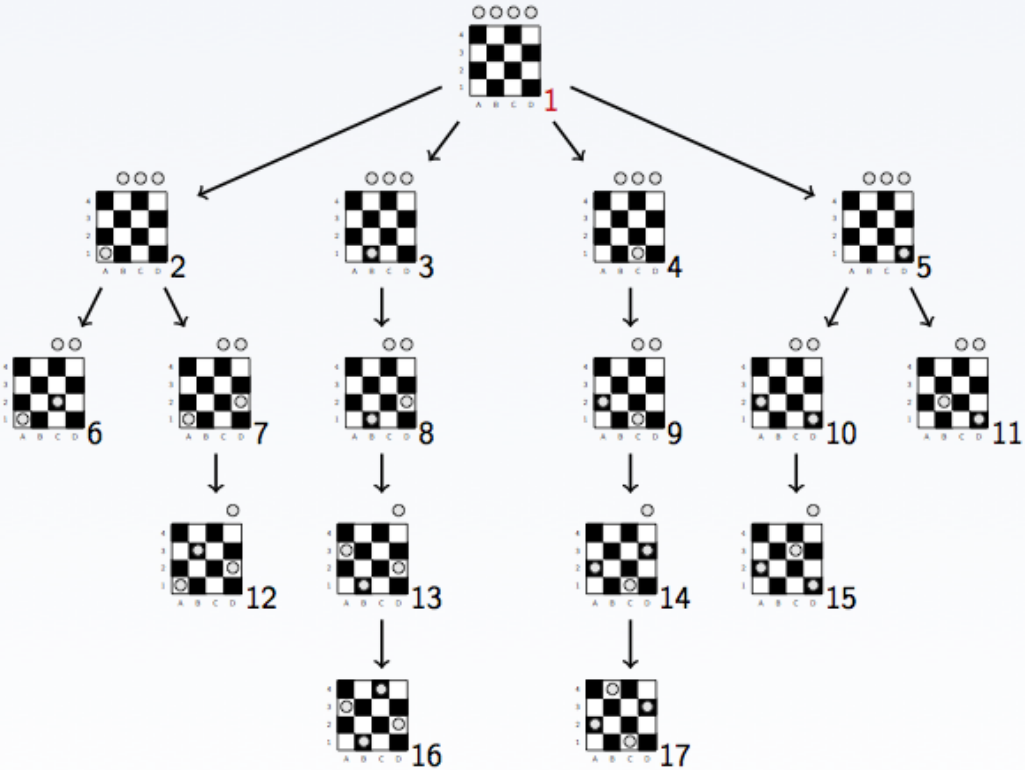
Datenstrukturen: L = Menge von Knoten markiert mit Weg
Eingabe: Füge die initialen Knoten in die Menge L ein.
Algorithmus:

1. Wenn L leer ist, dann breche ab.
2. Wenn L einen Zielknoten K enthält, dann gebe aus: K und Weg dorthin.
3. Sonst, sei $N(L)$ Menge aller direkten Nachfolger der Knoten von L, mit einem Weg dorthin markiert. Mache weiter mit Schritt 1 und $L := N(L)$.

Beispiel Breitensuche



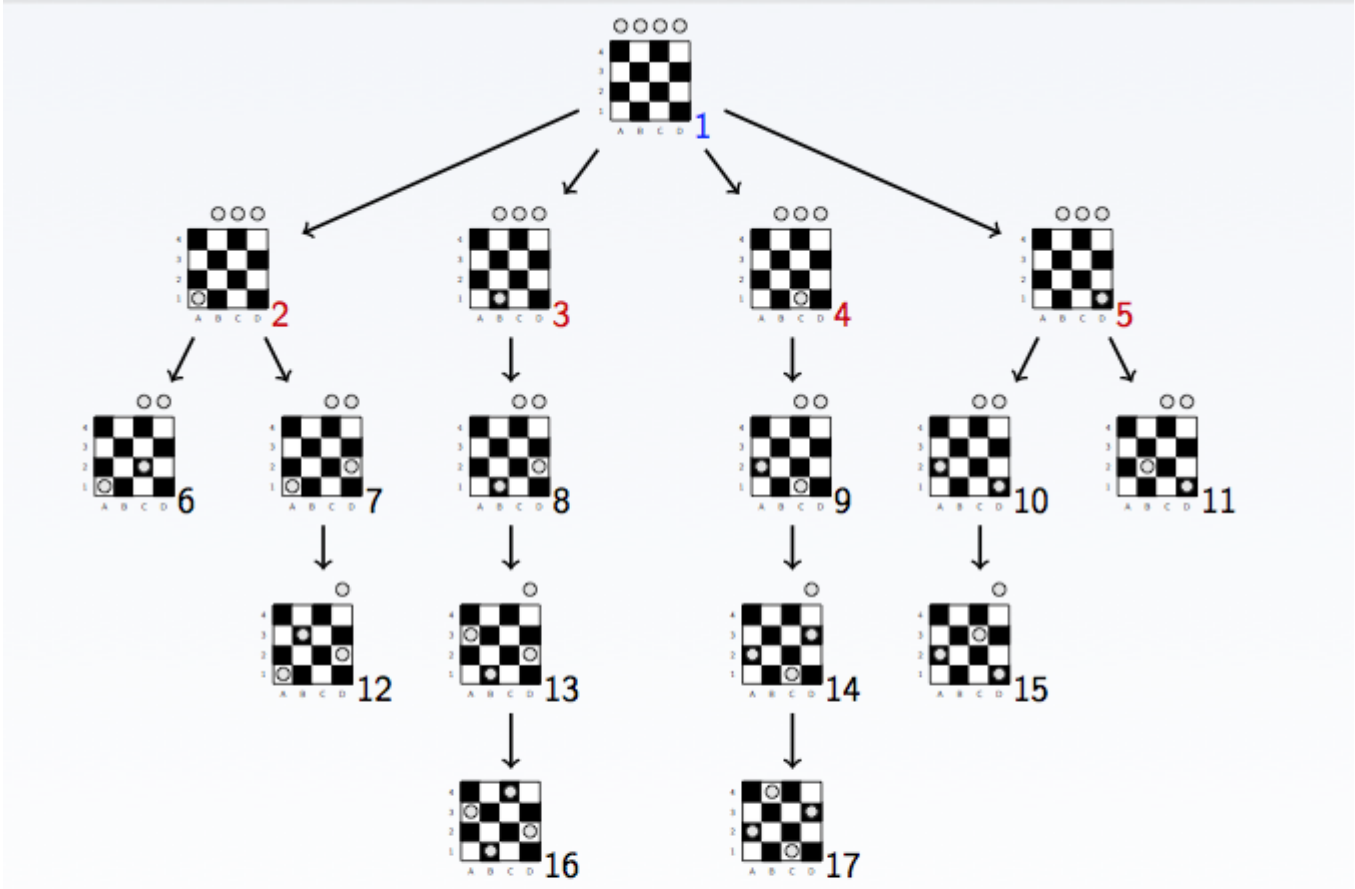
Beispiel Breitensuche



Am Anfang:

$L := \{(1, [])\}$

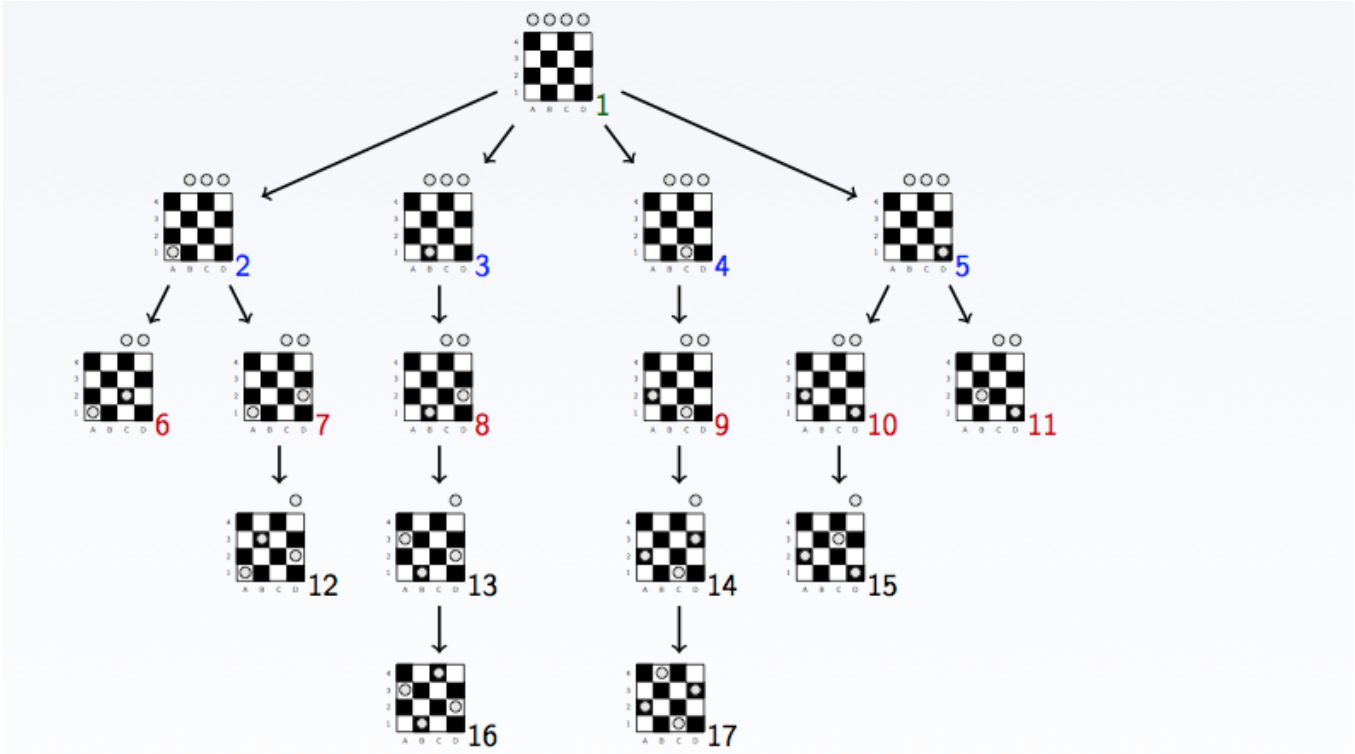
Beispiel Breitensuche



1. Iteration

$L := \{(1, [])\}$
$NF(L) = \{2, 3, 4, 5\}$
$L := \{(2, [1]), (3, [1]), (4, [1]), (5, [1])\}$

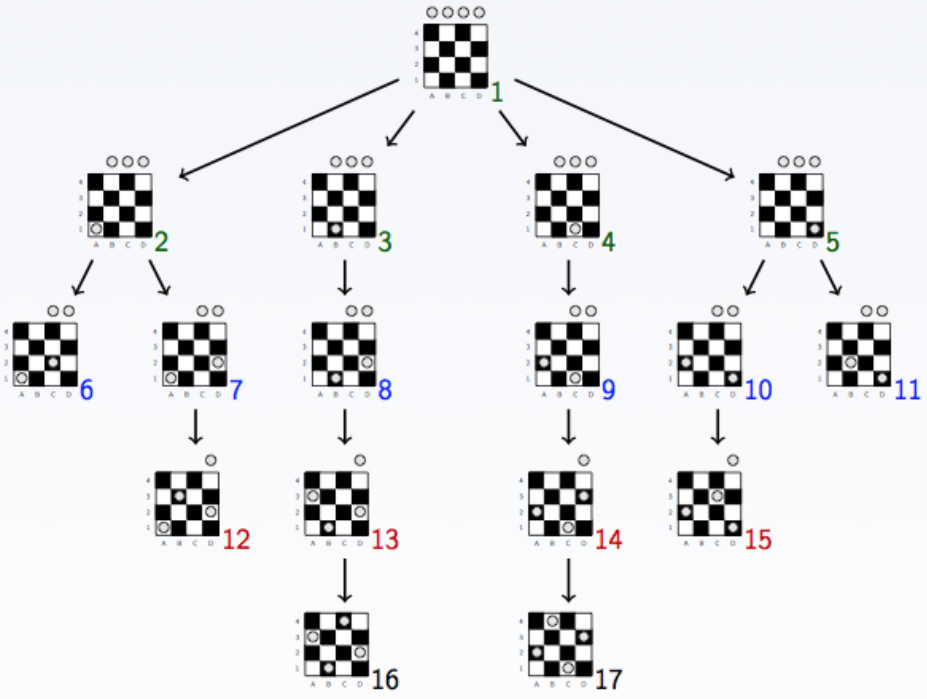
Beispiel Breitensuche



2. Iteration

$L := \{(2,[1]),(3,[1]),(4,[1]),(5,[1])\}$
$NF(L) = NF(2) \cup NF(2) \cup NF(4) \cup NF(5) = \{6,7,8,9,10,11\}$
$L := \{(6,[1,2]),(7,[1,2]),(8,[1,3]),(9,[1,4]),(10,[1,5]),(11,[1,5])\}$

Beispiel Breitensuche



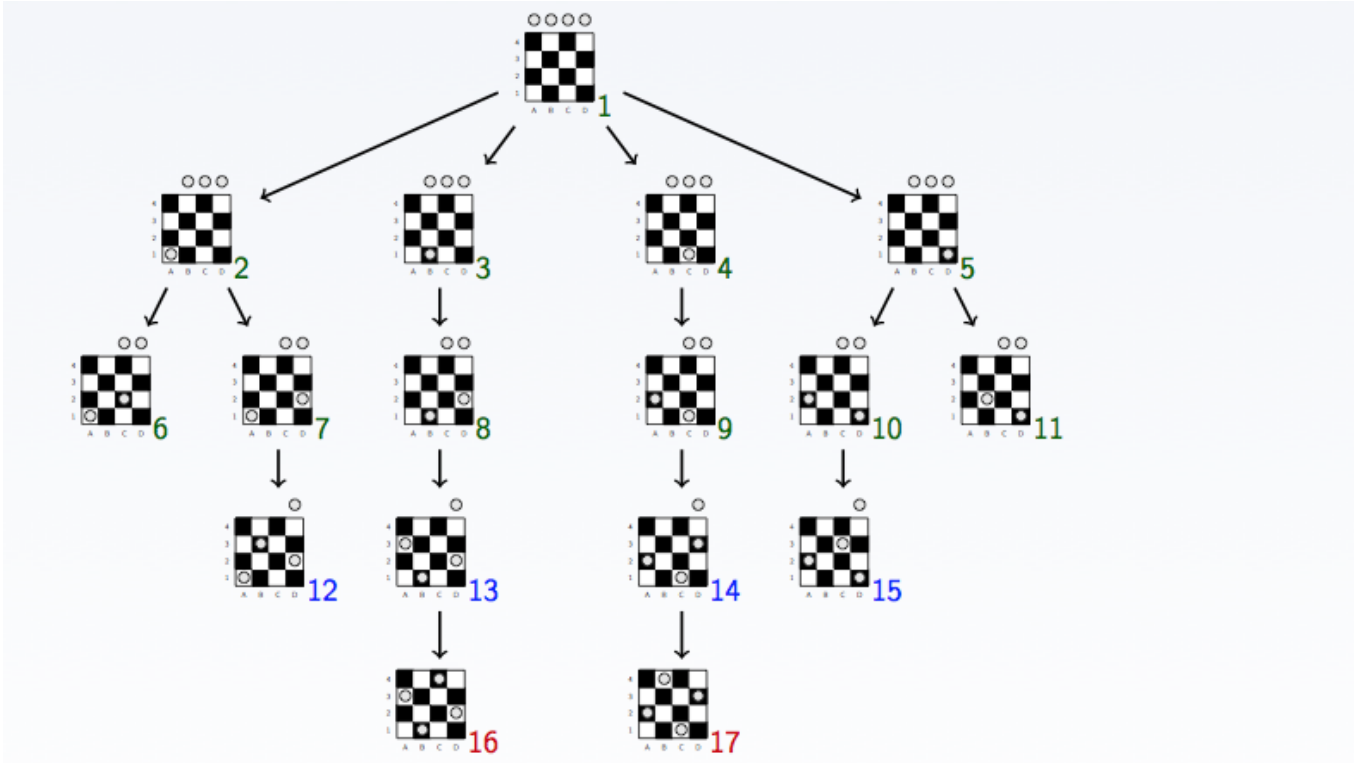
3. Iteration

$L := \{(6, [1, 2]), (7, [1, 2]), (8, [1, 3]), (9, [1, 4]), (10, [1, 5]), (11, [1, 5])\}$

$NF(L) = NF(6) \cup NF(7) \cup NF(8) \cup NF(8) \cup NF(9) \cup NF(10) \cup NF(11) = \{12, 13, 14, 15\}$

$L := \{(12, [1, 2, 7]), (13, [1, 3, 8]), (14, [1, 4, 9]), (15, [1, 5, 10])\}$

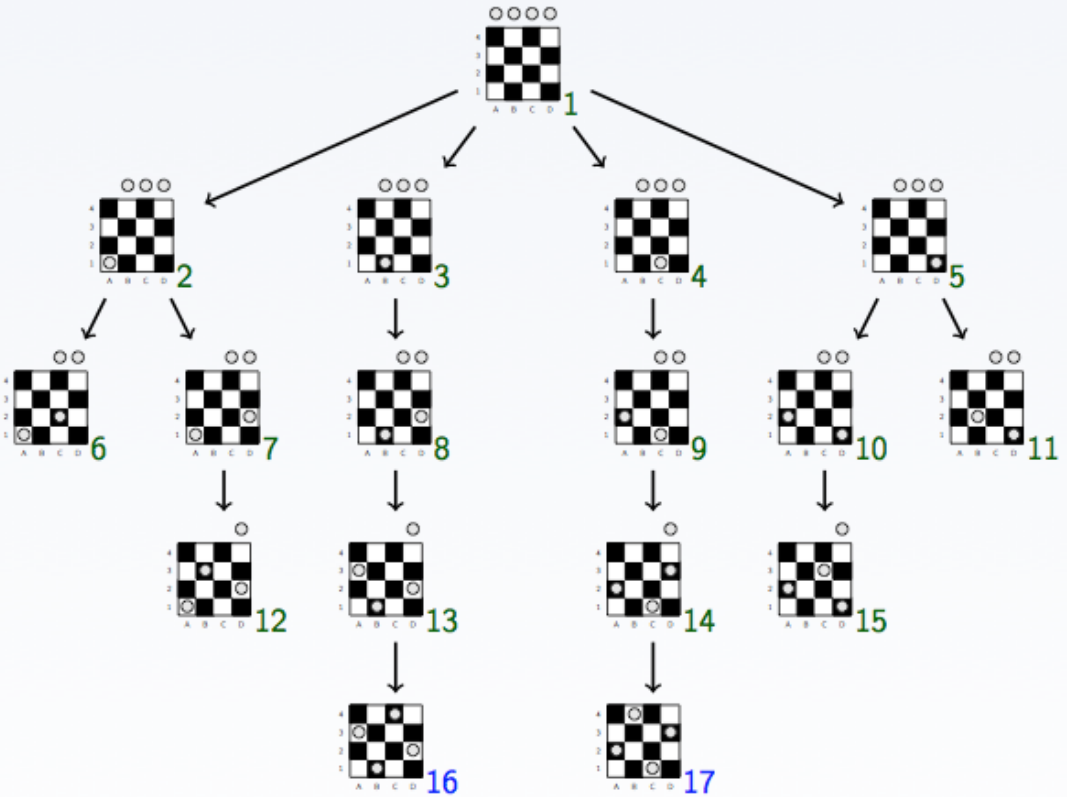
Beispiel Breitensuche



4. Iteration

$L := \{(12, [1, 2, 7]), (13, [1, 3, 8]), (14, [1, 4, 9]), (15, [1, 5, 10])\}$
$NF(L) = NF(12) \cup NF(13) \cup NF(14) \cup NF(15) = \{16, 17\}$
$L := \{(16, [1, 3, 8, 13]), (17, [1, 4, 9, 14])\}$

Beispiel Breitensuche



5. Iteration

$L := \{(16, [1, 3, 8, 13]), (17, [1, 4, 9, 14])\}$

L enhält Zielknoten $\Rightarrow$ gebe 1,3,8,13,16 aus

Eigenschaften

Komplexität (worst-case) bei fester Verzweigungrate $c > 1$

- **Platz**: Anzahl der Knoten in Tiefe d, d.h. $O(c^d) =$ exponentiell in der Tiefe d!
- **Zeit**: $\underbrace{\text{Anzahl der Knoten in Tiefe } d}_{n} + \underbrace{(n \log n)}_{\text{Mengenbildung}} =$

  $O(c^d(1 + d * \log\ c))$

Vollständigkeit

- Die Breitensuche ist vollständig! (bei endlicher Verzweigungrate)

Fazit: Tiefen- und Breitensuche

	Tiefensuche	Breitensuche
Zeit	$O(c^d)$	$O(c^d(1 + d \log c))$
Platz	$O(d)$	$O(c^d)$
Vollständig	nein	ja

Iteratives Vertiefen

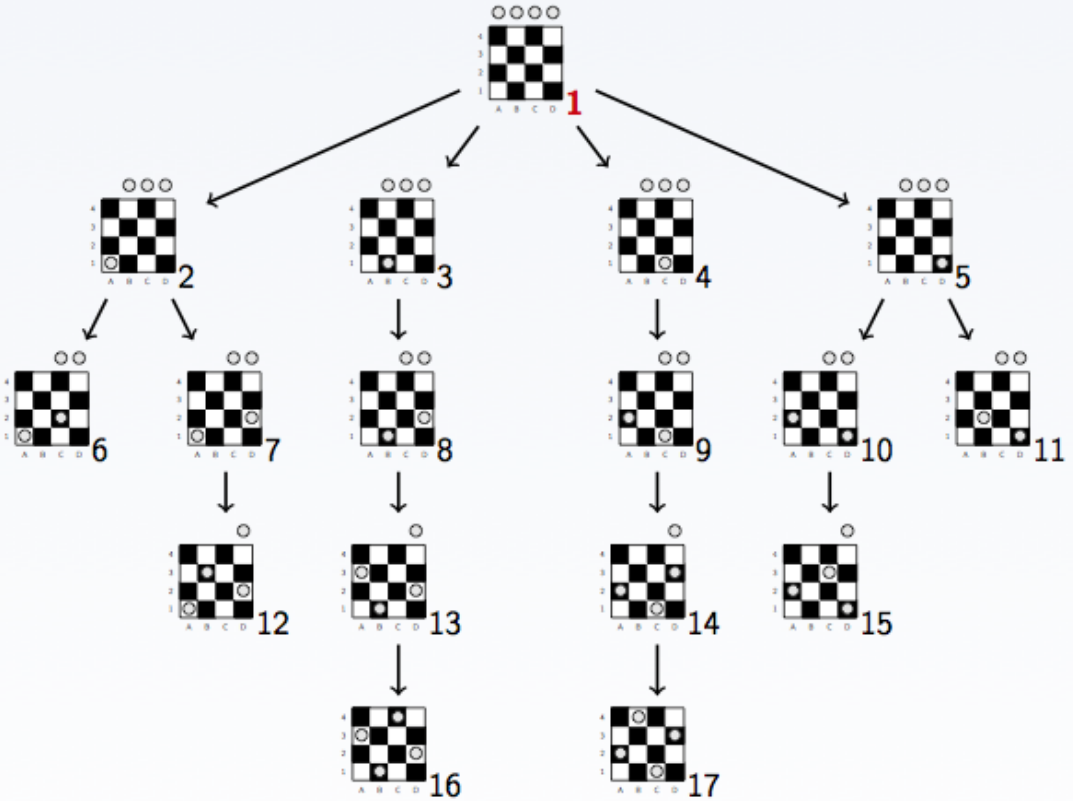
- iterative deepening
- Kompromiss a la *Vollständige Tiefensuche*

Pseudo-Code

:

1. $k := 0$;
2. Tiefensuche mit Tiefenschranke k;
3. **wenn** Ziel gefunden **dann**
   breche ab, und gebe Ziel mit Weg aus
   sonst
4. $k := k + 1$;
5. gehe zu 2

Beispiel: Iterative Tiefensuche

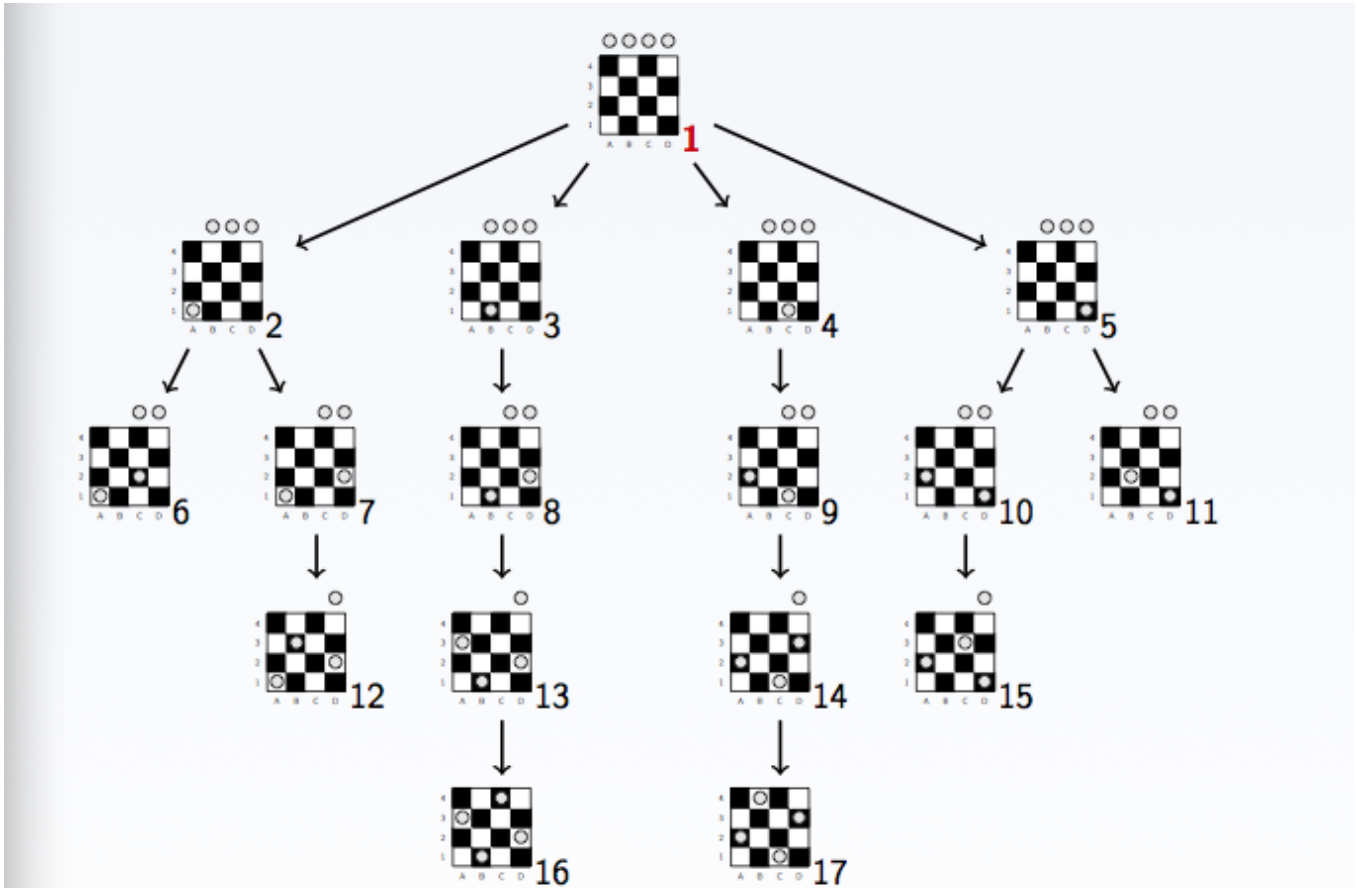


Reihenfolge der Knotenbesuche

$k = 1$

1

Beispiel: Iterative Tiefensuche

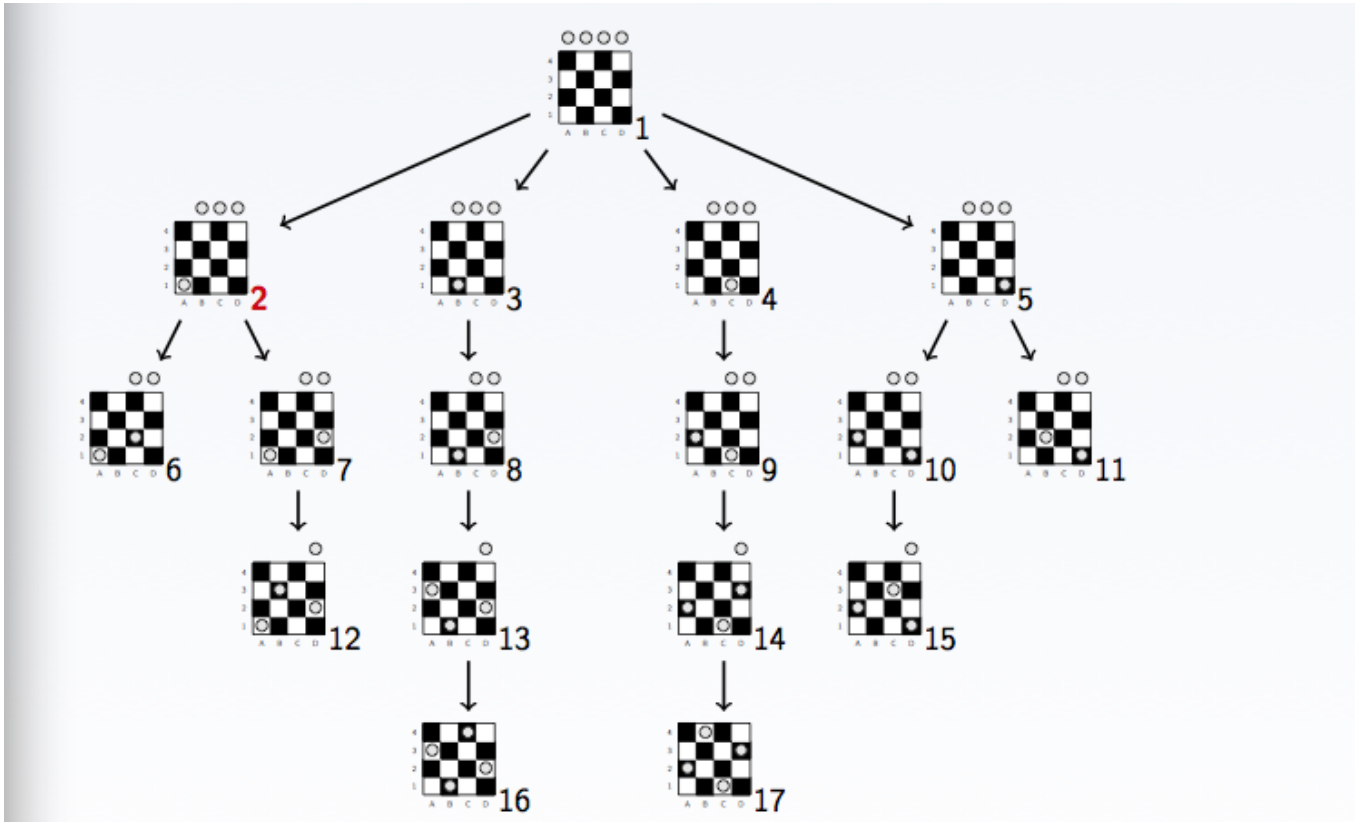


Reihenfolge der Knotenbesuche

$k = 2$

1,2,3,4,5

Beispiel: Iterative Tiefensuche

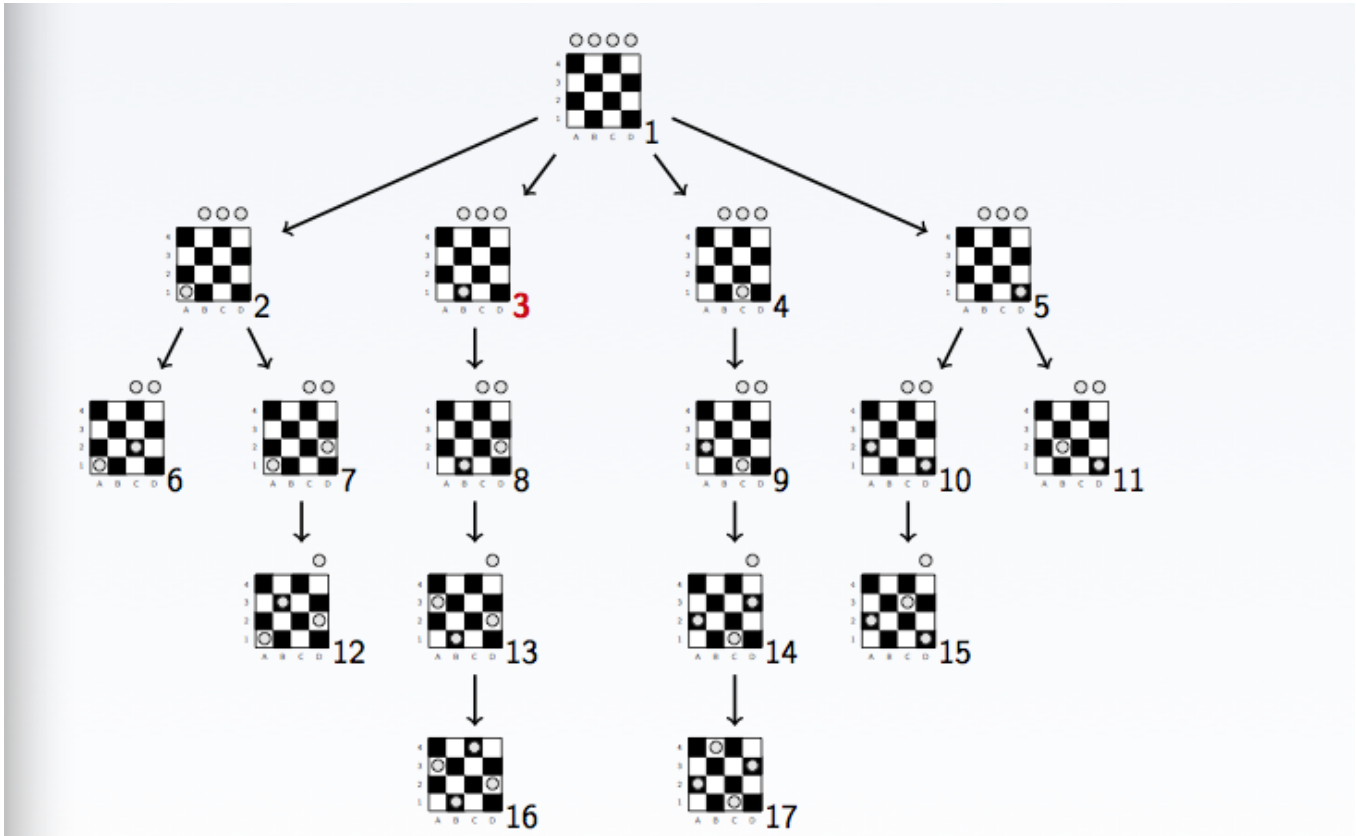


Reihenfolge der Knotenbesuche

$k = 2$

1,2,3,4,5

Beispiel: Iterative Tiefensuche

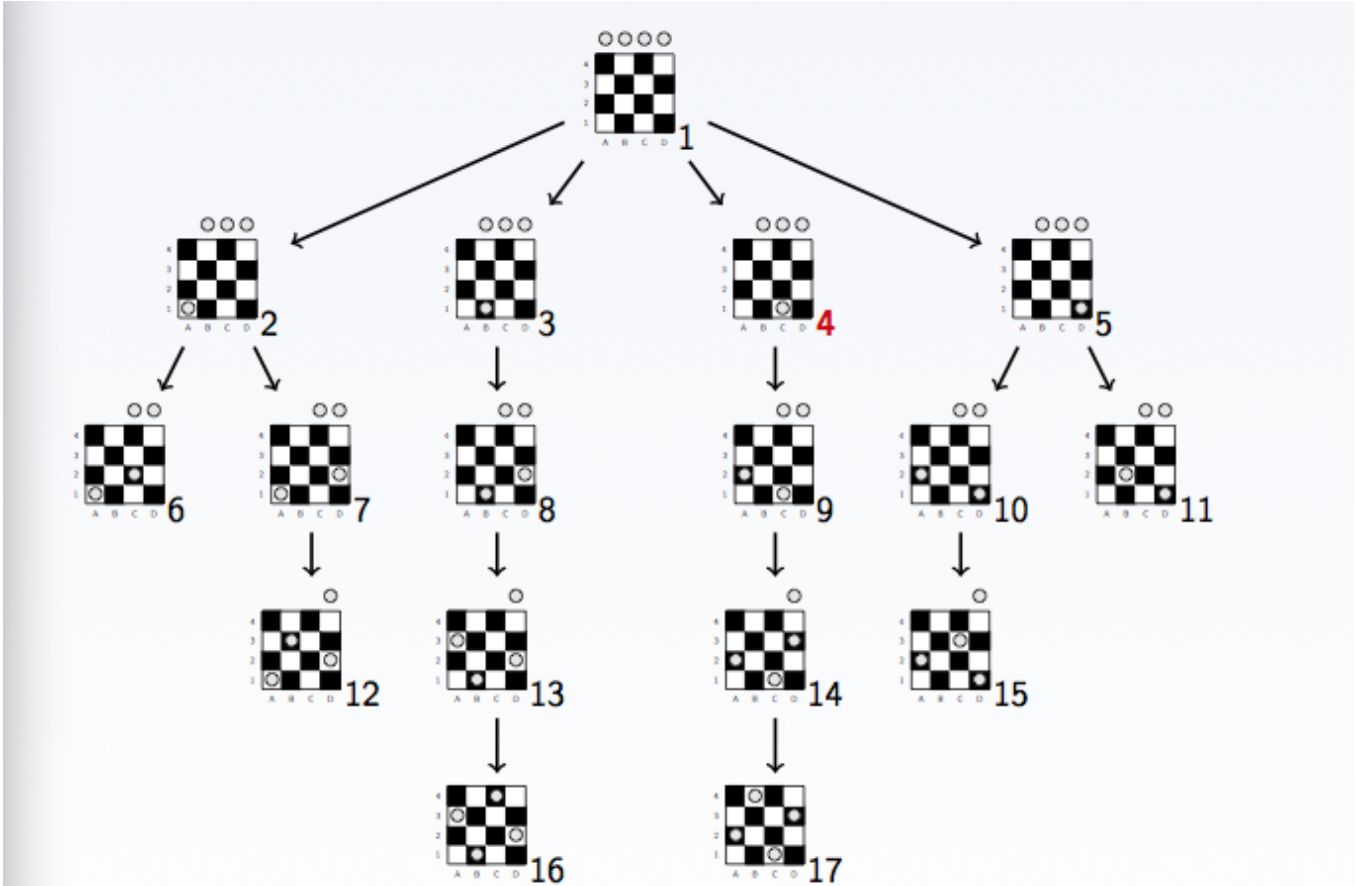


Reihenfolge der Knotenbesuche

$k = 2$

1,2,3,4,5

Beispiel: Iterative Tiefensuche

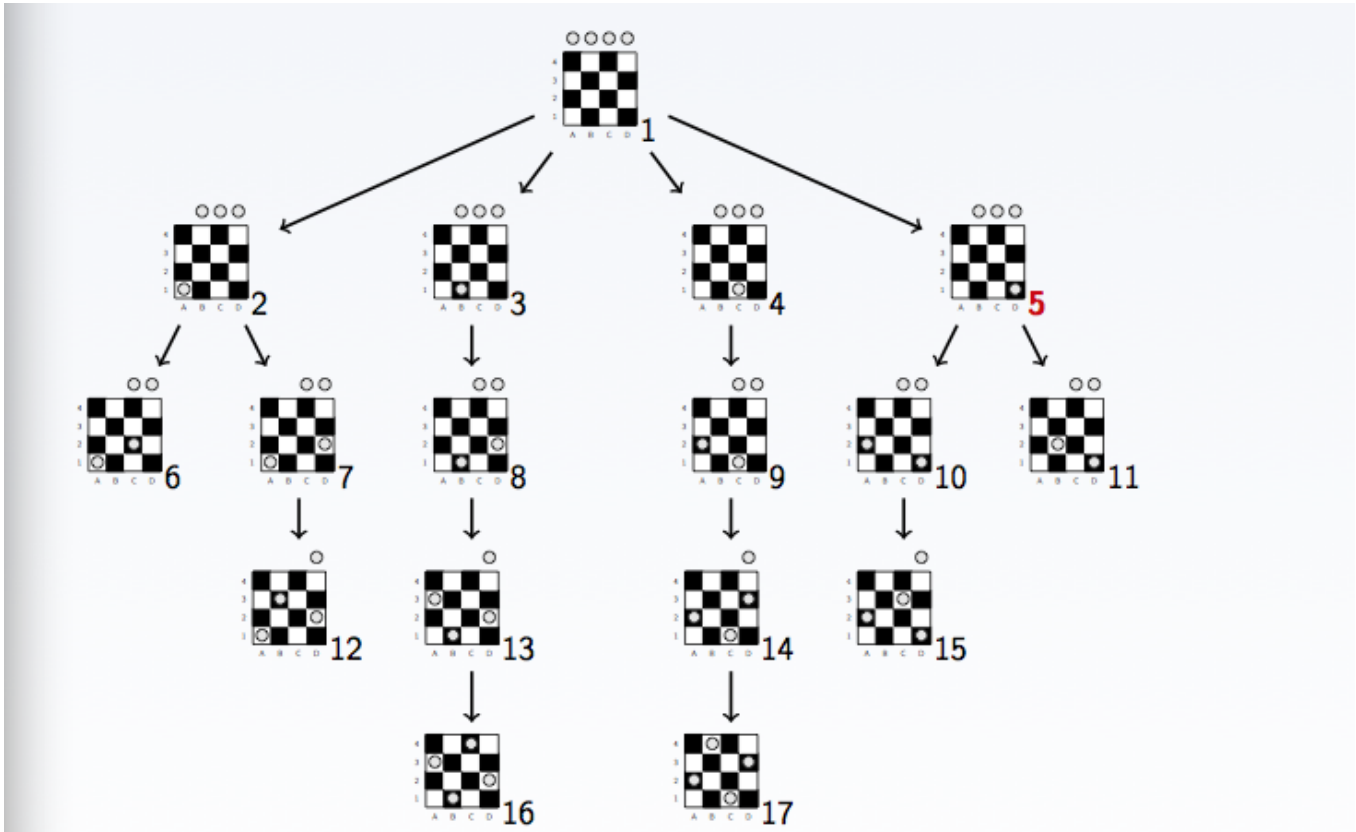


Reihenfolge der Knotenbesuche

$k = 2$

1,2,3,4,5

Beispiel: Iterative Tiefensuche

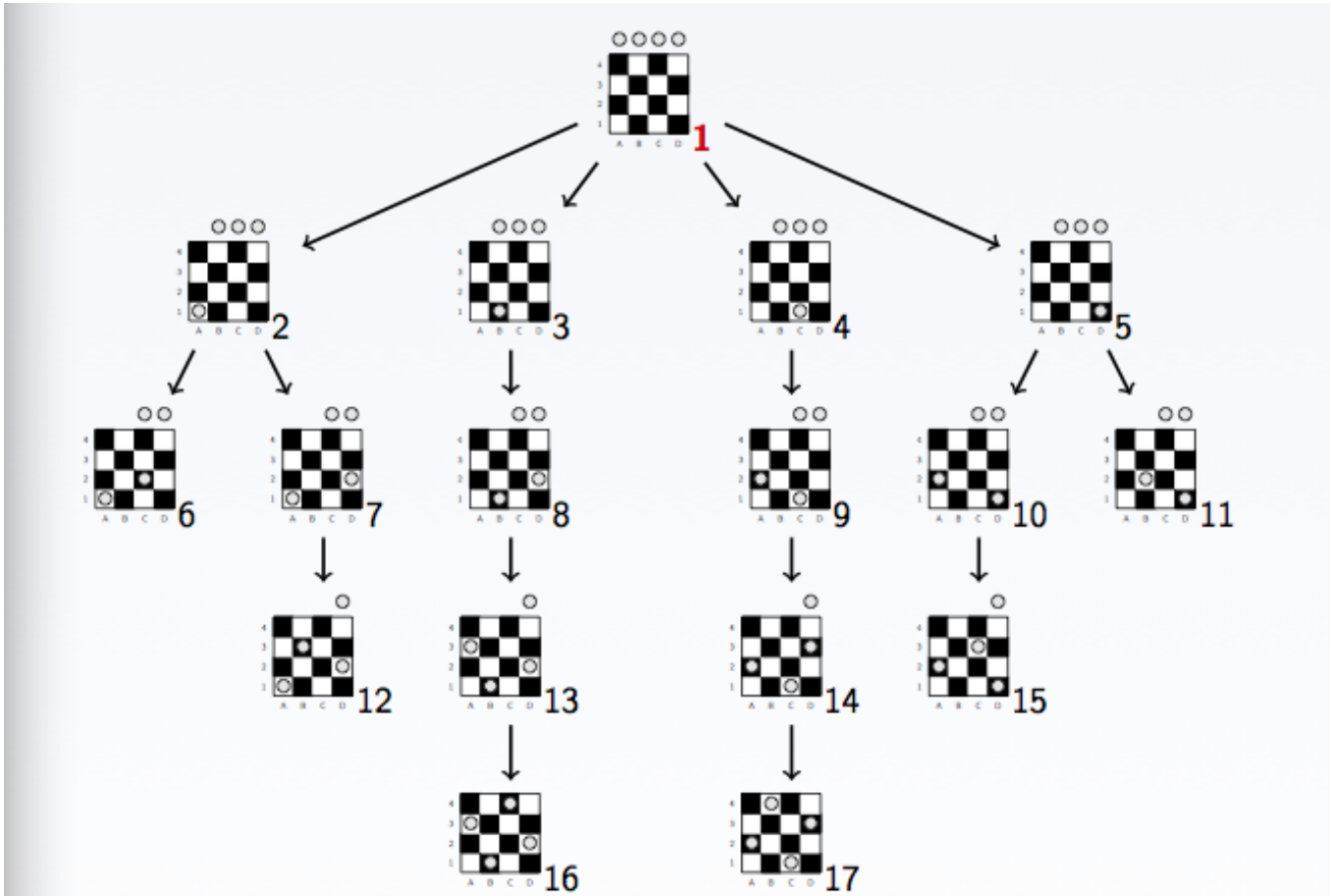


Reihenfolge der Knotenbesuche

$k = 2$

1,2,3,4,5

Beispiel: Iterative Tiefensuche

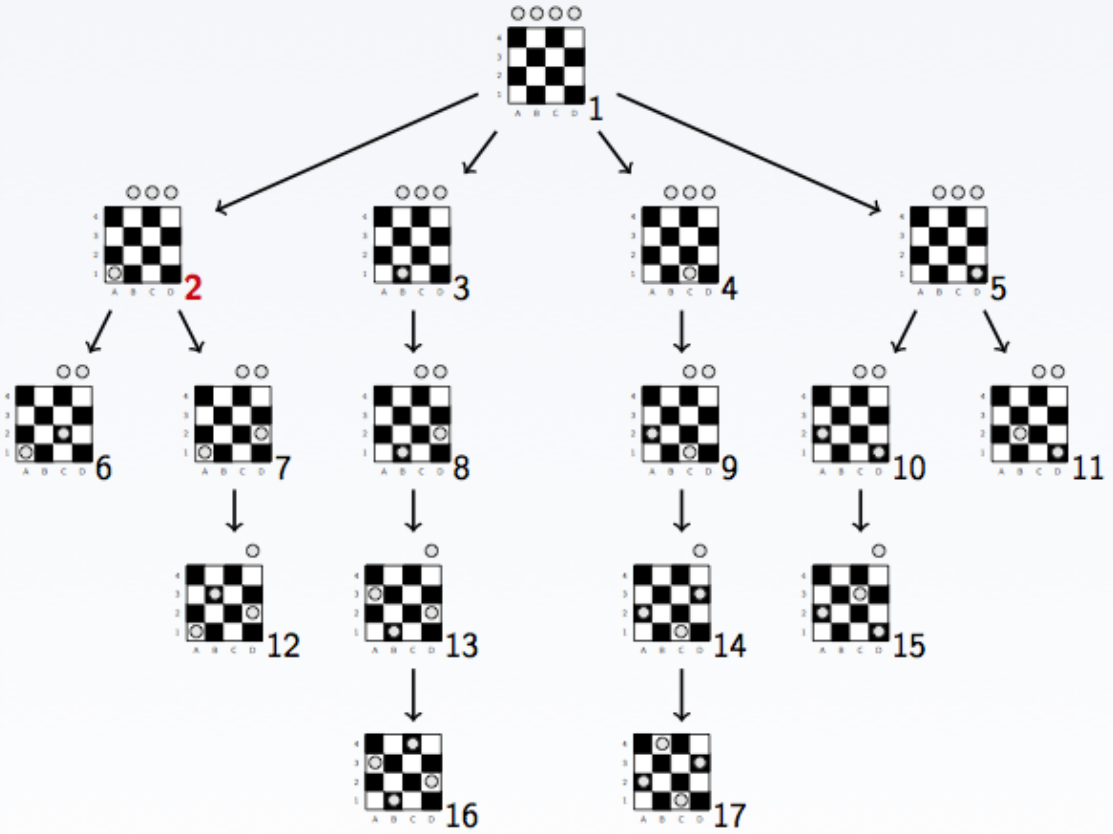


Reihenfolge der Knotenbesuche

$k = 3$

1,2,6,7,3,8,4,9,5,10,11

Beispiel: Iterative Tiefensuche

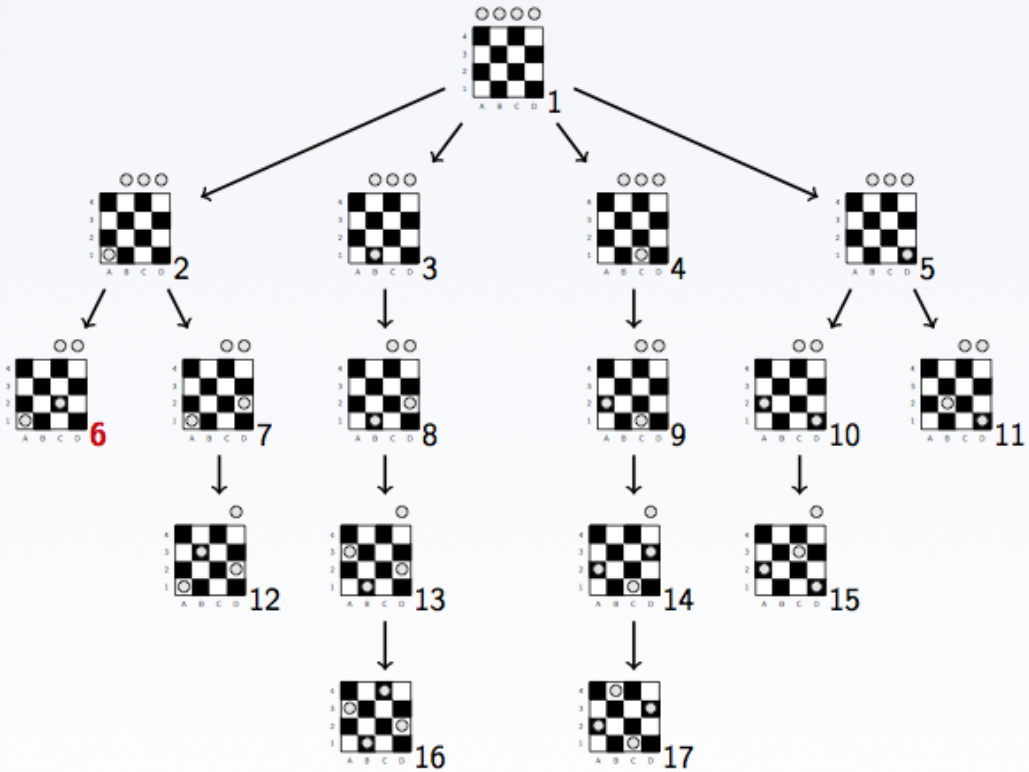


Reihenfolge der Knotenbesuche

$k = 3$

1,2,6,7,3,8,4,9,5,10,11

Beispiel: Iterative Tiefensuche

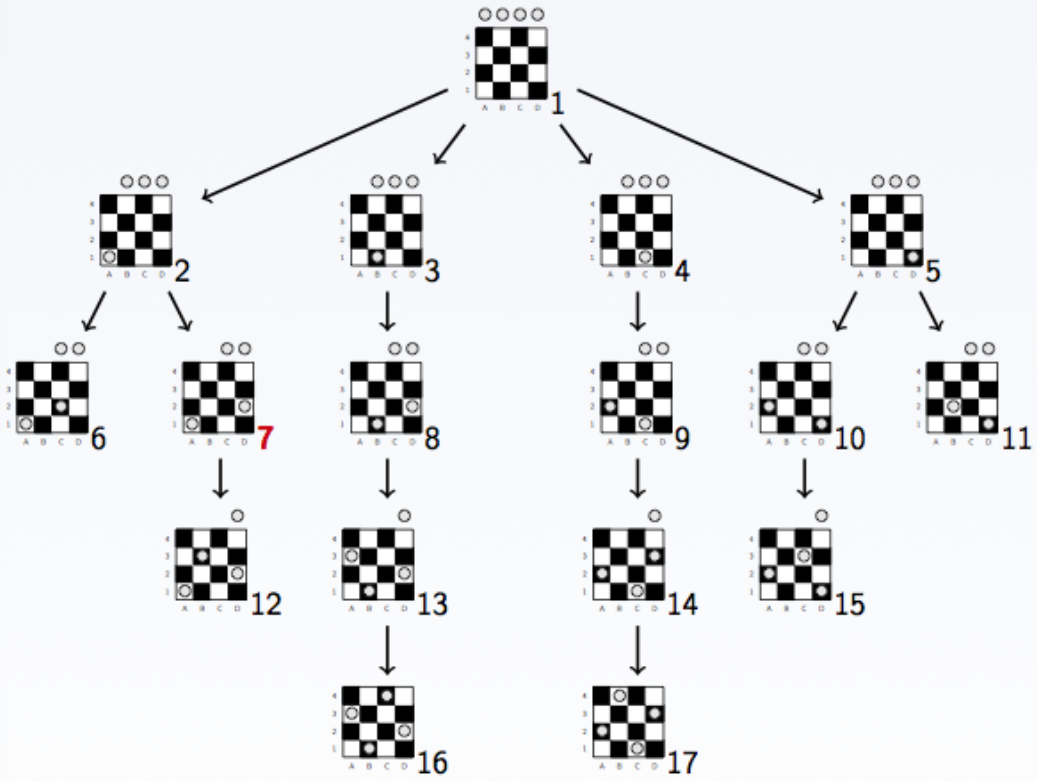


Reihenfolge der Knotenbesuche

$k = 3$

1,2,6,7,3,8,4,9,5,10,11

Beispiel: Iterative Tiefensuche

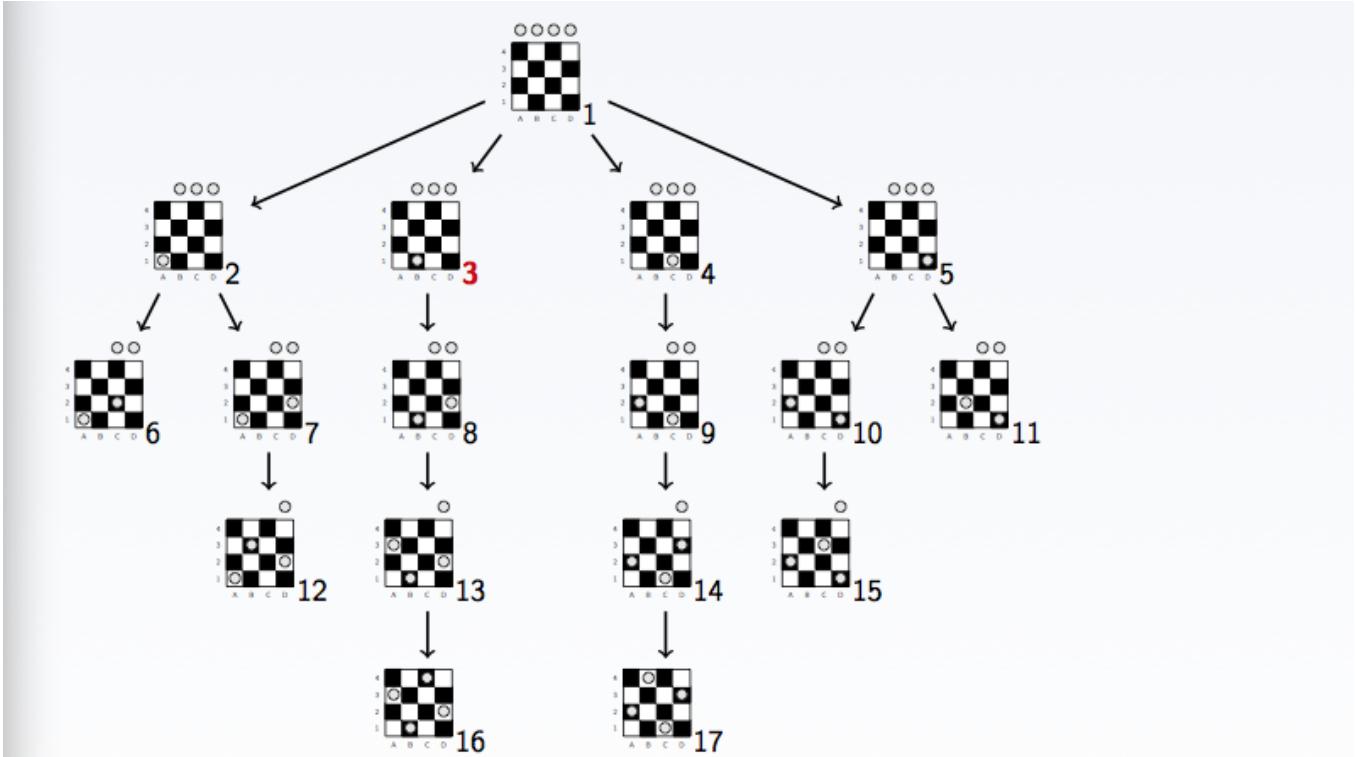


Reihenfolge der Knotenbesuche

$k = 3$

1,2,6,7,3,8,4,9,5,10,11

Beispiel: Iterative Tiefensuche

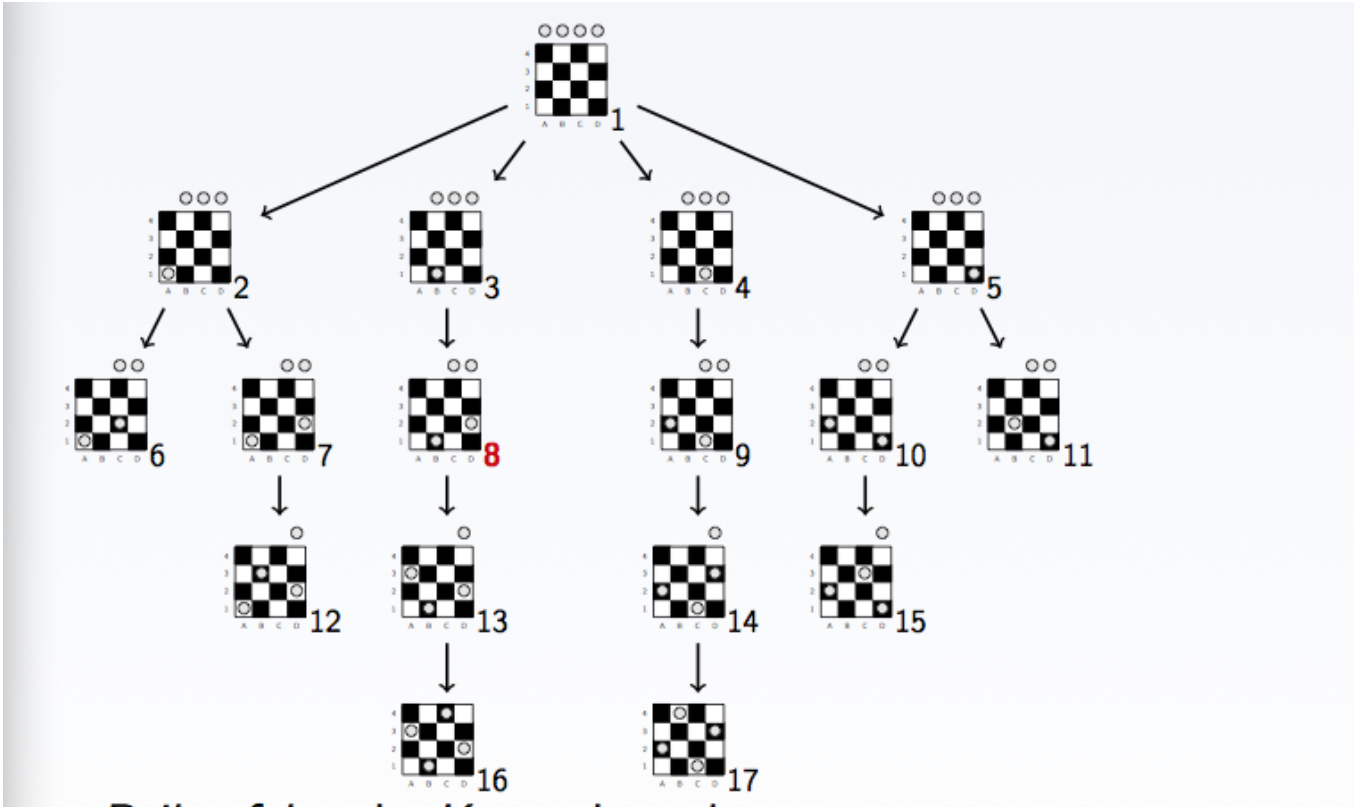


Reihenfolge der Knotenbesuche

$k = 3$

1,2,6,7,3,8,4,9,5,10,11

Beispiel: Iterative Tiefensuche

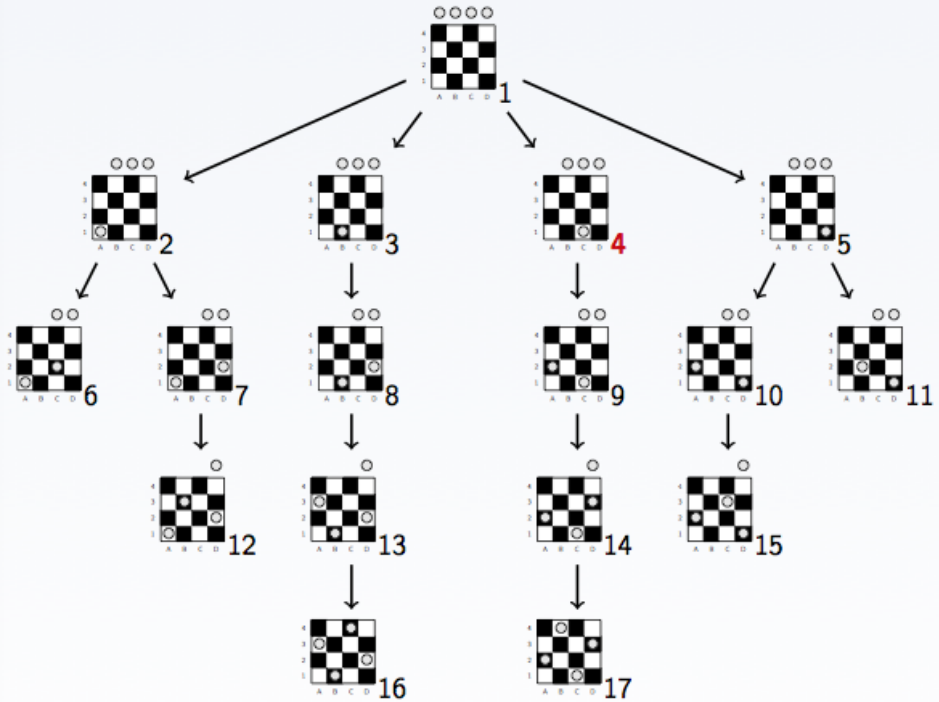


Reihenfolge der Knotenbesuche

$k = 3$

1,2,6,7,3,8,4,9,5,10,11

Beispiel: Iterative Tiefensuche

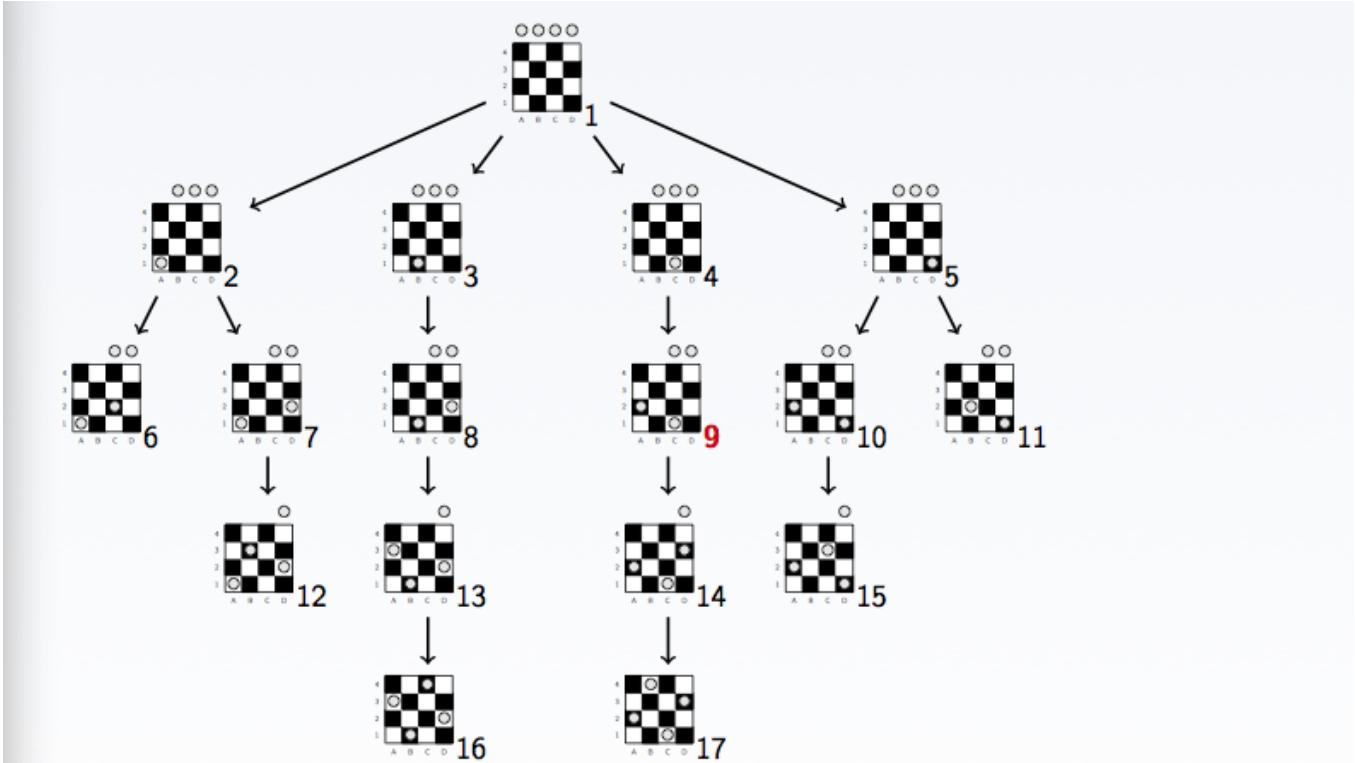


Reihenfolge der Knotenbesuche

$k = 3$

1,2,6,7,3,8,4,9,5,10,11

Beispiel: Iterative Tiefensuche

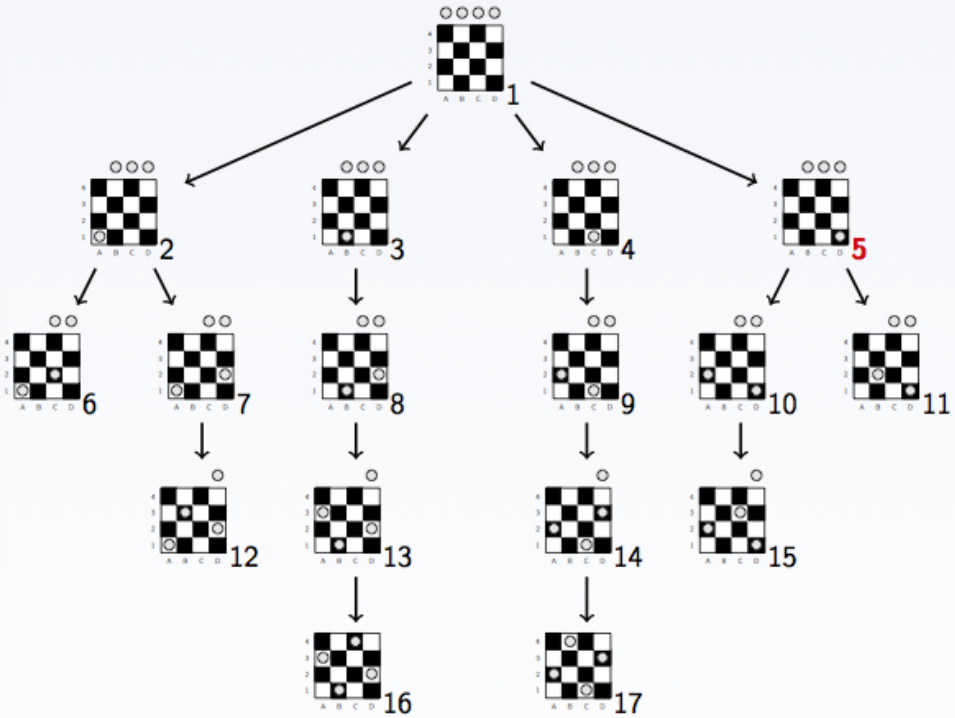


Reihenfolge der Knotenbesuche

$k = 3$

1,2,6,7,3,8,4,9,5,10,11

Beispiel: Iterative Tiefensuche

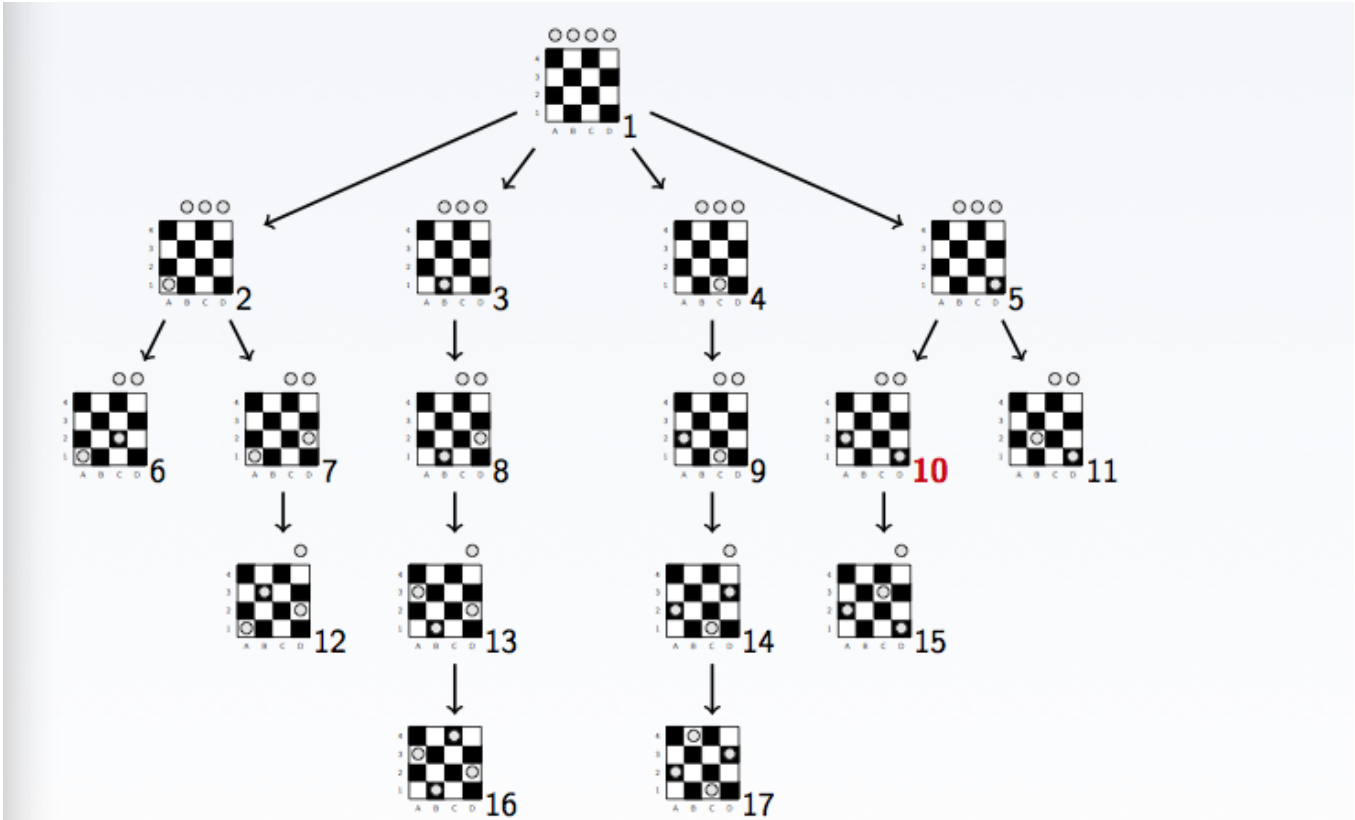


Reihenfolge der Knotenbesuche

$k = 3$

1,2,6,7,3,8,4,9,5,10,11

Beispiel: Iterative Tiefensuche

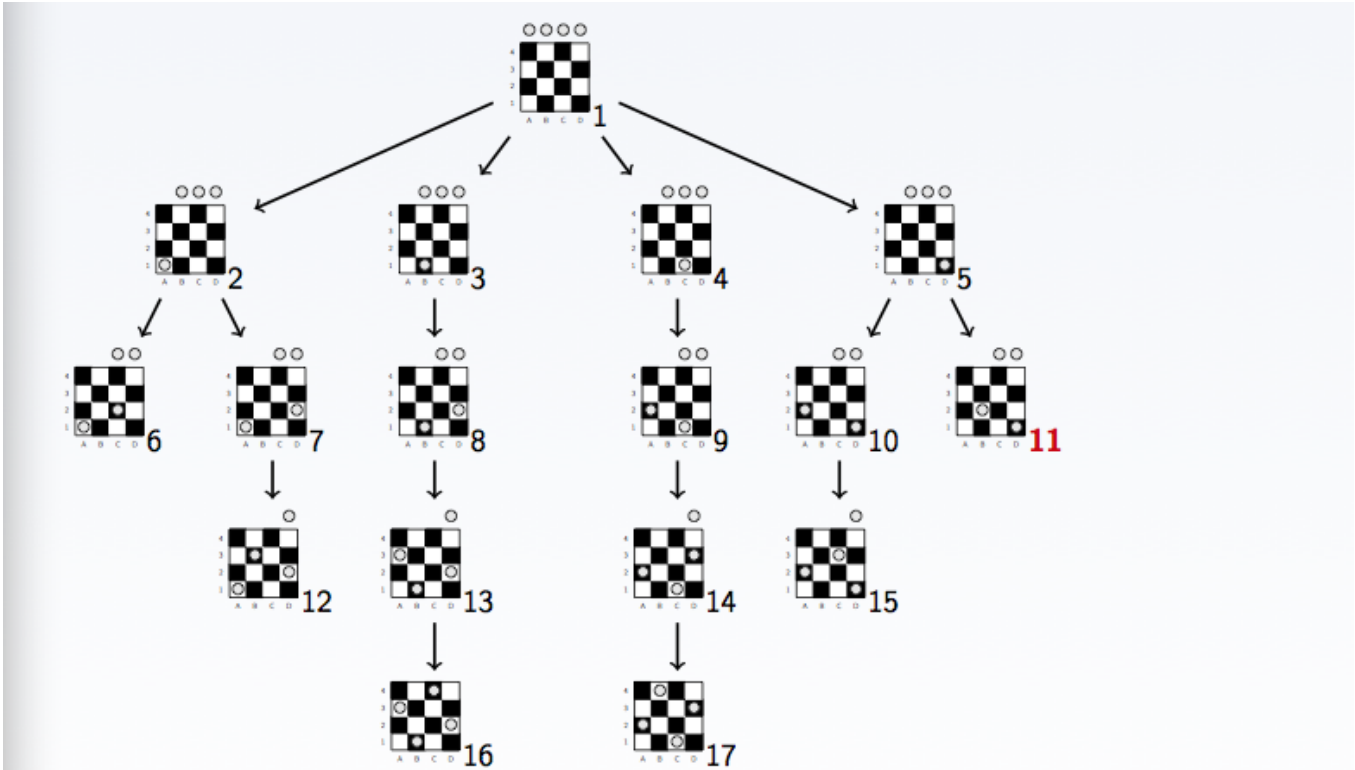


Reihenfolge der Knotenbesuche

$k = 3$

1,2,6,7,3,8,4,9,5,10,11

Beispiel: Iterative Tiefensuche

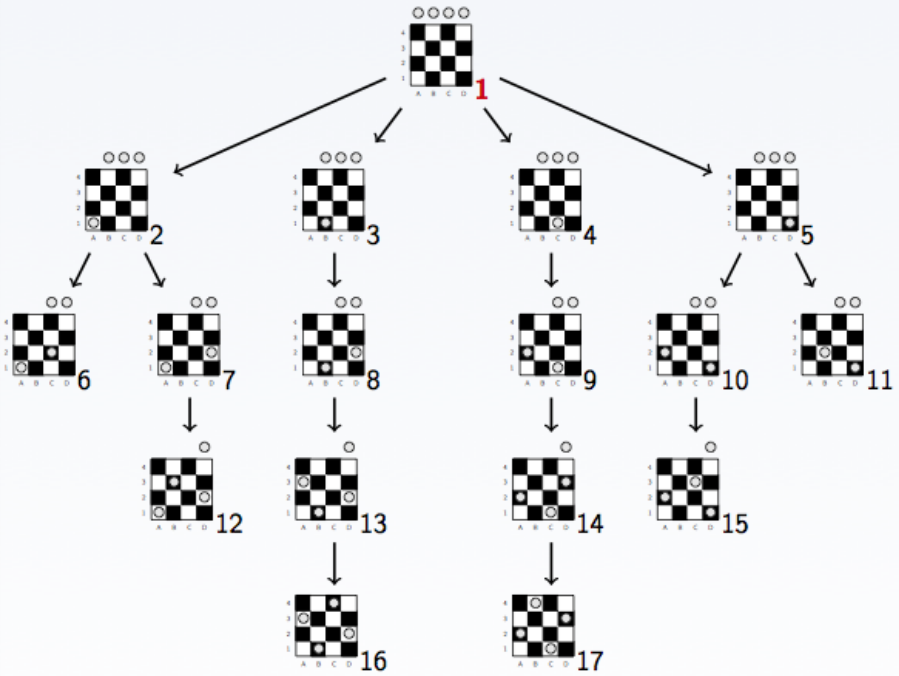


Reihenfolge der Knotenbesuche

$k = 3$

1,2,6,7,3,8,4,9,5,10,11

Beispiel: Iterative Tiefensuche

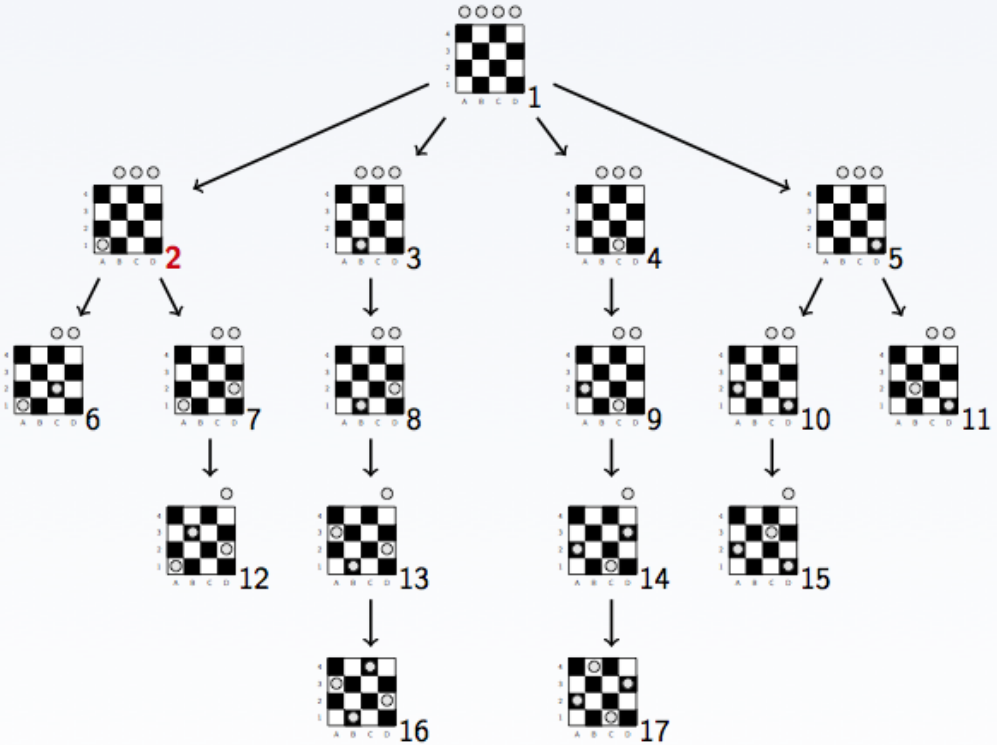


Reihenfolge der Knotenbesuche

$k = 4$

1,2,6,7,12,3,8,13,4,9,14,5,10,15,11

Beispiel: Iterative Tiefensuche

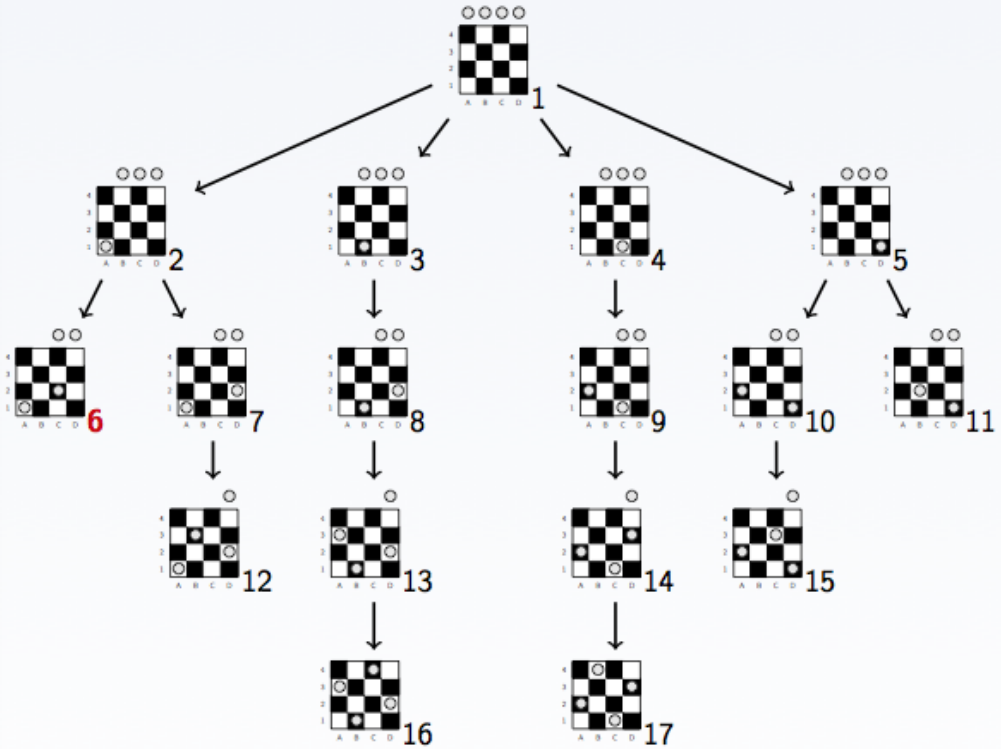


Reihenfolge der Knotenbesuche

$k = 4$

1,2,6,7,12,3,8,13,4,9,14,5,10,15,11

Beispiel: Iterative Tiefensuche

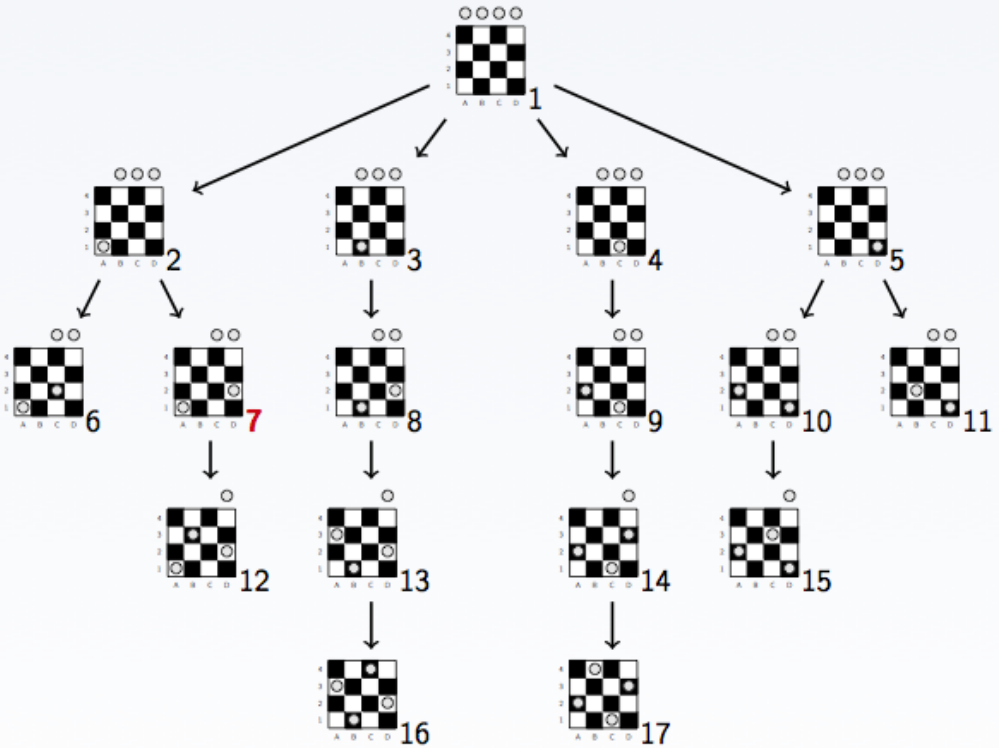


Reihenfolge der Knotenbesuche

$k = 4$

1,2,6,7,12,3,8,13,4,9,14,5,10,15,11

Beispiel: Iterative Tiefensuche

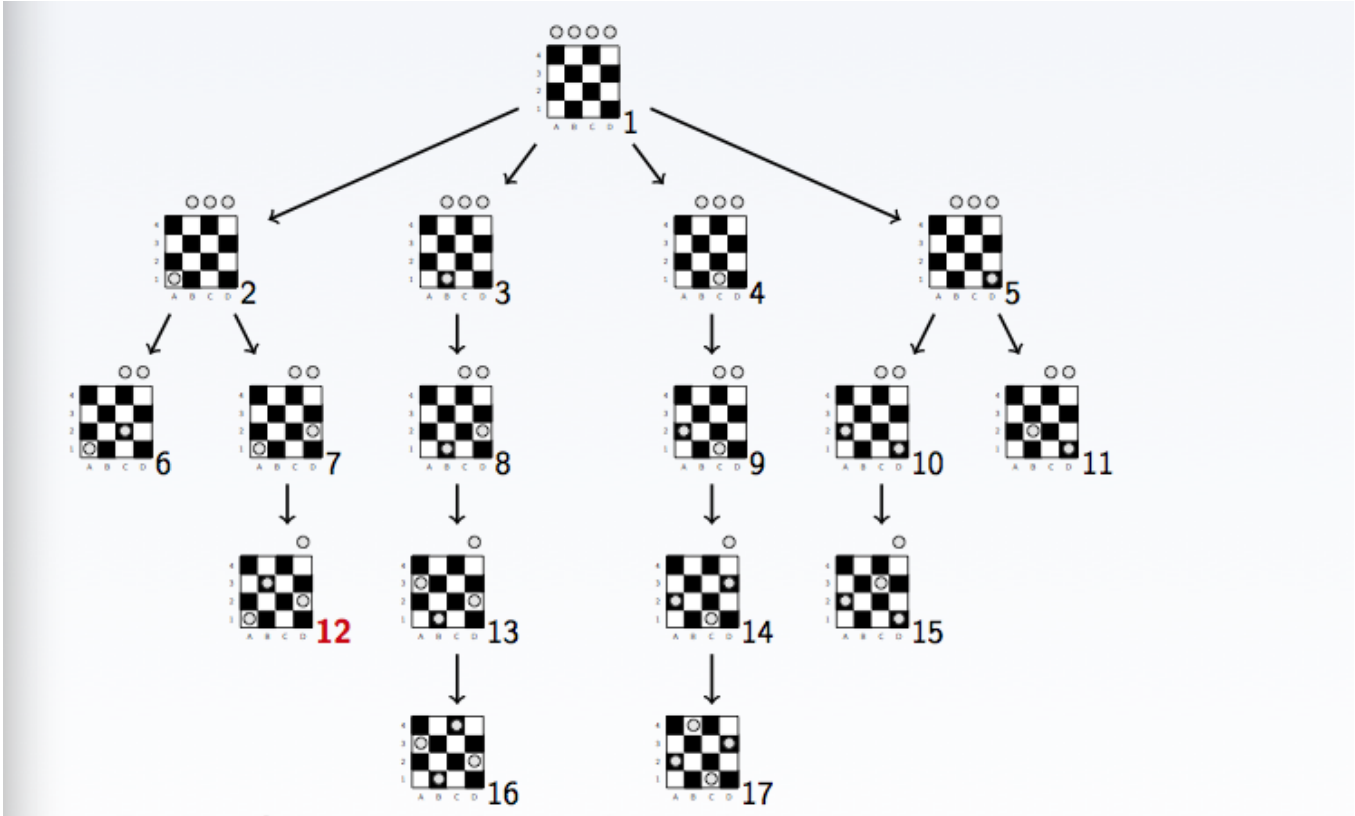


Reihenfolge der Knotenbesuche

$k = 4$

1,2,6,7,12,3,8,13,4,9,14,5,10,15,11

Beispiel: Iterative Tiefensuche

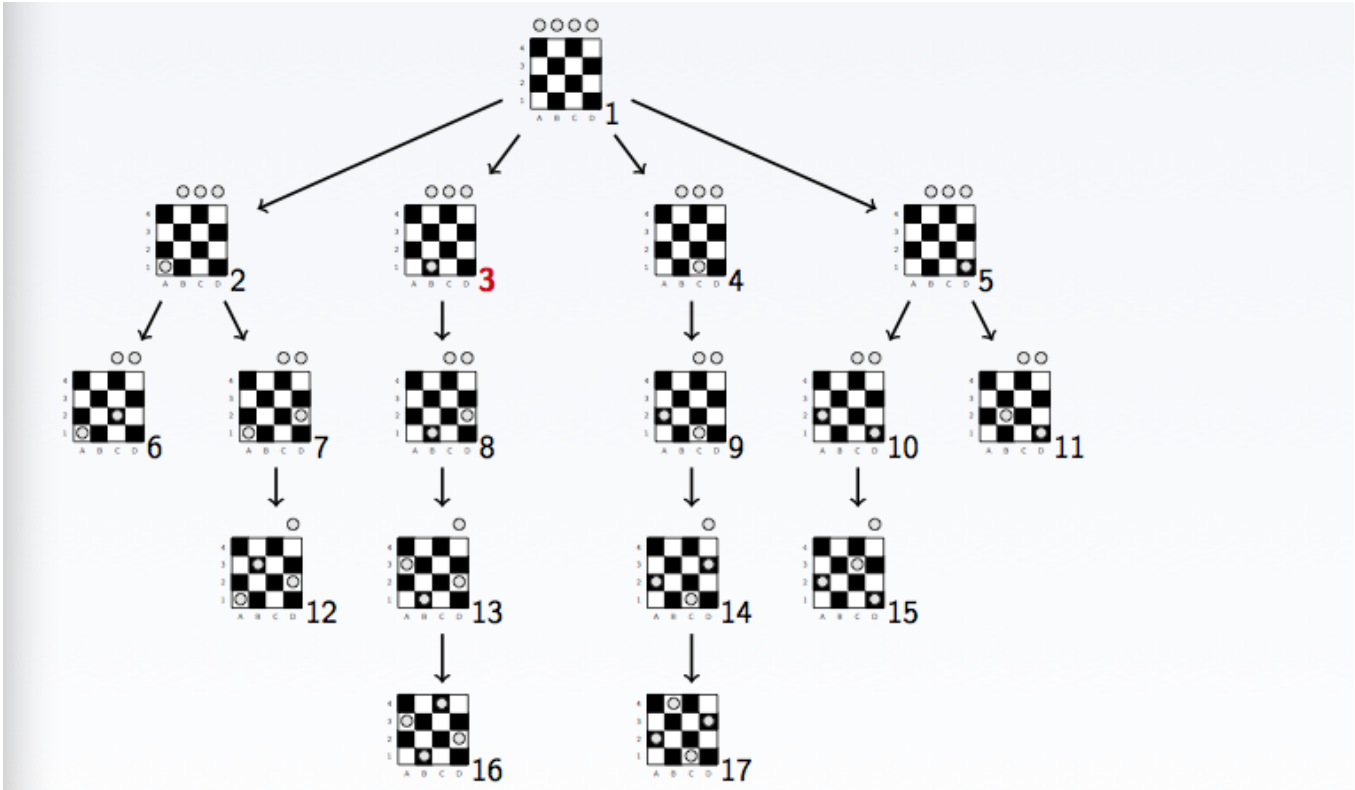


Reihenfolge der Knotenbesuche

$k = 4$

1,2,6,7,12,3,8,13,4,9,14,5,10,15,11

Beispiel: Iterative Tiefensuche

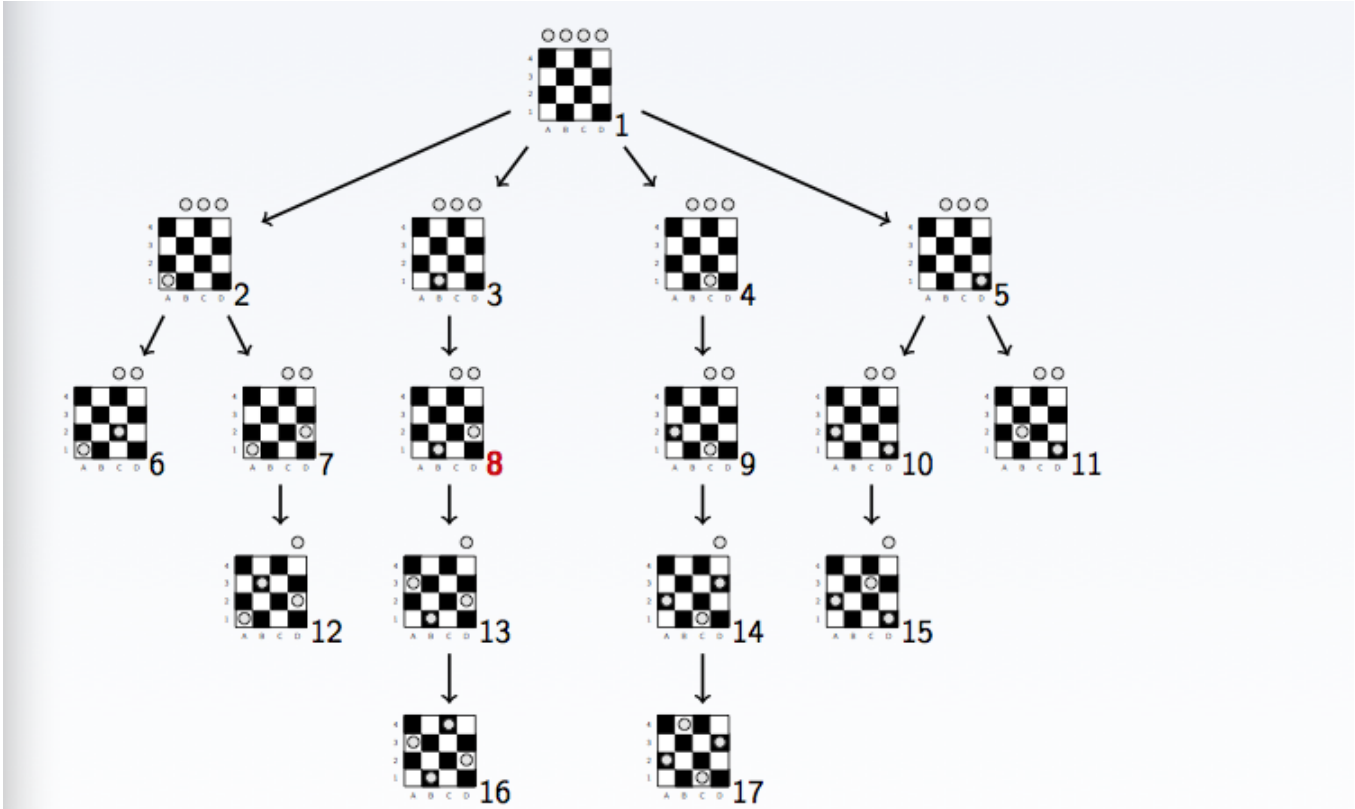


Reihenfolge der Knotenbesuche

$k = 4$

1,2,6,7,12,3,8,13,4,9,14,5,10,15,11

Beispiel: Iterative Tiefensuche

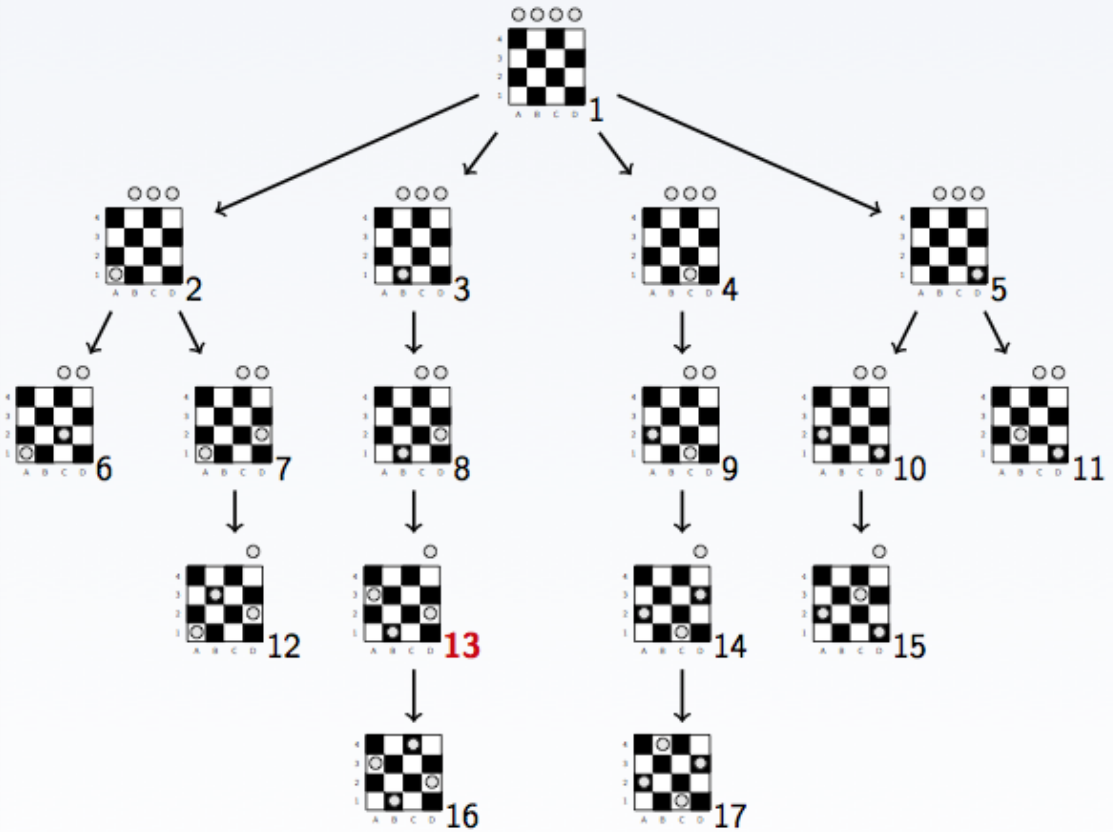


Reihenfolge der Knotenbesuche

$k = 4$

1,2,6,7,12,3,8,13,4,9,14,5,10,15,11

Beispiel: Iterative Tiefensuche

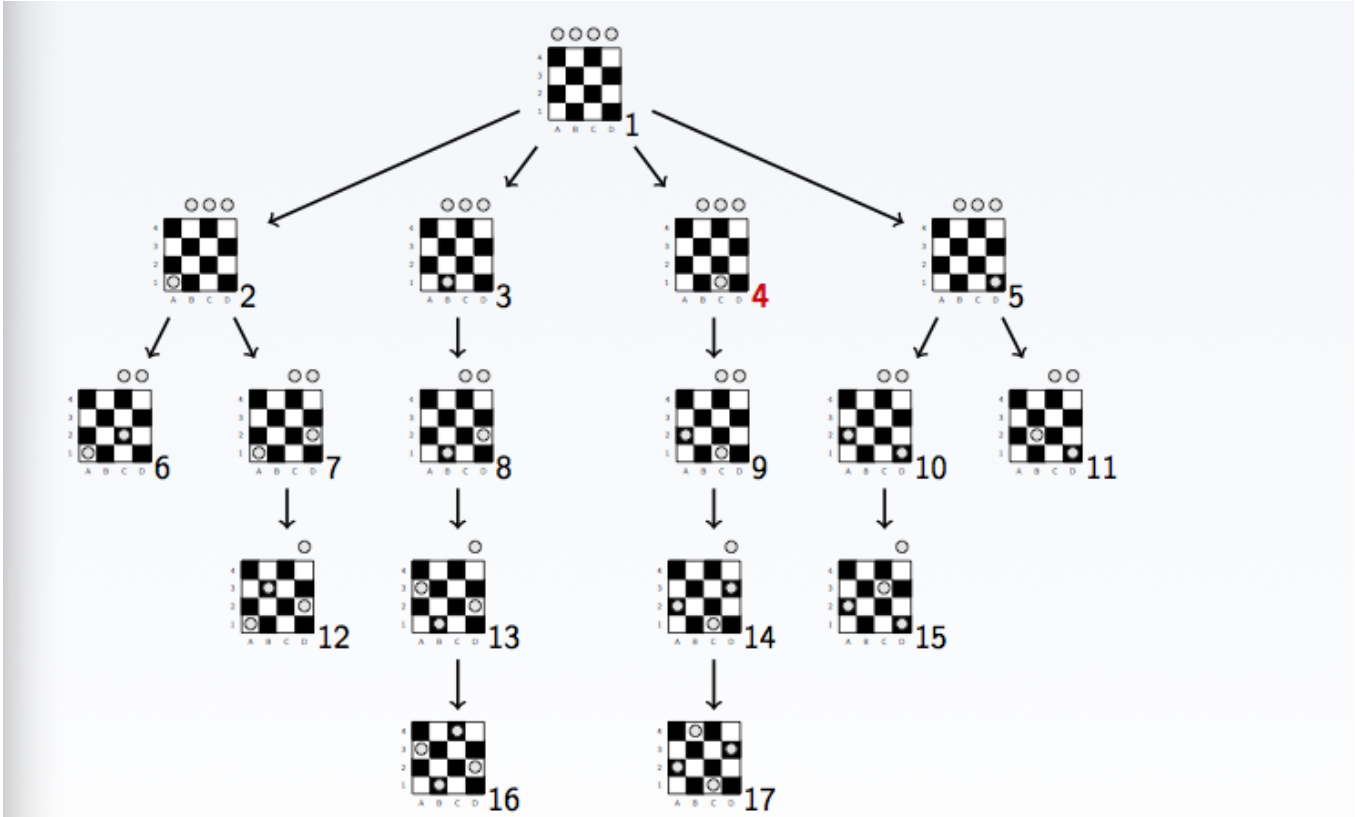


Reihenfolge der Knotenbesuche

$k = 4$

1,2,6,7,12,3,8,13,4,9,14,5,10,15,11

Beispiel: Iterative Tiefensuche

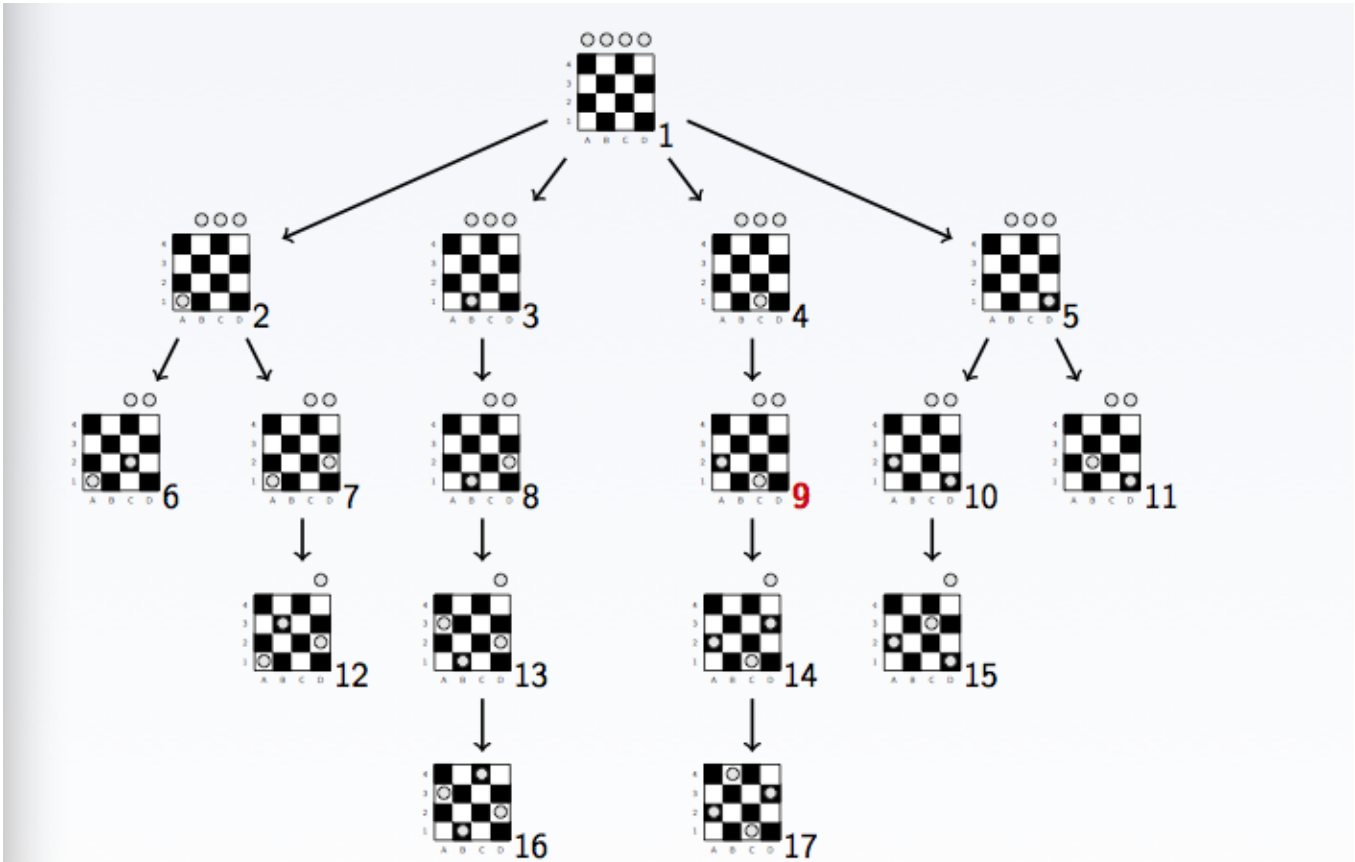


Reihenfolge der Knotenbesuche

$k = 4$

1,2,6,7,12,3,8,13,4,9,14,5,10,15,11

Beispiel: Iterative Tiefensuche

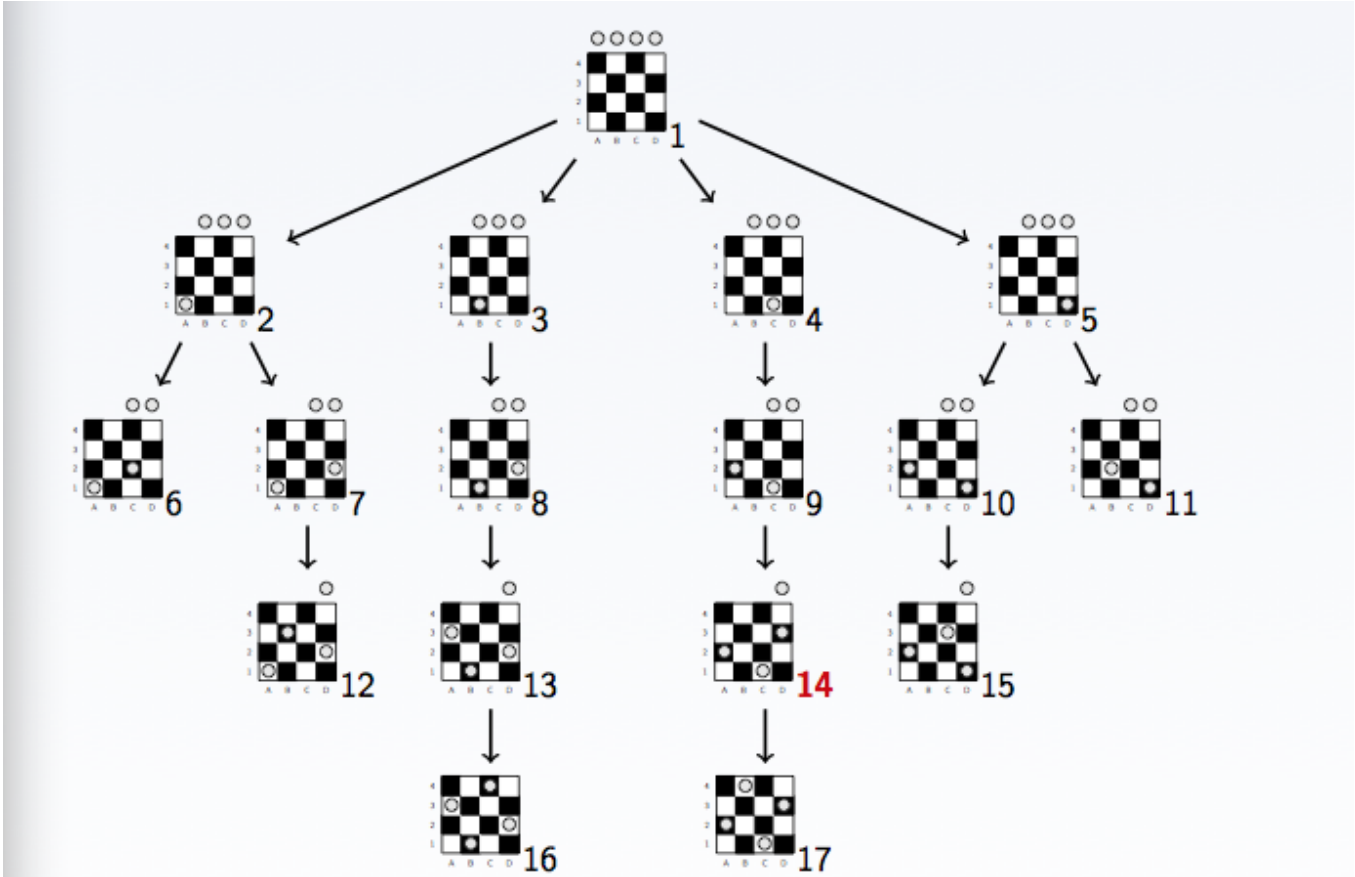


Reihenfolge der Knotenbesuche

$k = 4$

1,2,6,7,12,3,8,13,4,9,14,5,10,15,11

Beispiel: Iterative Tiefensuche

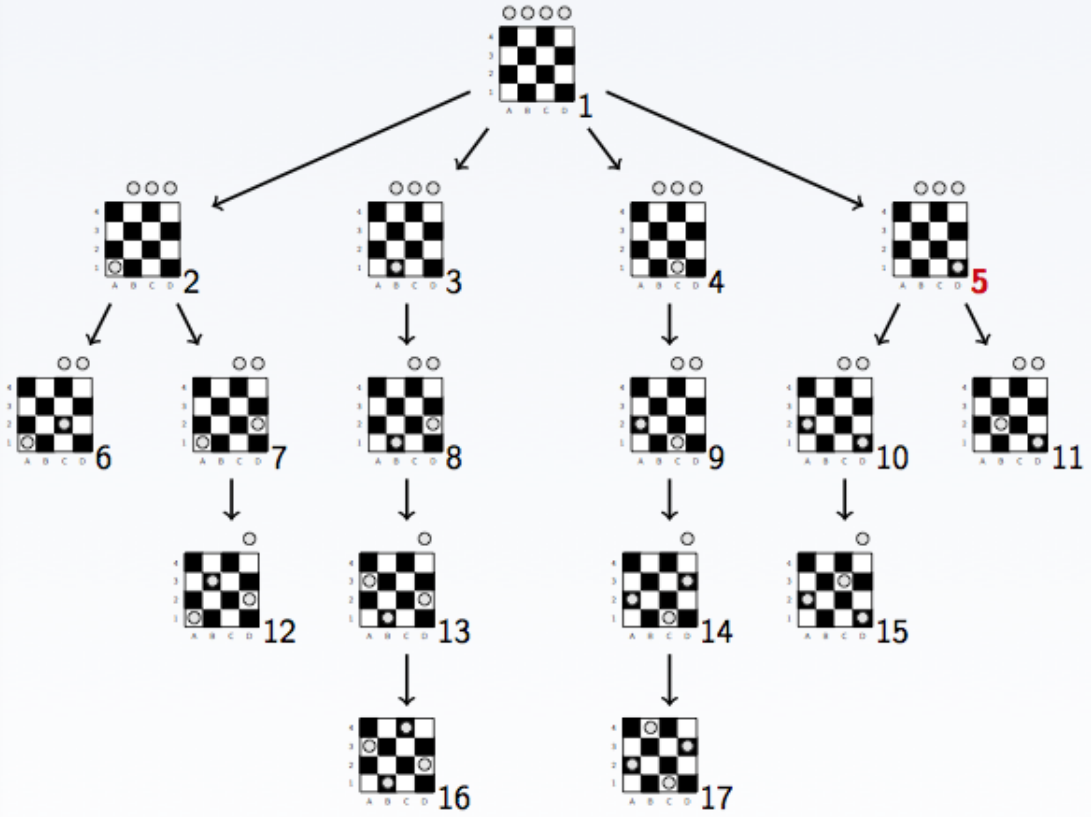


Reihenfolge der Knotenbesuche

$k = 4$

1,2,6,7,12,3,8,13,4,9,14,5,10,15,11

Beispiel: Iterative Tiefensuche

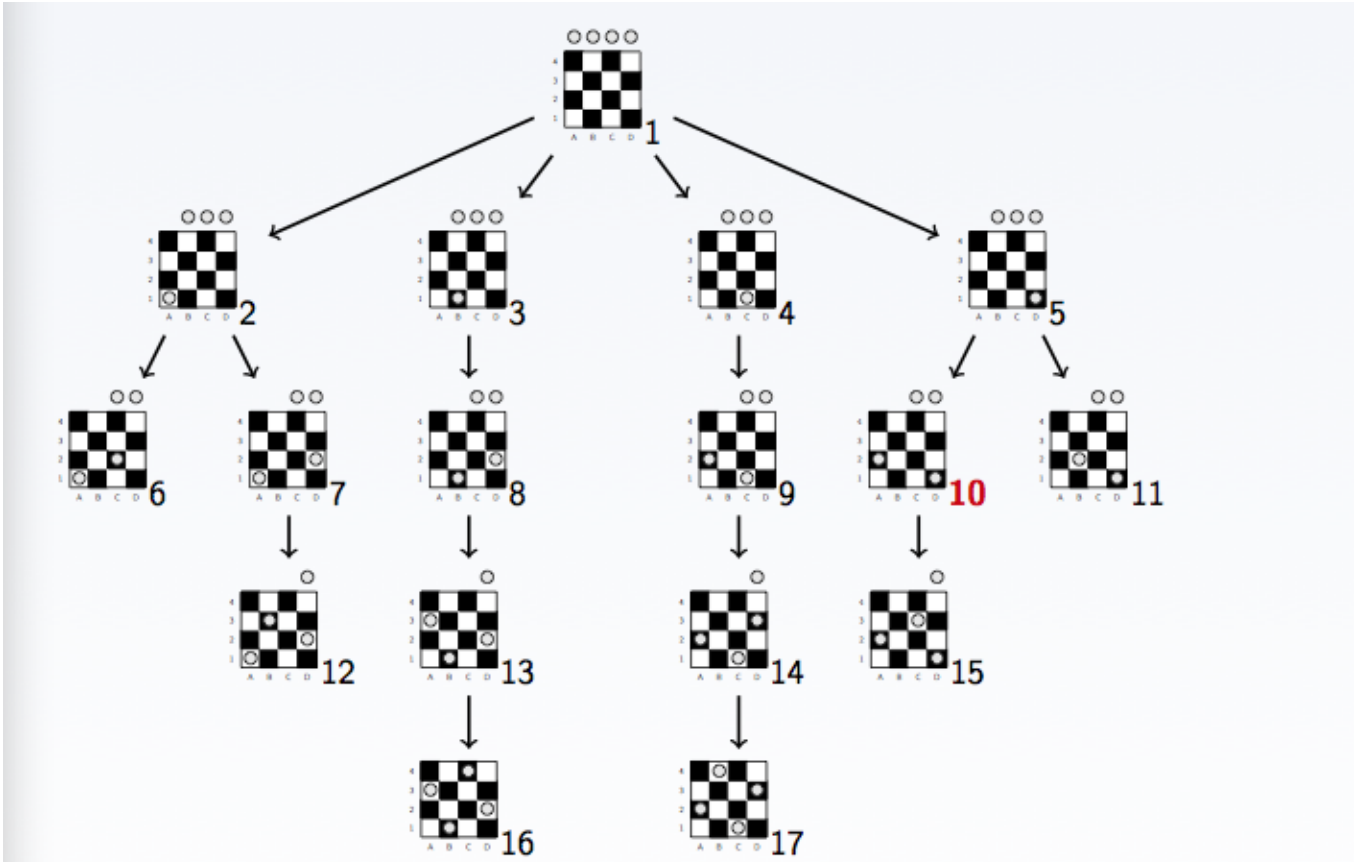


Reihenfolge der Knotenbesuche

$k = 4$

1,2,6,7,12,3,8,13,4,9,14,5,10,15,11

Beispiel: Iterative Tiefensuche

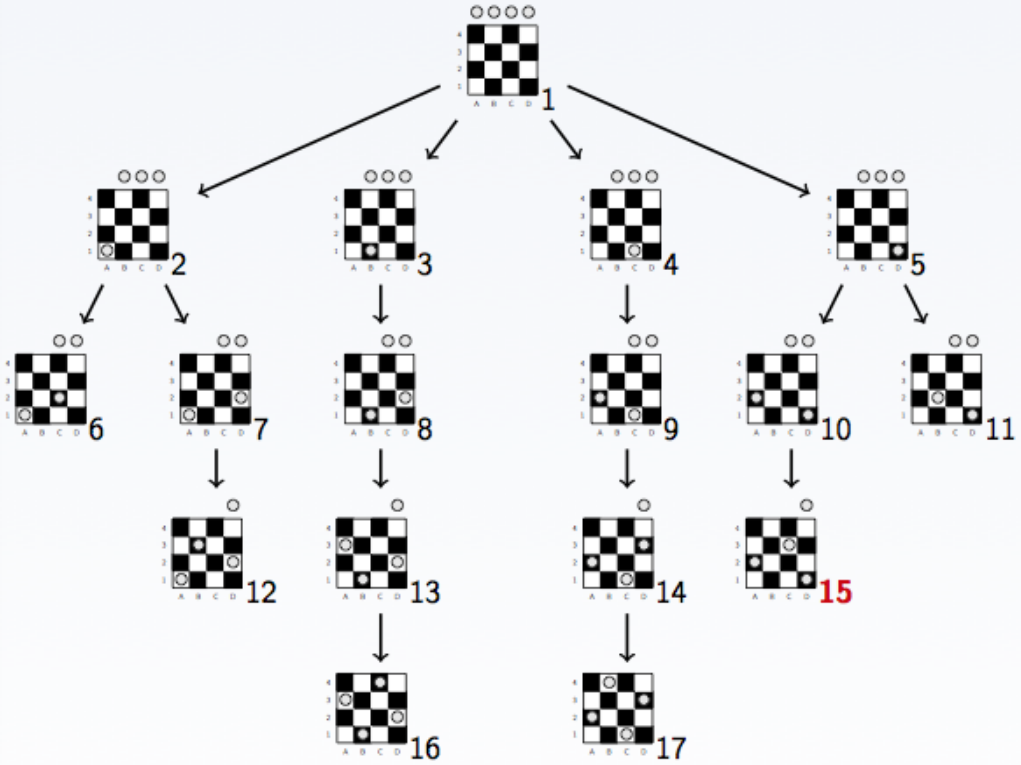


Reihenfolge der Knotenbesuche

$k = 4$

1,2,6,7,12,3,8,13,4,9,14,5,10,15,11

Beispiel: Iterative Tiefensuche

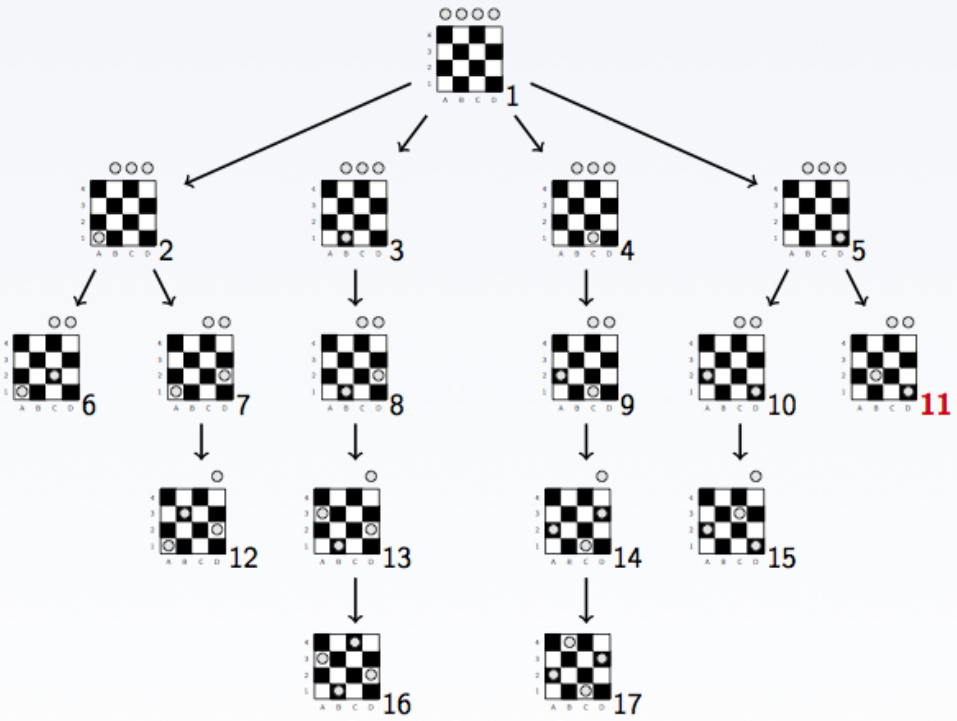


Reihenfolge der Knotenbesuche

$k = 4$

1,2,6,7,12,3,8,13,4,9,14,5,10,15,11

Beispiel: Iterative Tiefensuche

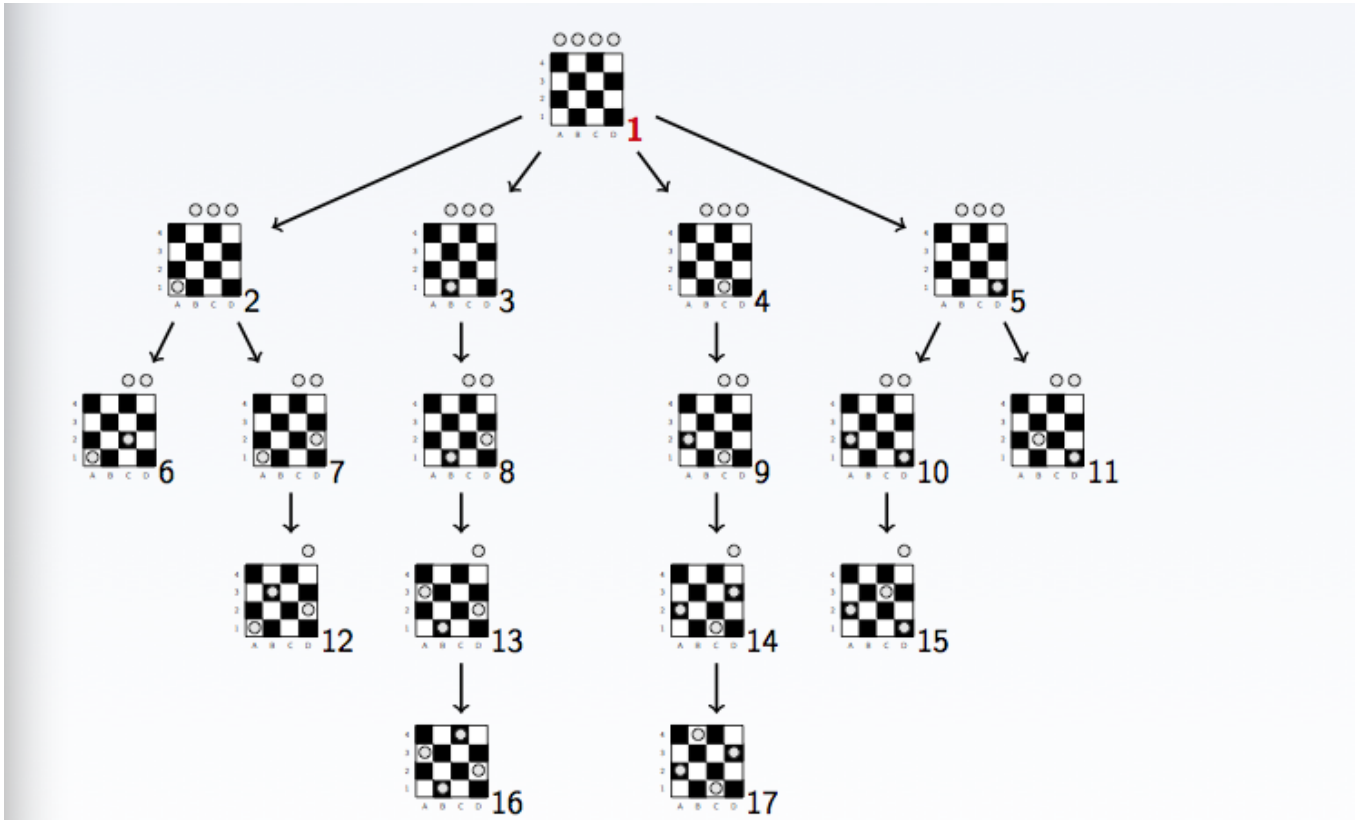


Reihenfolge der Knotenbesuche

$k = 4$

1,2,6,7,12,3,8,13,4,9,14,5,10,15,11

Beispiel: Iterative Tiefensuche

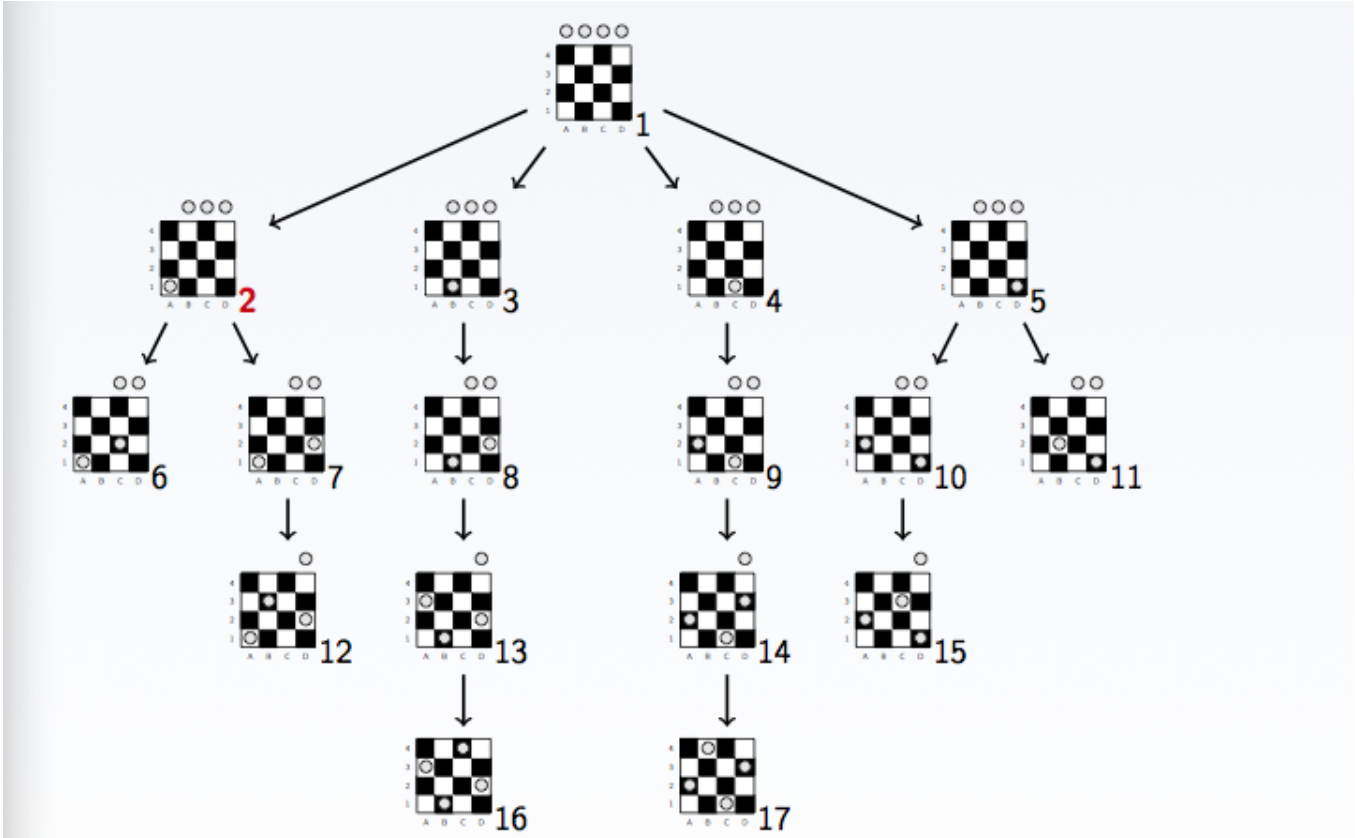


Reihenfolge der Knotenbesuche

$k = 5$

1,2,6,7,12,3,8,13,16 $\Rightarrow$ Ziel

Beispiel: Iterative Tiefensuche

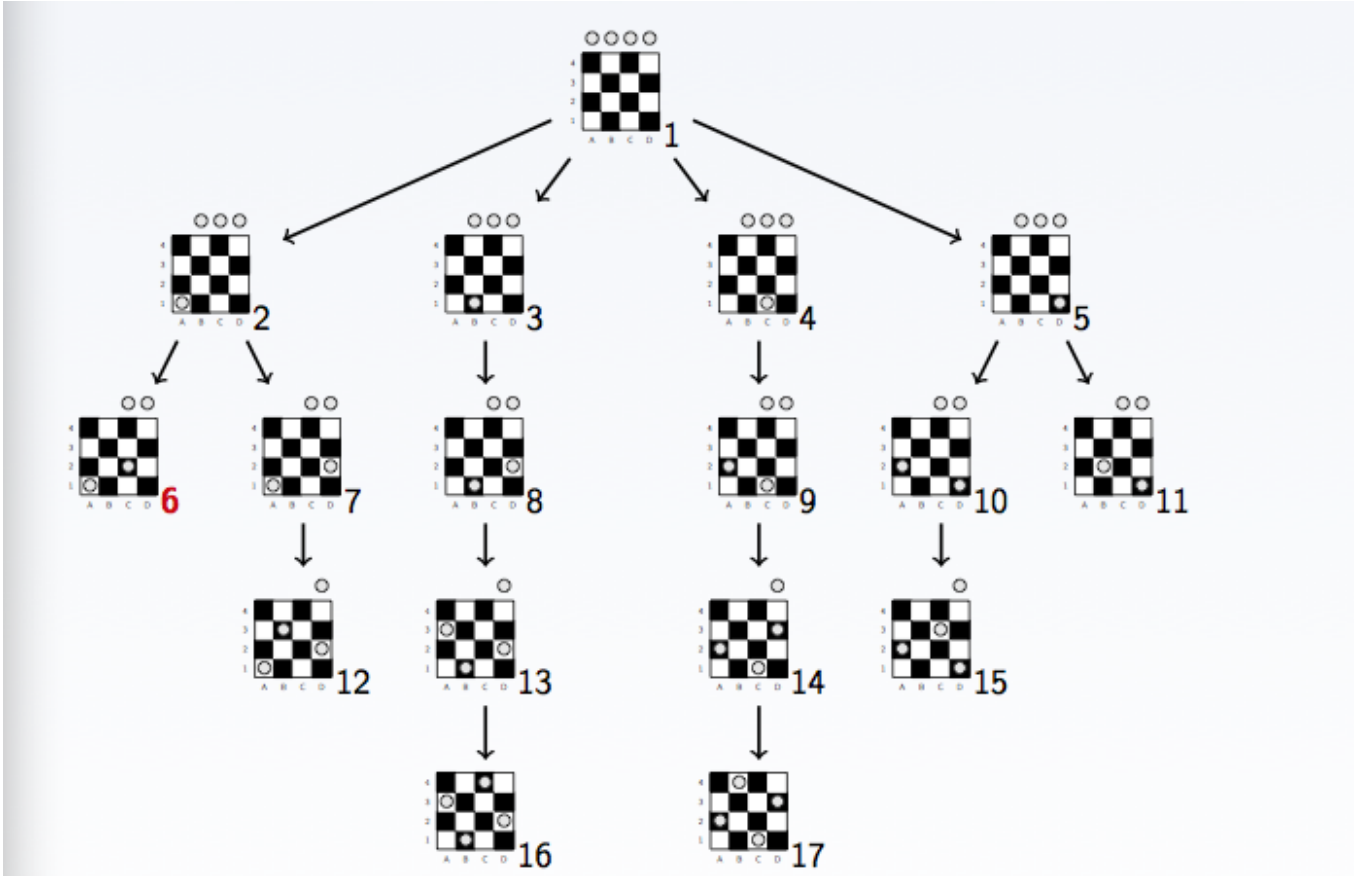


Reihenfolge der Knotenbesuche

$k = 5$

1,2,6,7,12,3,8,13,16 $\Rightarrow$ Ziel

Beispiel: Iterative Tiefensuche

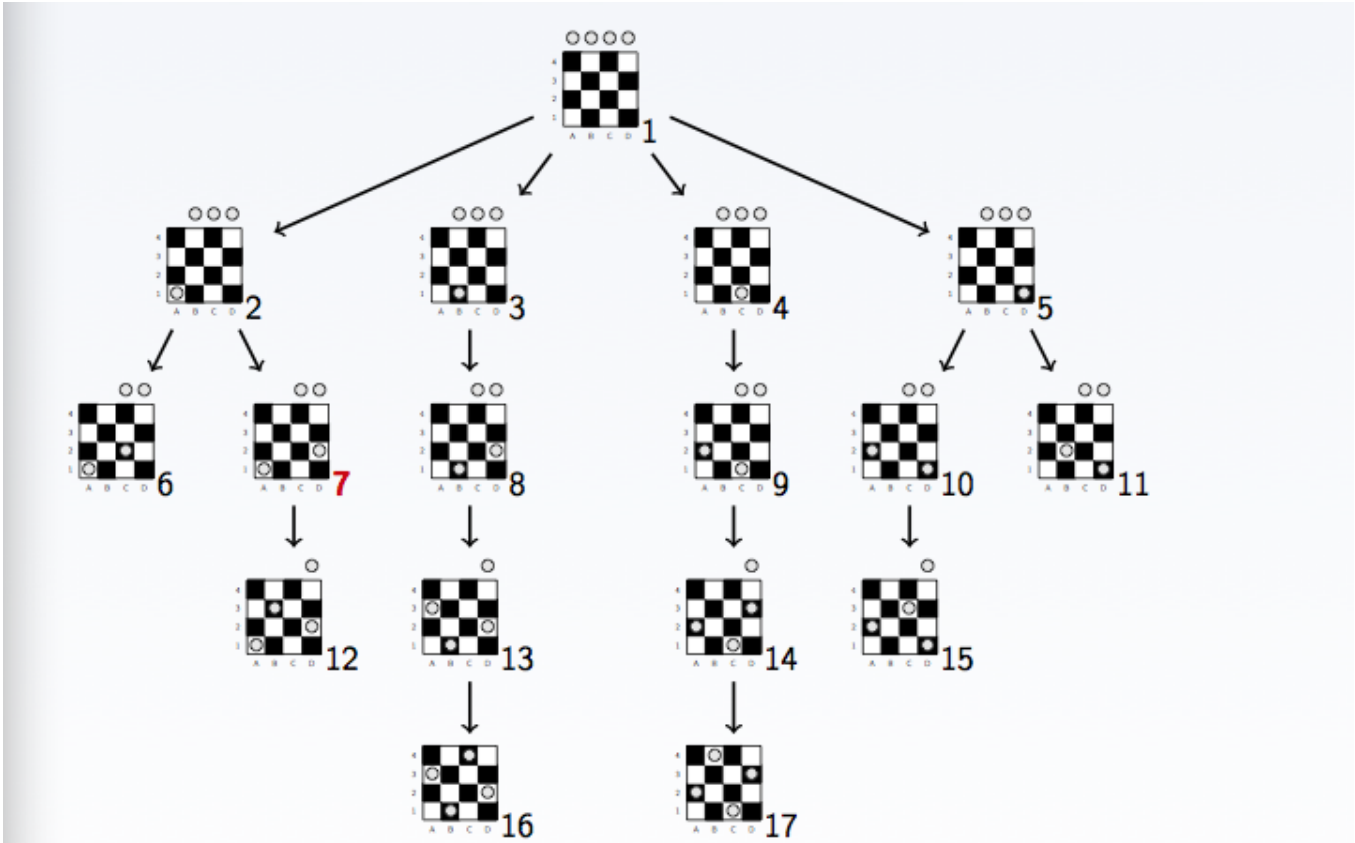


Reihenfolge der Knotenbesuche

$k = 5$

1,2,6,7,12,3,8,13,16 $\Rightarrow$ Ziel

Beispiel: Iterative Tiefensuche

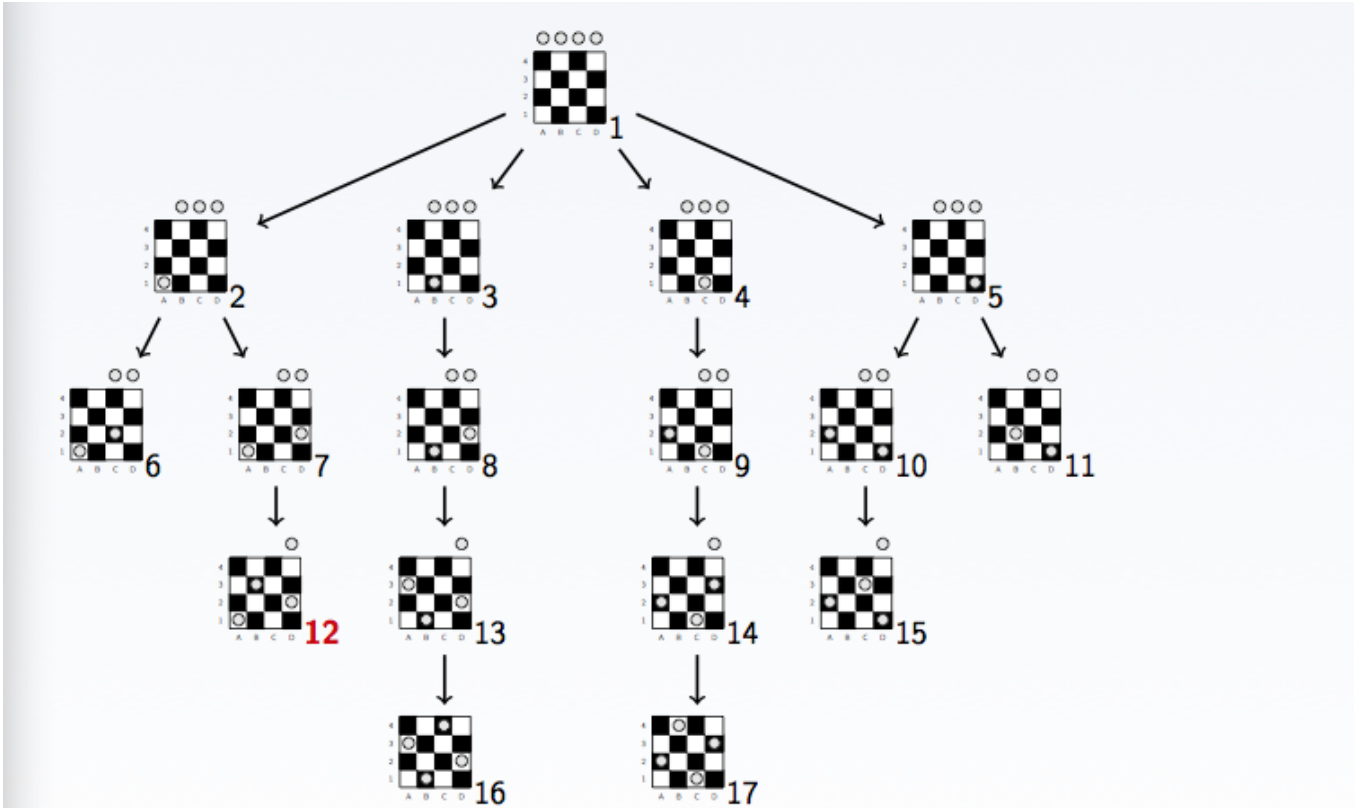


Reihenfolge der Knotenbesuche

$k = 5$

1,2,6,7,12,3,8,13,16 $\Rightarrow$ Ziel

Beispiel: Iterative Tiefensuche

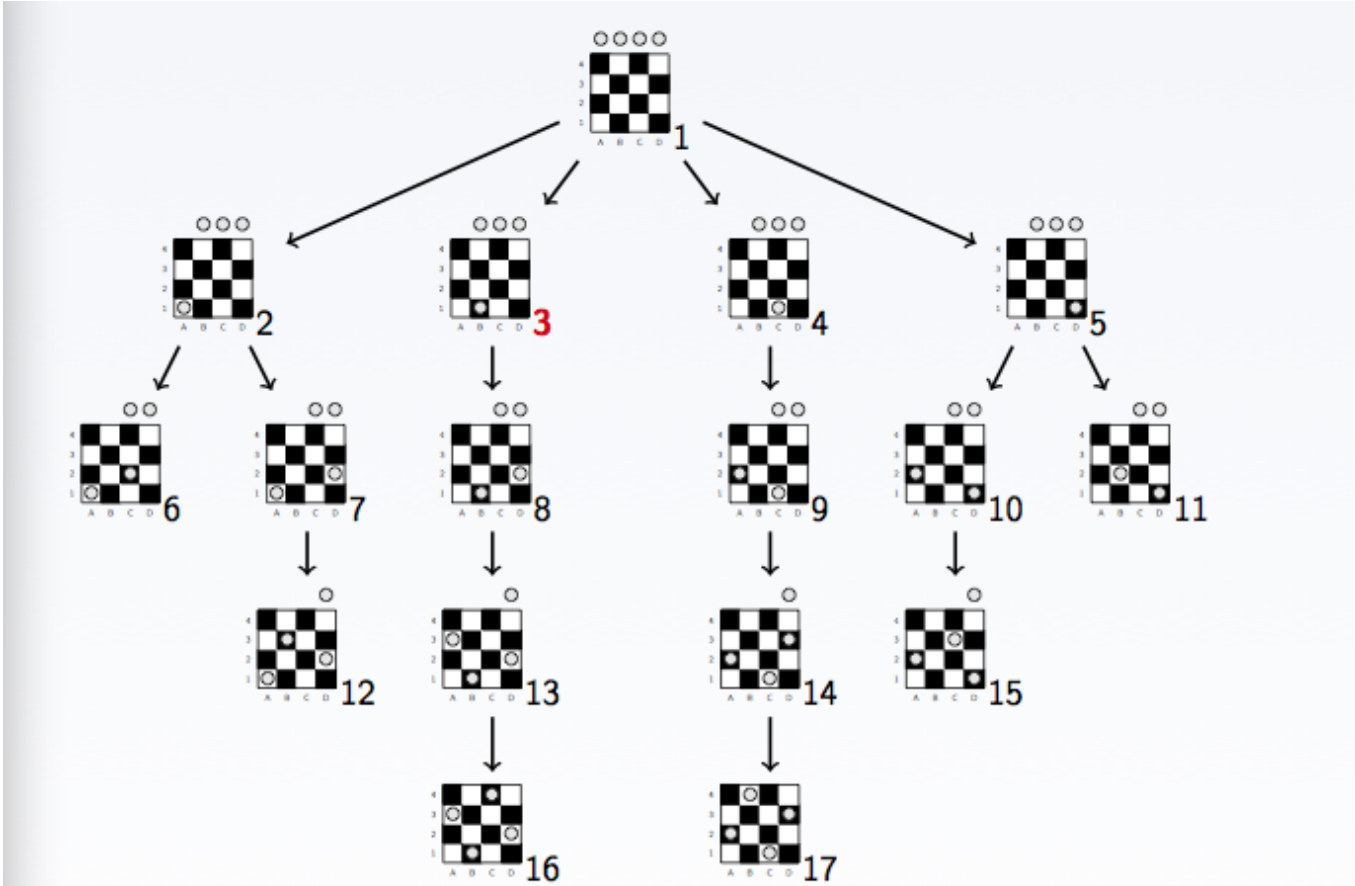


Reihenfolge der Knotenbesuche

$k = 5$

1,2,6,7,12,3,8,13,16 $\Rightarrow$ Ziel

Beispiel: Iterative Tiefensuche

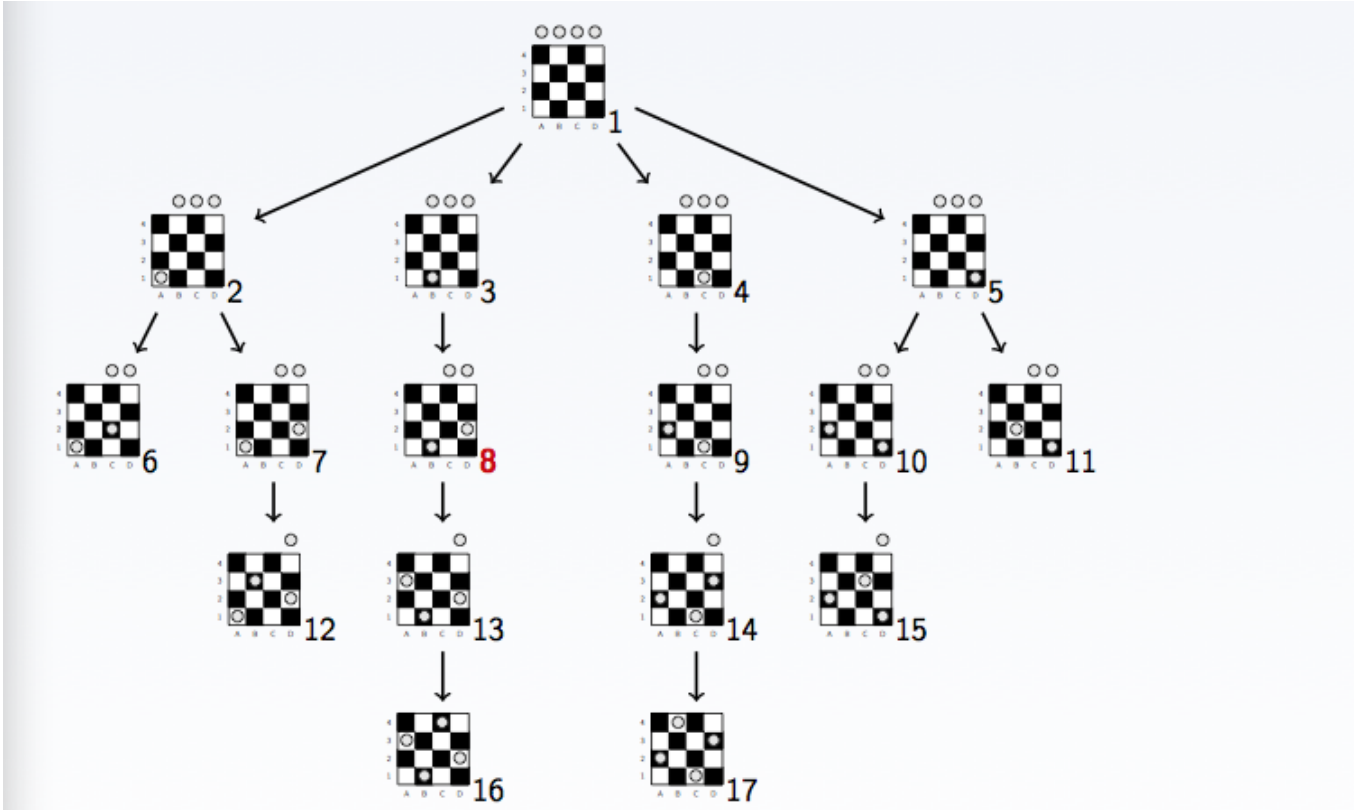


Reihenfolge der Knotenbesuche

$k = 5$

1,2,6,7,12,3,8,13,16 $\Rightarrow$ Ziel

Beispiel: Iterative Tiefensuche

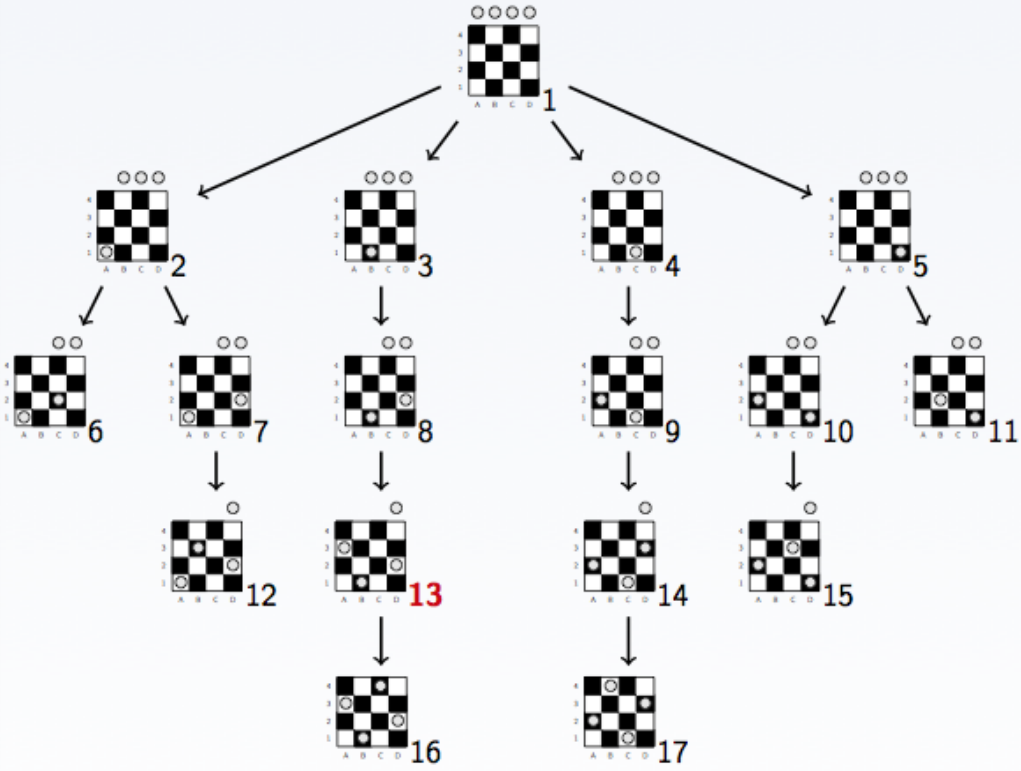


Reihenfolge der Knotenbesuche

$k = 5$

1,2,6,7,12,3,8,13,16 $\Rightarrow$ Ziel

Beispiel: Iterative Tiefensuche

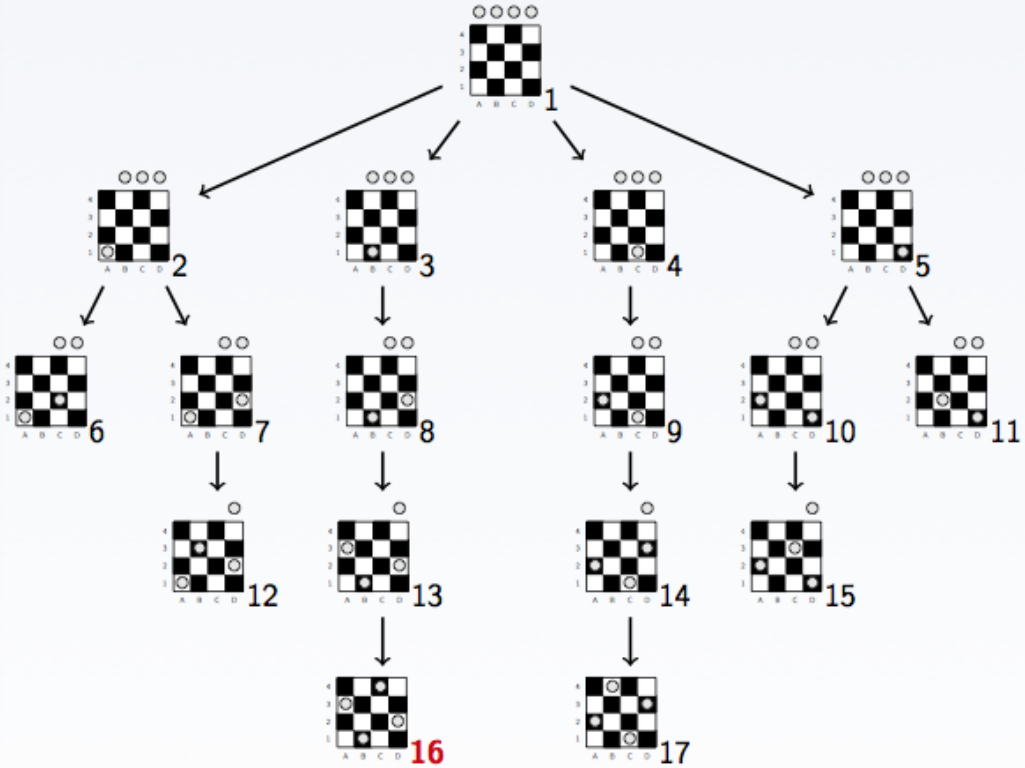


Reihenfolge der Knotenbesuche

$k = 5$

1,2,6,7,12,3,8,13,16 $\Rightarrow$ Ziel

Beispiel: Iterative Tiefensuche

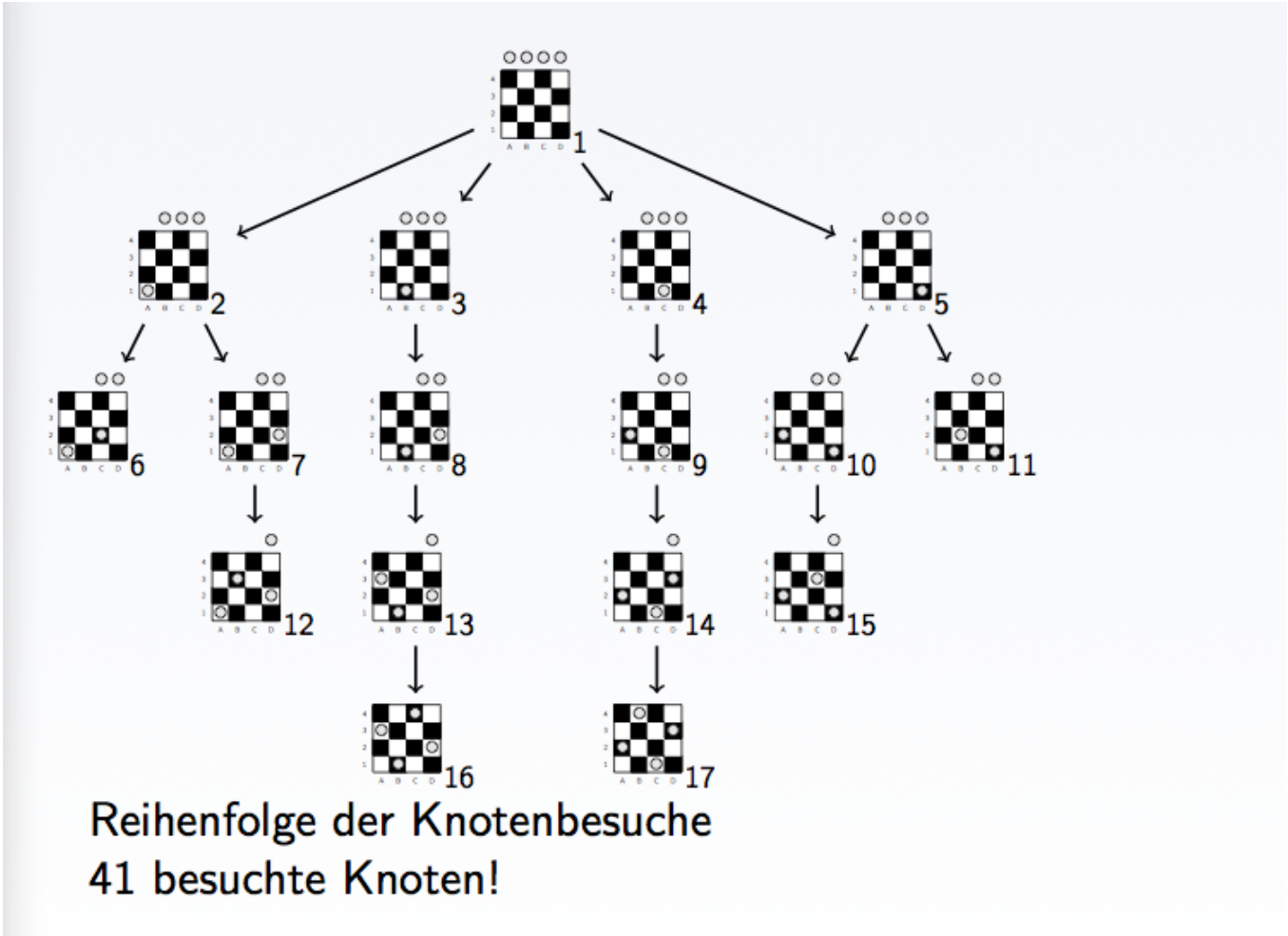


Reihenfolge der Knotenbesuche

$k = 5$

1,2,6,7,12,3,8,13,16 $\Rightarrow$ Ziel

Beispiel: Iterative Tiefensuche



Eigenschaften

Komplexität (worst-case)

bei mittlerer Verzweigungsrate $c > 1$

- Platz: Linear in der Tiefe
- Zeit: ? (gleich)

Vollständigkeit

- Die iterative Tiefensuche ist vollständig (bei endlicher Verzweigungrate)

Zeitbedarf iteratives Vertiefen

Näherung: $\sum_{i=1}^{n} a^i \approx \frac{a^{n+1}}{a-1}$

Tiefensuche mit Tiefenbeschränkung k

- Alle Knoten besuchen: $\sum_{i=1}^{k} c^i$
- Im Mittel besuchte Knoten (Zielknoten in Tiefe k):

  $0.5 * (\sum_{i=1}^{k} c^i) \approx 0.5 * (\frac{c^{k+1}}{c-1})$

Iteratives Vertiefen bis Tiefe k, im Mittel, Zielknoten in Tiefe k:

$k-1$ Tiefensuchen für Tiefe $1, \ldots, k-1$

+Tiefensuche für Tiefe k (Hälfte der Knoten)

$$= \sum_{i=1}^{k-1} \frac{c^{i+1}}{c-1} + 0.5 * (\frac{c^{k+1}}{c-1})$$

Zeitbedarf iteratives Vertiefen

Näherung: $\sum_{i=1}^{n} a^i \approx \frac{a^{n+1}}{a-1}$

$$\sum_{i=1}^{k-1} \frac{c^{i+1}}{c-1} + 0.5 * (\frac{c^{k+1}}{c-1}) = \frac{\sum_{i=1}^{k-1} c^{i+1}}{c-1} + 0.5 * (\frac{c^{k+1}}{c-1})$$

$$= \frac{(\sum_{i=1}^{k} c^i) - c^k}{c-1} + 0.5 * (\frac{c^{k+1}}{c-1}) \approx \frac{1}{c-1}((\frac{c^{k+1}}{c-1}) - c^1) + 0.5 * (\frac{c^{k+1}}{c-1})$$

$$= (\frac{c^{k+1}}{(c-1)^2}) - \frac{c}{c-1} + 0.5 * (\frac{c^{k+1}}{c-1}) \approx (\frac{c^{k+1}}{(c-1)^2}) + 0.5 * (\frac{c^{k+1}}{c-1})$$

Zeitbedarf iteratives Vertiefen

Faktor $\frac{\text{Iteratives Vertiefen}}{\text{Tiefensuche bis } k}$:

$$\frac{\frac{c^{k+1}}{(c-1)^2} + 0.5 * (\frac{c^{k+1}}{c-1})}{0.5 * (\frac{c^{k+1}}{c-1})} = \frac{\frac{c^{k+1}}{(c-1)^2}}{0.5 * (\frac{c^{k+1}}{c-1})} + 1 = \frac{2}{c-1} + 1$$

Tabelle der ca.-Werte des Faktors $d = \frac{2}{c-1} + 1$ ist

c	2	3	4	5	...	10
d	3	2	1,67	1,5	...	1,22

Bemerkungen zur iterativen Tiefensuche

Allgemeine Idee dabei:

- Spare Platz, opfere Zeit (speichern vs. neu berechnen)
- Gegensätzlich zum dynamischen Programmieren:
  Dort: Opfere Platz für schnellere Zeit.

Varianten

Nutze Platz:

- Speichere Knoten, die bereits expandiert wurden, und betrachte sie nicht neu
  $\Rightarrow$ keine wiederholten Expansionen
- Speichere einen Teil des letzten Suchbaums (z.B. den linken Teil), damit er beim nächsten mal nicht betrachtet werden muss.
- Kombiniere Breitensuche mit Tiefensuche: erst in die Breite, ab dort dann Tiefensuche

Rückwärtssuche

Idee

- Suche nicht vom Start das Ziel, sondern umgekehrt
- Statt Nachfolgerfunktion benutze Vorgängerfunktion

Lohnswert?

- Lohnt, wenn die Verzweigungsrate der Vorgängerfunktion kleiner ist, als die der Nachfolgerfunktion
- Problematisch: Finde algorithmische Beschreibung der Vorgängerfunktion

Bidirektionale Suche

- Suche vorwärts und rückwärts
- Erfordert Vergleich der momentan expandierten Knoten (Schnittmenge)
- Vorteil z.B. bei Breitensuche: Platz: statt c^d nur $2 * (c^{d/2})$ $\approx$ Doppelt so tief suchen in gleichem Platz (wenn #Eingangsknoten $\approx$ #Ausgangsknoten)
- Nachteil: Schnittbildung (Zeitintensiv), und Vorgänger- und Nachfolgerfunktion nötig.