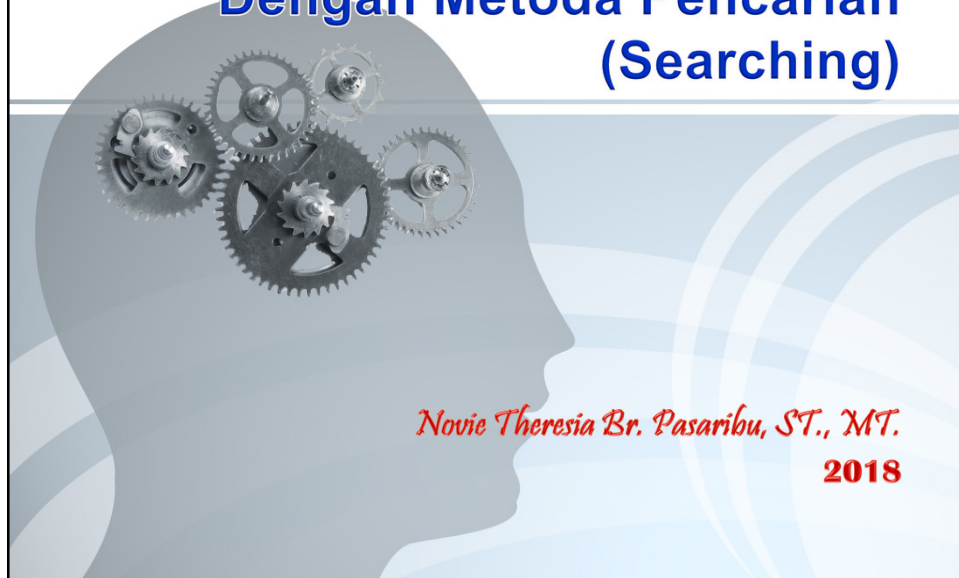


3. Pemecahan Masalah Dengan Metoda Pencarian (Searching)

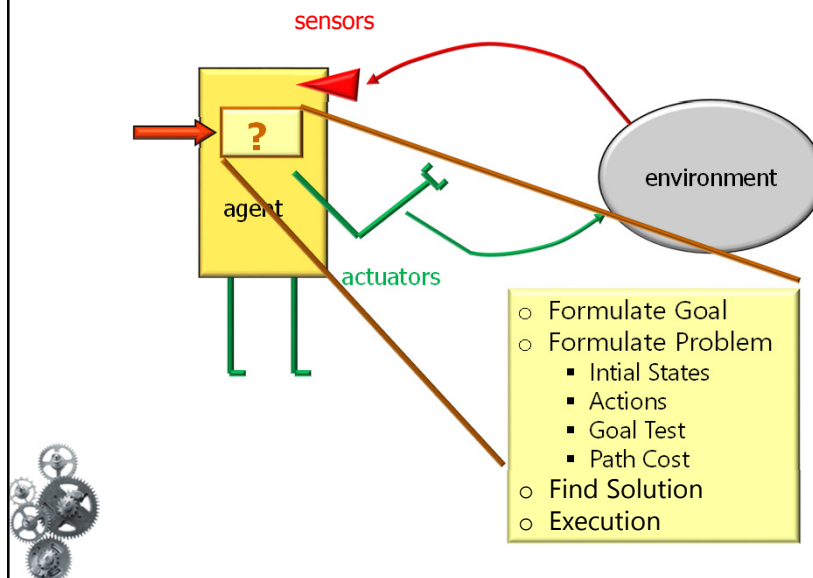


Pemecahan Masalah dengan Metoda Pencarian (Searching)

- **Problem-Solving Agent (PSA)**
 - Memutuskan tindakan yang harus dilakukan untuk mencapai hasil yang diinginkan.
 - Dengan cara : mengidentifikasi tiap urutan aksi yang mendahuluinya.
- Sasaran Problem-Solving Agent, adalah mencapai suatu Goal yang diinginkan (PSA merupakan Tipe Goal-Based Agents).



Problem-Solving Agent (PSA)



Mekanisme kerja Problem-Solving Agent

- **Formulate Goal (Merumuskan Tujuan)**
Tentukan tujuan yang ingin dicapai.
- **Formulate Problem (Merumuskan Permasalahan)**
Tentukan tindakan (action) & keadaan (state) yang dipertimbangkan dalam mencapai tujuan.
- **Find Solution (Mencari Solusi Masalah)**
Tentukan rangkaian tindakan yang perlu diambil untuk mencapai tujuan.
- **Execution (Pelaksanaan Solusi)**
Laksanakan rangkaian tindakan yang sudah ditentukan di tahap sebelumnya.



Contoh : Turis di Romania

- Liburan ke Romania, khususnya : Arad.
Penerbangan dilakukan besok dari Bucharest.

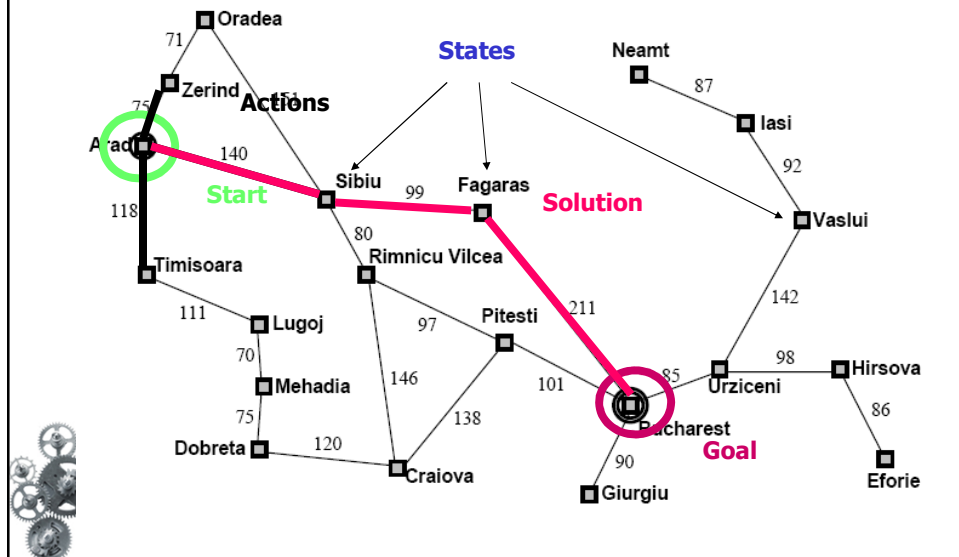


Contoh : Turis di Romania

- **Formulate Goal**
Berada di Bucharest
- **Formulate Problem**
 - States : Kota-kota
 - Actions : Perjalanan antar kota-kota
- **Find Solution**
 - Sederatan kota-kota yang menghasilkan jarak terpendek : Arad, Sibiu, Fagaras, Bucharest.
- **Execution**



Contoh : Turis di Romania



Tipe Masalah (Problem)

- **Single State Problem**
Deterministic, Fully Observable
- **Multiple State Problem**
Deterministic, Partially Observable
- **Contingency Problem**
Stochastic, Partially Observable
- **Exploration Problem**
Unknown state space



Single State Problem

- **Deterministic, Fully Observable**

- Agent mengetahui tentang lingkungannya (exact state)



- Dapat memperhitungkan sederetan action yang optimal untuk mencapai goal state

- Contoh :

Bermain catur , setiap action akan menghasilkan state yang pasti



Multiple State Problem

- **Deterministic, Partially Observable**

- Agent tidak mengetahui exact state (bisa beberapa kemungkinan state)

- Mendapatkan state ketika bekerja mencapai goal state.

- Contoh :

Berjalan diruangan gelap :

- Jika berada dipintu, berjalan terus akan membawa kita kedapur.
 - Jika kita berada didapur, belok kiri akan membawa kita ke kamar mandi,dll.



Contingency Problem



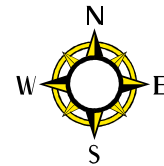
- **Stochastic, Partially Observable**

- Harus menggunakan sensor selama eksekusi
- Solusi adalah sebuah tree atau kebijakan
- Prediksi selanjutnya tidak dapat diketahui dengan pasti

Contoh :

Pemain skate baru di suatu area

- Sliding problem
- Pemain skate lainnya yang disekitarnya

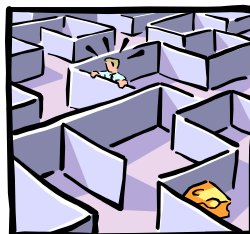


Exploration Problem

- **Unknown state space**

- Agen harus menemukan & mempelajari “peta” dari lingkungan untuk digunakan sebagai pemecah masalah.

Contoh :
Labirin



Problem & Solution

- Problem, adalah :
kumpulan informasi yang digunakan agent untuk menentukan apa yang harus dilakukannya (bertindak)
- Elemen dasar dari problem definition:
 - State (Keadaan)
 - Action (Tindakan)




Formulate Problem

- **An Initial State** : Kondisi awal
- **A Set of Actions**
Kumpulan dari kemungkinan-kemungkinan tindakan yang dapat dilakukan oleh agent
 - Operator : digunakan untuk menunjukkan suatu tindakan, dengan kondisi keadaan yang bagaimana tindakan akan tercapai, bila melakukan suatu tindakan pada suatu keadaan khusus
- **A Goal Test**
Apakah goal/tujuan akhir tercapai?
- **A Path Cost**
Jumlah dari biaya individual tiap tindakan selama perjalanan menuju tercapainya tujuan akhir



Mengukur Performance Problem-Solving

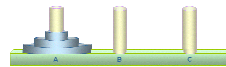
Ada 3 cara mengukur performance problem solving :

- Apakah ditemukan suatu solusi akhir ? 
- Apakah solusi yang diberikan adalah solusi yang terbaik (low path cost)?
- Bagaimana hubungan *search cost* terhadap waktu dan memory yang diperlukan untuk menemukan solusi ?

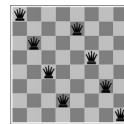
$$\text{Total Cost} = \text{Search Cost} + \text{Path Cost}$$

Contoh Problem

Toy problems



- Singkat dan deskripsi yang tepat.
- Contoh :
 - 8-puzzle
 - 8-queens problem
 - Cryptarithmic
 - Vacuum world
 - Missionaries & cannibals
 - Simple route finding
 - Towers of Hanoi
 - Water jug problem



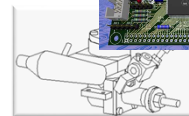
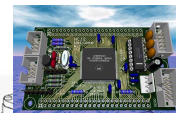
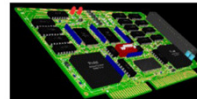
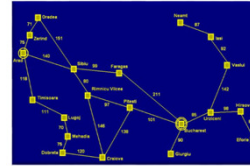
Contoh Problem

Real-world problems

- Lebih sulit

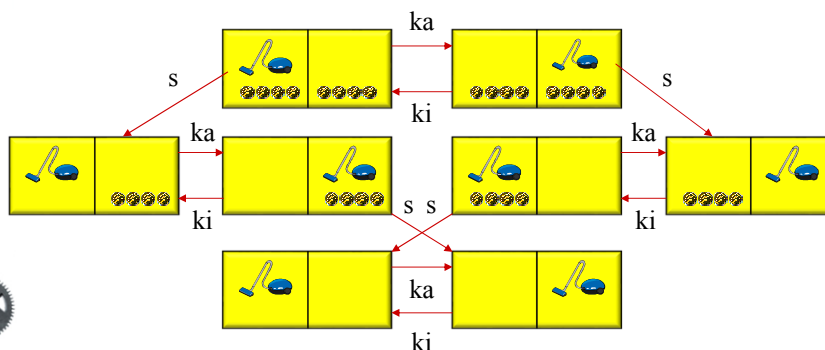
- Contoh :

- Route finding
- Touring & traveling salesperson problems
- VLSI layout
- Robot navigation
- Process or assembly planning



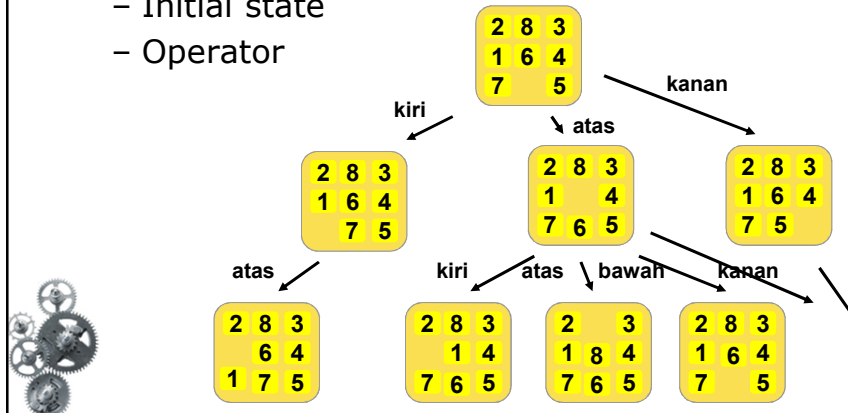
State Space Implisit & Eksplisit

- State space dapat direpresentasikan secara eksplisit dengan suatu grafik.
(Tidak mudah untuk memodelkan state space dari suatu problem)



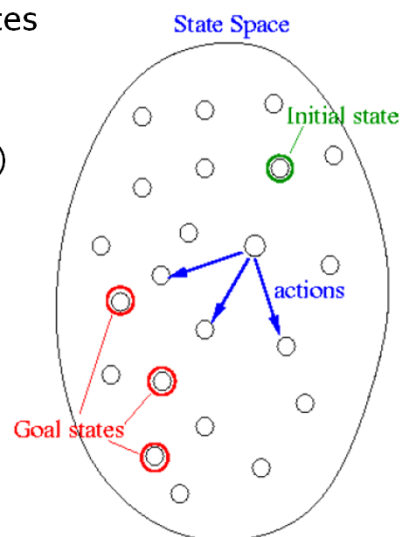
State Space Implisit & Eksplisit

- State space dapat direpresentasikan secara Implisit dan dikembangkan ketika dibutuhkan. Sehingga Agent harus mengetahui :
 - Initial state
 - Operator



Search Problem

- S : Himpunan dari states
- s_0 : Initial state
- A : Operator ($S \rightarrow S$)
- G : Goal states ($G \subseteq S$)



Contoh : Turis di Romania

- Liburan ke Romania, khususnya : Arad.
Penerbangan dilakukan besok dari Bucharest.



Formulate Problem Turis di Romania

- **An Initial State** : Dari Arad
- **Operator (atau succesor function):**
Contoh : Arad → Zerind, Arad → Sibiu, dll.
- **A Goal Test** : Berada di Bucharest
- **A Path Cost** :
Penjumlahan jarak, jumlah operator yang dieksekusi



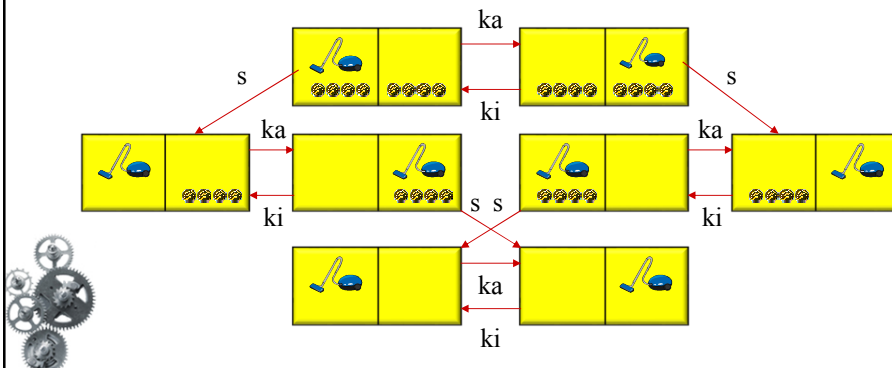
Contoh : The vacuum world

- The world hanya mempunyai 2 lokasi.
- Setiap lokasi, dapat mengandung sampah atau tidak.
- Agent bisa berada disuatu lokasi /dilokasi yang lain.
- Ada 8 kemungkinan world states.
- Ada 3 kemungkinan actions:
kiri (ki) , kanan (ka), sedot (s).
- Goal: membersihkan semua yang kotor



Contoh : The vacuum world

- State? Satu dari 8 keadaan (state)
- Operator? Ke kiri, kanan, sedot
- Goal Test? Tidak ada sampah pada daerah kotak
- Path Cost? 1 Path cost = jumlah langkah dalam path



Contoh : 8-puzzle



Initial State



Goal State

- **State?** Lokasi 8 buah angka dalam matriks 3x3.
- **Operator?** Geser yang kosong ke kiri, kanan, atas, bawah.
- **Goal Test?** Apakah state sesuai dengan goal state ?
- **Path Cost?** 1 per move.



Contoh: Criptharithmetic

FORTY	Solusi :	29786
+ TEN		+ 850
+ TEN		+ 850
-----		-----
SIXTY		31486

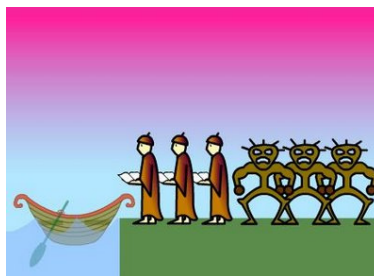
F=2 O=9 R=7, dst

- **State ?** Criptharithmetic puzzle dg beberapa huruf diganti dengan digit (angka).
- **Operator ?** Ganti semua karakter (huruf) yang ada dengan digit (angka) yang belum muncul dlm puzzle.
- **Goal Test ?** Puzzle hanya berisi angka-angka dan mewakili penjumlahan yang benar.
- **Path Cost ?** 0 (nol), seluruh solusi persamaan harus memenuhi(valid).



Contoh : Misionaris & Kanibal

- Ada 3 misionaris dan 3 Kanibal, yang berharap akan menyebrangi suatu sungai.
- Mereka hanya mempunyai sebuah perahu, yang hanya muat untuk dua orang saja.
- Kondisi tidak akan aman jika jumlah kanibal lebih banyak daripada jumlah misionaris.

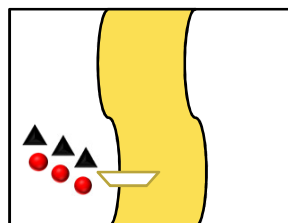


Contoh : Misionaris & Kanibal

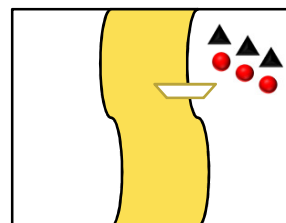
- **State ?** Konfigurasi misionaris, kanibal dan perahu (di satu sisi)
- **Operator ?** Pindahkan perahu
- **Goal Test ?** Apakah semua misionaris dan kanibal sudah berada disisi sebelah kanan?
- **Path Cost ?** 1 pergerakan=1 cost

▲▲▲ = Misionaris

●●● = Kanibal



Intial State



Goal State



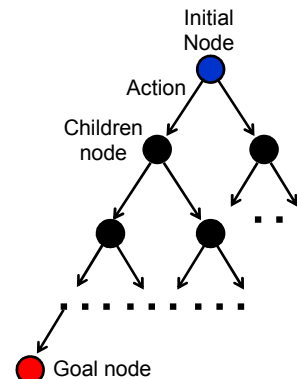
Mencari solusi Melalui Search Tree

- 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.



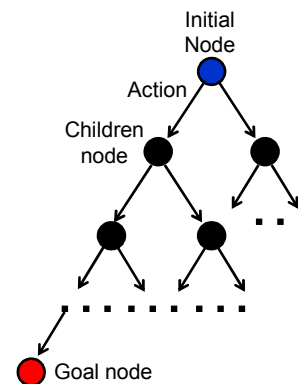
Mencari solusi Melalui Search Tree

- **Root node (Initial Node)**
dari search tree merepresentasikan **initial state**.
- Children node adalah merepresentasikan state pengganti dari suatu node (hasil ekspansi) .
- Kumpulan semua node yang belum diekspansi disebut fringe (pinggir) dari suatu search tree.



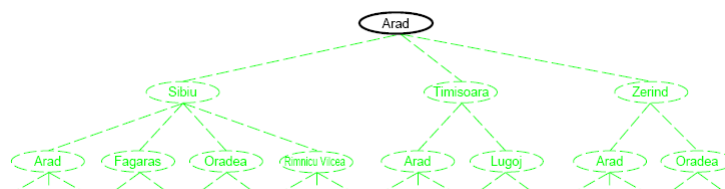
Mencari solusi Melalui Search Tree

- Jalur dari suatu search tree merepresentasikan serangkaian tindakan yang merupakan solusi yang dimulai dari initial state dan berakhir di goal state.



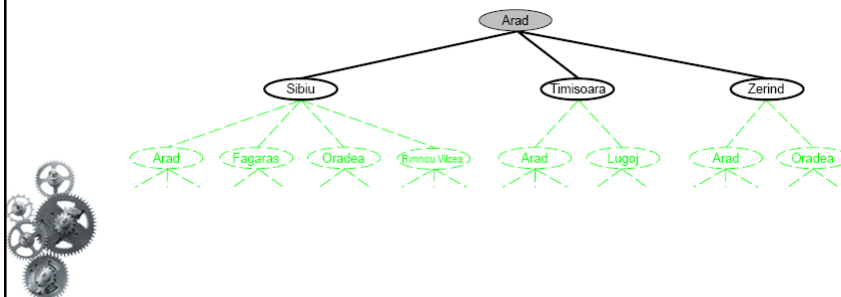
Contoh Penelusuran Search Tree

- Mulai dari **root node** (Arad) sebagai current node.



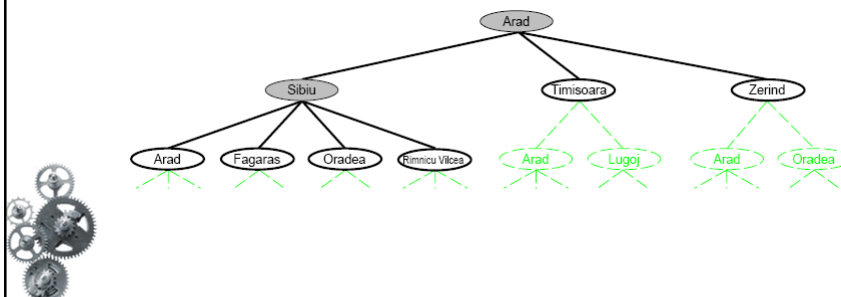
Contoh Penelusuran Search Tree

- Mulai dari **root node** (Arad) sebagai current node.
- Lakukan node expansion terhadapnya.



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.



Strategi pencarian (Search Strategy)

- Terdapat berbagai jenis strategi untuk melakukan search.
- Semua strategi ini berbeda dalam satu hal : urutan dari node expansion.



Kelompok Strategi Pencarian

- **Uninformed Search (Blind Search)**
 - Hanya menggunakan informasi yang ada pada problem definition.
 - Tidak ada informasi tentang banyaknya langkah atau path cost dari keadaan sekarang (current state) sampai goal.
- **Informed Search (Heuristic Search)**
 - Mengeksplorasi informasi.



Uninformed Search (Blind Search)

◀ **Breadth-first search**

◀ **Uniform-cost search**

◀ **Depth-first search**

◀ **Depth-limited search**

◀ **Iterative deepening search**



Informed Search (Heuristic Search)

◀ **Best-first search**

◀ **Greedy best-first search**

◀ **A* search**

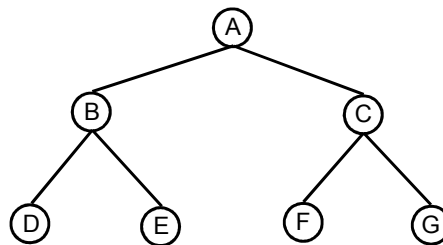


Breadth-First Search (BFS)

- Strategi ini node utama diekspansi, selanjutnya node hasil ekspansi (node_eks) diekspansi lagi dan seterusnya.
- Implementasi: fringe adalah sebuah queue, data struktur FIFO (First In First Out)



Breadth-First Search (BFS)



Breadth-First Search (BFS)

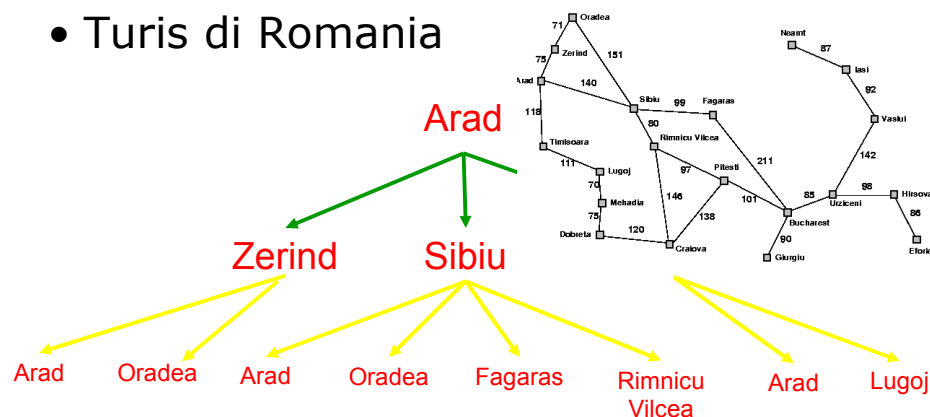
```

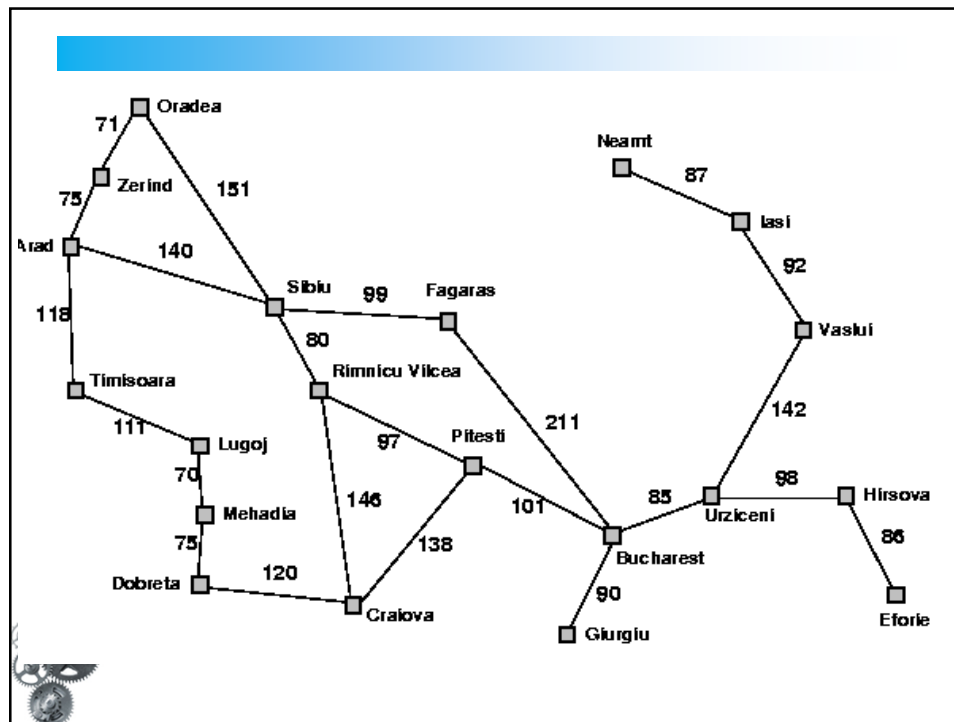
BREADTH-FIRST-SEARCH(NodeList, Goal)
NewNodes =  $\emptyset$ 
For all Node  $\in$  NodeList
  If GoalReached(Node, Goal)
    Return("Solution found", Node)
  NewNodes = Append(NewNodes, Successors(Node))
If NewNodes  $\neq \emptyset$ 
  Return(BREADTH-FIRST-SEARCH(NewNodes, Goal))
Else
  Return("No solution")
  
```



Breadth-First Search (BFS)

- Turis di Romania

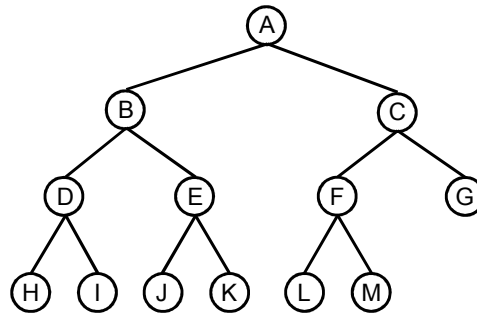




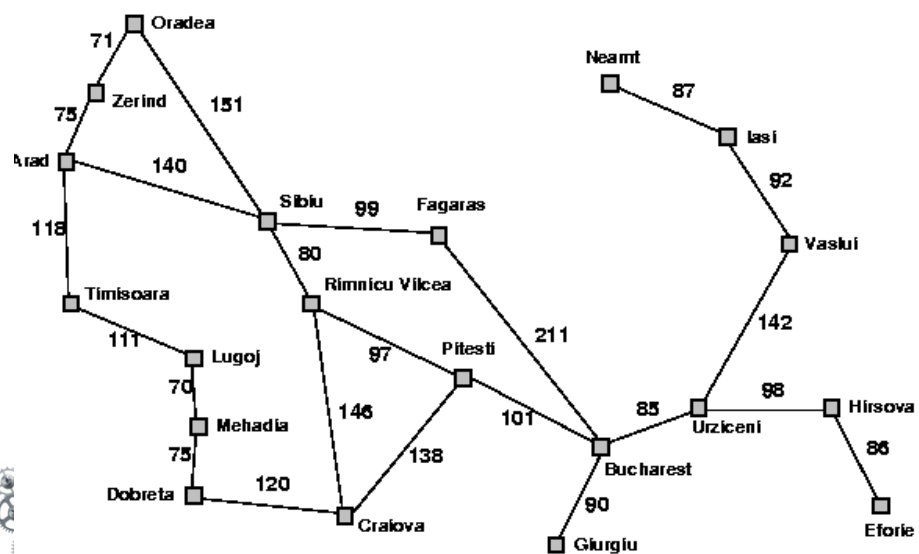
Depth-First Search (DFS)

- Depth-first search selalu melakukan ekspansi salah satu node pada level terdalam dalam pohon pencarian, jika gagal maka akan kembali ke atas, dan melakukan ekspansi pencarian pada/ melalui node yang lain, dst.
- Implementasi: fringe adalah sebuah stack, data struktur LIFO (Last In First Out)
- Depth-first search sangat cocok diimplementasikan secara rekursif.

Depth-First Search (DFS)



Depth-First Search (DFS)



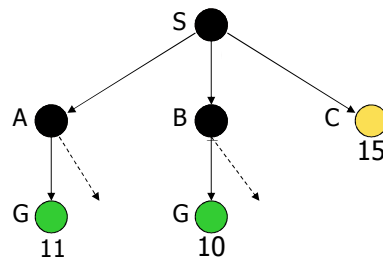
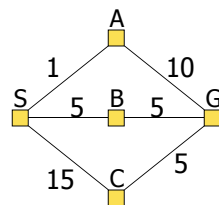
Uniform-Cost Search

- (Dijkstra, 1959)
- Merupakan modifikasi dari Breadth-First Search dengan tambahan melibatkan biaya terendah dari `node_eks`
- Implementasi: fringe adalah sebuah priority queue di mana node disortir berdasarkan path cost function $g(n)$.



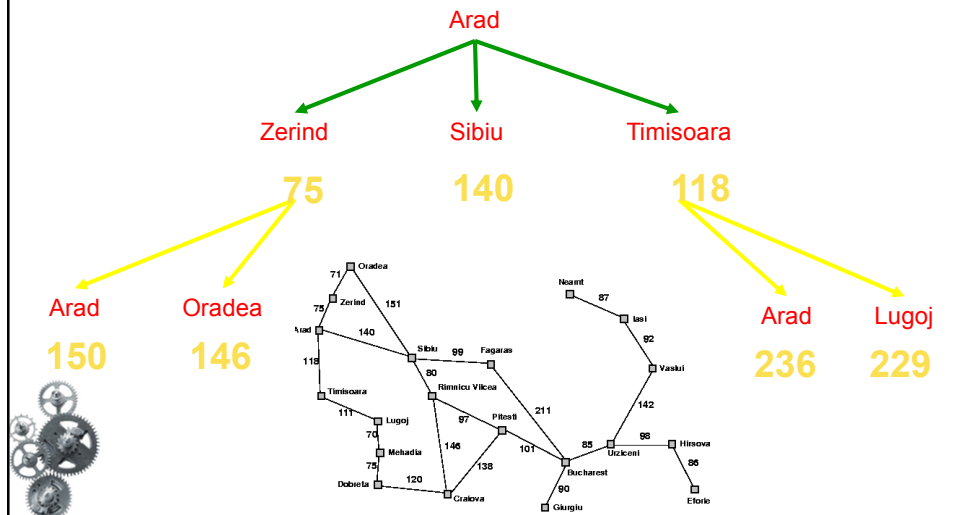
Uniform-Cost Search

- Contoh :

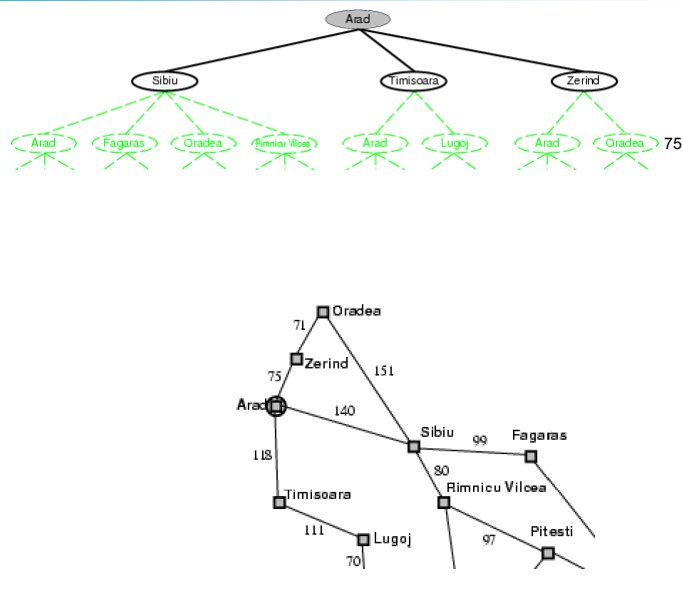


Uniform-Cost Search

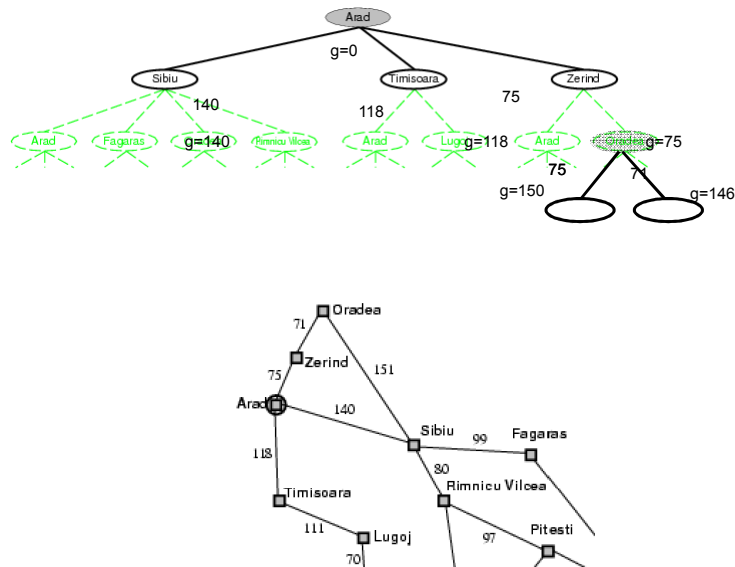
- Turis di Romania



Uniform-cost search



Uniform-cost search



Depth-Limited Search

- Merupakan modifikasi dari Depth-First Search untuk menghindari terjerusnya ke lubang perangkat yang terlalu dalam, dengan cara memotong maksimum dari kedalaman path.
- **Contoh :**
Maksimum path dalam peta ada 13, jika pencarian lebih dari 13, berarti salah dan pencarian langsung dihentikan, dicari melalui node lain.

Iterative Deepening Search

- Merupakan strategi pencarian dengan memilih batas limit kedalaman melalui percobaan seluruh kemungkinan batas limit (dari kedalaman 0 sampai n).

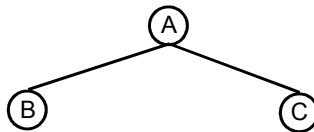


Iterative Deepening Search (L= 0,1,2)

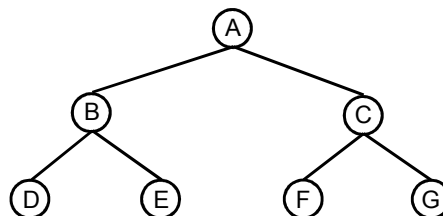
Limit = 0

(A)

Limit = 1

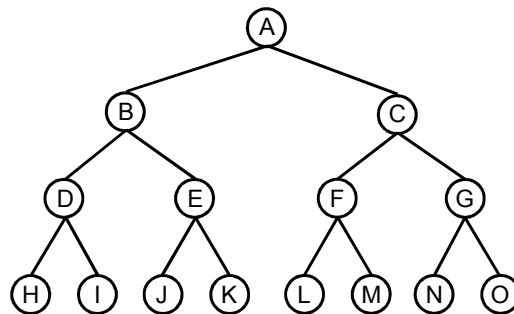


Limit = 2



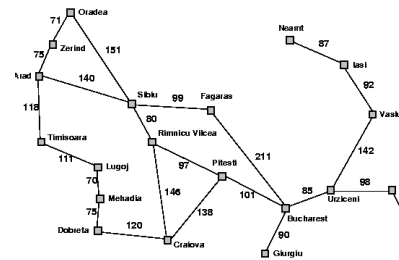
Iterative Deepening Search (L= 3)

Limit = 3



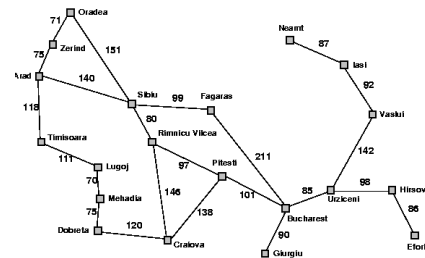
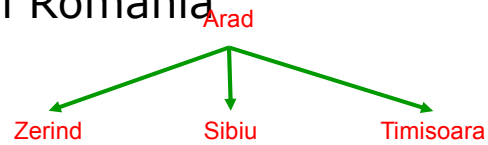
Iterative Deepening Search Depth=0

- Turis di Romania Arad



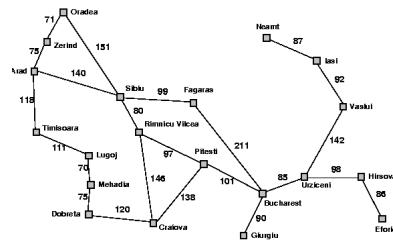
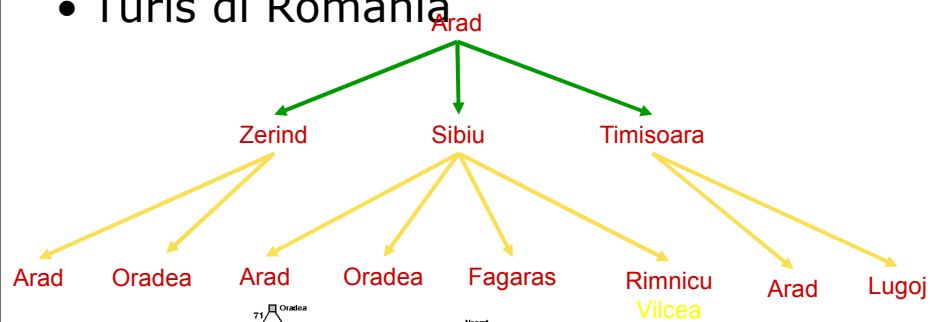
Iterative Deepening Search: depth=1

• Turis di Romania



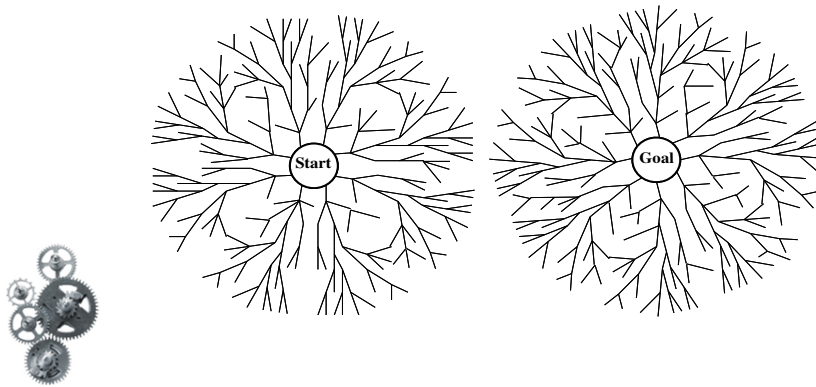
Iterative Deepening Search: depth=2

• Turis di Romania

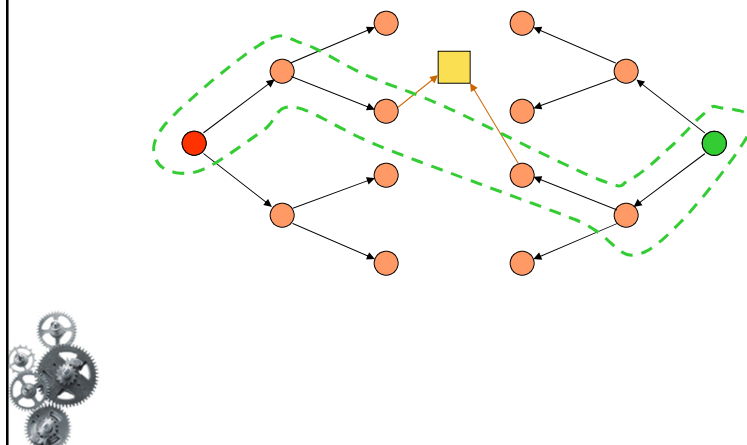


Bidirectional Search

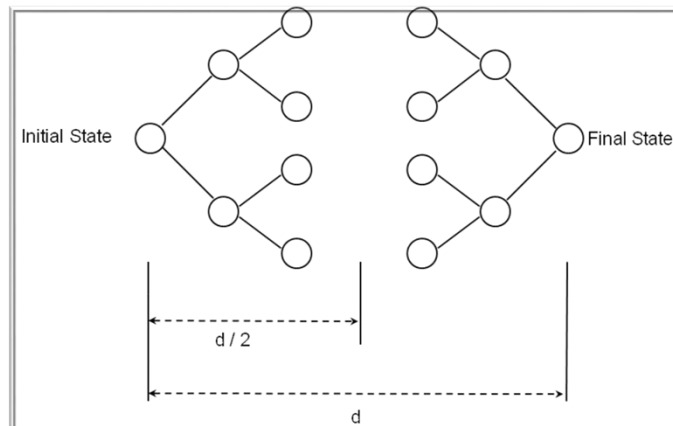
- Ide metoda ini adalah, pencarian secara simultan bersamaan dari initial state maju (forward), sementara dari goal mundur (backward), dan berhenti pada saat keduanya bertemu.



Bidirectional Search



Bidirectional Search



Strategi Pencarian dievaluasi berdasarkan

- **Completeness :**

Apakah strategi menjamin akan menemukan solusi (minimal satu) ?

- **Time complexity:**

Berapa lama waktu yang dibutuhkan untuk menemukan solusi ?

- **Space complexity:**

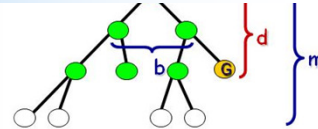
Berapa memory yang dibutuhkan untuk membentuk pencarian (maksimal ukuran node list)?

Time & space complexity diukur berdasarkan





Strategi pencarian dievaluasi berdasarkan



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!)

- **Optimality:**

Apakah strategi menghasilkan kualitas solusi tertinggi (minimum cost), jika dibandingkan dengan solusi-solusi lain?



Breadth-first search

- **Complete** : Ya (jika b terbatas)
- **Optimal** : Ya (jika cost = 1 per langkah)
- **Time** : $1 + b + b^2 + b^3 + b^4 + \dots + b^d = b^d$
- **Space** : b^d (Setiap node disimpan di memory)

b : Branching factor
 d : Kedalaman dari solusi
 m : Kedalaman Maximum



Kompleksitas BFS

- **Asumsi** : 1 simpul = 100 bytes dan kecepatan komputer = 10^6 simpul/detik.

<i>b</i>	<i>d</i>	Simpul	Waktu	Memory
10	6	10^6	1 detik	100 MB
10	8	10^8	100 detik	10 GB
10	12	10^{12}	11,57 hari	100 TB
10	14	10^{14}	3,17 tahun	10.000 TB



Depth-first search

- **Complete** : Tidak , gagal jika state-space tak terbatas (kondisi berulang), tanpa batasan kedalaman.
- **Optimal** : Tidak
- **Time** : b^m (Masalah jika $m > d$)
- **Space** : $b.m$ (kedalaman yang linier)

b : Branching factor
m: Kedalaman Maximum



Uniform Cost Search

- **Complete** : Ya (jika b terbatas)
- **Optimal** : Ya
- **Time** : b^d
- **Space** : b^d

b : Branching factor
 m : Kedalaman Maximum



Depth Limited Search

- **Complete** : Ya (Jika $l > d$)
- **Optimal** : Tidak
- **Time** : b^l
- **Space** : $b.l$

b : Branching factor
 m : Kedalaman Maximum
 l : Kedalaman cut-off



Iterative Deepening Search

- **Complete** : Ya
- **Optimal** : Ya
- **Time** : b^d
- **Space** : $b \cdot d$

- Maximum space sama dengan DFS.
- Time complexity hampir sama dengan BFS.

Bidirectional Search

- **Complete** : Ya
- **Optimal** : Ya
- **Time** : $b^{d/2}$
- **Space** : $b^{d/2}$

b : Branching factor
d : Kedalaman dari solusi

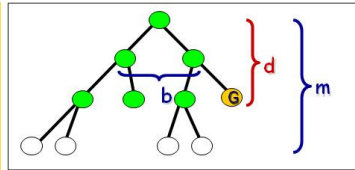
Perbandingan Strategi Pencarian

Criterion	Breadth-first	Uniform cost	Depth-first	Depth-limited	Iterative deepening (if applicable)	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{(d/2)}$
Space	b^d	b^d	bm	bl	bd	$b^{(d/2)}$
Optimal?	Ya	Ya	Tidak	Tidak	Ya	Ya
Complete?	Ya	Ya	Tidak	Ya	Ya	Ya

jika $l \geq d$

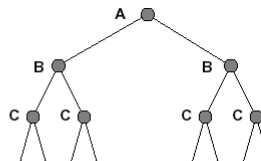
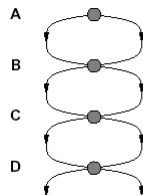


b : Branching factor
 d : Kedalaman dari solusi
 m : Kedalaman Maximum
 l : Kedalaman cut-off



Menghindari Keadaan yang berulang

- Masalah dalam proses Uninformed-Search :
 - Kemungkinan ada waktu terbuang karena "pengembangan" cabang.
 - Kemungkinan proses yang berulang



3 Cara Menghindari Keadaan Berulang

- Jangan kembali **ke keadaan yang baru dilewati** (dengan membuat fungsi/operator pengembang yang dapat menolak pada saat terbentuknya keadaan yang sama dengan keadaan parent).
- Jangan **membuat cabang dengan lingkaran didalamnya** (dengan membuat fungsi/operator pengembang yang dapat menolak pada saat terbentuknya keadaan yang sama dengan ancestor).
- Jangan **membentuk keadaan yang pernah dibentuk sebelumnya** (diperlukan memory yang dapat menyimpan setiap keadaan yang pernah dihasilkan).



Constraint Satisfaction Search

- Constraint Satisfaction Problem (CSP) :
Suatu masalah khusus yang adanya beberapa tambahan struktur diluar kebutuhan dasar dari masalah umum.
- Keadaan dalam CSP didefinisikan oleh :
sederetan nilai variabel dan goal test ditetapkan dengan suatu batasan (constraint) yang harus dipatuhi oleh nilai variabel tersebut.



Constraint Satisfaction Search

- Jenis Batasan :
 - UNARY CONSTRAINT (Menyangkut 1 variabel)
 - BINARY CONSTRAINT (Menghubungkan 2 variabel)
 - HIGHER ORDER CONSTRAINT (Meliputi ≥ 3 variabel)
- Setiap variabel V_i dalam CSP mempunyai "Domain" D_i , yang merupakan sekumpulan nilai yang dapat dipergunakan oleh variabel tersebut.
- "Domain" dapat berupa DISKRIT atau KONTINYU.
- Contoh :
 - Domain Kontinyu : Mendesain mobil berat
 - Domain Diskrit : Perakitan komponen



Backtracking Search & Forward Checking

• Backtracking Search

Suatu algoritma (perbaikan) yang menambahkan suatu test, sebelum melaksanakan langkah selanjutnya, yaitu test apakah ada batasan yang telah dilanggar oleh variabel yang telah disetujui sampai titik tersebut.

• Forward Checking

Suatu algoritma yang melihat kedepan untuk menghindari kemungkinan tidak terpecahkan/ terselesaikannya masalah.



Best-first Search

- **Best-First Search :**

suatu metoda dalam strategi pemecahan masalah dengan menggunakan fungsi evaluasi (evaluation function), dimana node diurutkan, sehingga hasil evaluasi terbaik (minimum cost nodes) akan dikembangkan lebih dahulu.

- Strategi untuk meminimumkan estimasi biaya untuk mencapai goal.

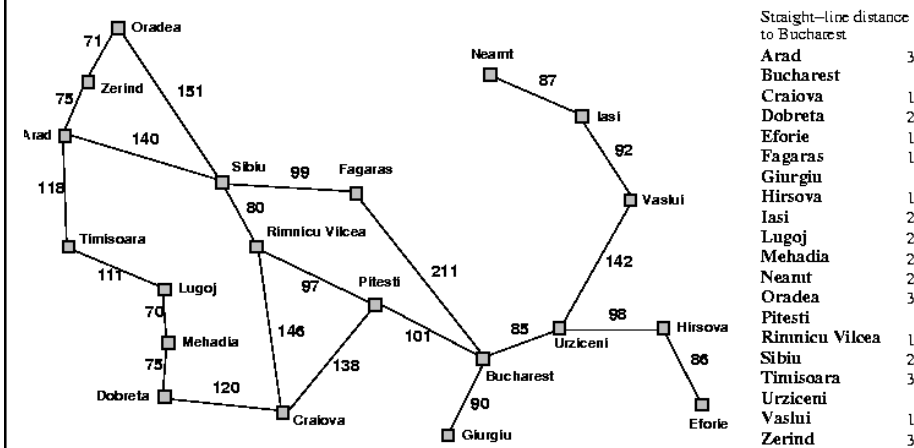


Best-first Search

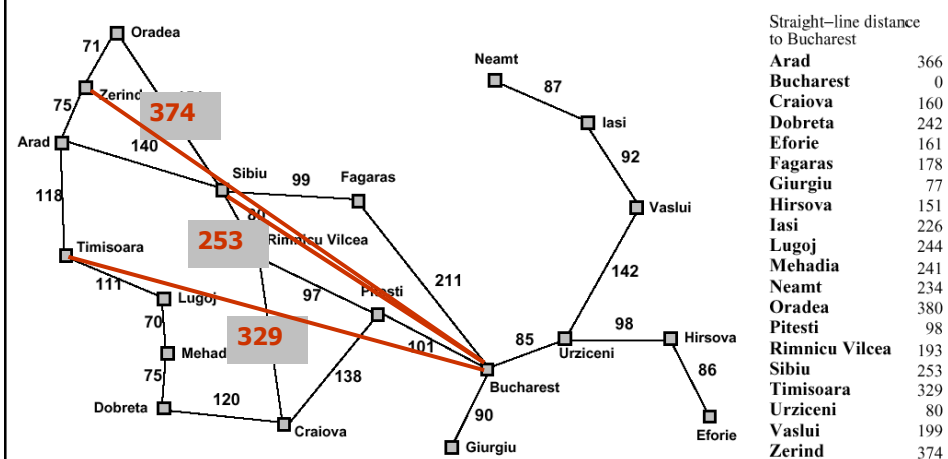
- Kunci keberhasilan best-first search terletak di heuristic function.
- Heuristic function $h(n)$ adalah :
fungsi yang menyatakan estimasi/
perkiraan cost dari n ke goal state.



Turis di Romania



Turis di Romania

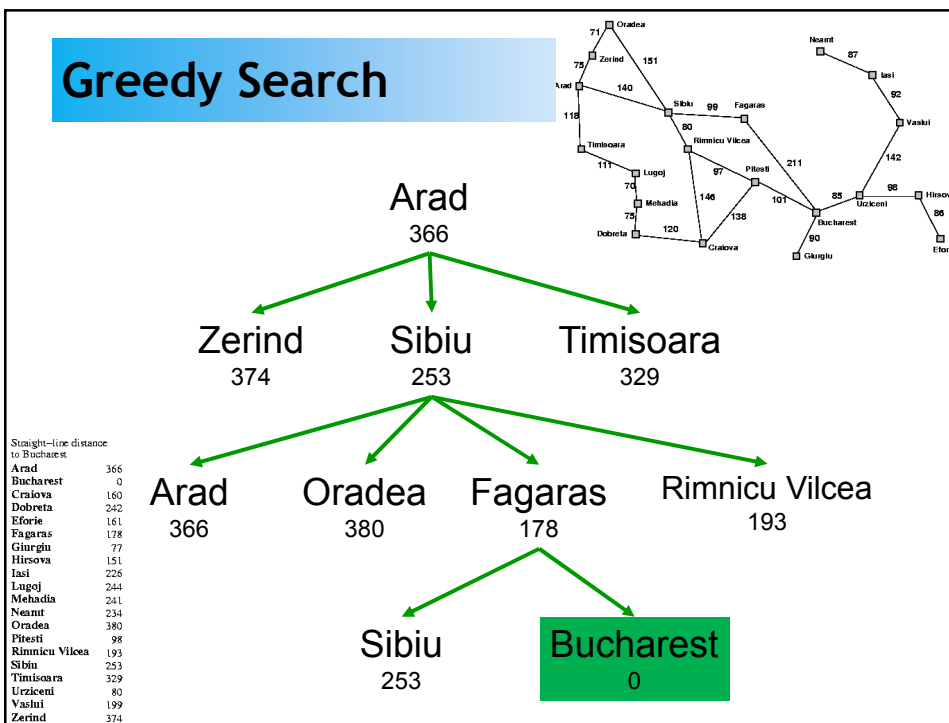


Greedy Search

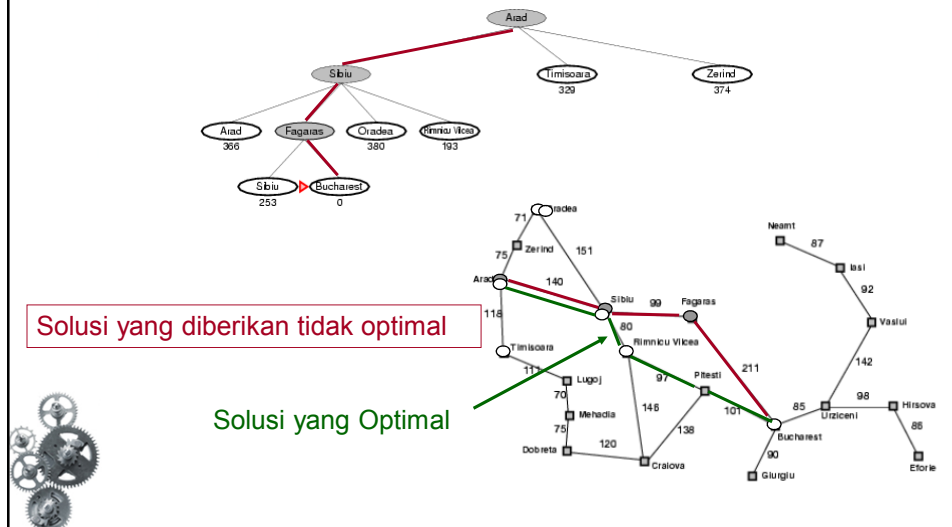
- Adalah strategi BFS yang menggunakan $f(n)=h(n)$ untuk menentukan node berikutnya yang akan dikembangkan (expand)
- Greedy best-first search selalu memilih node yang kelihatannya paling dekat ke goal (yang memiliki $h(n)$ paling kecil)



Greedy Search



Contoh Greedy Best-First Search



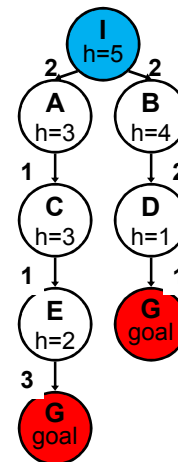
Greedy Search

- Tidak Complete
- Tidak Optimal

Greedy search akan mendapatkan goal disebelah kiri (solution cost : 7).

Solusi optimal solution adalah jalur dengan goal yang disebelah kanan (solution cost : 5).

Greedy search memilih yang terbaik, yang bisa saja yang terbaik secara lokal saja.

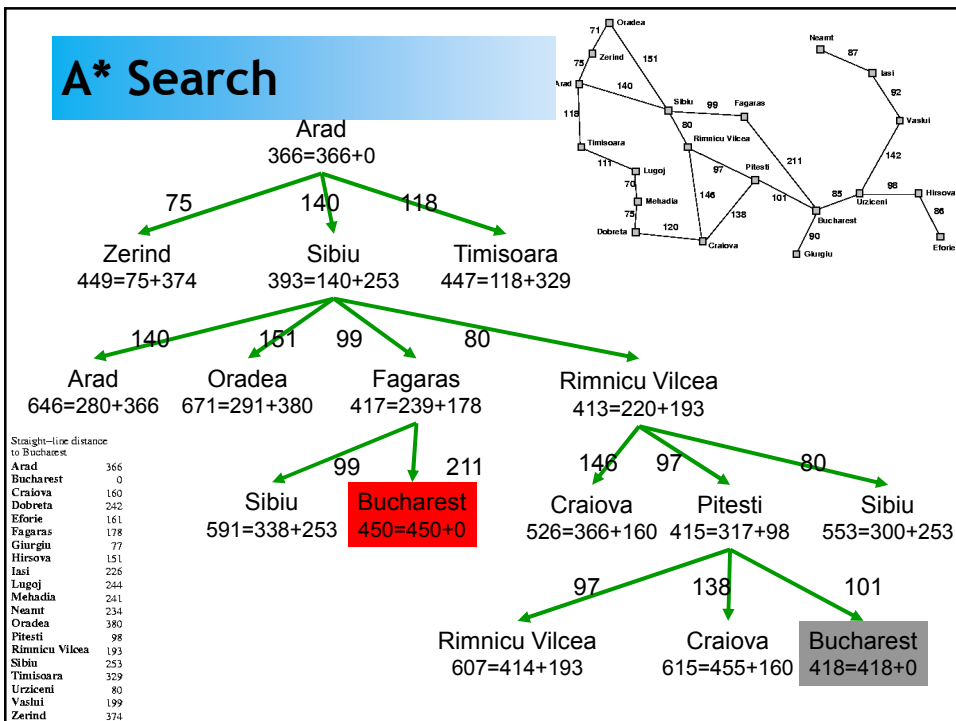


A* Search

- Metode search yang efisien dan optimal.
- Idenya : menghindari node yang berada di path yang "mahal"
- Kombinasi antara greedy search dan uniform-cost search.
- Fungsi evaluasi : $f(n) = g(n) + h(n)$
 - $g(n)$ = biaya perjalanan (path cost) ke node n .
 - $h(n)$ = biaya estimasi (estimated path cost) dari node n ke *goal*.
 - $f(n)$ = Solusi biaya estimasi termurah (estimated total cost of path) node n ke goal

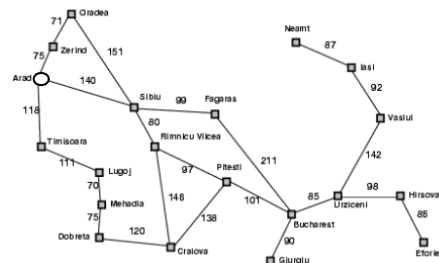


A* Search



A* search example

Arad
366=0+366

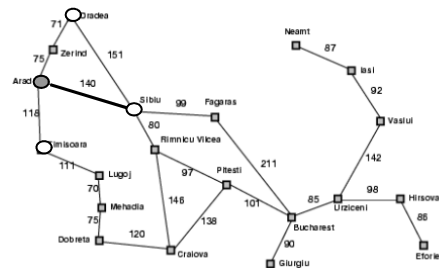


3/12/2018 MJ VeCoS ITU

87

A* search example

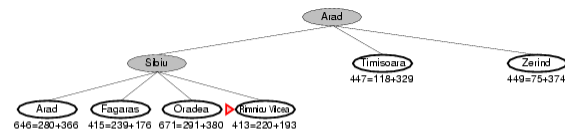
Arad
Sibiu 393=140+253
Timisoara 447=118+329
Zerind 449=75+374



3/12/2018 MJ VeCoS ITU

88

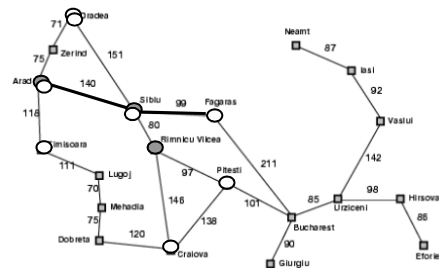
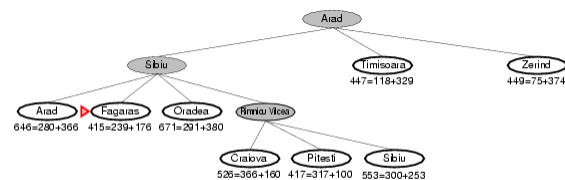
A* search example



3/12/2018 MJ VeCoS ITU

89

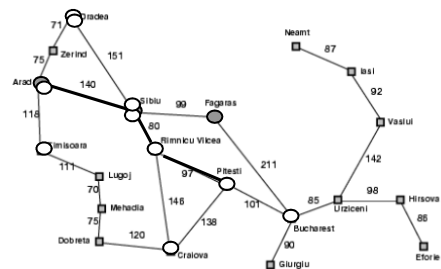
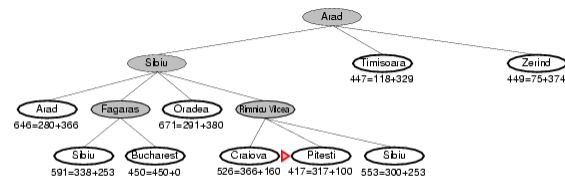
A* search example



3/12/2018 MJ VeCoS ITU

90

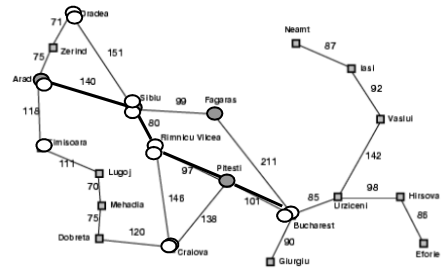
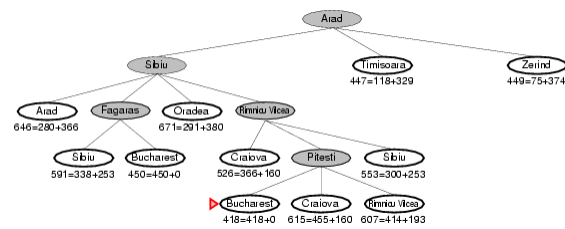
A* search example



3/12/2018 MJ VeCoS ITU

91

A* search example

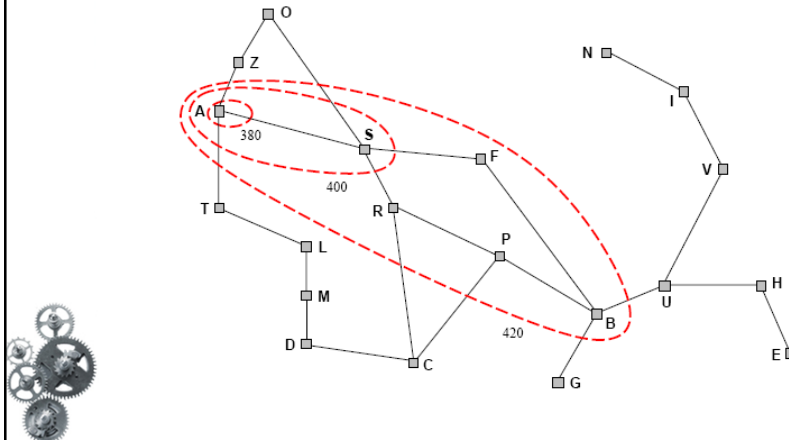


3/12/2018 MJ VeCoS ITU

92

Sifat A*

- Peta Romania memperlihatkan contour pada $f=380$, $f=400$, dan $f=420$ (Kondisi awal Arad)



IDA* (Iterative Deepening A*) Search

- Kombinasi A* dan iterative deepening
- Digunakan untuk mengurangi jumlah memory yang dipakai.
- Iterasi yang digunakan dengan Depth First Search yang dimodifikasi dengan menggunakan f-cost limit.
- Kelemahan IDA* :
Setiap contour naik satu state, waktu proses akan lebih lama



SMA*(Simplified Memory Bounded A*)Search

- IDA* memiliki jumlah memory yang kecil, tetapi tidak dapat menyimpan "sejarah" pencarian, sehingga ada kemungkinan proses pencarian yang terulang.
- SMA* menggunakan semua memory yang tersedia untuk menangani pencarian.



Sifat SMA*

- Tidak tergantung pada memory yang tersedia.
- SMA* menghindari pengulangan bagian sejauh memory mampu menampung.
- SMA* akan selesai jika memory yang tersedia cukup untuk menyimpan jalan solusi yang terdangkal.
- SMA* akan optimal, jika memory yang tersedia cukup untuk jalan solusi optimal yang terdangkal, jika tidak maka akan dikembalikan solusi yang terbaik yang dapat dicapai dengan memory yang tersedia.
- Bila memory cukup, memungkinkan untuk seluruh pencarian tree, maka pencarian tersebut optimal dan efisien.

