Problem Solving By Searching

Chastine Fatichah
Departemen Teknik Informatika
Februari 2023

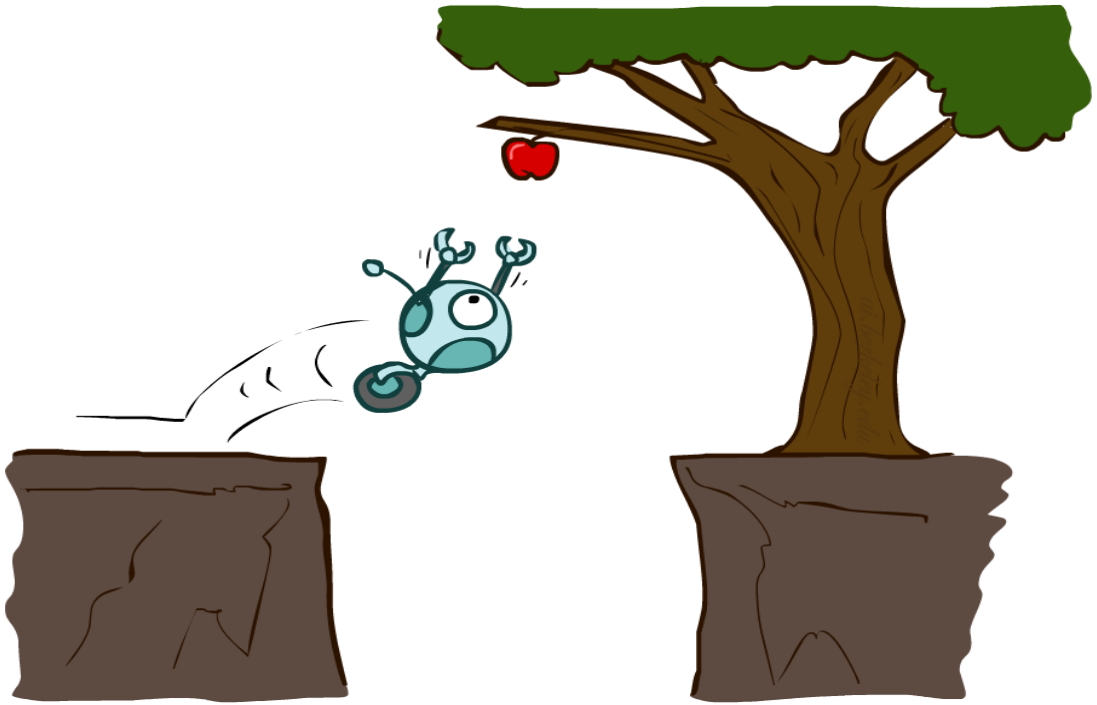
Capaian Pembelajaran Matakuliah

Mahasiswa mampu menjelaskan, mengidentifikasi, merancang, dan menerapkan *intelligent agent* untuk *problem* yang sesuai dengan memanfaatkan algoritma pencarian yang meliputi *uninformed search*, *informed search*, *heuristic search*, *adversarial search*, serta algoritma *search* untuk *Constraint Satisfaction Problem*

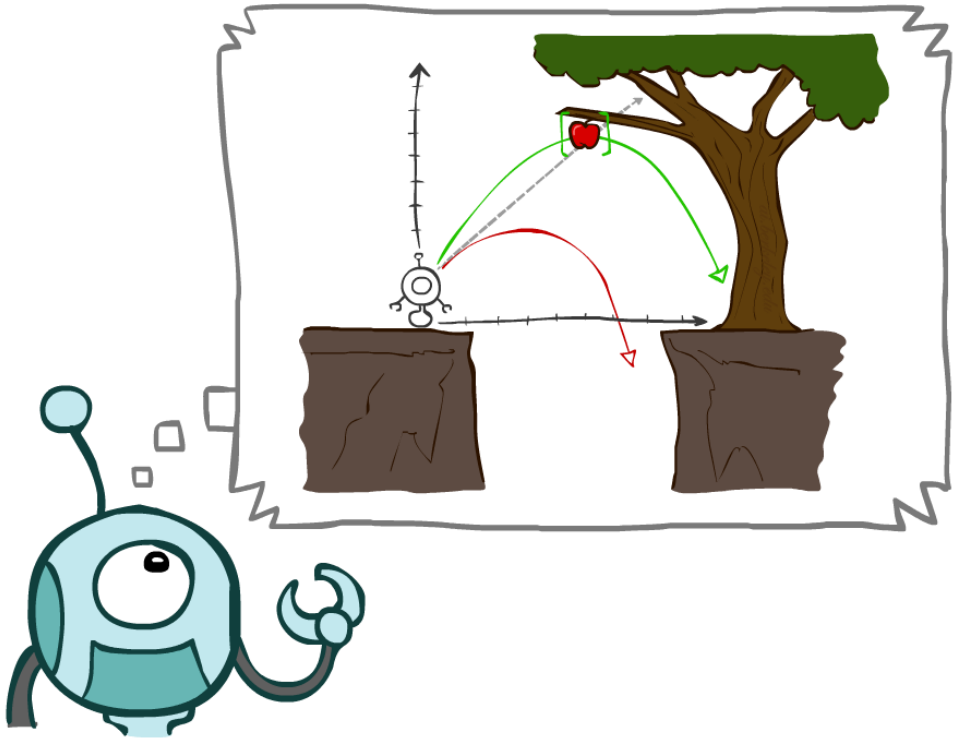
Pokok Bahasan

- *Problem-solving agent*
- Representasi masalah: *state space*
- Pencarian solusi: *Search strategies*
- *Uninformed search strategy*
  - Depth-First Search
  - Breadth-First Search
  - Uniform Cost Search

Reflex Agents

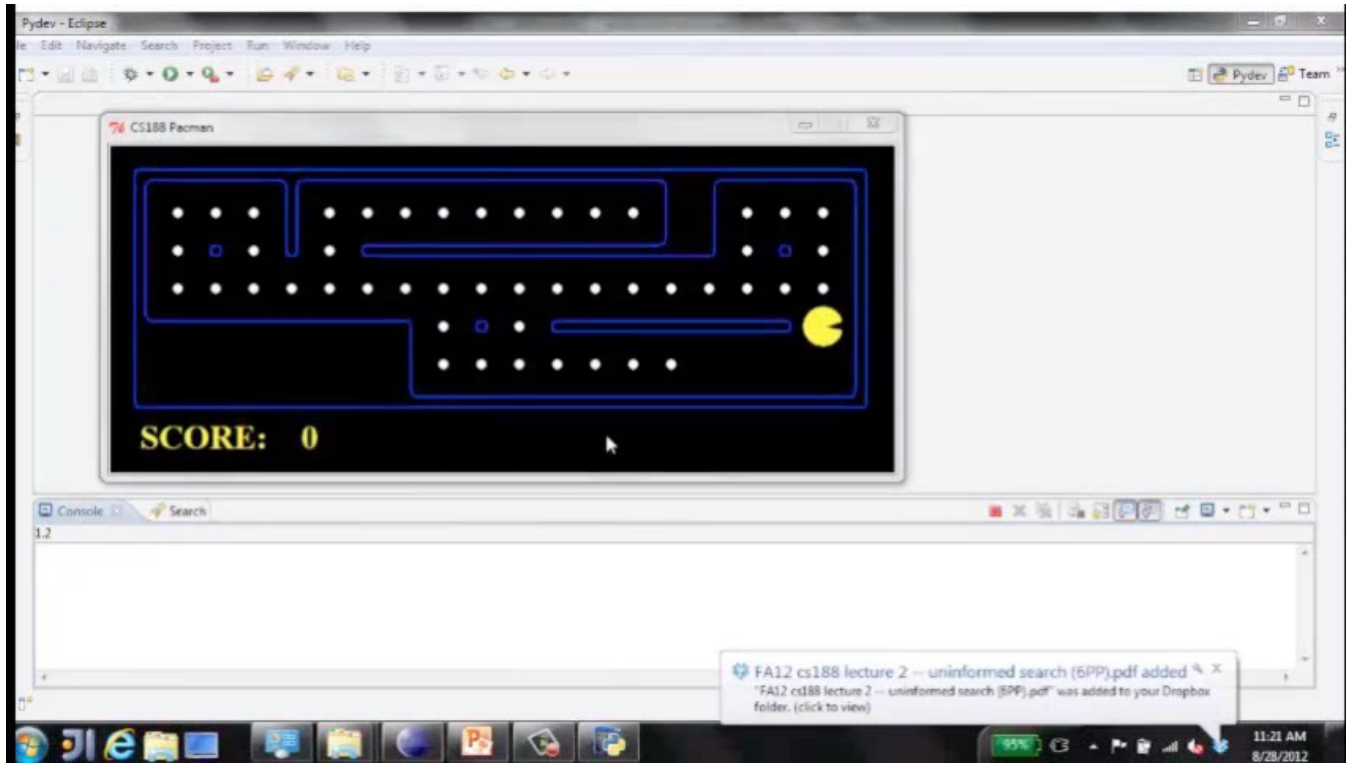


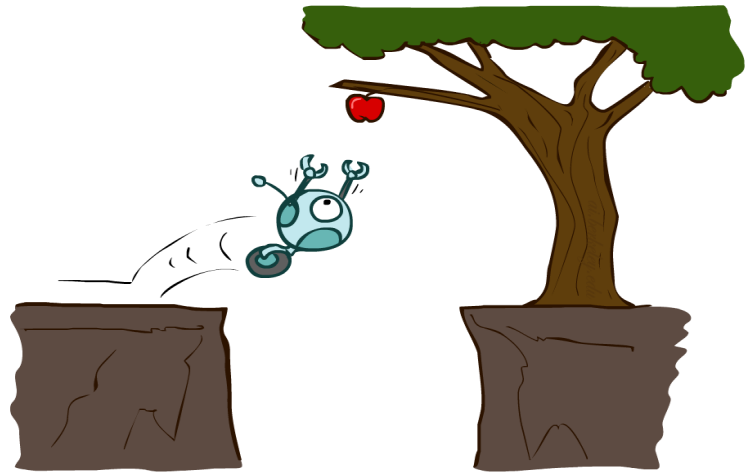
Agents that Plan



Sumber: Sergey Levine & Stuart Russell, University of California, Berkeley

Reflex Agents

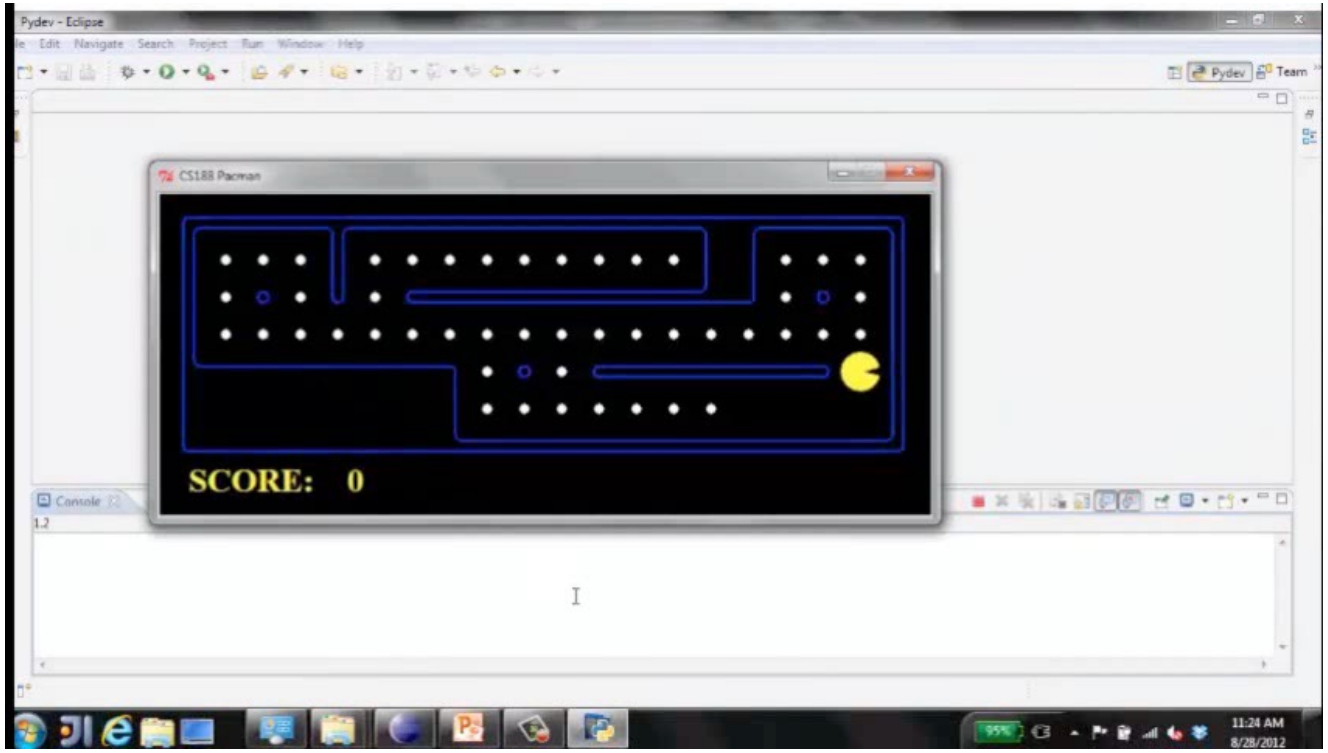


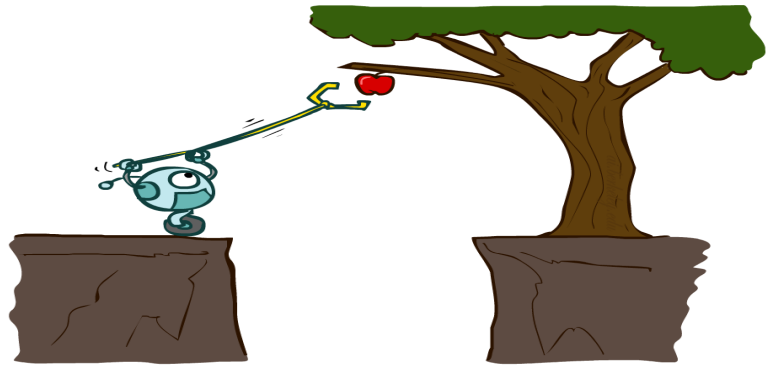


- Pemilihan aksi berdasarkan persepsi sekarang (dan mungkin memori)
- Bisa mempunyai memori atau model dari kondisi (*state*) sekarang
- Tidak mempertimbangkan konsukensi kedepan dari aksi yang dilakukan

Sumber: Sergey Levine & Stuart Russell, University of California, Berkeley

Planning Agents



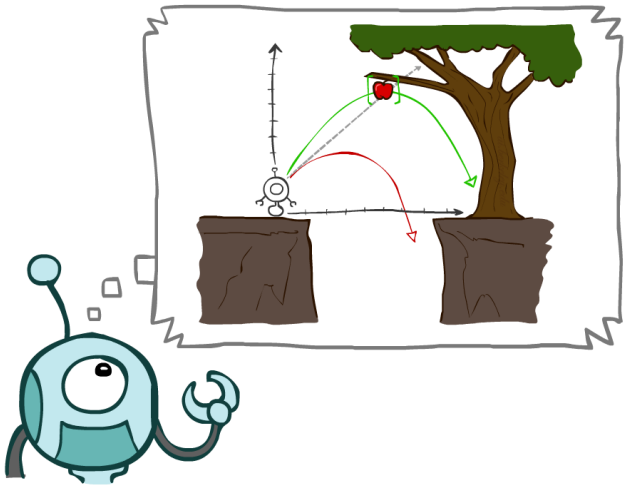


- Ask “what if”
- Keputusan berdasarkan hipotesis konsukensi dari aksi yang dilakukan
- Harus mempunyai model bagaimana seharusnya dunia mempertimbangkan sebuah aksi (Consider how the world WOULD BE)
- Harus merumuskan sebuah tujuan (*goal*)
- *Optimal or not optimal*
- *Complete or not*
- *Planning vs. replanning*

Sumber: Sergey Levine & Stuart Russell, University of California, Berkeley

Problem-Solving Agents

- *Goal-based agent* → mempertimbangkan aksi-aksi yang akan datang dan hasil yang ingin dicapai
- *Agent problem solving* → Menemukan rangkaian aksi (*sequence action*) untuk mencapai tujuannya
- Algoritma *Uninformed* → Tidak ada informasi, hanya deskripsi pada masalah tersebut



Simple Problem-Solving Agent

```
Function Simple-Problem-Solving-Agent(percept) return an action
Input : percept //a percept
Static : 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 do
         goal ← Formulate-Goal(state)
         problem ← Formulate-Problem(state,goal)
         seq ← Search(Problem)

Action ← First(seq)
Seq ← Rest(seq)
Return action
```

Mekanisme Kerja Problem-Solving Agent

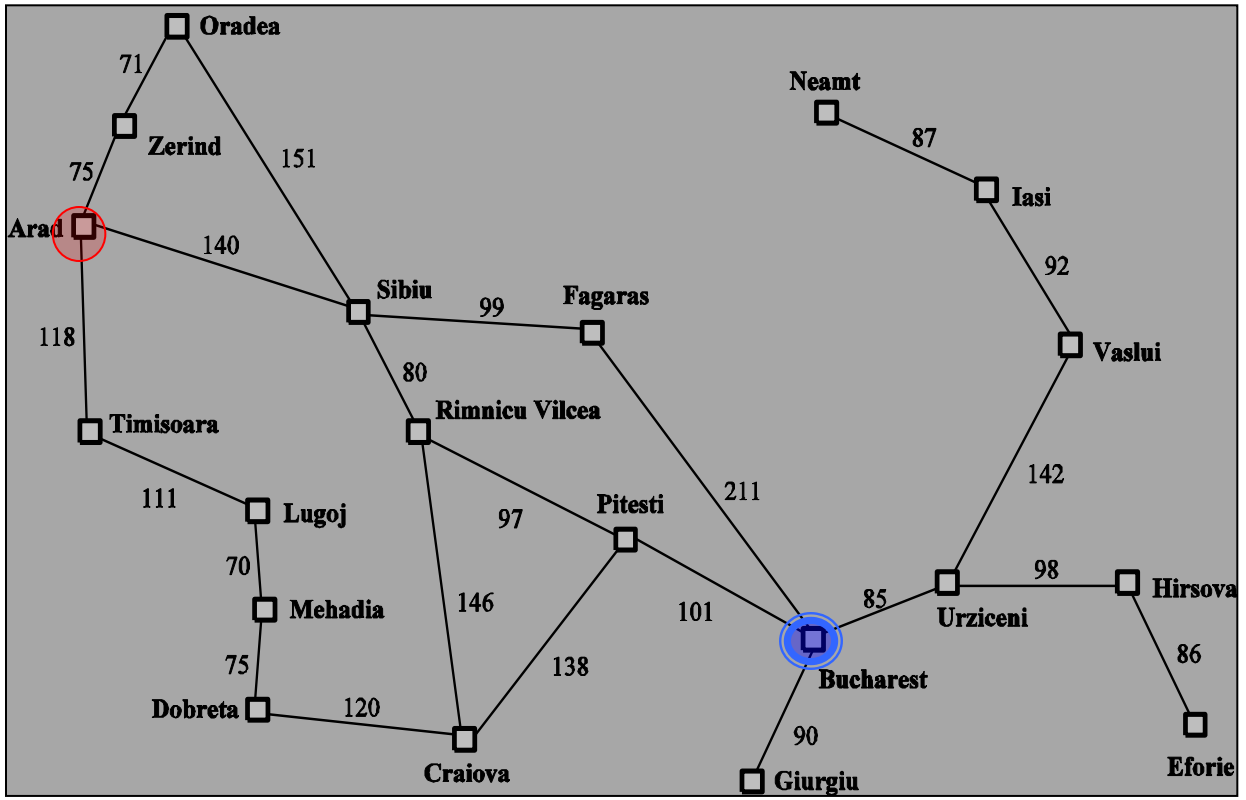
- Perumusan tujuan (*goal formulation*): tentukan tujuan yang ingin dicapai
  - *Kondisi saat ini*
  - *Performance measure*
- Perumusan masalah (*problem formulation*): tentukan tindakan/aksi (*action*) dan keadaan (*state*) yang dipertimbangkan dalam mencapai tujuan
- Pencarian solusi masalah (*searching*) : tentukan rangkaian aksi yang perlu diambil untuk mencapai tujuan
  - Input: problem, output: solusi dalam bentuk rangkaian aksi
- Pelaksanaan solusi (*execution*): laksanakan rangkaian aksi yang sudah ditentukan di tahap sebelumnya

Mendefinisikan Problem dan Solusi

- Problem dapat dirumuskan dengan 4 komponen
  - *Initial state*
  - *Actions : Successor function* :
    - `Successor-Fn(x) = <Action,Successor>`
  - *Goal Test*
  - *Path Cost*
    - Optimal Solution → *path cost* terkecil

Contoh Kasus: Romania

- Seorang *agent* sedang berlibur dan sekarang sedang di kota Arad Romania
- Besok dia harus naik pesawat dari Bucharest
- *Goal* dari *agent* sekarang adalah pergi ke Bucharest
- *Action* yang tidak berhubungan dengan *goal* akan dibuang -> *decision agent* lebih sederhana



Contoh Kasus: Romania

- *Agent* → Mencapai tujuan (ke Bucharest) dengan naik mobil
- Kemana akan pergi setelah dari Arad ?
  - Ada tiga jalan : ke Sibiu, Timisoara, Zerind
  - Agent kita ini masih belum tahu jalan disana (mana yang tercepat) tapi hanya memiliki peta.
  - Dari informasi peta, dilakukan hipotesa terhadap ketiga jalur tersebut untuk sampai ke Bucharest

Contoh Kasus: Romania (Agent & Environment)

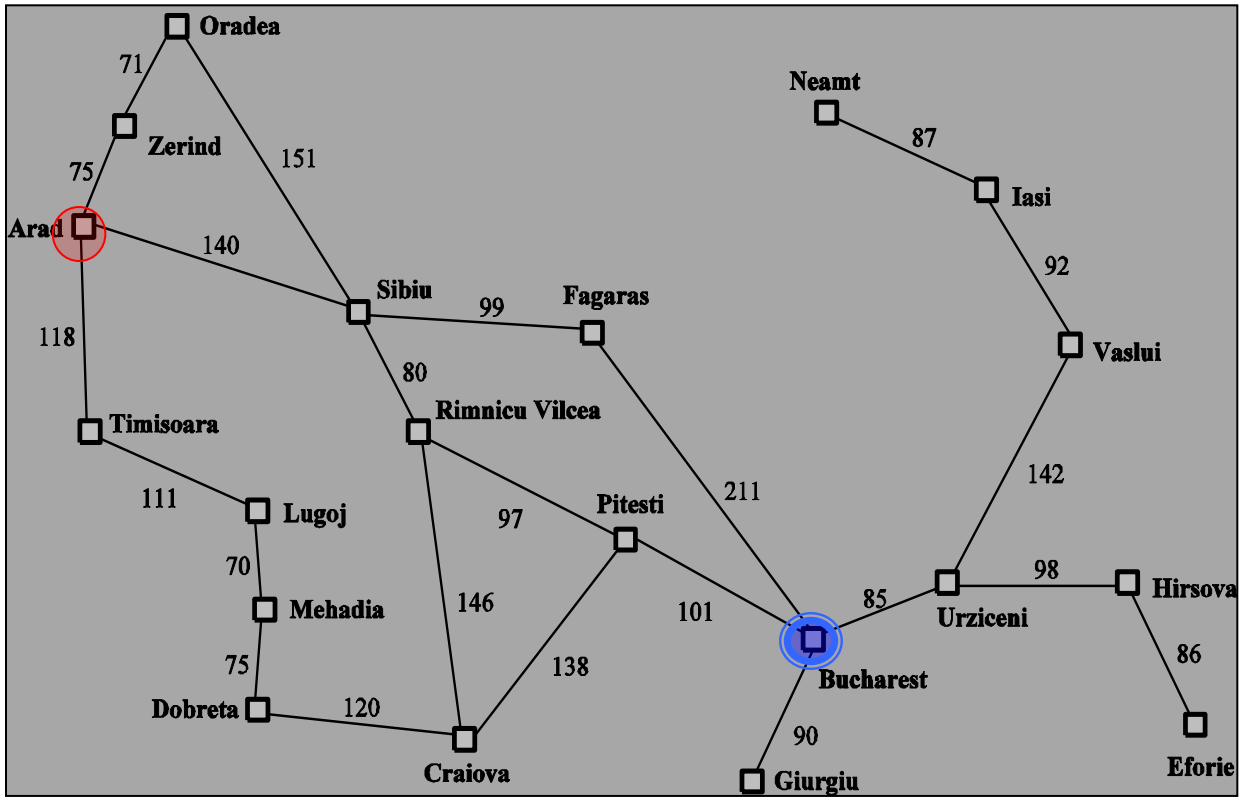
- Static
  - Tidak perlu memperhatikan perubahan yang terjadi pada *environment*
- Observable
  - Ada peta, *initial state* diketahui (di Arad)
- Discrete
  - *Enumeration action*
- Deterministic
  - Tidak bisa menangani terhadap hal-hal yang tidak diperkirakan

Contoh Kasus: Romania (Perumusan Tujuan, Masalah, & Solusi)

- Perumusan Tujuan
  - Tiba di Bucharest besok
- Perumusan Masalah
  - States : kota-kota
  - Actions : mengemudi antar kota
- Pencarian Solusi
  - Rangkaian kota : Arad, Sibiu, Fagaras, Bucharest

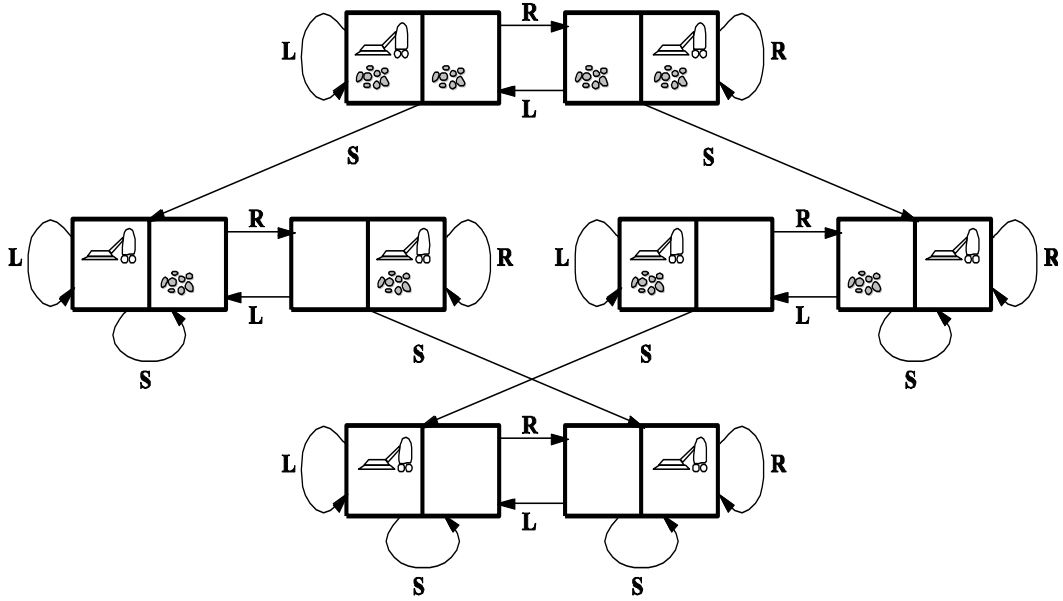
Contoh Kasus: Romania

- Initial State →In(Arad)
- Actions : Successor function →
  - {<Go(Sibiu),In(Sibiu)>,
    <Go(Timisoara),In(Timisoara)>,
    <Go(Zerind),In(Zerind)>}
- Goal Test → In(Bucharest)
- Path Cost →



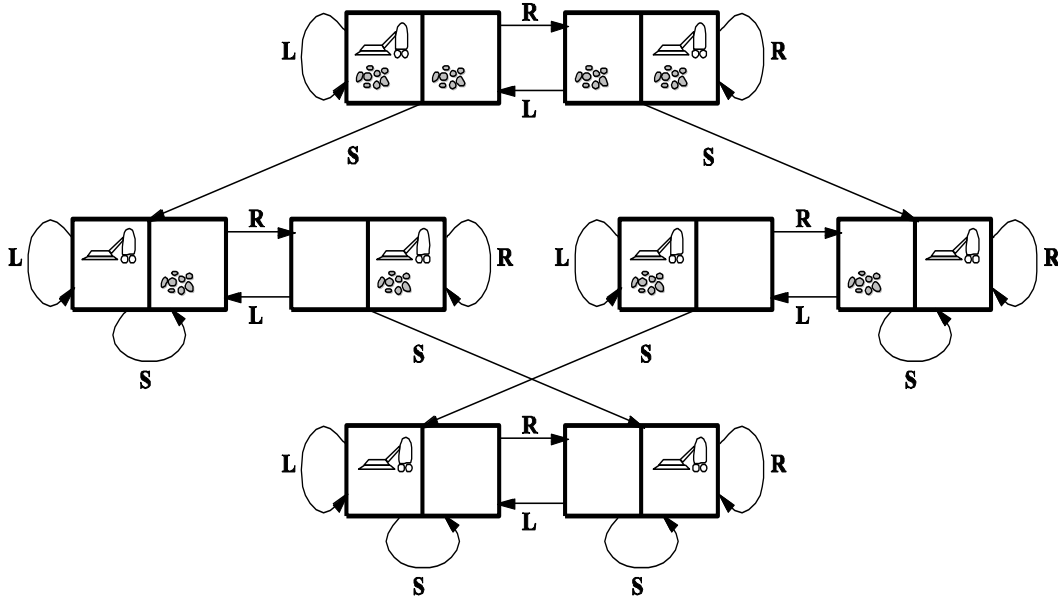
Contoh: Vacuum Cleaner

- *States* : Berada di salah satu dari dua lokasi yang ada, setiap lokasi mungkin bersih atau kotor. Jadi jumlah kemungkinan state = $2 * 2^2$
- *Initial State* ?
- *Successor Function* ?
- *Goal Test* ?
- *Path Cost* ?



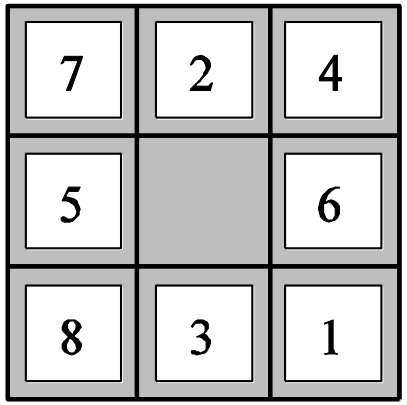
Contoh: Vacuum Cleaner

- *States* : Berada di salah satu dari dua lokasi yang ada, setiap lokasi mungkin bersih atau kotor. Jadi jumlah kemungkinan state = $2 * 2^2$
- *Initial State* ? Sembarang state
- *Successor Function* ? (ke kiri, ke kanan, bersihkan)
- *Goal Test* ? Semua lokasi bersih
- *Path Cost* ? Setiap aksi = 1 point

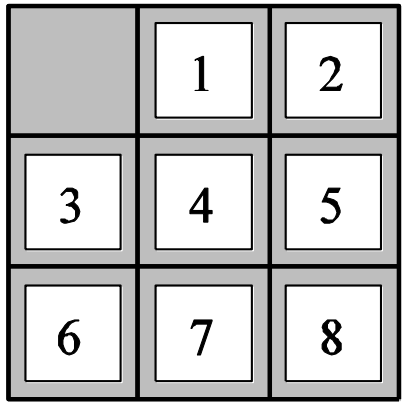


Contoh Kasus: 8-Puzzle

- *State* : ?
- *Initial State* : ?
- *Successor Function* : ?
- *Goal Test* : ?
- *Path Cost* : ?



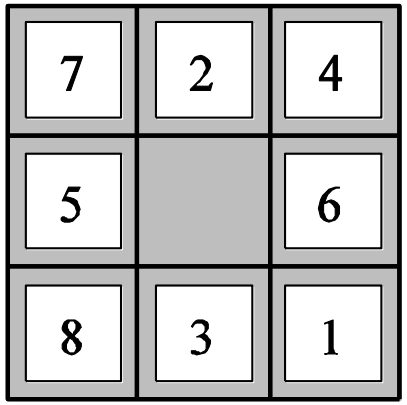
Start State



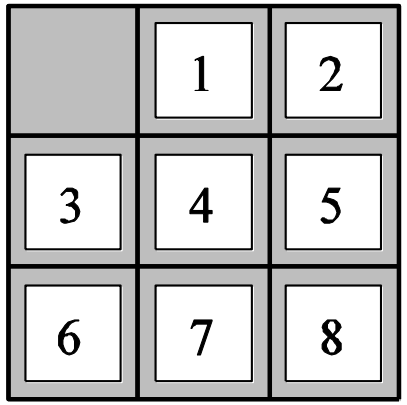
Goal State

Contoh Kasus: 8-Puzzles

- *State* : 8 kotak angka dan 1 kotak kosong
- *Initial State* : Sembarang state
- *Successor Function* : (ke kiri, ke kanan, ke atas atau ke bawah)
- *Goal Test* : Tersusun kotak angka yang diinginkan
- *Path Cost* : Setiap aksi bernilai 1 point



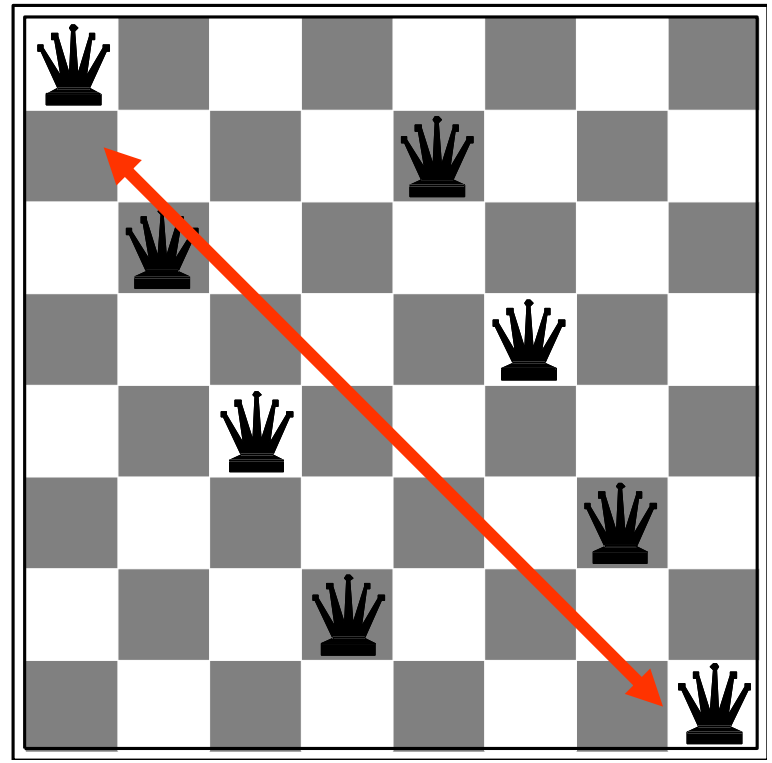
Start State



Goal State

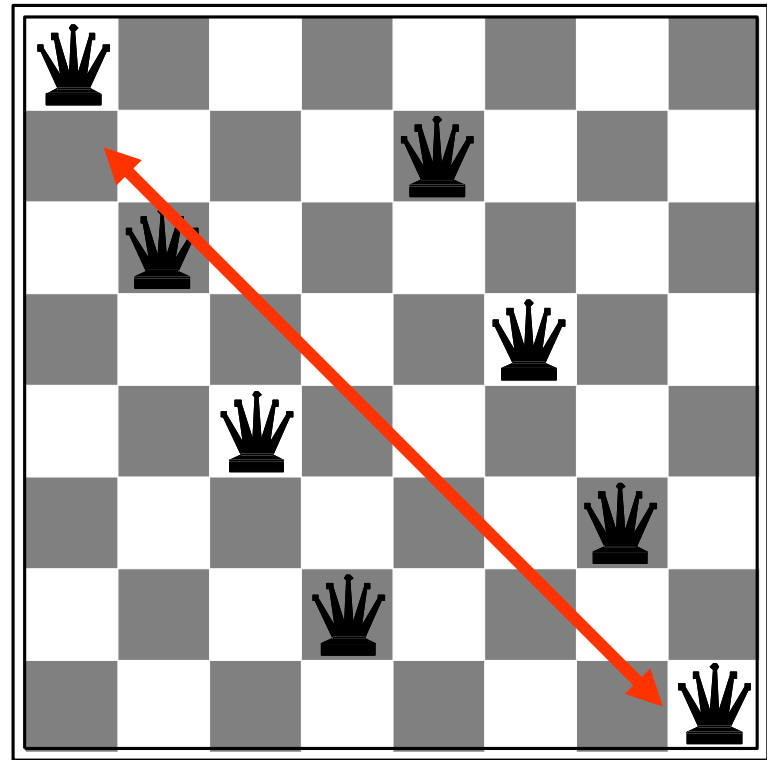
Contoh Kasus: 8-Queens

- *State* : ?
- *Initial State* : ?
- *Successor Function* : ?
- *Goal Test* : ?



Contoh Kasus: 8-Queens

- *State* : susunan 0..8 ratu pada papan catur
- *Initial State* : Tidak ada ratu pada papan catur
- *Successor Function* : Masukkan ratu ke papan catur
- *Goal Test* : Tidak ada ratu yang saling serang

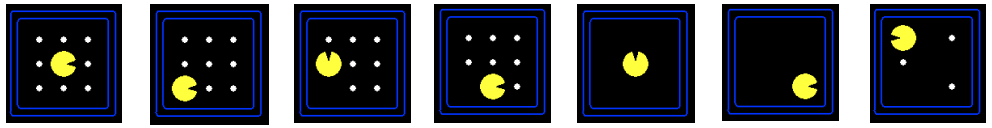


Real World Problem

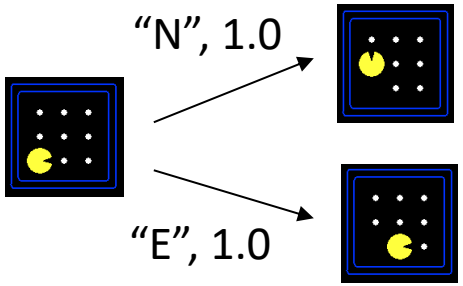
- Airline Travel Problem
- Touring Problem
- Traveling Salesman Problem
- VLSI Layout
- Robot Navigation
- Automatic Assembly Sequencing
- Internet Searching

Searching for Solution

- *Search problem* terdiri dari:
  - State space



  - Successor function
    (with actions, costs)



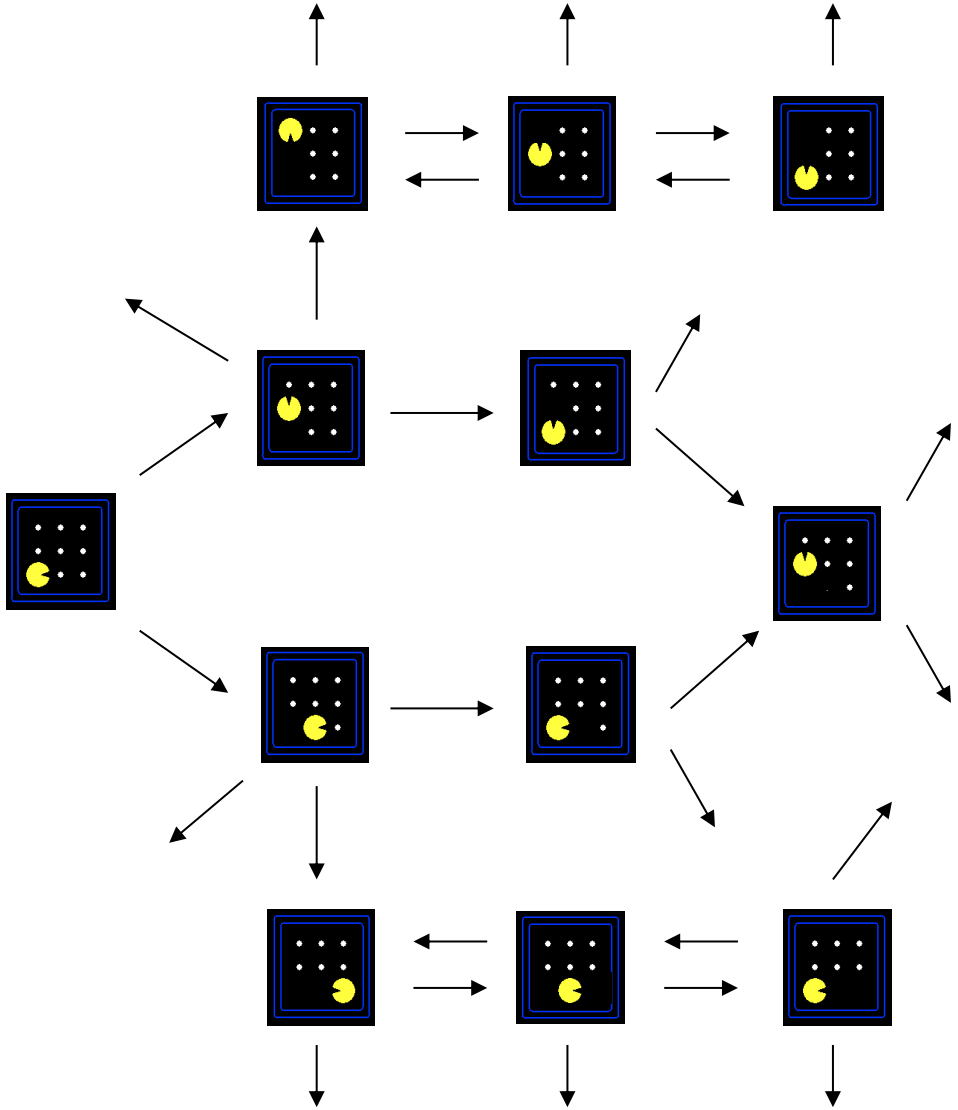
  - Start state dan Goal test

- Solusi adalah urutan aksi (rencana) yang akan dilakukan dari *start state* ke *goal state*

Sumber: Sergey Levine & Stuart Russell, University of California, Berkeley

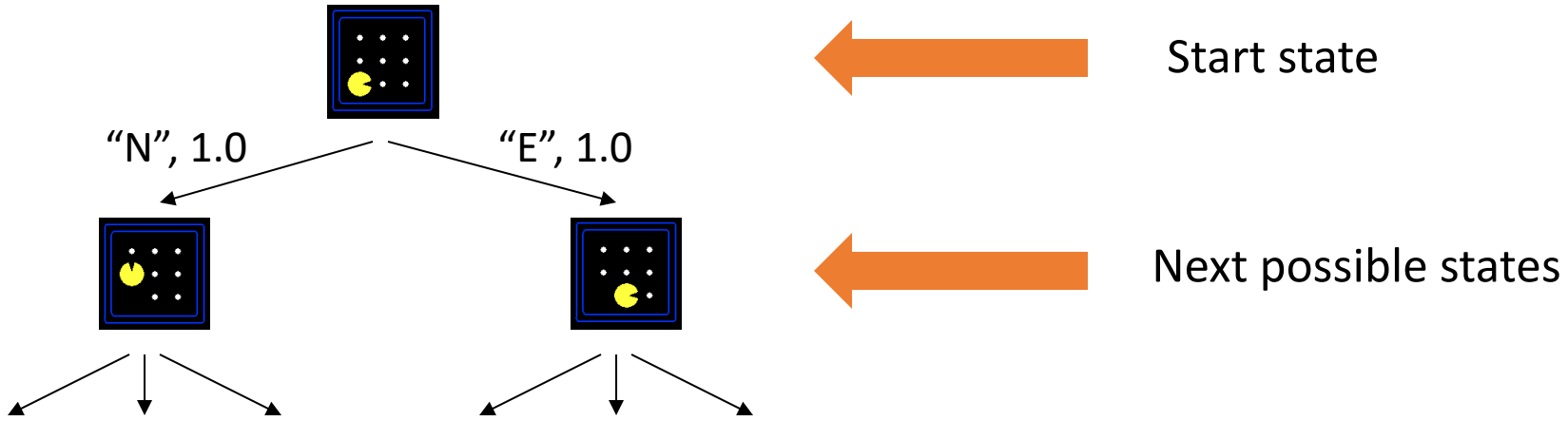
State Space Graph

- Sebuah *state space graph* merupakan representasi matematika pada sebuah *search problem*
  - Nodes merepresentasikan konfigurasi state
  - Arcs merepresentasikan successors (hasil aksi)
  - Goal test adalah himpunan *goal nodes* (bisa hanya satu node)
- Setiap state hanya muncul sekali!



Sumber: Sergey Levine & Stuart Russell, University of California, Berkeley

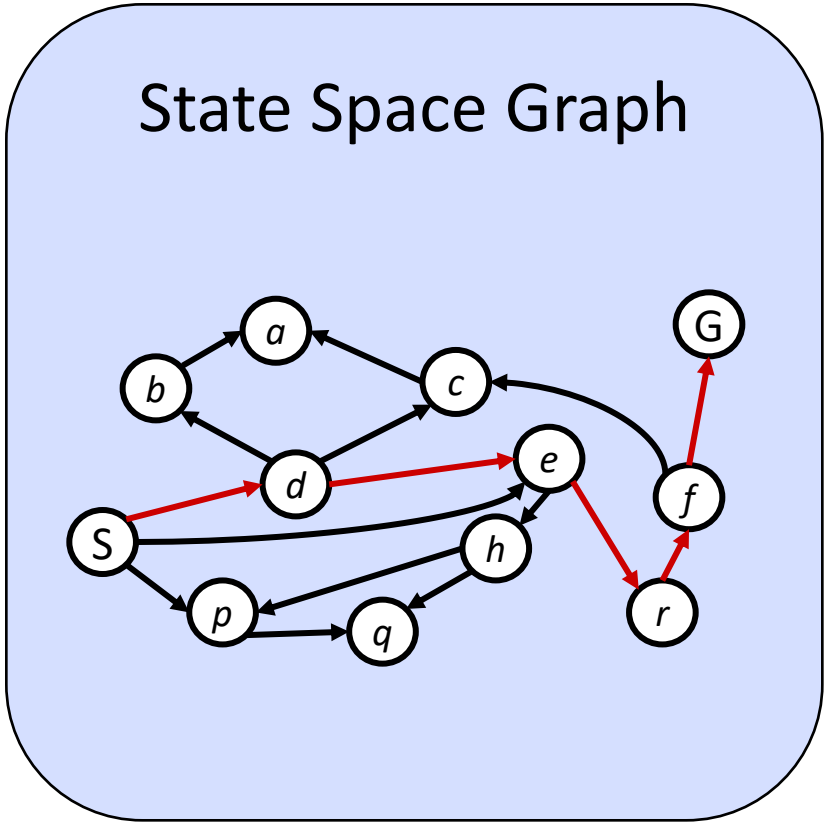
Search Trees

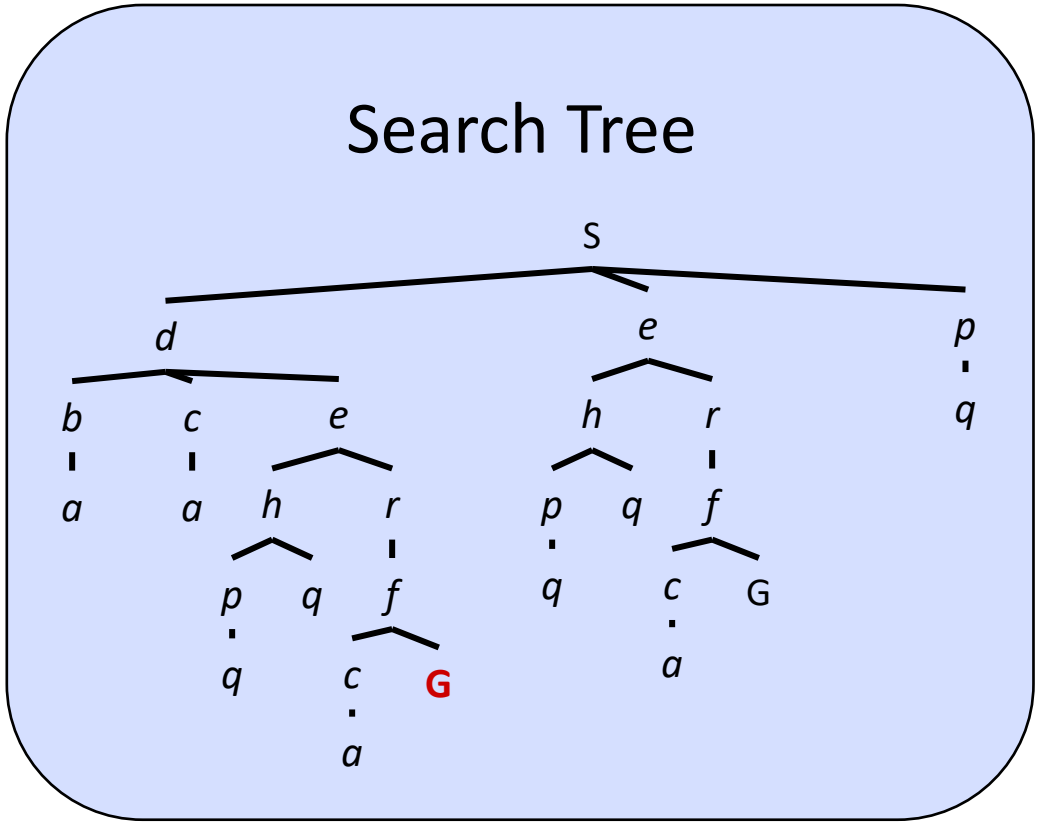


- Sebuah "what if" tree pada rencana dan luaran yang dihasilkan
- *Start state* adalah *root node*
- Anak merefer pada *successors*
- Nodes menunjukkan states
- Pada banyak problem, sebenarnya tidak pernah membangun *tree* secara utuh

Sumber: Sergey Levine & Stuart Russell, University of California, Berkeley

State Space Graphs vs. Search Trees

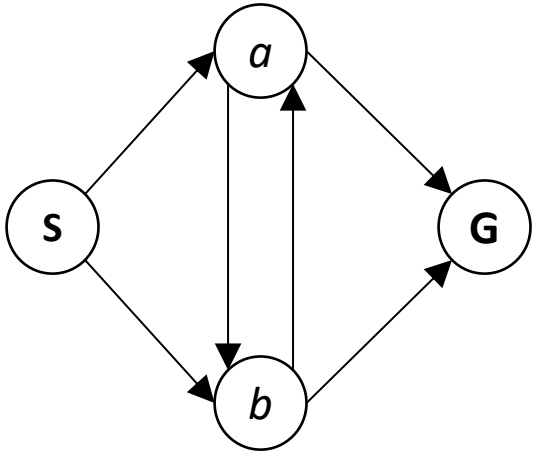




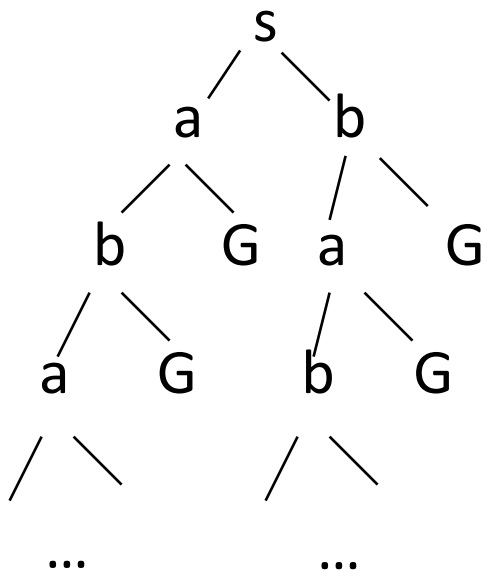
Sumber: Sergey Levine & Stuart Russell, University of California, Berkeley

State Space Graphs vs. Search Trees

Consider this 4-state graph:



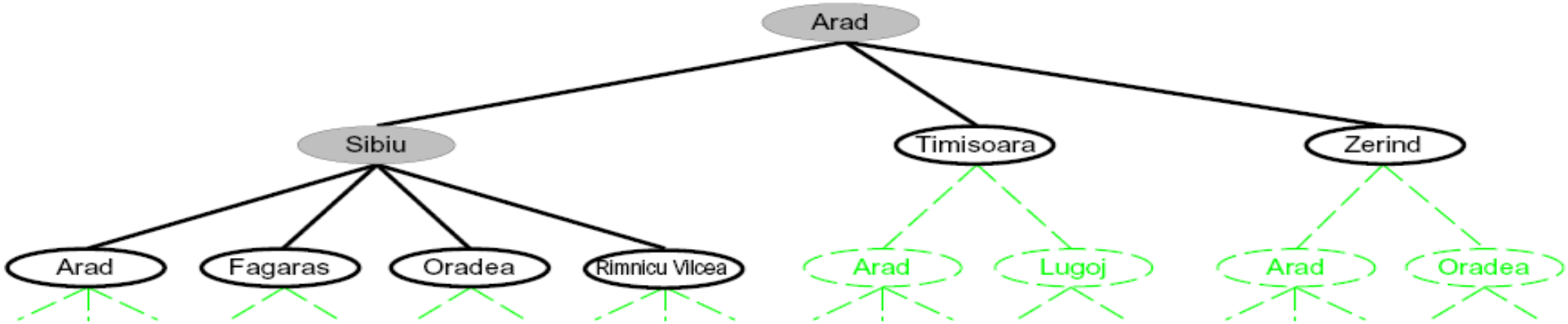
How big is its search tree (from S)?

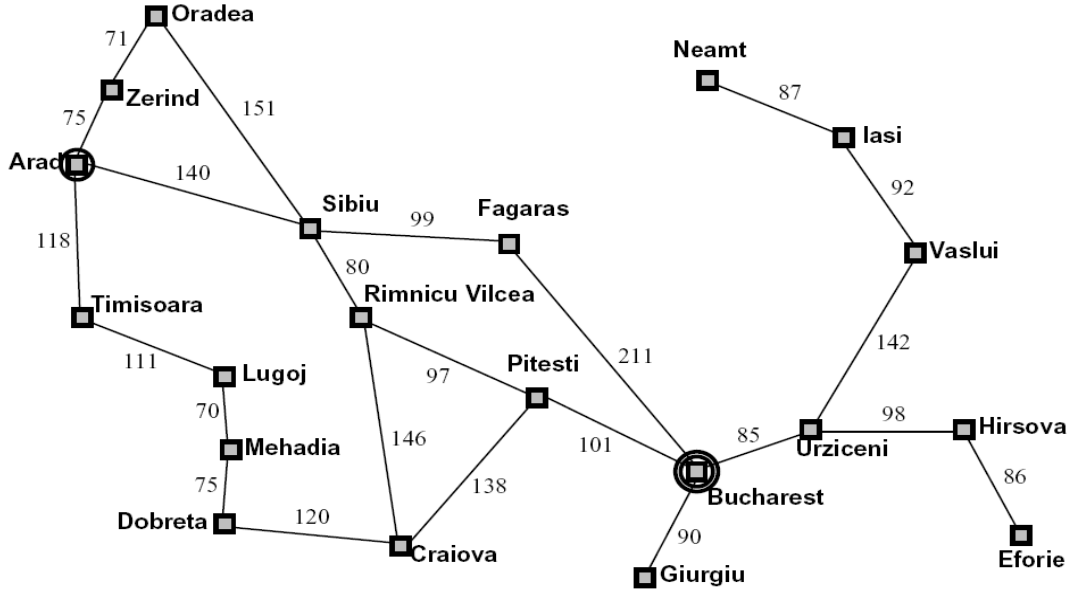


Important: Lots of repeated structure in the search tree!

Sumber: Sergey Levine & Stuart Russell, University of California, Berkeley

Pencarian dengan sebuah *Search Tree*

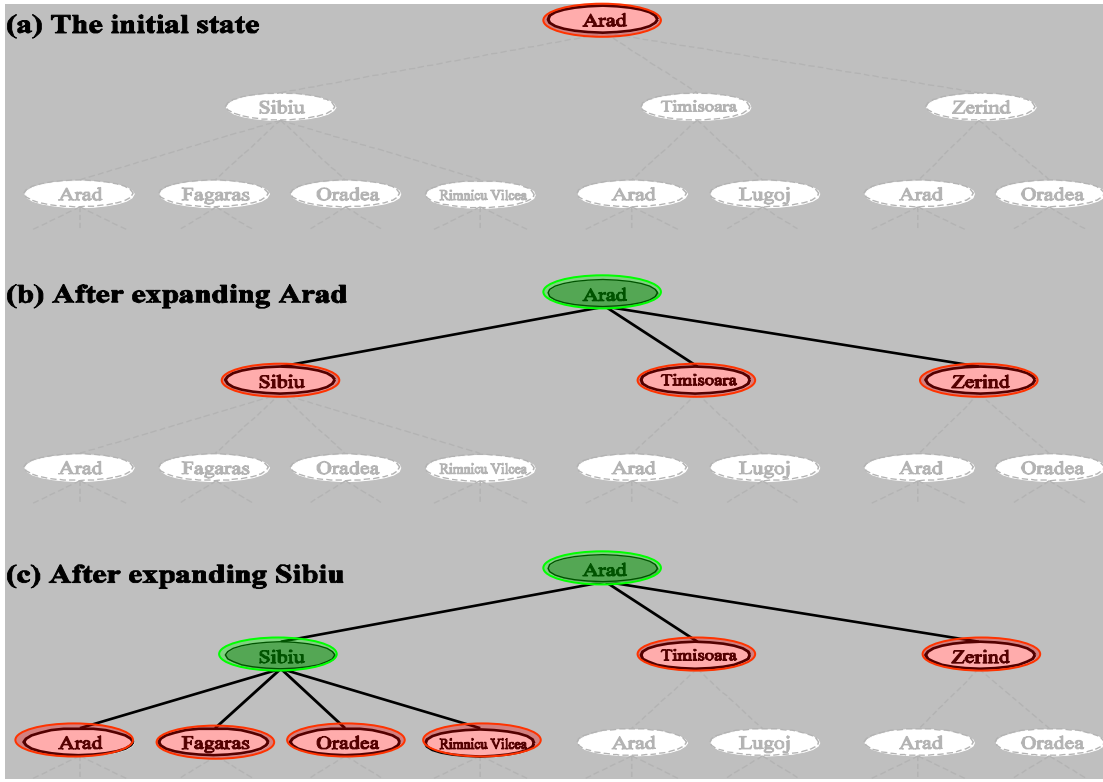




- Search:
  - Ekspansi potensi (tree nodes)
  - Mencoba untuk ekspansi sedikit mungkin node pada tree
  - Fringe → kumpulan *node* hasil generate yang belum diekspansi

General Tree Search

- Search Node → mulai *root of search tree* sebagai *Initial State*
- Cek node apakah *Goal*?
- Jika bukan, *expanding* → menghasilkan state baru
- Pertanyaan utama: *fringe nodes* yang akan di explore berikutnya? → *search strategy*
- Ide utama:
  - *Fringe*
  - *Expansion*
  - *Exploration strategy*

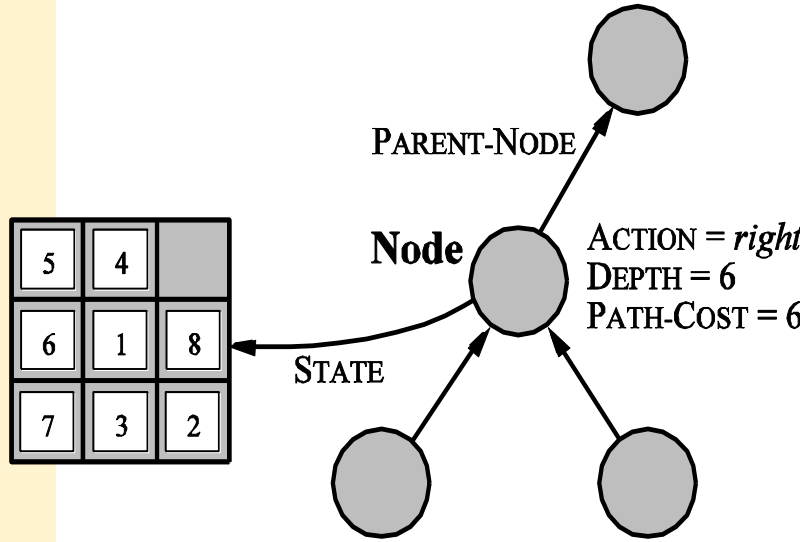



```
Function Tree-Search(problem,strategy) return a solution or a failure
   initialize the search tree using the initial state
   loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according strategy
        if the node contains a goal state then return the coresponding solution
        else expand the node and add the resulting node to the search tree
```

General Tree Search

```
Function Tree-Search(problem,fringe) return a solution or a failure
        fringe ← Insert(Make-Node(Initial-State[problem]),fringe)
        loop do
                if empty ?(fringe)  then return failure
                node ← Remove-First(fringe)
                if Goal-Test[problem] applied to State[node] succeds then
                        return Solution(node)
                fringe ← Insert-All(Expand(node,problem),fringe)
```

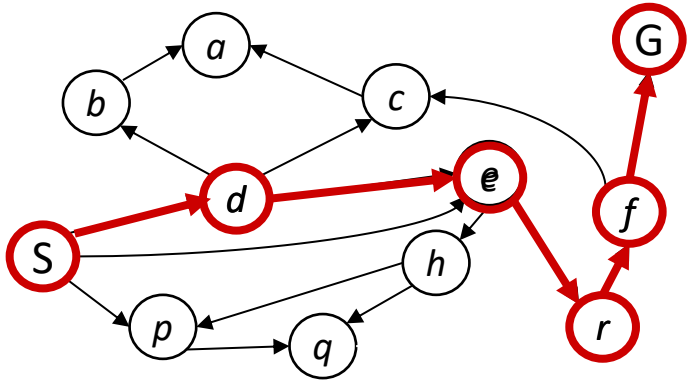
```
Function Expand(node,problem) return a set of nodes
        successor ← the empty set
        for each (action,result) in Successor-Fn[problem](State[node]) do
                s ← a new Node
                State[s] ← result
                Parent-node[s] ← node
                Action[s] ← action
                Path-Cost[s] ← Path-Cost[node] + Step-Cost(node,action,s)
                Depth[s] ← Depth[node] + 1
                add s to successor
         return successors
```

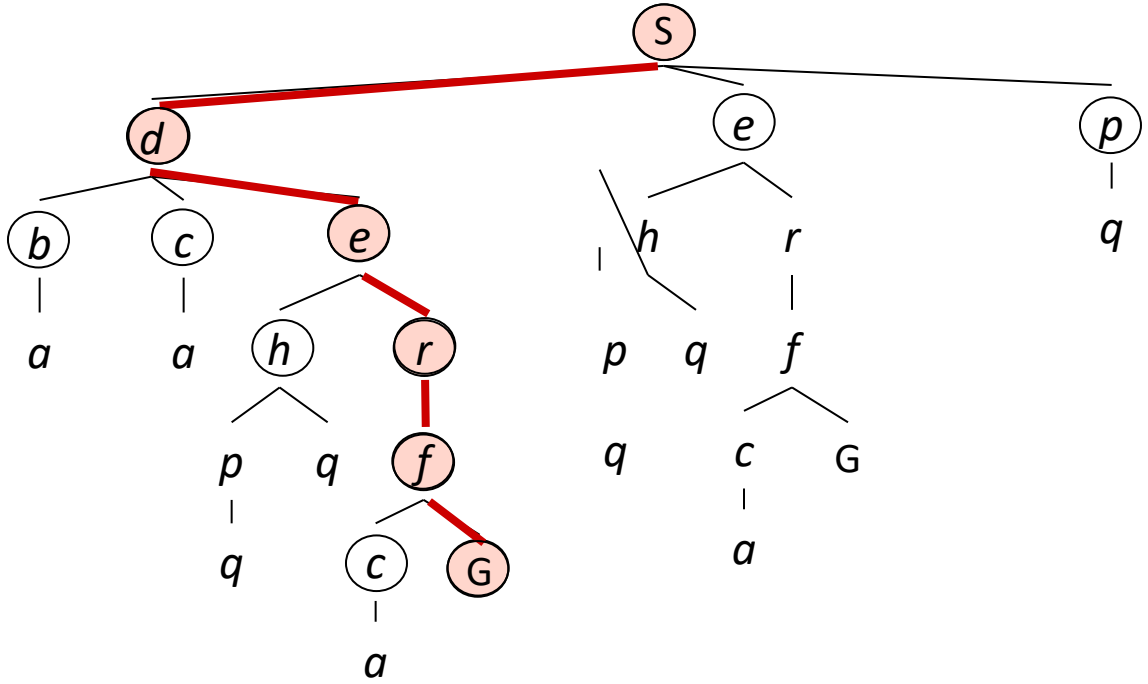


Representasi Node:

- State
- Parent-Node
- Action
- Path-Cost
- Depth

Contoh *Tree Search*





~~s~~

~~s → d~~

s → e

s → p

s → d → b

s → d → c

~~s → d → e~~

s → d → e → h

~~s → d → e → r~~

~~s → d → e → r → f~~

s → d → e → r → f → c

~~s → d → e → r → f → G~~

Search Strategies

- *Strategy* → memilih node yang akan diekspansi:
  - *Completeness* →Menemukan solusi jika ada
  - *Optimality* → Optimal solution (cost terkecil)
  - *Time Complexity* → berapa lama menemukan solusinya ?
  - *Space Complexity* → berapa banyak memory yang dibutuhkan ?

- *Time* dan *Space Complexity* bisa dilihat dari:
  - *b* → maks branch factor
  - *d* → depth dari solusi terkecil
  - *m* → maks depth dari state space

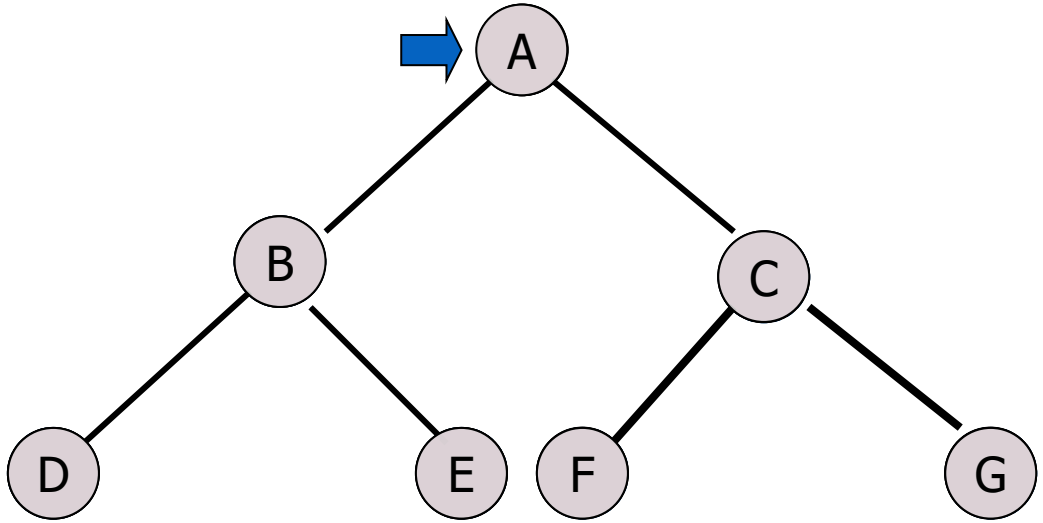
Uninformed Search Strategies

- Depth-first search
- Breadth-first search
- Depth-limited search
- Iterative deepening search
- Uniform-cost search
- Bidirectional search

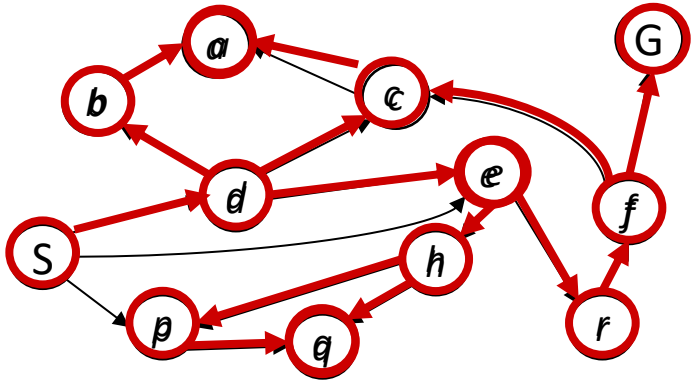
Depth-first Search

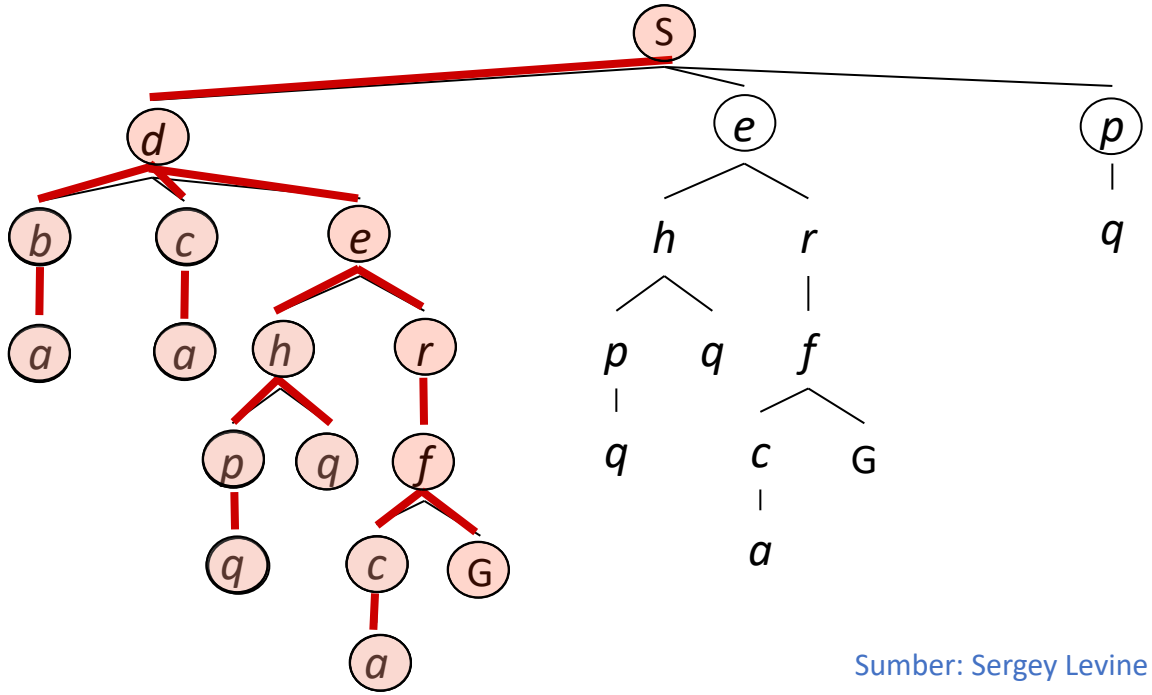
- Expand node yang terdalam
- Fringe → LIFO stack

- Complete ?
  - Tidak, jika memiliki *depth* tak terbatas,
  - Modifikasi dengan *limited depth*
  - Ya, jika depth terbatas
- Time ? O(b^m)
  - Bermasalah jika *m* jauh lebih besar dari *d*
- Space ? O(*bm*), linier space
- Optimal ? Tidak



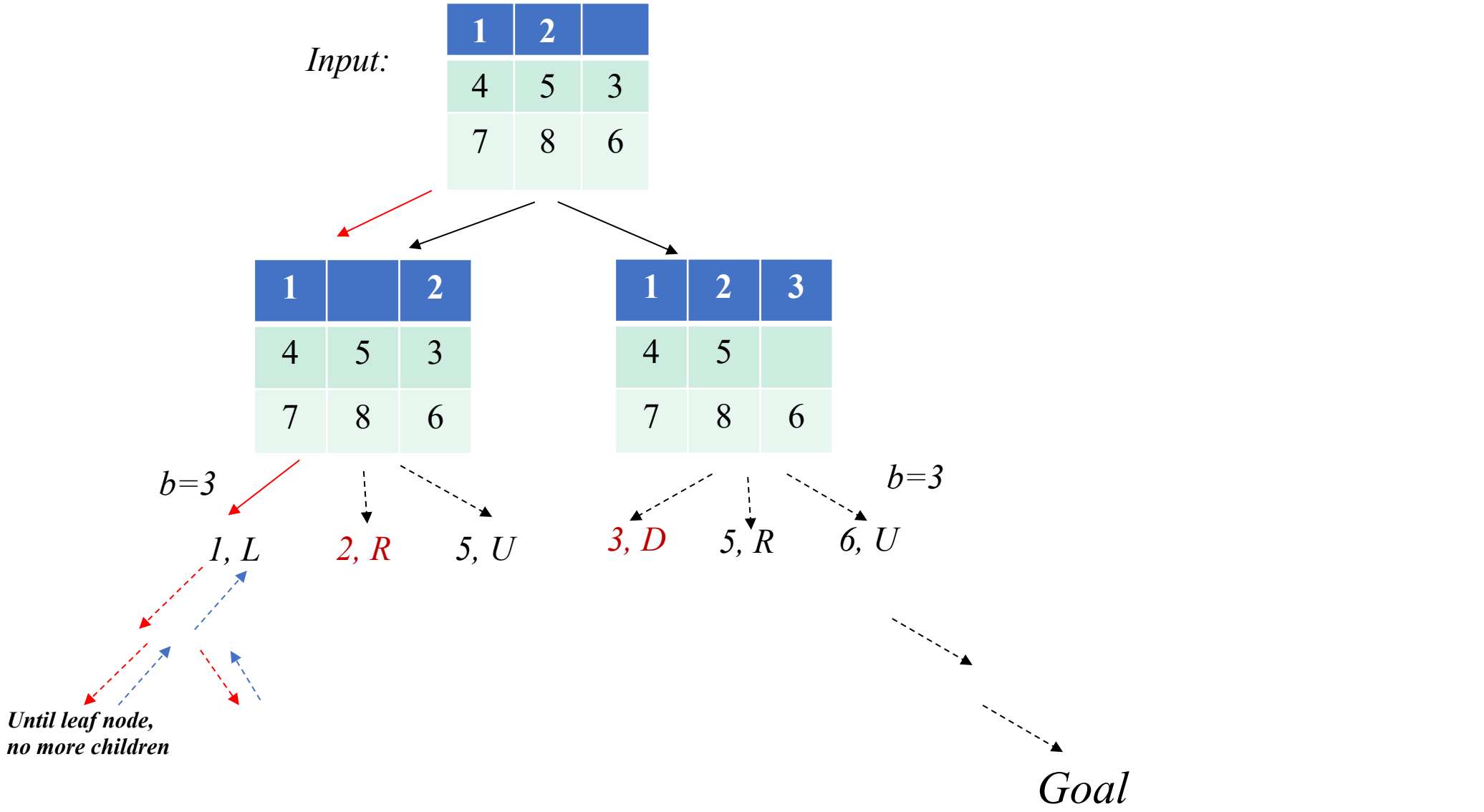
Depth-first Search



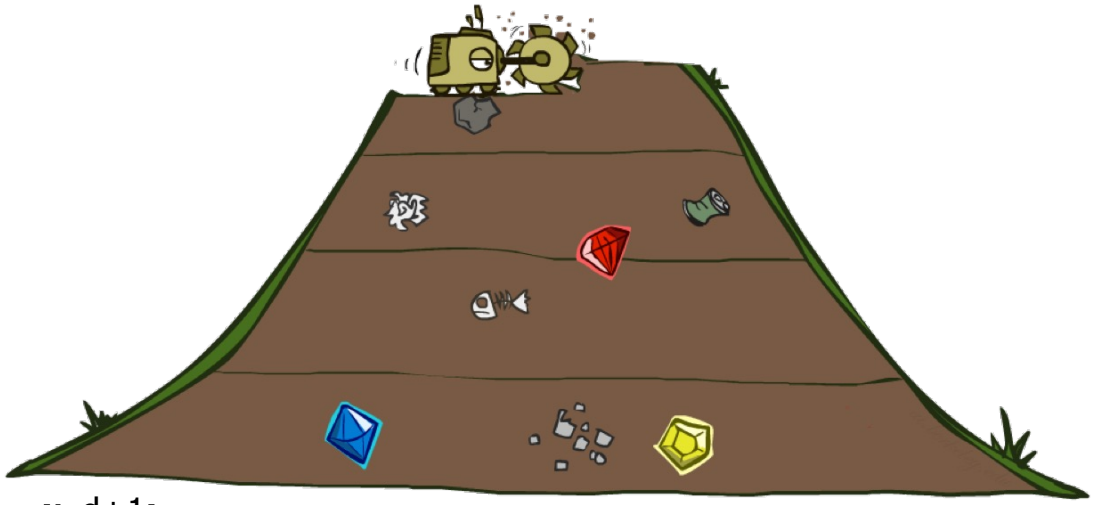


Sumber: Sergey Levine & Stuart Russell, University of California, Berkeley

Depth-first Search: 8-puzzles

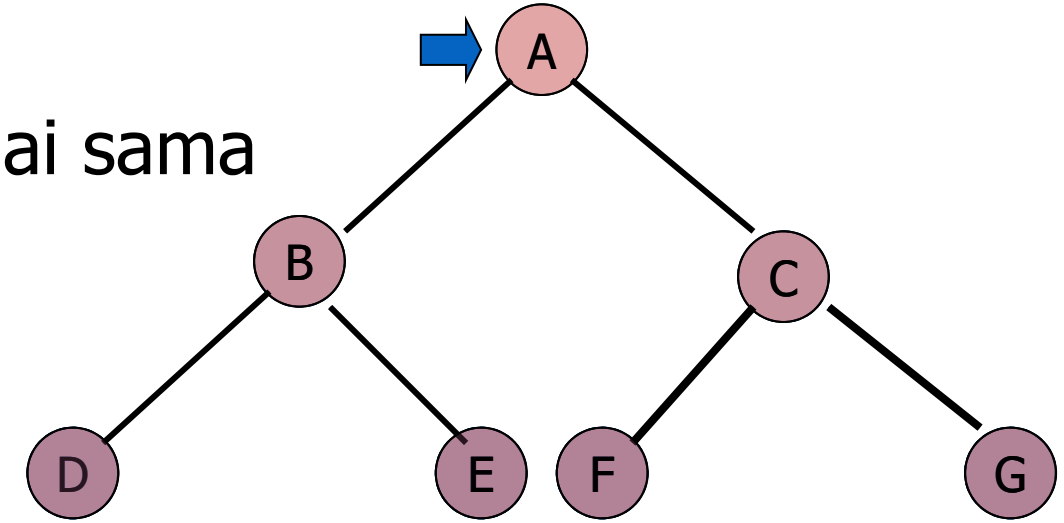


Breadth-first Search

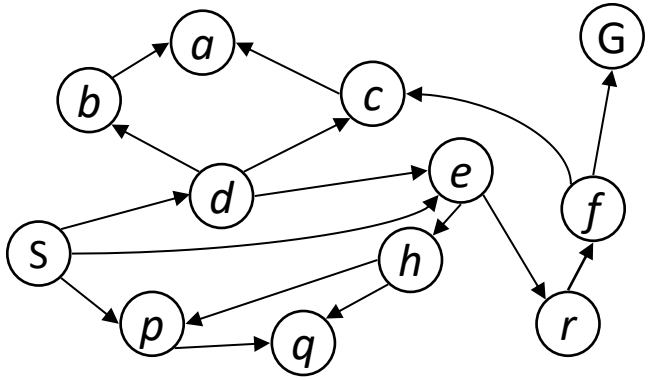


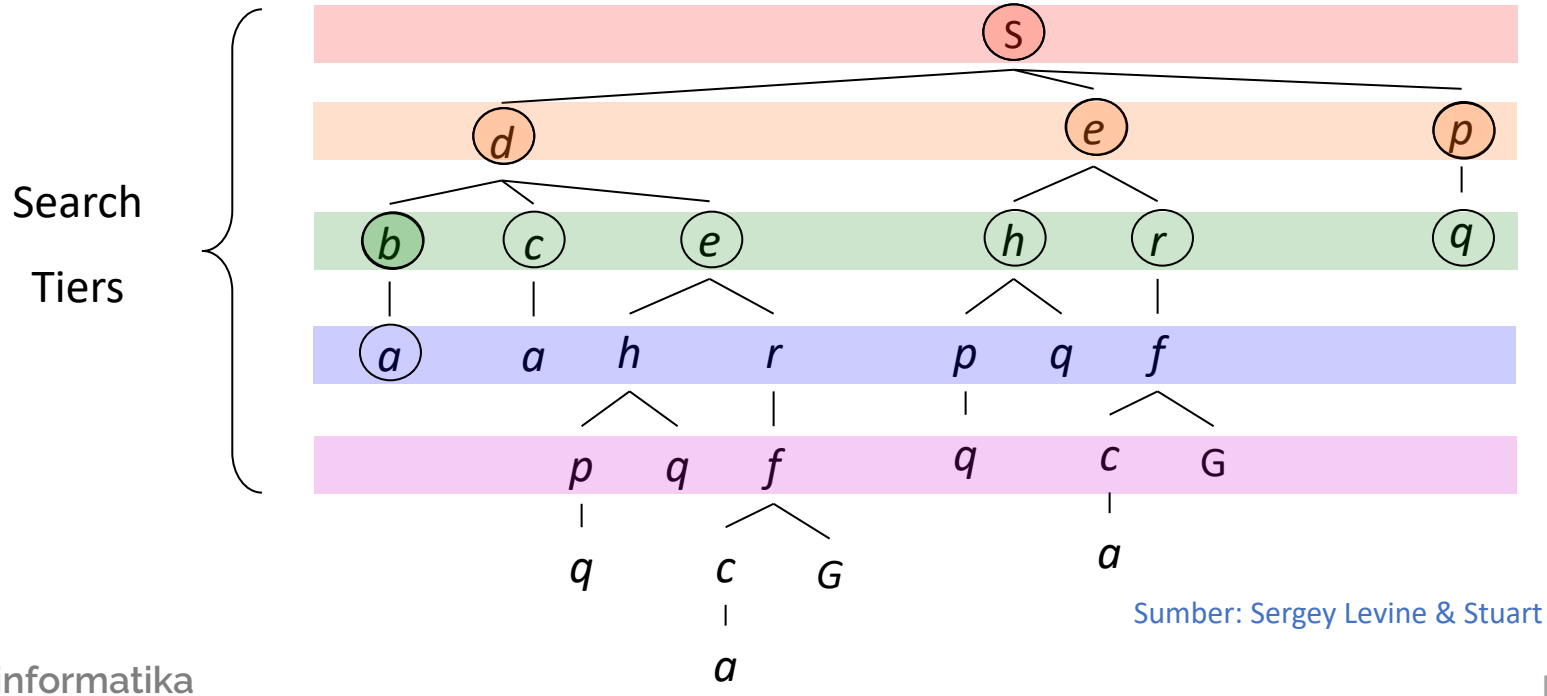
- Ekspand node ke samping
- Fringe → FIFO queue

- Complete ? Ya, jika *b* terbatas
- Time ? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d-1) = O(b^{d+1})$
- Space ? $O(b^{d+1})$
- Optimal ? Ya, jika semua aksi bernilai sama



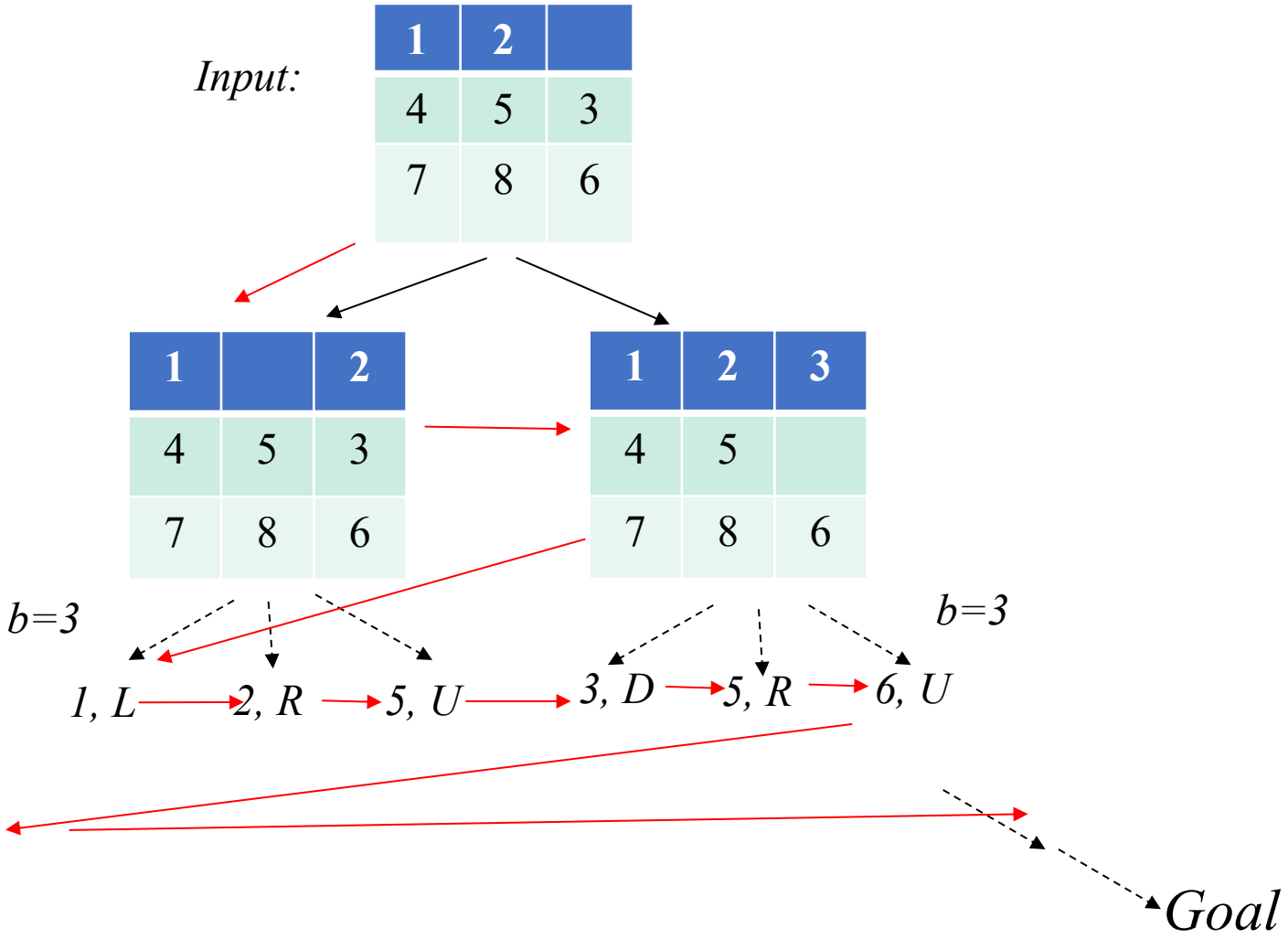
Breadth-first Search





Sumber: Sergey Levine & Stuart Russell, University of California, Berkeley

Breadth-first Search: 8-puzzles



Depth-limited Search

- Sama dengan DFS dengan deep limit l
- Implementasi dengan Rekursif

```
Function Depth-Limited-Search(problem,limit) return a solution or a failure/cutoff
         return Recursive-DLS(Make-Node(Initial-State[problem]),problem,limit)
Function Recursive-DLS(node,problem,limit) return a solution or failure/cutoff
         cutoff_occured? ← false
         if Goal-Test[problem](State[node]) then return solution
         else if Depth[node] = limit then return cutoff
         else for each successor in Expand(node,problem) do
                  result ← recursive-DLS(successor,problem,limit)
                  if result ← cutoff then cutoff_occured? ← true
                  else if result <> failure then return result
         if cutoff_occured? Then return cutoff
         else return failure
```

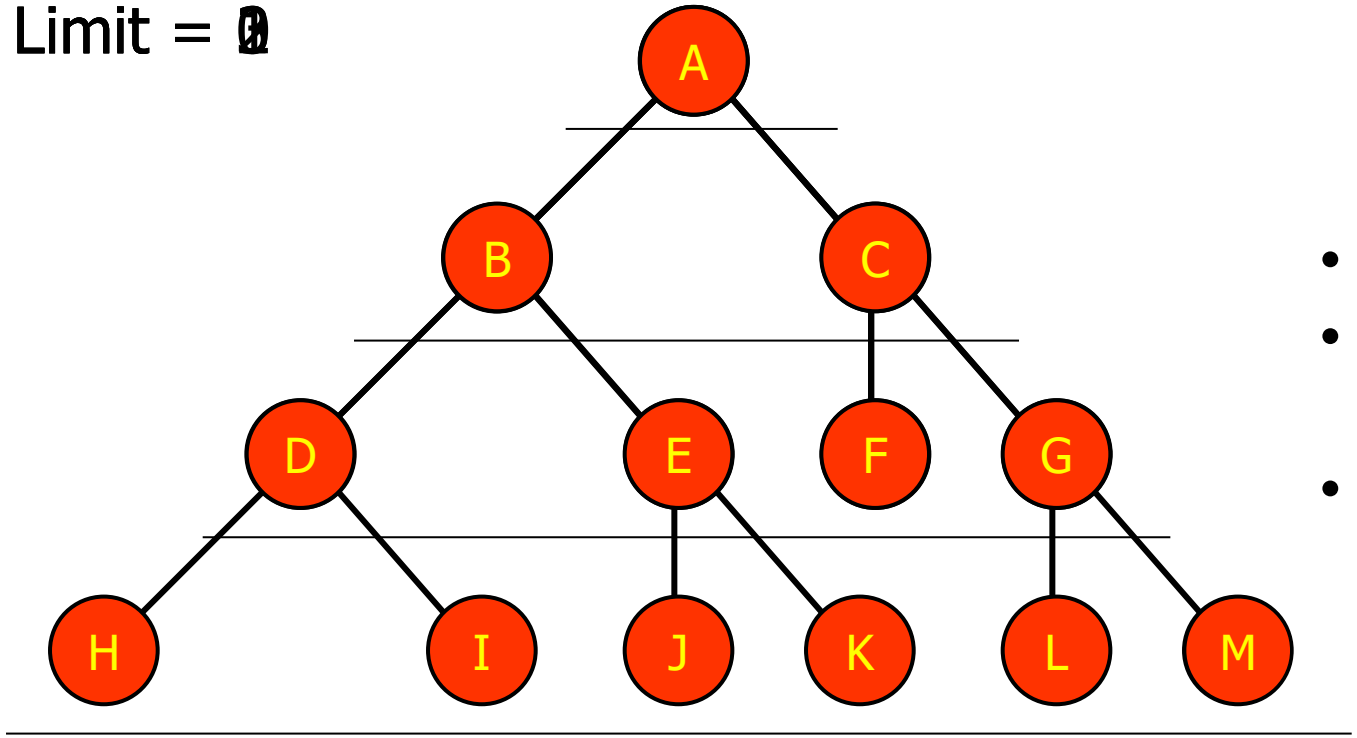
Iterative Deepening Search

- Merupakan DFS yang diiterasi berdasarkan kedalamannya
- Mengkombinasi DFS dan BFS
- Pencarian pada level 1 terlebih dahulu jika tidak menemukan goal pencarian berikutnya digenerate level 2, 3, dan seterusnya
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..

```
Function Iterative-Deepening-Search(problem) return a solution or a failure
        input problem // a problem
        for depth ← 0 to ∞ do
                result ← Depth-Limited-Search(problem,depth)
                if result <> cutoff then return result
```

Iterative Deepening Search

Limit = 0



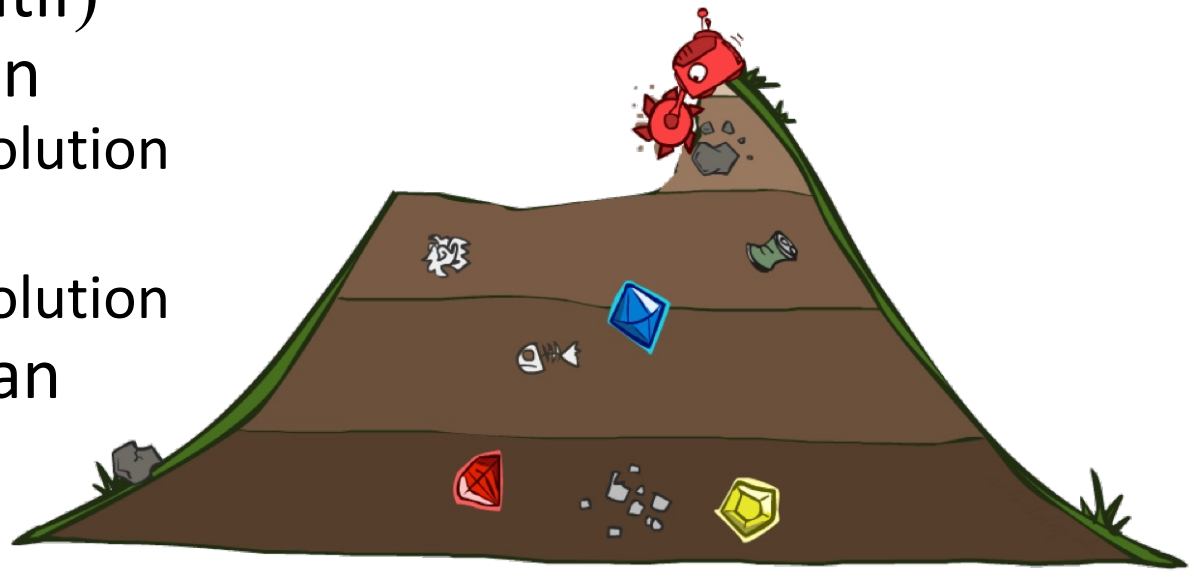
- Complete ? Ya
- Time ? $(d+1)b^0 + (d)b + (d-1)b^2 + \ldots + b^d = O(b^d)$
  - BFS menggenerate node sampai d + 1 sementara IDS hanya d → IDS lebih cepat dari BFS
- Space ? $O(bd)$
- Optimal ? Ya, jika memiliki cost yang sama untuk setiap aksi
- Perbandingan IDS vs BFS:

  ➔ b = 10 dan d = 5
  - N(IDS) : 50 + 400 + 3.000 + 20.000 + 100.000 = 123.450
  - N(BFS) : 10 + 100 + 1.000 + 10.000 + 100.000 + 999.990 = 1.111.100

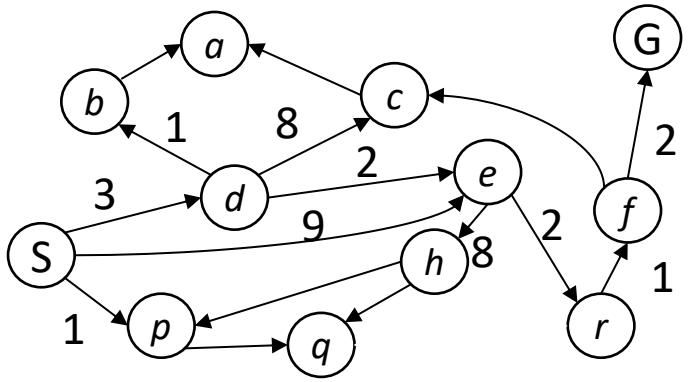
Uniform-cost Search

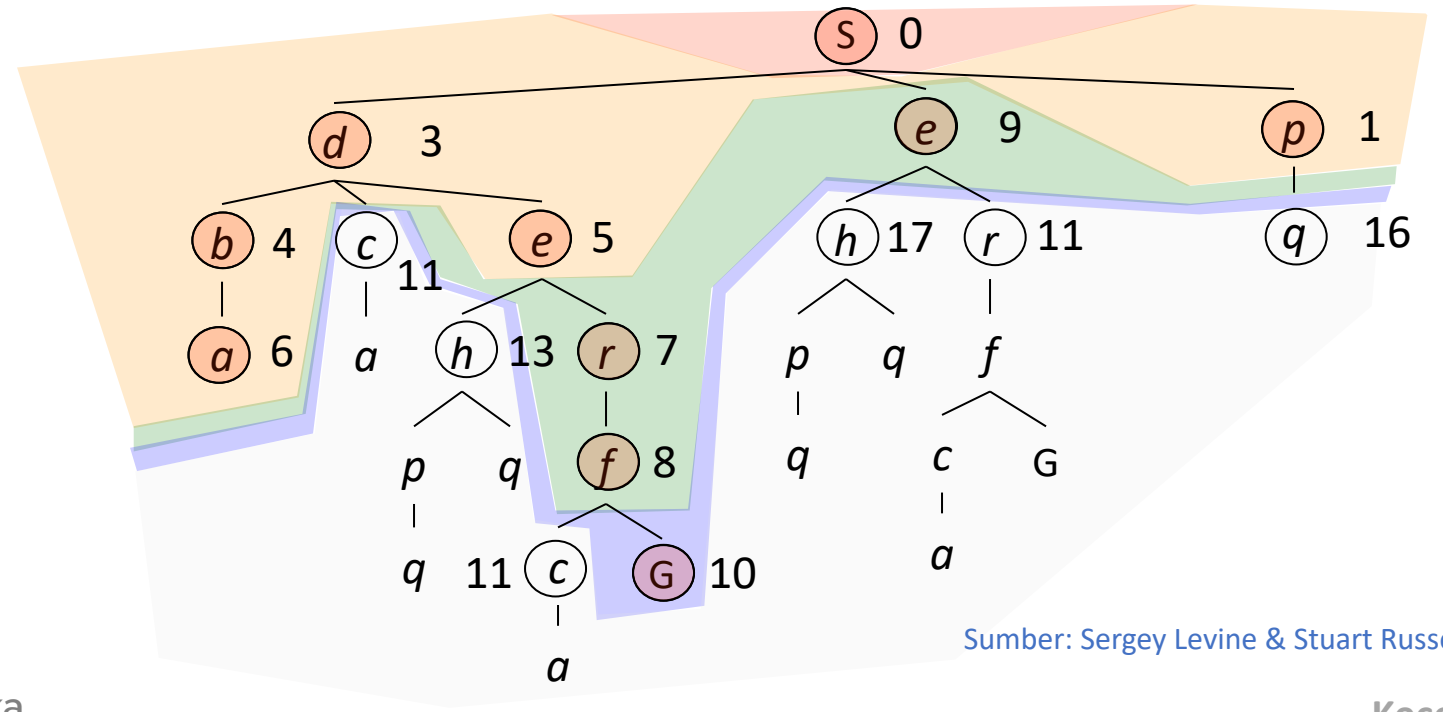
- Expand node dengan cost terkecil
- *Fringe* ➔ *priority queue (priority: cumulative cost)*
- Jika masing-masing node memiliki cost yang sama = BFS

- Complete ? Ya jika step cost >= Є (positif)
- Time ? O($b^{C^*/\epsilon}$) ; C* = Optimal solution
  - Jumlah node dengan g <= cost optimal solution
- Space ? O($b^{C^*/\epsilon}$)
  - Jumlah node dengan g <= cost optimal solution
- Optimal ? Ya, node diekspand → urutan penambahan *g(n)*



Uniform-cost Search



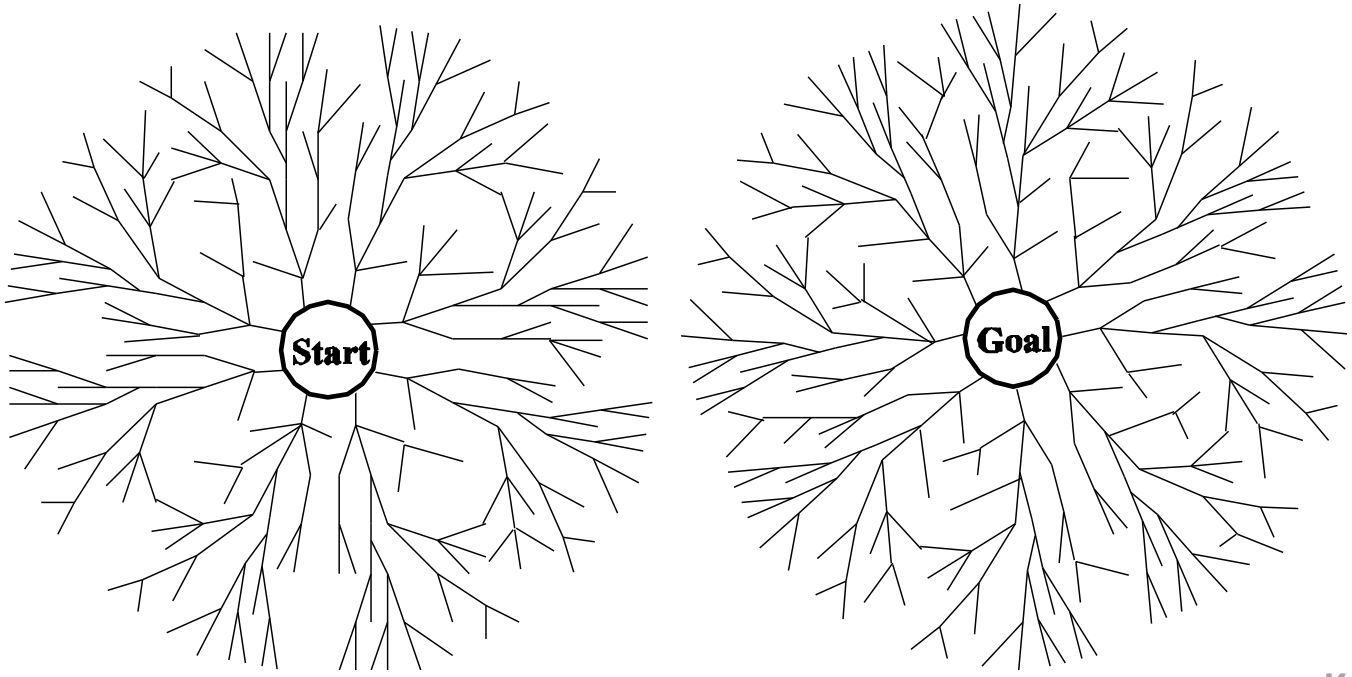


Cost contours

Sumber: Sergey Levine & Stuart Russell, University of California, Berkeley

Bidirectional Search

- Mencari dari 2 arah secara simultan
- Motivasi → $b^{d/2} + b^{d/2}$ jauh lebih kecil dari b^d
- Misal d = 6, masing2 menggunakan BFS → depth=3, b=10; dengan bidirectional hanya 22.200 node sedangkan BFS standar mencapai 11.111.000 node



Perbandingan: Uninformed search strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

Evaluation of search algorithms. b is the branching factor; m is the maximum depth of the search tree; d is the depth of the shallowest solution, or m is when there is no solution; ℓ is the depth limit.
Superscript caveats are as follows: [1]complete if b is finite, and the state space either has a solution or is finite.
[2]complete if all action costs are $\geq \epsilon > 0$; [3]cost-optimal if action costs are all identical; [4]if both directions are breadth-first or uniform-cost.

Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS?

Sumber: Sergey Levine & Stuart Russell, University of California, Berkeley

Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS?

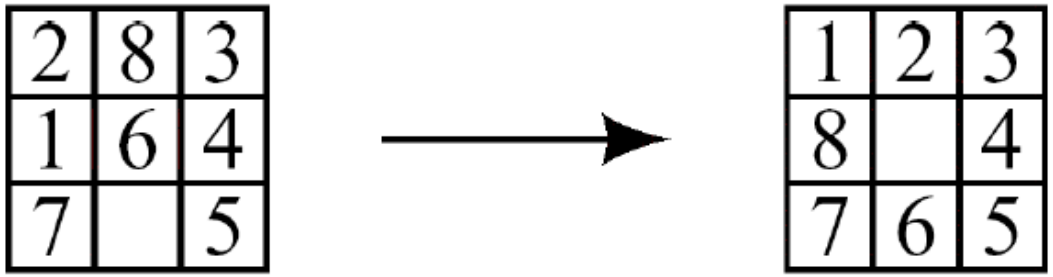
Sumber: Sergey Levine & Stuart Russell, University of California, Berkeley

Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS?

Sumber: Sergey Levine & Stuart Russell, University of California, Berkeley

Tugas 1 Individu

- Batas waktu pengumpulan: 1 Maret 2023
- Terdapat kasus 8-puzzles dengan informasi state awal dan goal sebagai berikut:



- Selesaikan kasus 8-puzzles diatas menggunakan 2 metode *uninformed search*

- TERIMA KASIH -