Problem solving agent

Representasi masalah: state space

Pencarian solusi: search

Search strategies

Ringkasan

IKI 30320: Sistem Cerdas
Kuliah 3: Problem-Solving Agent & Search

Ruli Manurung

Fakultas Ilmu Komputer
Universitas Indonesia

3 September 2007

Outline

1. Problem solving agent
2. Representasi masalah: state space
3. Pencarian solusi: search
4. Search strategies
5. Ringkasan

Outline

1 Problem solving agent

2 Representasi masalah: state space

3 Pencarian solusi: search

4 Search strategies

5 Ringkasan

Problem-Solving Agent

- Di kuliah yang lalu kita melihat contoh reflex agent: tidak cocok untuk masalah besar!
- Goal-based agent: memiliki tujuan, memungkinkannya evaluasi tindakan dan memilih yang terbaik.
- Di kuliah ini kita membahas satu kemungkinan jenis goal-based agent: problem-solving agent
- Problem-solving agent menghasilkan solusi dalam bentuk serangkaian tindakan yang diambil untuk mencapai tujuan.
- Apa problem-nya? Apa solution-nya?

Mekanisme kerja Problem-Solving Agent

- Perumusan tujuan (goal formulation): tentukan tujuan yang ingin dicapai
- Perumusan masalah (problem formulation): tentukan tindakan (action) dan keadaan (state) yang dipertimbangkan dalam mencapai tujuan
- Pencarian solusi masalah (searching): tentukan rangkaian tindakan yang perlu diambil untuk mencapai tujuan
- Pelaksanaan solusi (execution): laksanakan rangkaian tindakan yang sudah ditentukan di tahap sebelumnya

Agent program Problem Solving Agent

function SIMPLEPROBLEMSOLVINGAGENT (*percept*)
returns *action*

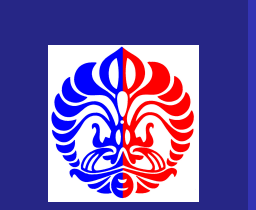
state ← UPDATESTATE(*state*, *percept*)
if seq is empty **then**
    goal ← FORMULATEGOAL (*state*, *goal*)
    problem ← FORMULATEPROBLEM (*state*, *goal*)
    seq ← SEARCH (*problem*)
action ← RECOMMENDATION (*seq*, *state*)
seq ← REMAINDER (*seq*, *state*)
return *action*

Sifat Problem-Solving Agent

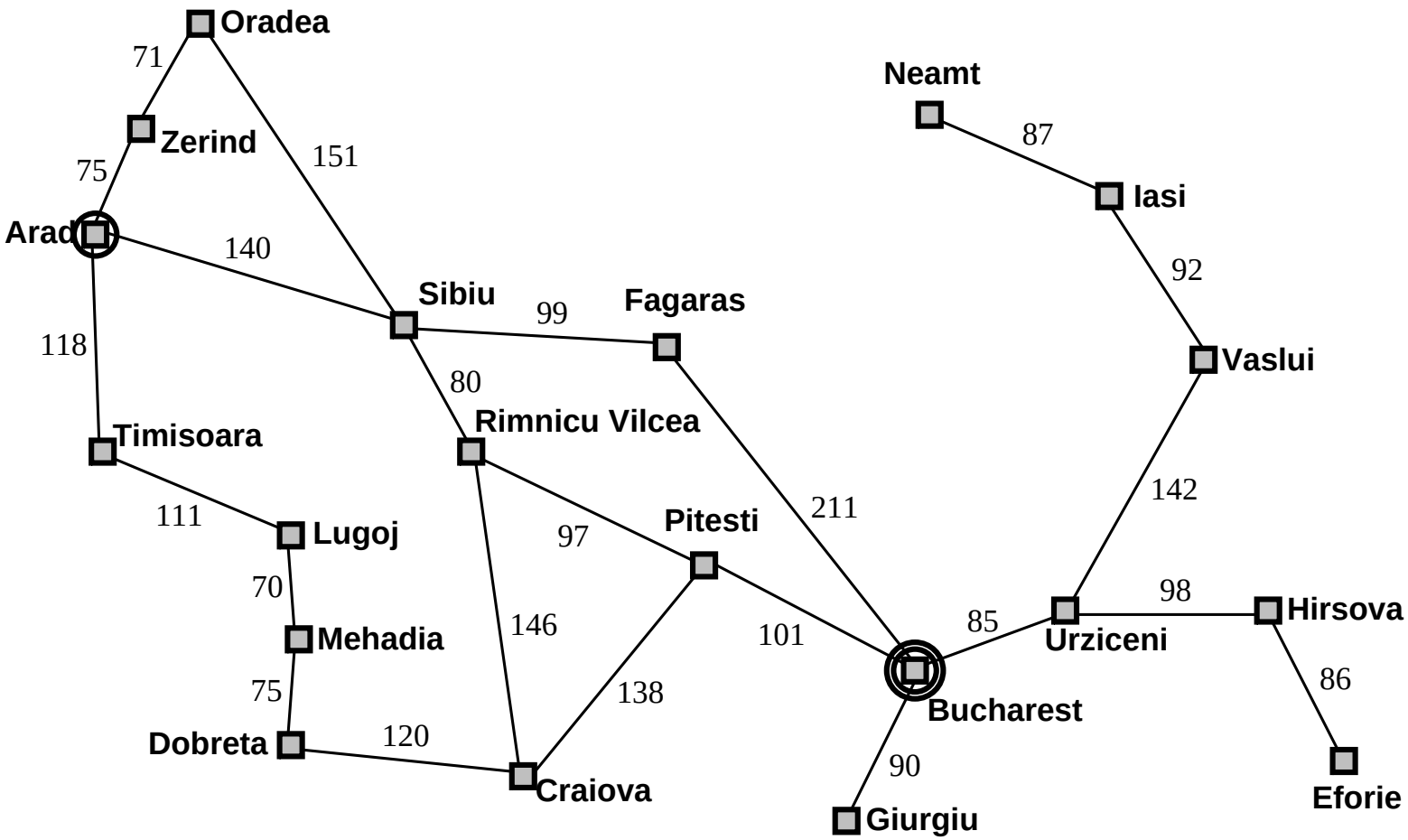
- Biasanya problem solving agent mengasumsikan bahwa environment-nya:
  - fully observable
  - deterministic
  - sequential
  - static
  - discrete
- Setelah mencari solusi, agent ini melaksanakan tindakan dengan "mata tertutup" $\rightarrow$ tidak melihat percept!

Contoh: Turis di Rumania

- Suatu "tourist agent" yang sedang berlibur di Rumania, kini berada di Arad. Besok, dia harus terbang dari bandara Bucharest.
- Perumusan tujuan: berada di Bucharest
- Perumusan masalah:
  - Tindakan (action): menyetir dari kota ke kota
  - Keadaan (state): kota-kota di Rumania
- Pencarian solusi: rangkaian kota yang dituju, mis: Arad, Sibiu, Fagaras, Bucharest



Peta Rumania



Outline

1 Problem solving agent

2 Representasi masalah: state space

3 Pencarian solusi: search

4 Search strategies

5 Ringkasan

Perumusan masalah sebagai state space

- Initial state: keadaan awal di mana si agent mulai, mis: $BeradaDi(Arad)$
- Possible actions: tindakan yang dapat dilakukan si agent, mis: $Nyetir(Arad, Zerind)$.
- Sebuah successor function S menentukan untuk suatu state X, himpunan tindakan yang mungkin diambil beserta state yang dihasilkan. Contoh:
  $X = BeradaDi(Arad)$
  $S(X) = \{< Nyetir(Arad, Zerind), BeradaDi(Zerind) >, ...\}$
- Initial state dan successor function mendefinisikan state space: himpunan semua state yang dapat dicapai dari initial state. Dapat direpresentasikan sebagai graph. Path dalam state space adalah serangkaian state (dihubungkan serangkaian action).

Menelusuri sebuah state space

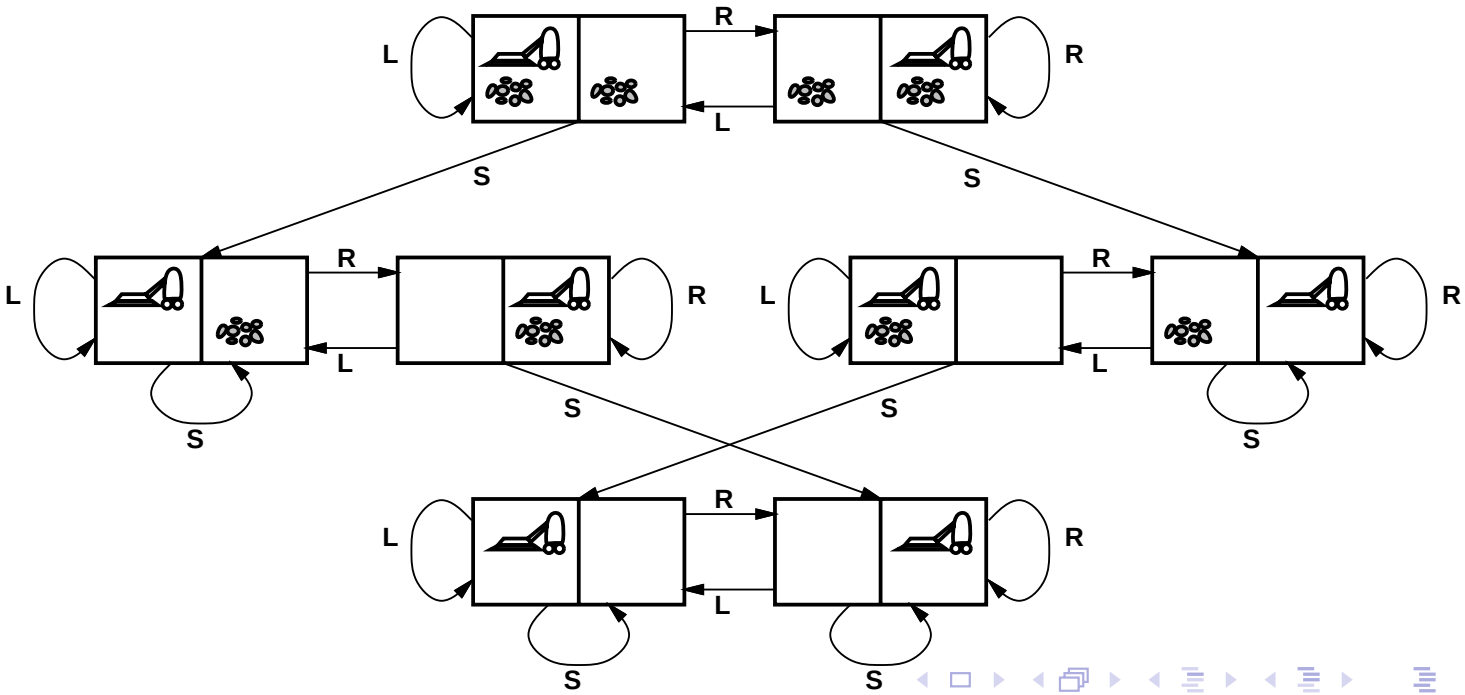
- Goal test: penentuan apakah suatu state adalah tujuan yang ingin dicapai.
  - Eksplisit: himpunan goal state, mis: $\{BeradaDi(Bucharest)\}$.
  - Implisit: deskripsi tujuan, mis: dalam catur $\rightarrow$ skak mat.
- Path cost function: sebuah fungsi yang memberikan nilai numerik terhadap setiap path. Fungsi ini merefleksikan performance measure si agent.
- Asumsi path cost function = $\sum$ step cost: cost action a dari state x ke y: $c(x, a, y)$.
- Sebuah solusi adalah path dari initial state ke goal state.
- Sebuah solusi optimal adalah solusi dengan path cost function minimal.

Memilih state space

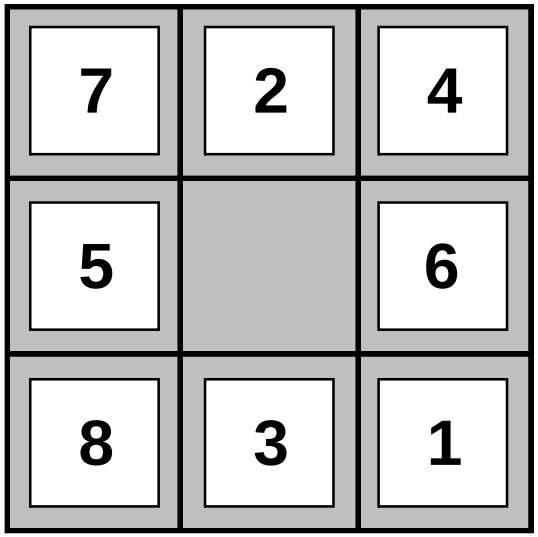
- Dunia nyata luar biasa kompleks dan rumit! State space harus merupakan abstraksi masalah supaya bisa dipecahkan.
  - State = himpunan "keadaan nyata". Mis: $BeradaDi(Arad)$ – dengan siapa? kondisi cuaca?
  - Action = kombinasi berbagai "tindakan nyata". Mis: $Nyetir(Arad, Sibiu)$ – jalan tikus, isi bensin, istirahat, dll.
  - Solution = representasi berbagai "path nyata" yang mencapai tujuan
- Abstraksi ini membuat masalah yang nyata lebih mudah dipecahkan.

Contoh: VacuumCleanerWorld

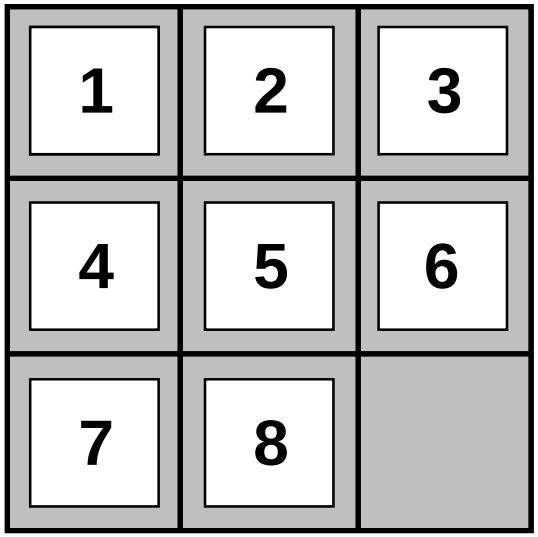
- State: lokasi agent, status debu
- Possible action: *DoKeKiri*(L), *DoKeKanan*(R), *DoSedot*(S)
- Goal test: apakah semua ruangan bebas debu?
- Path cost: asumsi step cost sama untuk semua action, mis: 1. Path cost = jumlah langkah dalam path.
- Successor function mendefinisikan state space sbb:



Contoh: 8-Puzzle



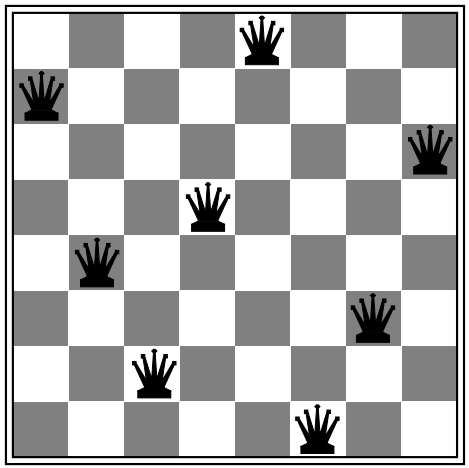
Start State



Goal State

- State: lokasi 8 buah angka dalam matriks 3x3
- Possible action: *Kiri*, *Kanan*, *Atas*, *Bawah*
- Goal test: apakah konfigurasi angka seperti goal state di atas
- Path cost: asumsi step cost = 1. Path cost = jumlah langkah dalam path.

Contoh: 8-Queens Problem



Letakkan 8 bidak menteri (*queen*!) sedemikian sehingga tidak ada yang saling "makan" (menteri bisa makan dalam satu baris, kolom, diagonal).

- State: Papan catur dengan n bidak menteri, $0 \leq n \leq 8$.
- Initial state: Papan catur yang kosong.
- Possible action: Letakkan sebuah bidak menteri di posisi kosong.
- Goal test: 8 bidak menteri di papan, tidak ada yang saling makan.

Masalah state space... combinatorial explosion!

- Dengan definisi masalah demikian, ada $64 \times 63 \times \ldots \times 57 \approx 1.8 \times 10^{14}$ path!
- Mustahil kita selesaikan dengan komputer tercanggih apapun. Definisi masalah bisa diperjelas:
  - State: Papan catur dengan n bidak menteri, $0 \leq n \leq 8$, satu per kolom di n kolom paling kiri.
  - Possible action: Letakkan sebuah bidak menteri di posisi kosong di kolom paling kiri yang belum ada bidaknya sehingga tidak ada yang saling makan.
- State space sekarang ukurannya tinggal 2057, dan mudah dipecahkan.
- Perumusan masalah yang tepat bisa berakibat drastis!
- Meskipun demikian, untuk $n = 100$: 10^{400} vs. 10^{52}...

Outline

1 Problem solving agent

2 Representasi masalah: state space

3 Pencarian solusi: search

4 Search strategies

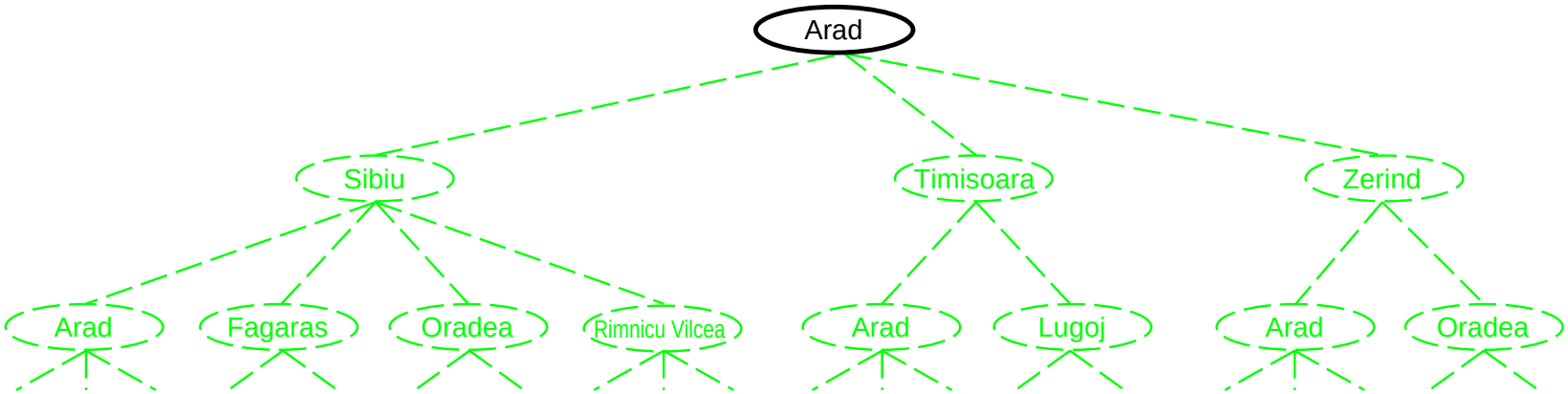
5 Ringkasan

Mencari solusi melalui search tree

- Setelah merumuskan masalah → cari solusinya menggunakan sebuah search algorithm
- Search tree merepresentasikan *state space*.
- Search tree terdiri dari kumpulan node: struktur data yang merepresentasikan suatu *state* pada suatu *path*, dan memiliki parent, children, depth, dan path cost.
- Root node merepresentasikan initial state.
- Penerapan *successor function* terhadap (*state* yang diwakili) *node* menghasilkan *children* baru → ini disebut node expansion.
- Kumpulan semua node yang belum di-*expand* disebut fringe (pinggir) sebuah *search tree*.

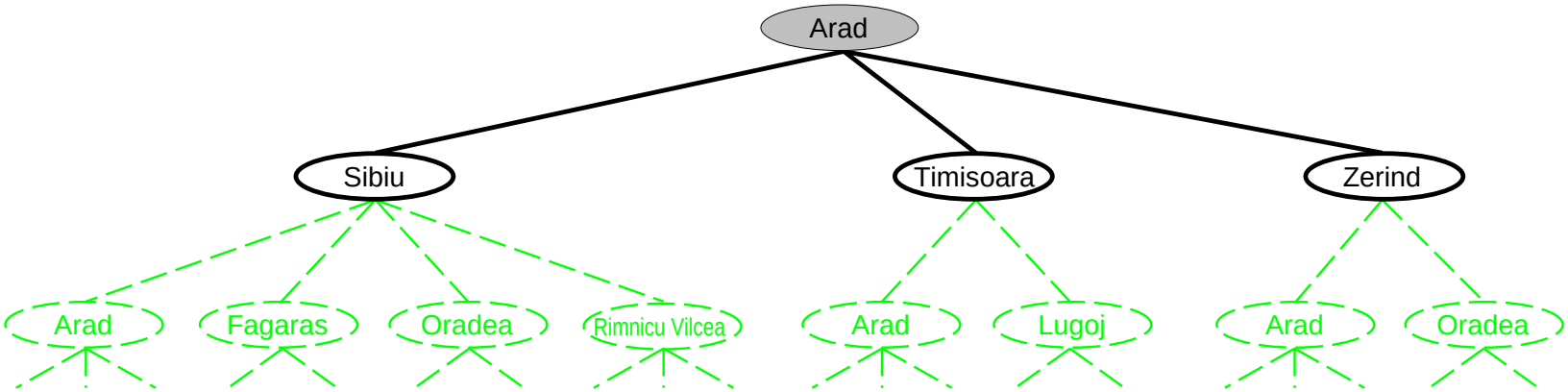
Contoh penelusuran search tree

- Mulai dari root node (Arad) sebagai current node.
- Lakukan *node expansion* terhadapnya.
- Pilih salah satu node yang di-expand sebagai current node yang baru. Ulangi langkah sebelumnya.



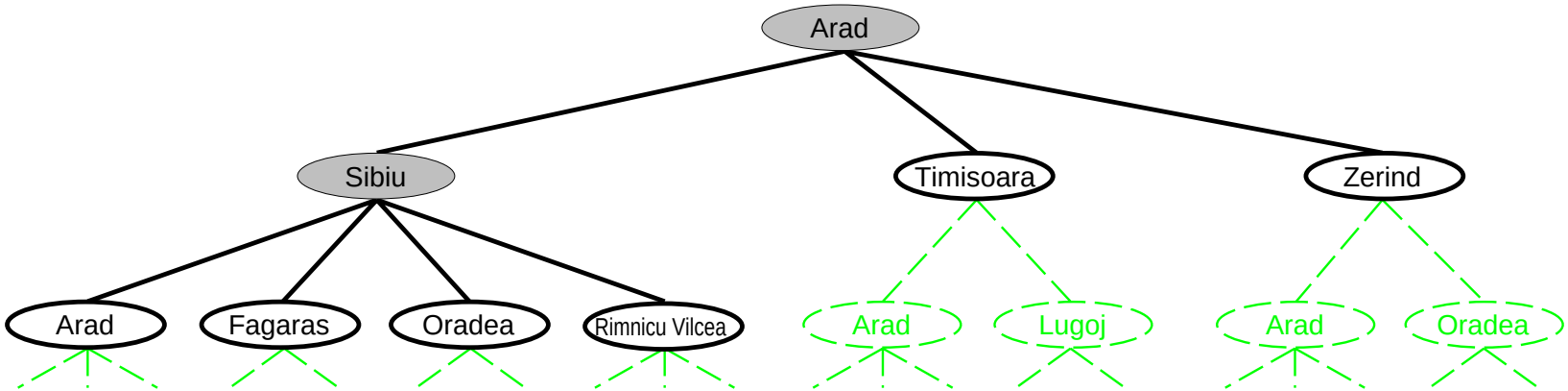
Contoh penelusuran search tree

- Mulai dari root node (Arad) sebagai current node.
- Lakukan *node expansion* terhadapnya.
- Pilih salah satu node yang di-expand sebagai current node yang baru. Ulangi langkah sebelumnya.



Contoh penelusuran search tree

- Mulai dari root node (Arad) sebagai current node.
- Lakukan *node expansion* terhadapnya.
- Pilih salah satu node yang di-expand sebagai current node yang baru. Ulangi langkah sebelumnya.



Algoritme penelusuran search tree

1. Pada awalnya, *fringe* = himpunan node yang mewakili *initial state*.
2. Pilih satu node dari *fringe* sebagai current node (Kalau *fringe* kosong, selesai dengan gagal).
3. Jika node tsb. lolos *goal test*, selesai dengan sukses!
4. Jika tidak, lakukan node expansion terhadap *current node* tsb. Tambahkan semua node yang dihasilkan ke *fringe*.
5. Ulangi langkah 2.

function TREESEARCH (*problem*, *fringe*) **returns** *solution* or *failure*

fringe ← INSERT(MAKENODE(INITIALSTATE(*problem*)),*fringe*)
loop do
  if EMPTY?(*fringe*) **then return** failure
  node ← REMOVEFIRST(*fringe*)
  if GOALTEST(*problem*) applied to STATE(*node*) succeeds
    then return SOLUTION(*node*)
  fringe ← INSERTALL(EXPAND(*node*,*problem*),*fringe*)

Algoritme penelusuran search tree

1. Pada awalnya, *fringe* = himpunan node yang mewakili *initial state*.
2. Pilih satu node dari *fringe* sebagai current node (Kalau *fringe* kosong, selesai dengan gagal).
3. Jika node tsb. lolos *goal test*, selesai dengan sukses!
4. Jika tidak, lakukan node expansion terhadap *current node* tsb. Tambahkan semua node yang dihasilkan ke *fringe*.
5. Ulangi langkah 2.

function TREESEARCH (*problem*, *fringe*) **returns** *solution* or *failure*

fringe ← INSERT(MAKENODE(INITIALSTATE(*problem*)),*fringe*)
loop do
  if EMPTY?(*fringe*) **then return** failure
  node ← REMOVEFIRST(*fringe*)
  if GOALTEST(*problem*) applied to STATE(*node*) succeeds
    then return SOLUTION(*node*)
  fringe ← INSERTALL(EXPAND(*node*,*problem*),*fringe*)

Algoritme penelusuran search tree

1. Pada awalnya, *fringe* = himpunan node yang mewakili *initial state*.
2. Pilih satu node dari *fringe* sebagai current node (Kalau *fringe* kosong, selesai dengan gagal).
3. Jika node tsb. lolos *goal test*, selesai dengan sukses!
4. Jika tidak, lakukan node expansion terhadap *current node* tsb. Tambahkan semua node yang dihasilkan ke *fringe*.
5. Ulangi langkah 2.

function TREESEARCH (*problem*, *fringe*) **returns** *solution* or *failure*

fringe ← INSERT(MAKENODE(INITIALSTATE(*problem*)),*fringe*)
loop do
  if EMPTY?(*fringe*) **then return** failure
  node ← REMOVEFIRST(*fringe*)
  if GOALTEST(*problem*) applied to STATE(*node*) succeeds
    then return SOLUTION(*node*)
  fringe ← INSERTALL(EXPAND(*node*,*problem*),*fringe*)

Algoritme penelusuran search tree

1. Pada awalnya, *fringe* = himpunan node yang mewakili *initial state*.
2. Pilih satu node dari *fringe* sebagai current node (Kalau *fringe* kosong, selesai dengan gagal).
3. Jika node tsb. lolos *goal test*, selesai dengan sukses!
4. Jika tidak, lakukan node expansion terhadap *current node* tsb. Tambahkan semua node yang dihasilkan ke *fringe*.
5. Ulangi langkah 2.

function TREESEARCH (*problem*, *fringe*) **returns** *solution* or *failure*

fringe ← INSERT(MAKENODE(INITIALSTATE(*problem*)),*fringe*)
loop do
  if EMPTY?(*fringe*) **then return** failure
  node ← REMOVEFIRST(*fringe*)
  if GOALTEST(*problem*) applied to STATE(*node*) succeeds
    then return SOLUTION(*node*)
  fringe ← INSERTALL(EXPAND(*node*,*problem*),*fringe*)

Algoritme penelusuran search tree

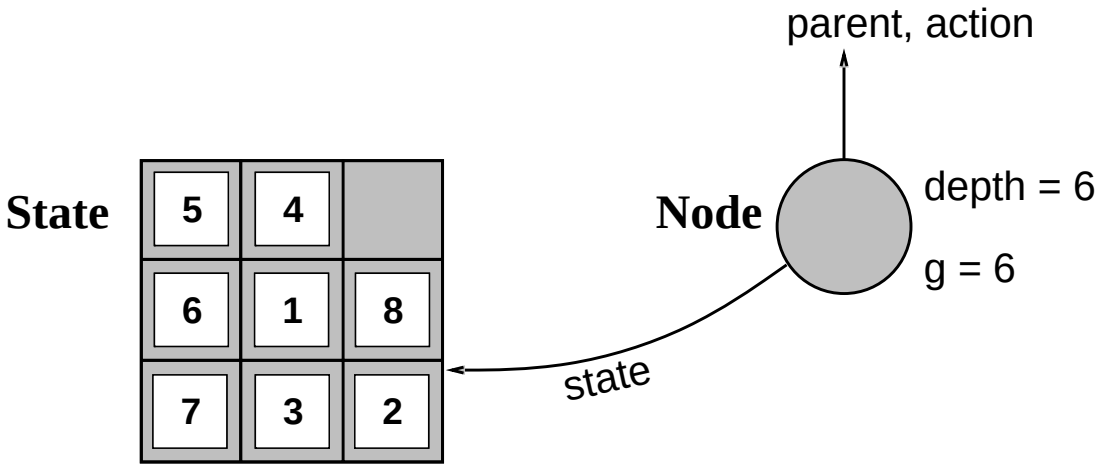
1. Pada awalnya, *fringe* = himpunan node yang mewakili *initial state*.
2. Pilih satu node dari *fringe* sebagai current node (Kalau *fringe* kosong, selesai dengan gagal).
3. Jika node tsb. lolos *goal test*, selesai dengan sukses!
4. Jika tidak, lakukan node expansion terhadap *current node* tsb. Tambahkan semua node yang dihasilkan ke *fringe*.
5. Ulangi langkah 2.

function TREESEARCH (*problem*, *fringe*) **returns** *solution* or *failure*

fringe ← INSERT(MAKENODE(INITIALSTATE(*problem*)),*fringe*)
loop do
  if EMPTY?(*fringe*) **then return** failure
  node ← REMOVEFIRST(*fringe*)
  if GOALTEST(*problem*) applied to STATE(*node*) succeeds
    then return SOLUTION(*node*)
  fringe ← INSERTALL(EXPAND(*node*,*problem*),*fringe*)

State vs. Node

- Sebuah state merepresentasikan abstraksi keadaan nyata dari masalah.
- Sebuah node adalah struktur data yang menjadi bagian dari search tree.
- State tidak memiliki parent, children, depth, path cost!
- Node = state pada path tertentu. Dua node berbeda bisa mewakili state yang sama!



Outline

1 Problem solving agent

2 Representasi masalah: state space

3 Pencarian solusi: search

4 Search strategies

5 Ringkasan

Strategi pencarian

- Terdapat berbagai jenis strategi untuk melakukan search.
- Semua strategi ini berbeda dalam satu hal: urutan dari node expansion.
- Search strategy di-evaluasi berdasarkan:
  - completeness: apakah solusi (jika ada) pasti ditemukan?
  - time complexity: jumlah node yang di-expand.
  - space complexity: jumlah maksimum node di dalam memory.
  - optimality: apakah solusi dengan minimum cost pasti ditemukan?
- Time & space complexity diukur berdasarkan
  - b - branching factor dari search tree
  - d - depth (kedalaman) dari solusi optimal
  - m - kedalaman maksimum dari search tree (bisa infinite!)

Uninformed search strategies

- Uninformed strategy hanya menggunakan informasi dari definisi masalah.
- Bisa diterapkan secara generik terhadap semua jenis masalah yang bisa direpresentasikan dalam sebuah state space.
- Ada beberapa jenis:
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative-deepening search

Outline

1 Problem solving agent

2 Representasi masalah: state space

3 Pencarian solusi: search

4 Search strategies

5 Ringkasan

Ringkasan

- Problem solving agents
- Perumusan masalah $\rightarrow$ state space
- Pencarian solusi $\rightarrow$ penelusuran search tree