

729G78 Artificiell intelligens

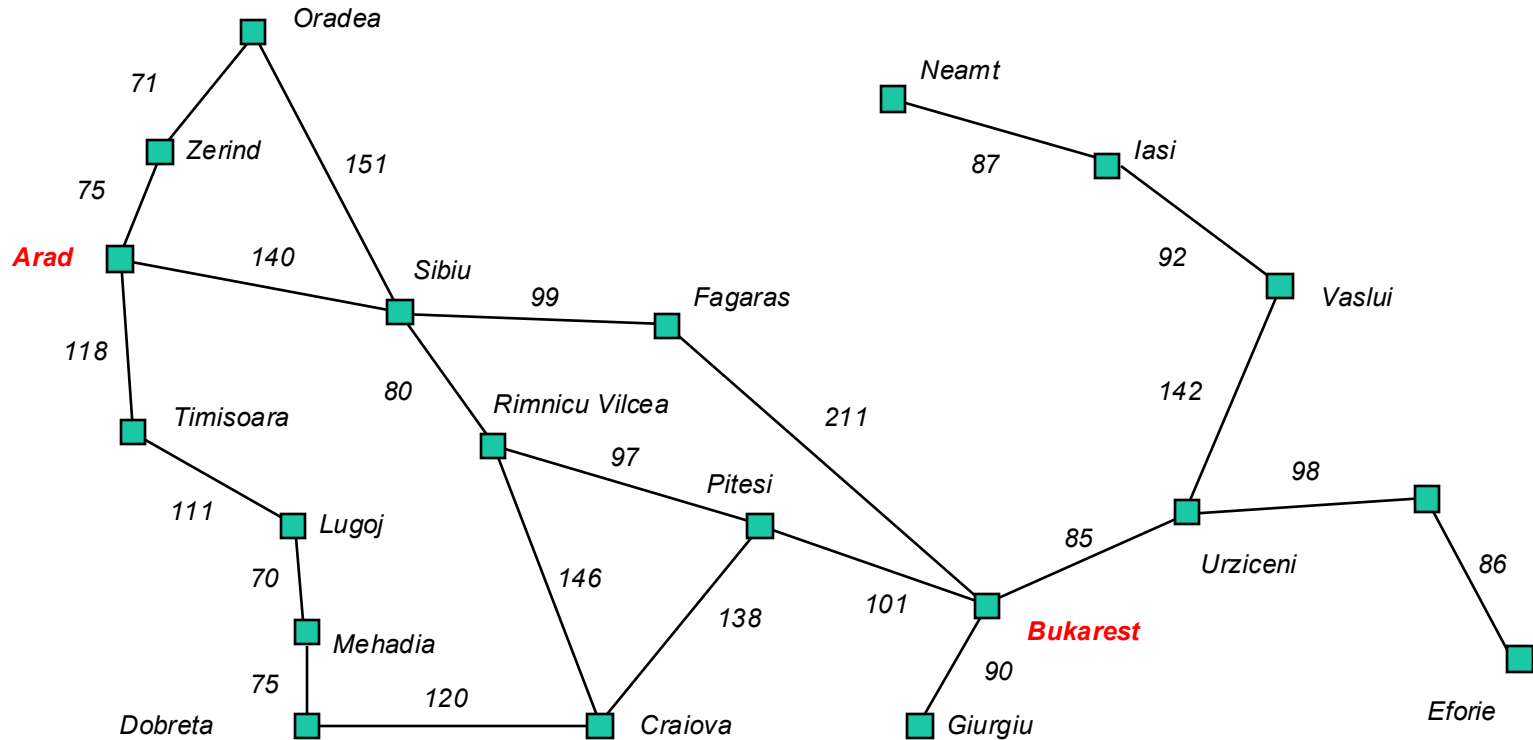
Sökning – Blind sökning

Robin Keskisärkkä

HCS/IDA

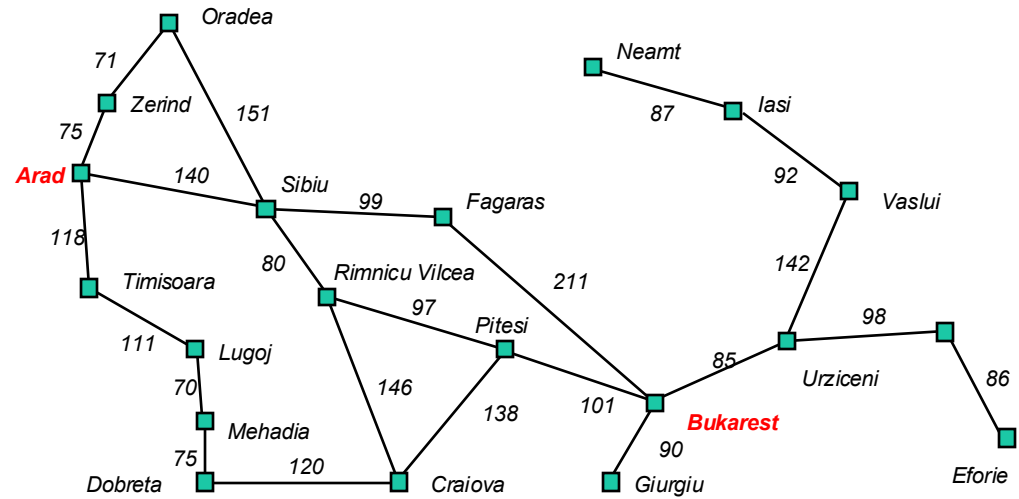
Problemformulering

Ex. Hitta bästa vägen från Arad till Bukarest



Tillståndsrymd

- Initialtillstånd
 $\{In(Arad)\}$
- Måltest
 $\{In(Bukarest)\}$
- Handlingar
 $\{Go(Sibiu), Go(Zerind), Go(Timisoara)\}$
- Övergångsmodell
 - Operationer som beskriver effekten av en handling som en funktion av tillståndsförändringar
 - $Result(In(Arad), Go(Sibiu)) = In(Sibiu)$
- Handlingskostnad
 - $Arad \rightarrow Timisoara \rightarrow Lugoj = 118 + 111 = 229$



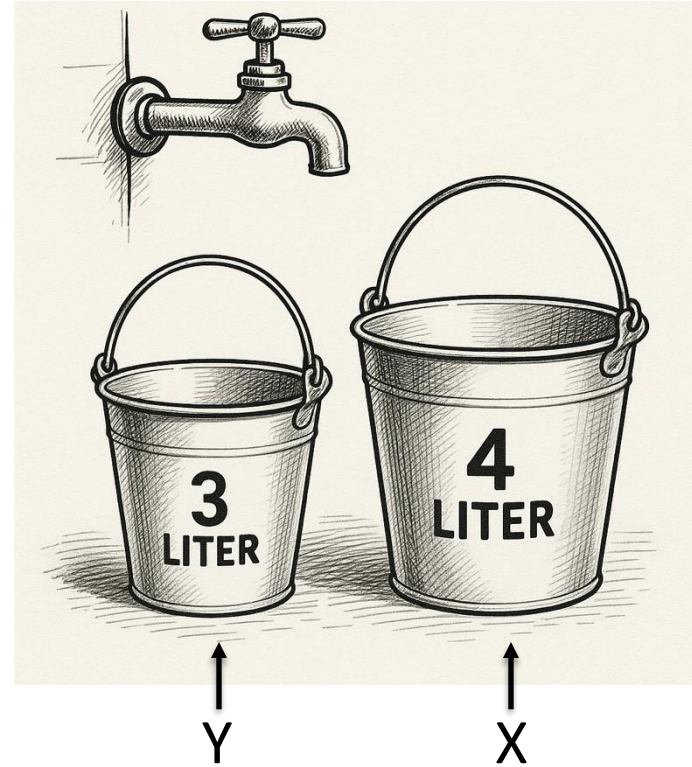
Exempel

Vattenhinksproblemet

- 2 hinkar: en på 4-liter och en på 3-liter
- En kran att fylla med
- Problem: att få exakt 2 liter i 4-litershinken

Extraantaganden

- Vatten i kranen
- Kan hälla vatten mellan hinkar
- Kan sluta hälla när som helst
- ...



Vattenhinksproblemet

Tillstånd:

X = 4-litershinken

Y = 3-litershinken

Initialtillstånd $(0, 0)$ dvs. $X=0$ och $Y=0$

Måltillstånd: $(2, Y)$ dvs. $X=2$ och Y vad som helst

Kostnad:

1 per operation

Operatorer:

Fyll en hink

Töm en hink

Fyll från en hink till en annan

Häll ut lite vatten



Vattenhinksproblemet

Operationer:

$$(X, Y: X < 4) \rightarrow (4, Y)$$

$$(X, Y: Y < 3) \rightarrow (X, 3)$$

$$(X, Y: X > 0) \rightarrow (0, Y)$$

$$(X, Y: Y > 0) \rightarrow (X, 0)$$

$$(X, Y: X + Y \leq 4 \wedge Y > 0) \rightarrow (4, Y - (4 - X))$$

$$(X, Y: X + Y \leq 3 \wedge X > 0) \rightarrow (X - (3 - Y), 3)$$

$$(X, Y: X + Y \leq 4 \wedge Y > 0) \rightarrow (X+Y, 0)$$

$$(X, Y: X + Y \leq 3 \wedge X > 0) \rightarrow (0, X+Y)$$



Sökning

- Datastrukturer och operationer
- Värdering av sökstrategier
- Blind sökning (flera olika varianter)
- Heuristisk sökning
- Genetiska algoritmer
- Spelförande program
- Constraint satisfaction

Sökning

- Datastruktur
 - $\text{nod} = [\text{tillstånd}, \text{förälder}, \text{operator}, \text{djup}, \text{vägkostnad}]$
- Expandera varje nod
 - Applicera operatorer på ett tillstånd och generera alla nya tillstånd
- Ex: Initialtillståndet $(0, 0)$ i vattenhinksproblemet
 - Expanderas till $(0, 3)$ eller $(4, 0)$
 - $(0, 3)$ expanderas till $(0, 0)$, $(3, 0)$ eller $(4, 3)$
 - etc.
- Sökstrategin avgör vilken nod som ska expanderas närmast

Köoperationer

- Sökalgoritmer sparar noder som ska expanderas i en kö
- Typiska köoperationer:
 - `isEmpty(queue)`
 - `pop(queue)`
 - `add(element, queue)`
- Queue: Först-in-först-ut (FIFO)
- Stack: Sist-in-först-ut (LIFO)
- Deque: FIFO eller LIFO
- List: Flexibel men LIFO som standard

Generell sökalgoritm

```
def search(problem):  
    frontier = []  
    add(initialNode(problem), frontier)  
    while True:  
        if isEmpty(frontier):  
            return failure  
        node = frontier.pop()  
        if goalTest(problem, state(node)):  
            return node  
        frontier = addAll(expand(node, problem), frontier)
```

Nodexpansion

`expand(node, problem)`

Skapa en lista av noder som är efterföljare till en nod:

`state(n):` ger tillståndet för noden `n`

`parent(n):` ger förälder till noden `n`

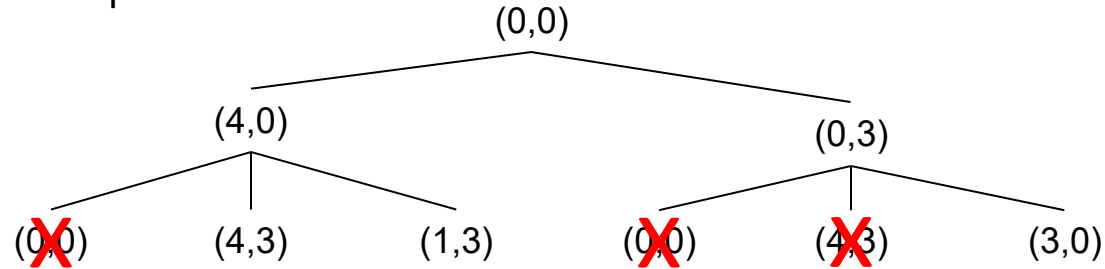
`action(n):` ger handlingen som ledde till noden `n`

`väggkostnad(n):` väggkostnad för att nå `n` + väggkostnad för förälder

`djup(n):` förälderns djup + 1

Undvika loopar

Ex vattenhinksproblemet



- Generera inte tillstånd som är lika med förälderns tillstånd
- Generera inte vägar med cykler
 - Behöver bara leta från noden som genererats upp till startnoden
- Generera inte ett tillstånd som genererats förut
 - Varje tillstånd måste lagras i minnet, $O(b^d)$

Sökning

- Effektivitet
 - Nås målet?
 - Till vilken kostnad?
 - Hur bra är lösningen?
- Totalkostnaden = handlingskostnaden + sökkostnaden

Värdering av sökstrategi

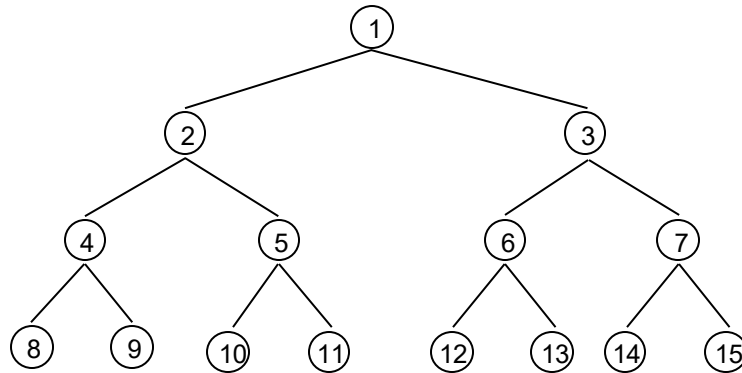
- Komplett
 - Hittar lösning om den finns
- Optimal
 - Hittar ”bästa” lösningen
- Tidsåtgång
- Minnesåtgång

Blind sökning

- Ingen information om hur långt det är kvar till målet
 - Bredden först
 - Uniform cost
 - Djupet först
 - Djupbegränsad sökning
 - Iterativ fördjupning
 - Dubbelriktad sökning

Bredden först

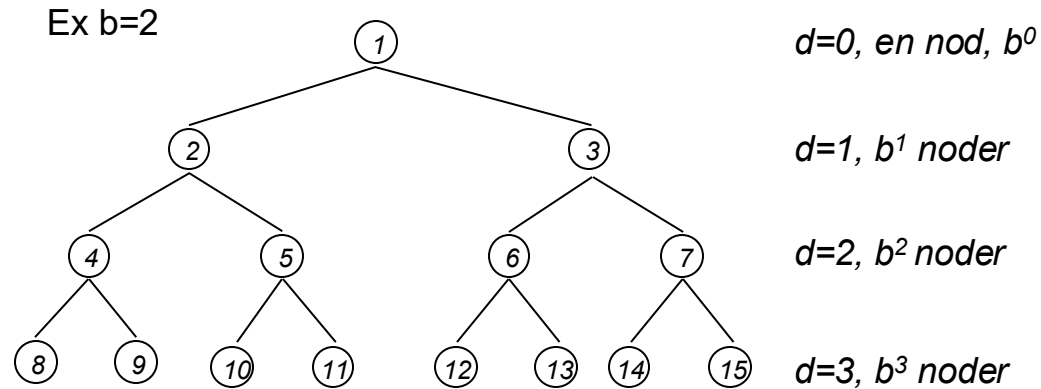
- Expandera alla noder på en nivå först



Implementeras med en FIFO-kö

Bredden först

- Komplett
- Optimal
- Komplexitet ett problem
 - Förgreningsfaktor b
 - Sökdjup d

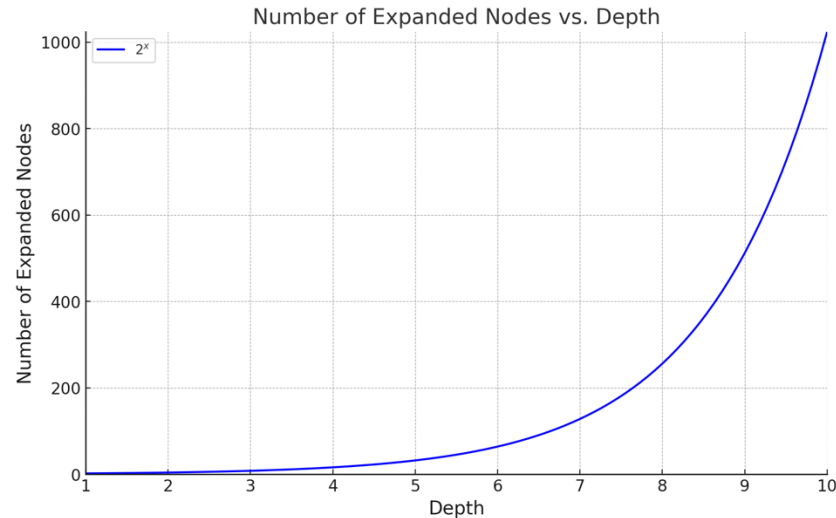


Bredden först

Antag lösning på djupet d , då har vi expanderat:

$$1 + b^1 + b^2 + b^3 + \dots + b^{d-1} + b^d \text{ noder}$$

Vid komplexitetsanalys säger vi att vi har $O(b^d)$, dvs exponentiell tillväxt



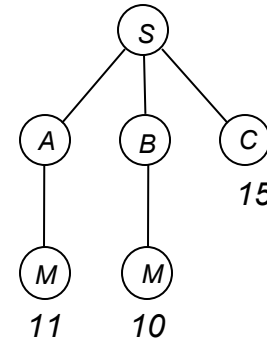
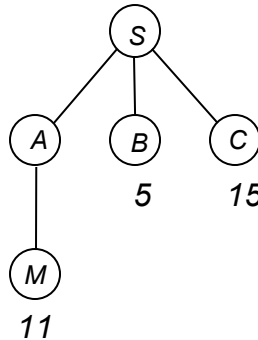
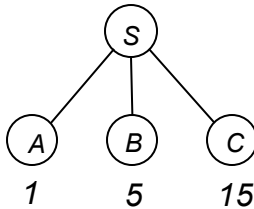
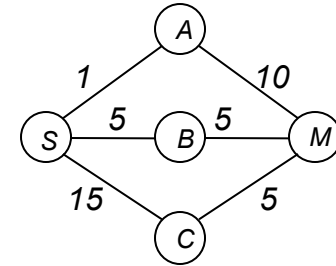
Bredden först, tids- och minneskomplexitet

Djup	Noder	Tid	Minne
2	100	0,1 ms	0,1 MB
4	10000	10 ms	10 MB
6	10^6	1 s	1 GB

Förgreningsfaktor = 10, 1 miljon noder/sekund, 1000 bytes/nod

Uniform Cost

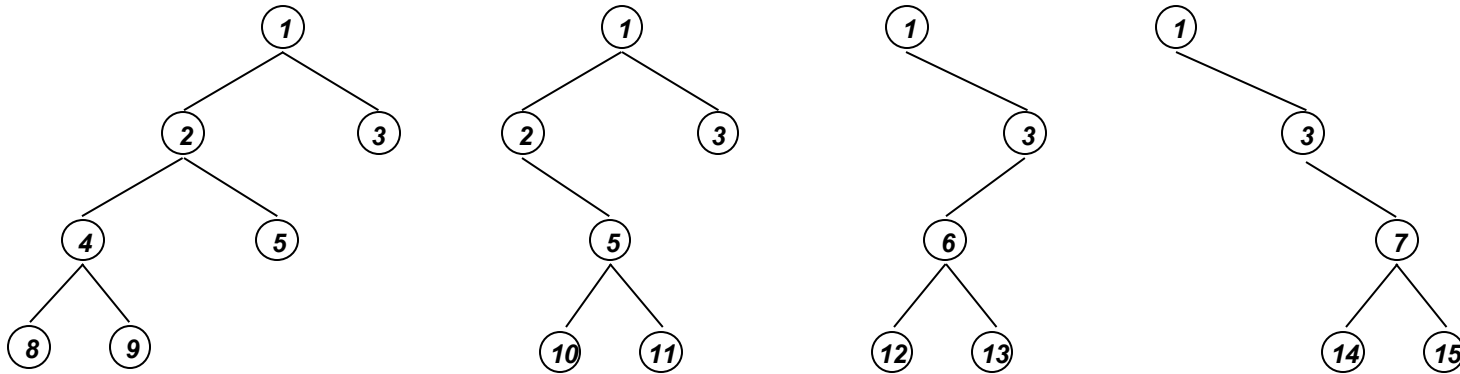
- Om inte alla operatorer har samma kostnad
- Expandera billigaste vägen längs "frontier"



- Optimal och komplett om det inte finns negativa kostnader
- I princip $O(b^d)$ för tids- och minneskomplexitet ($O(b^{1+C^*/\epsilon})$)

Djupet först

- Följer alltid en väg till slutet
- Om ingen lösning hittas testas nästa gren



Implementeras som en LIFO-kö (stack)

Djupet först

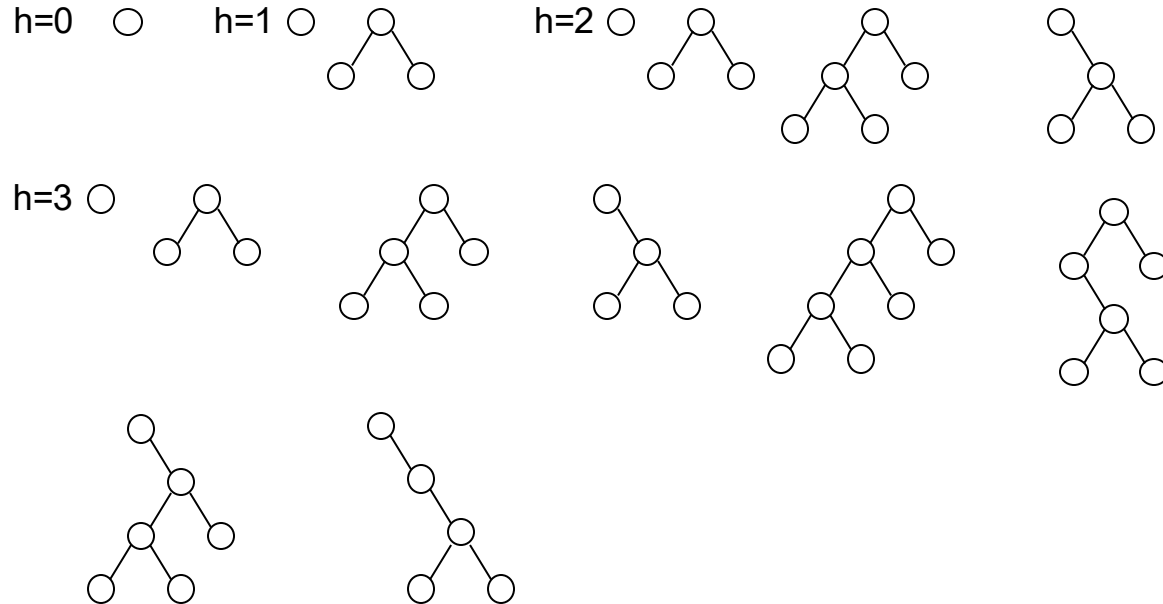
- Kräver mindre minne, $m \cdot b$ där $m = \text{maxdjup}$
 - Jämför $b=10$, $m=12$
djupet först = $10 \cdot 12 = 120$
bredden först = 10^{12}
- Samma tidskomplexitet $O(b^m)$
- Inte optimal
- Inte komplett, kan dyka ner i en oändligt lång sökväg

Djupbegränsad sökning

- Lägg in ett maxdjup, l , för djupet först
- Tidskomplexitet fortfarande exponentiell, $O(b^l)$
- Minneskomplexitet $O(b \cdot l)$
- Inte komplett om man inte kan skatta ett sök djup som garanterar en lösning, jfr, rumäniengrafen, max 20 städer, dvs $l = 20$ ger komplett
- Inte optimal

Iterativ fördjupning

- Låt sökdjupet, h , öka från 0 till det djup, d , där en lösning finns



Iterativ fördjupning

- Optimal och komplett
- Minneskomplexitet, $O(b \cdot d)$
- Tidskomplexitet, $O(b^d)$
- Expanderar dock noderna flera gånger
 - Bredden-först $1 + b^1 + b^2 + b^3 + \dots + b^{d-1} + b^d$
 - Iterativ fördjupning $1 + d \cdot b^1 + (d-1) \cdot b^2 + (d-2) \cdot b^3 + \dots + 3 \cdot b^{d-2} + 2 \cdot b^{d-1} + 1 \cdot b^d$
 - Men antal noder på nivå d är många fler än resten. Ex $b=10, d=5$:
 - Bredden först: $1 + 10 + 100 + 1000 + 10000 + 100000 = 111\ 111$
 - Iterativ fördjupning: $50 + 400 + 3000 + 20000 + 100000 = 123\ 450$

Dubbelriktad sökning

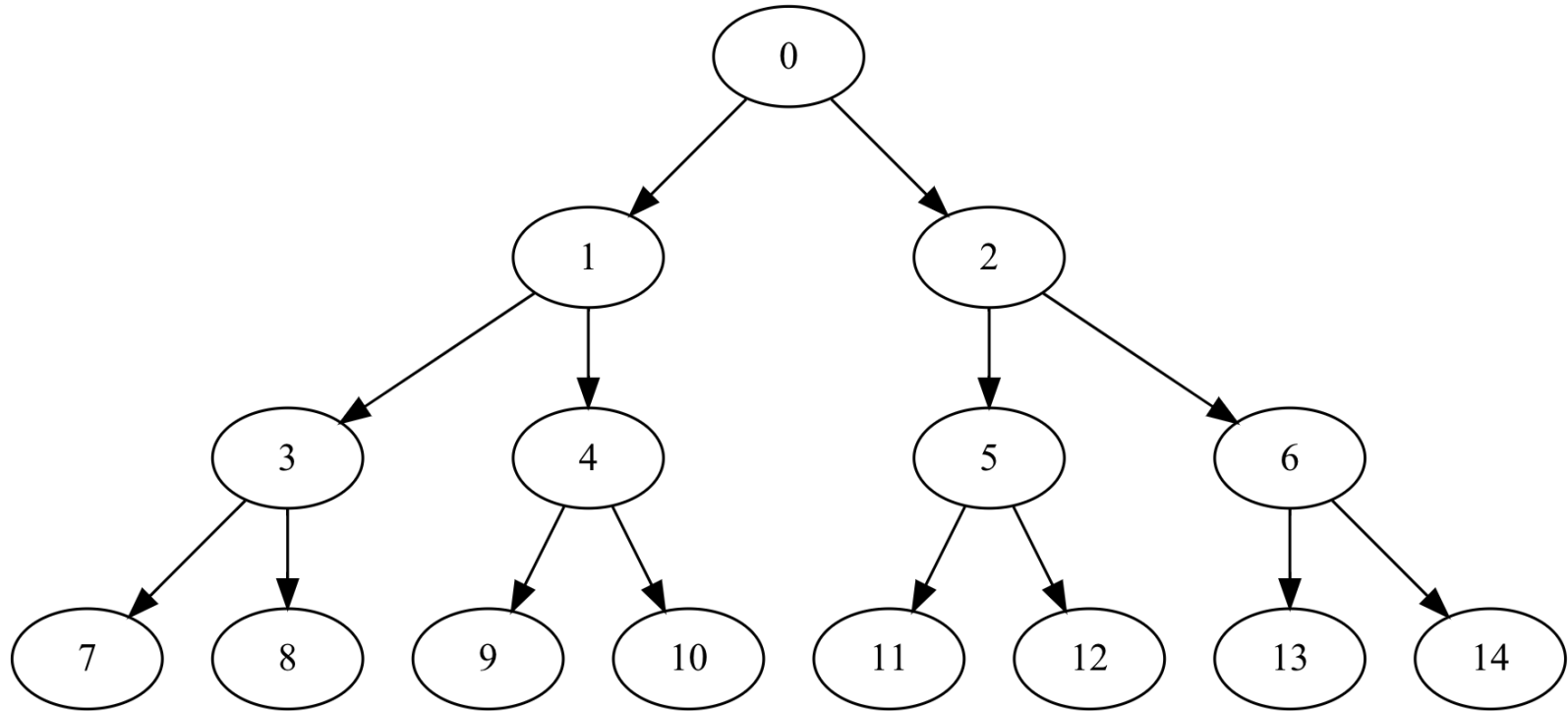
- Sök bredden-först från start och mål samtidigt
 - Inte alltid möjligt
- Komplett och optimal
- Tids- och minneskomplexitet, lösning djup d
 - $2b^{d/2}$, dvs $O(b^{d/2})$
 - Ex $b=10$, $d=6$
 - Bredden först, $O(b^d) = 10^6 = 1000000$
 - Dubbelriktad sökning, $O(b^{d/2}) = 10^{6/2} = 10^3 = 1000$

Sammanfattning: sökstrategier

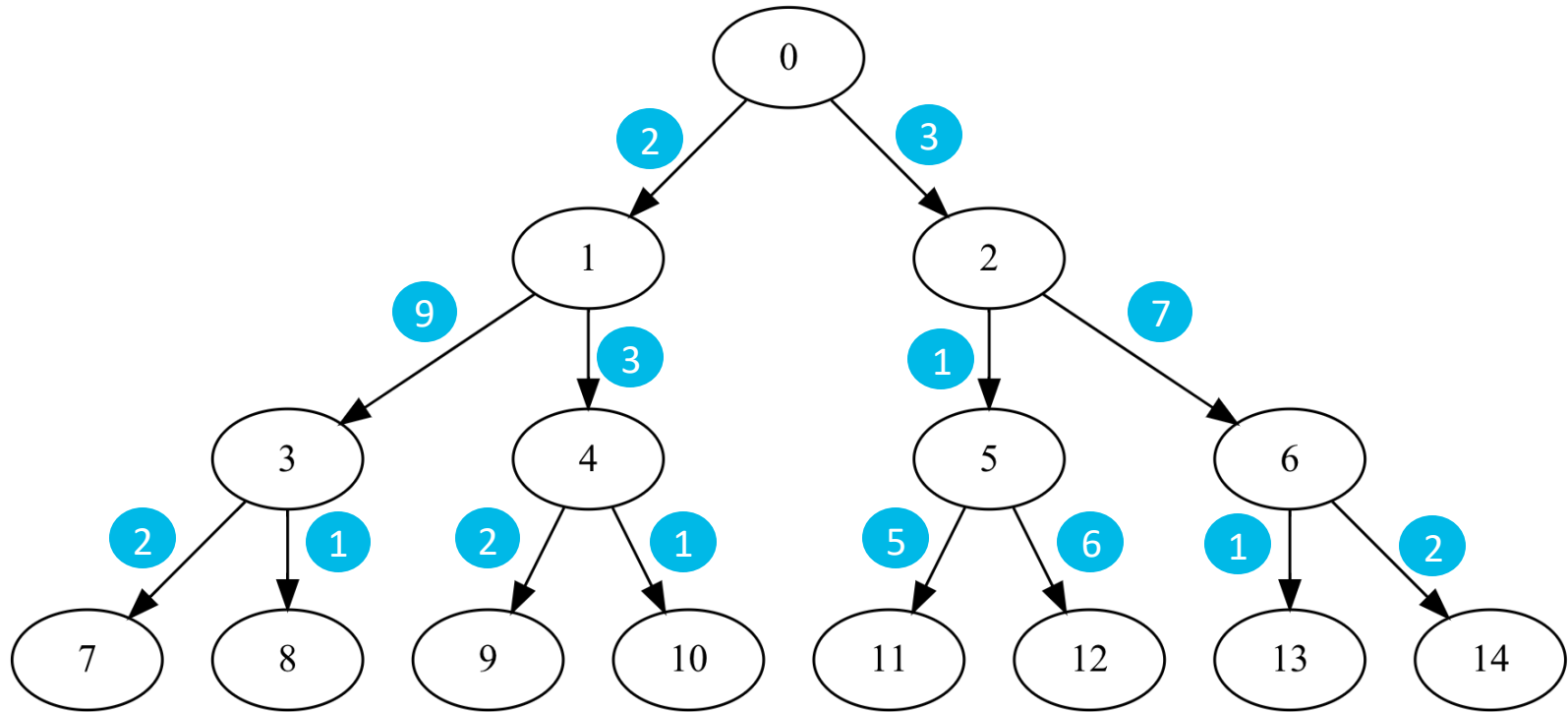
Kriterium	Bredden först	Uniform Cost	Djupet först	Djup-begränsad	Iterativ fördjupning	Dubbelriktad sökning
Komplett?	Ja	Ja	Nej	Nej	Ja	Ja
Optimal?	Ja	Ja	Nej	Nej	Ja	Ja
Tid	$O(b^d)$	$O(b^{1+C^*/\epsilon})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Minne	$O(b^d)$	$O(b^{1+C^*/\epsilon})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$

Övningar

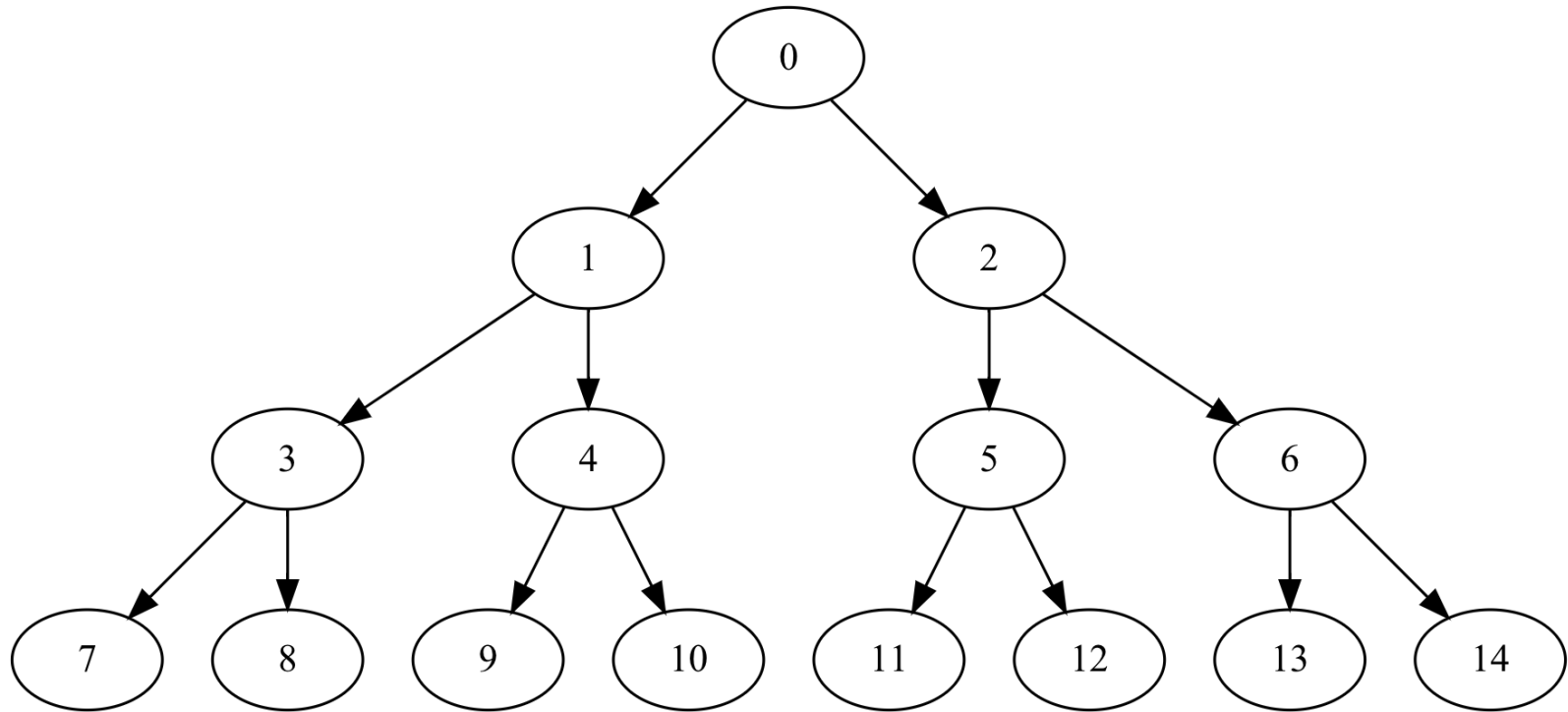
Bredden först sökning



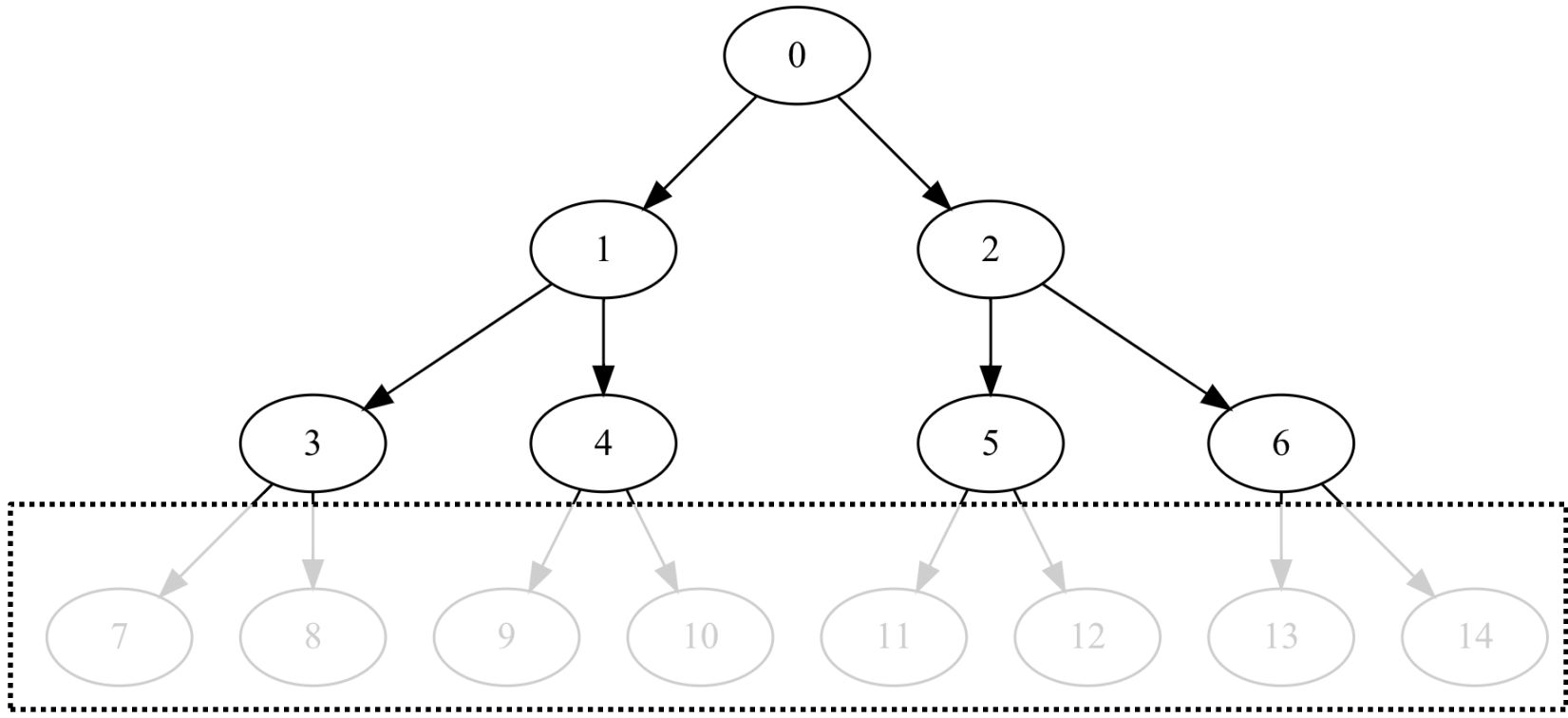
Uniform cost



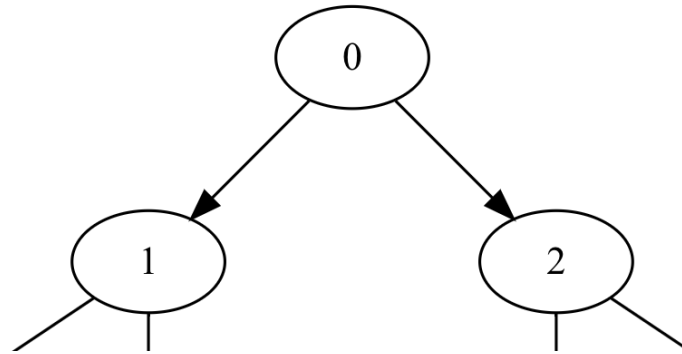
Djupet först sökning



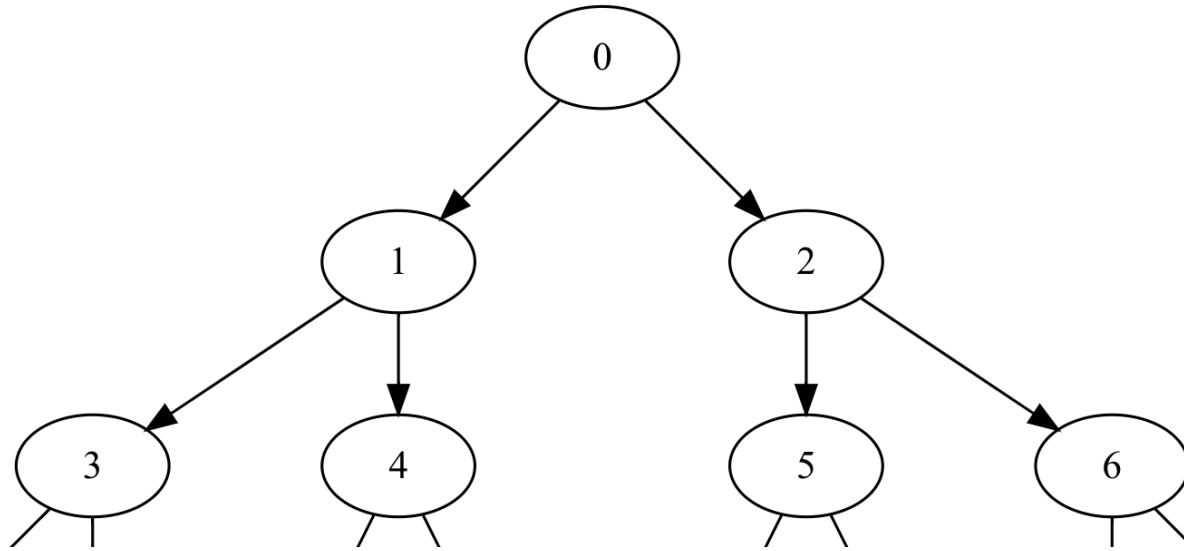
Djupbegränsad sökning



Iterativ fördjupning



Iterativ fördjupning



Iterativ fördjupning

