Solving problems by Searching for Solutions

AIMA Chapter 3.1 – 3.4

So far …

What is AI?

What are agents?

- Rational agents

Task environment

- Specifying
- Properties

Types of agents

- Reflex (simple & model based)
- Goal based
- Utility based

Searching for Solutions

Agent knows the current state

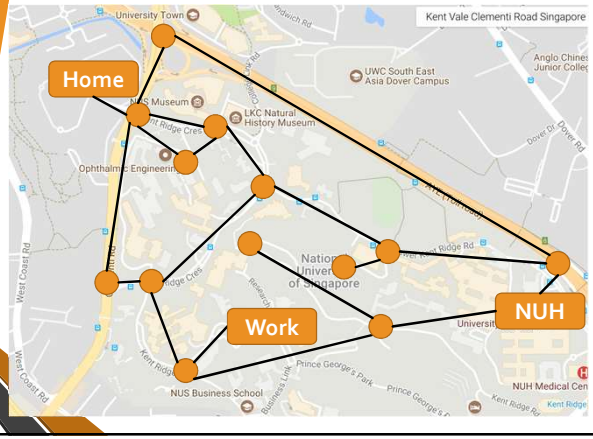
There are finite number of actions

- Fully observable, deterministic, discrete environment

Action have exactly one outcome

- Examples
  - Shortest route going through all cities
  - Winning sequence of moves in a **single player** game
  - Assembling a machine.
  - Find path through a maze
  - Route planning
  - Solution to Boolean satisfiability problem

Example: Route Planning



Problem Formulation

Initial state

- The start state

Actions

- $ACTION(s)$: set of actions applicable in s; e.g., $GO(NUH)$

Transition model

- Description of what each action does
- $s' = RESULT(s, a)$ state that results from doing a in s

Problem Formulation

Goal Test

- Is the state s equal the goal state?
- Explicit set of goal states, e.g., $\{In(Work)\}$
- Implicit function, e.g., $IsCheckmate(s)$

Path Cost

- Additive. e.g., sum of distances, number of actions executed
- $c(s, a, s')$: the step cost of taking action a in state s to reach state s'. Assumed to be ≥ 0

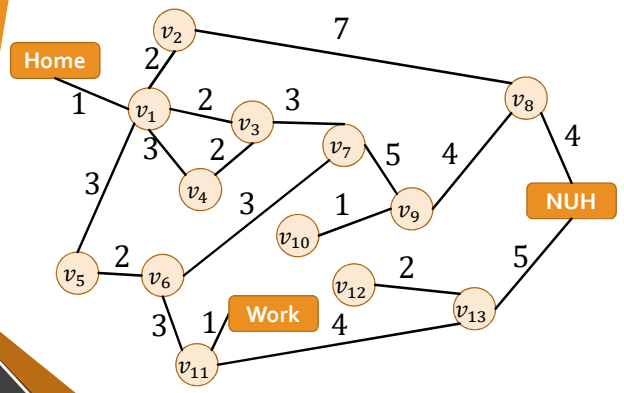
Solution: sequence of actions leading from initial to goal state

Is the solution unique?

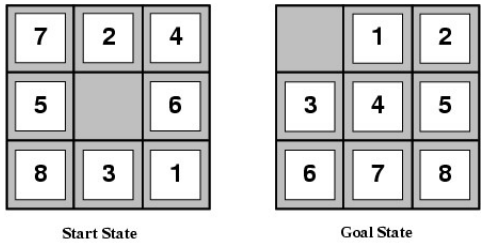
Is it optimal?

Can it be efficiently found?

Example: Route Planning

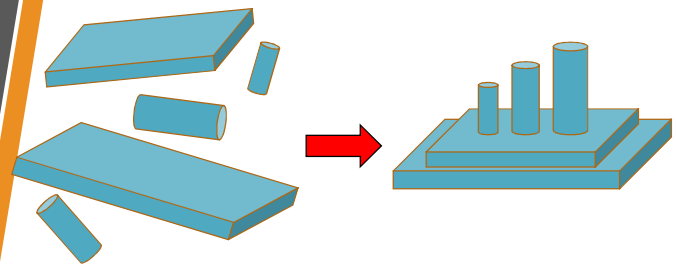


Example: 8 Puzzle



1. States (Initial State)?
2. Actions?
3. Transition Model?
4. Goal Test?
5. Cost Function?

Example: Item Assembly



1. States (Initial State)?
2. Actions?
3. Transition Model?
4. Goal Test?
5. Cost Function?

Tree Search Algorithms

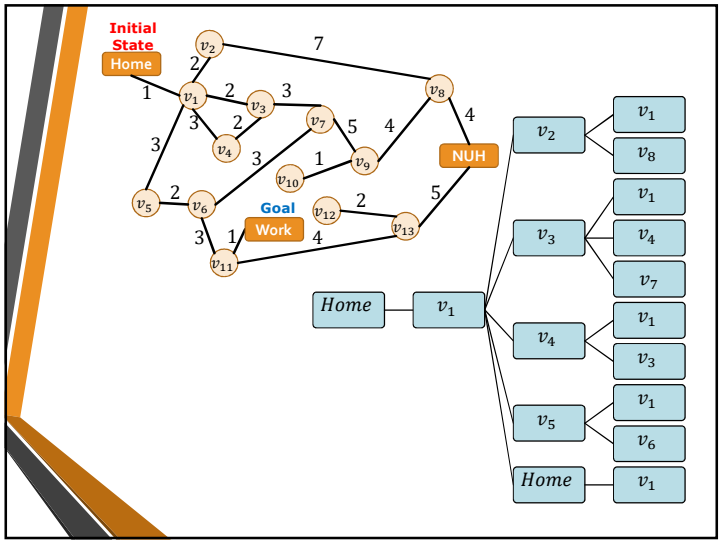
- Start at the source node, keep searching until we reach a goal state.
- Frontier: nodes that we have seen but haven't explored yet (at initialization: the frontier is just the source)
- At each iteration, choose a node from the frontier, explore it, and add its neighbors to the frontier.

Tree Search Algorithms

- Start at the source node, keep searching until we

```
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
```

- At each iteration, choose a node from the frontier, explore it, and add its neighbors to the frontier.



Traversal Methods

- In what order to we explore the frontier nodes? Search strategy
- Which neighbors do we add? All of them?

"Algorithms which do not remember their (search) history are doomed to repeat it"

Graph Search

- Graph Search "Remembers"!
- A node that's been explored once will not be revisited.

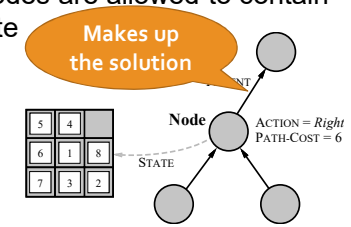
Graph Search

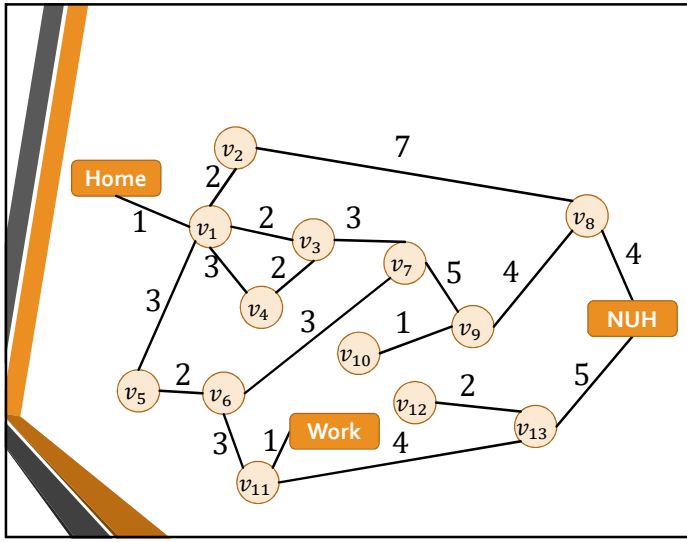
function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
initialize the frontier using the initial state of *problem*
initialize the explored set to be empty
loop do
if the frontier is empty **then return** failure
choose a leaf node and remove it from the frontier
if the node contains a goal 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

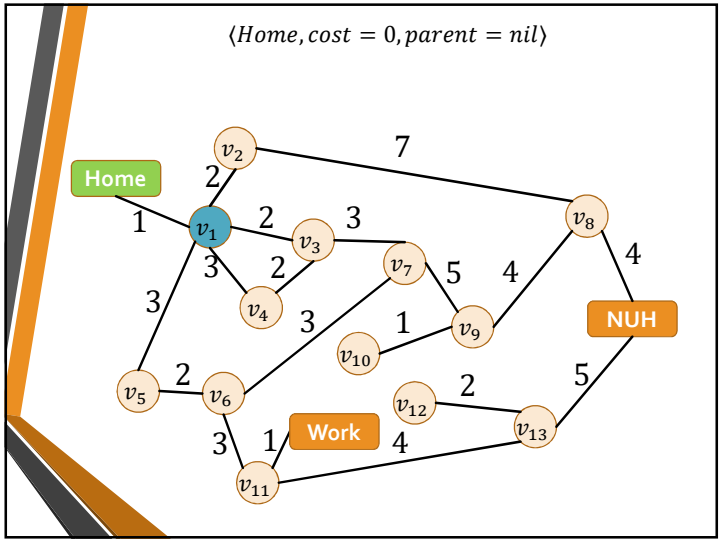
The names 'Graph Search' and 'Tree Search' can be unintuitive: remember their definitions!

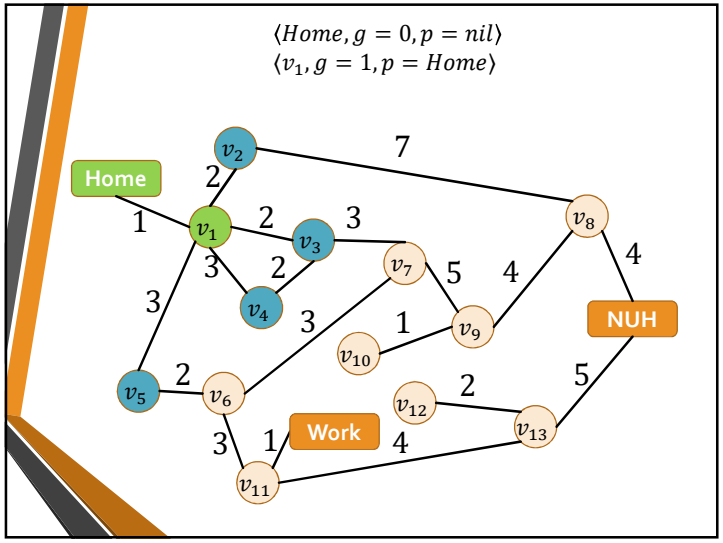
Implementation: States vs. Nodes

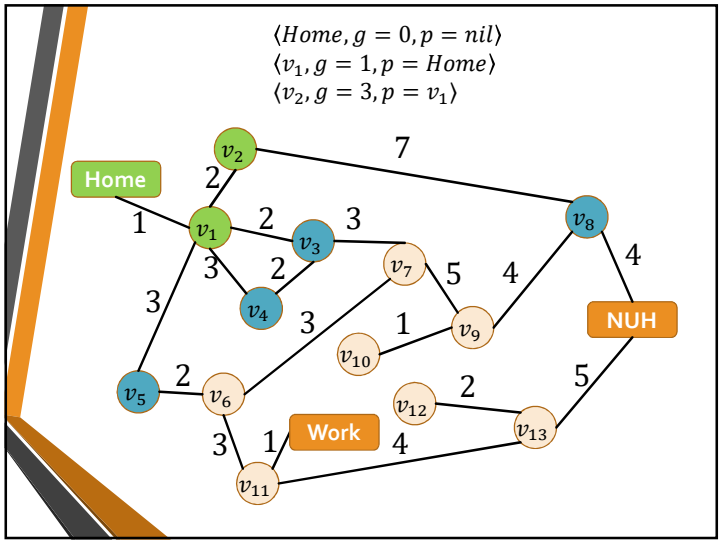
- A state represents a physical configuration
- A node is a data structure constituting part of search tree. It includes state, parent node, action, and path cost $g(n)$.
- Two different nodes are allowed to contain same world state

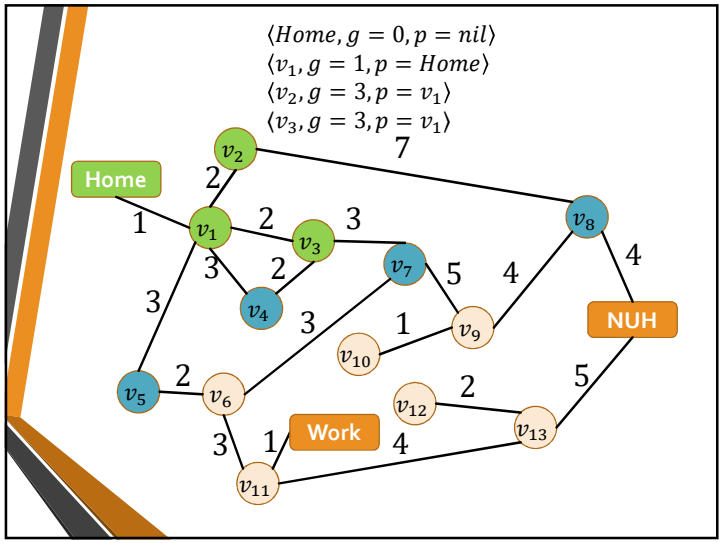


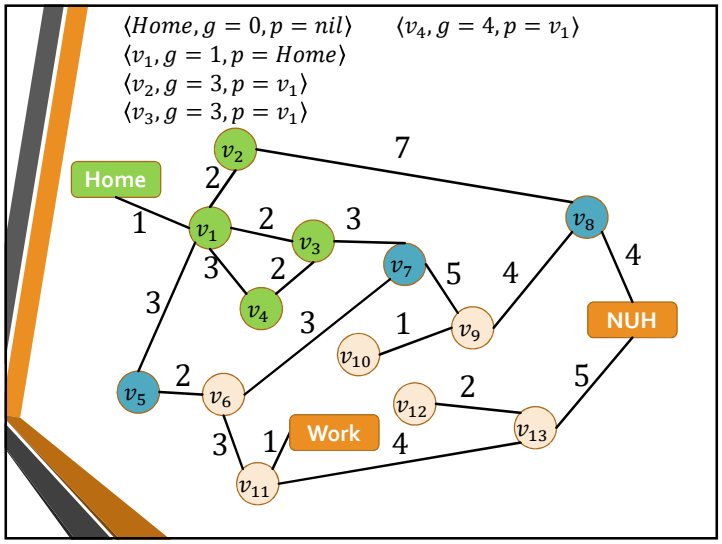












Search Strategies

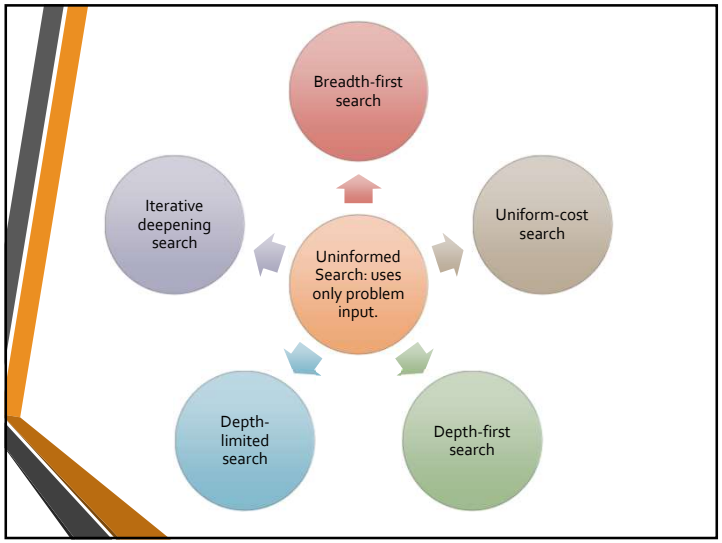
Any deterministic search algorithm is determined by the order of node expansion.

Evaluation Criteria

- completeness: always find a solution if exists
- optimality: find a least-cost solution
- time complexity: number of nodes generated
- space complexity: max. number of nodes in memory

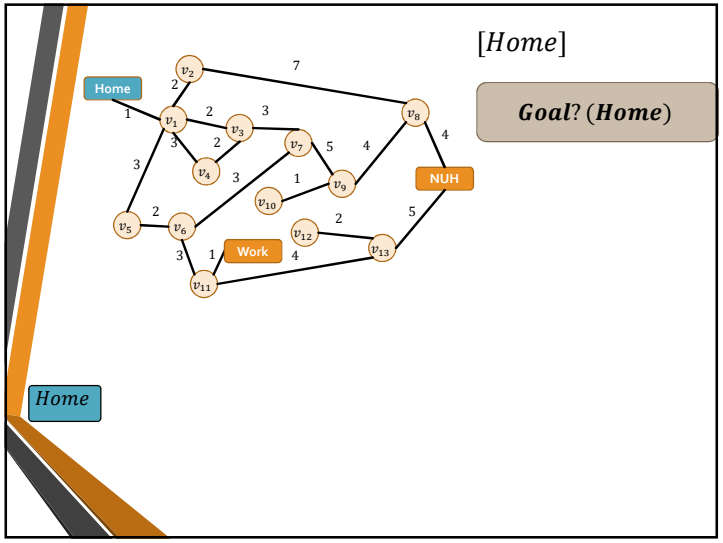
Problem Parameters

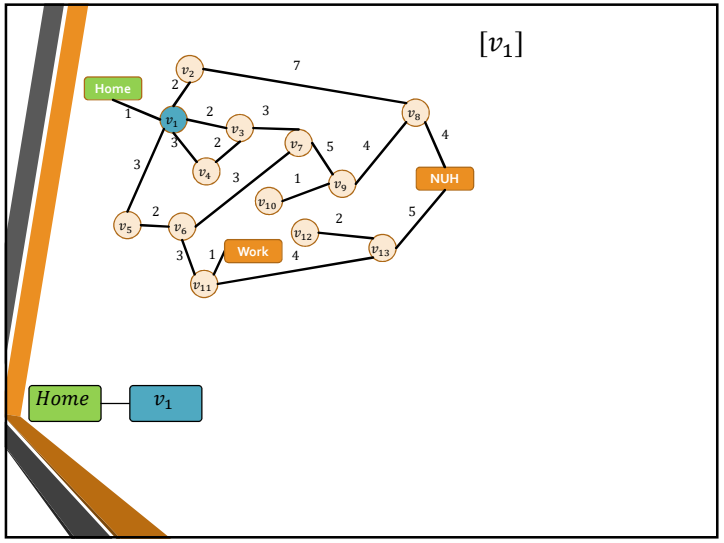
- b: maximum # of successors of any node (may be ∞)
- d: depth of shallowest **goal** node
- m: maximum depth of search tree (may be ∞)

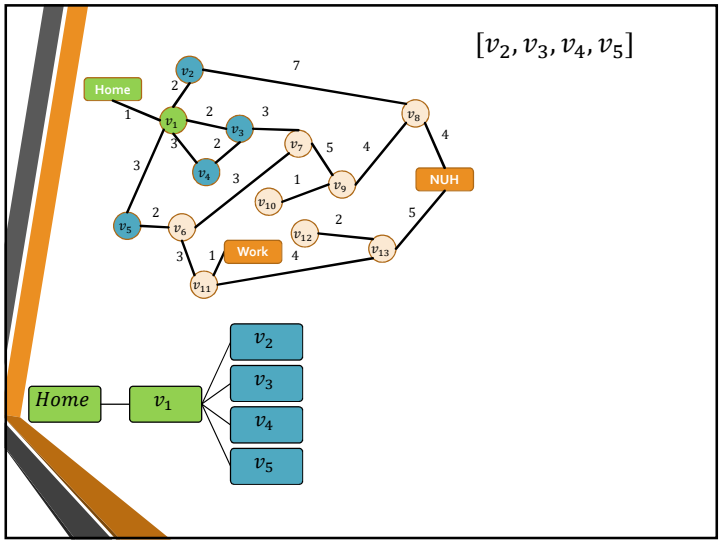


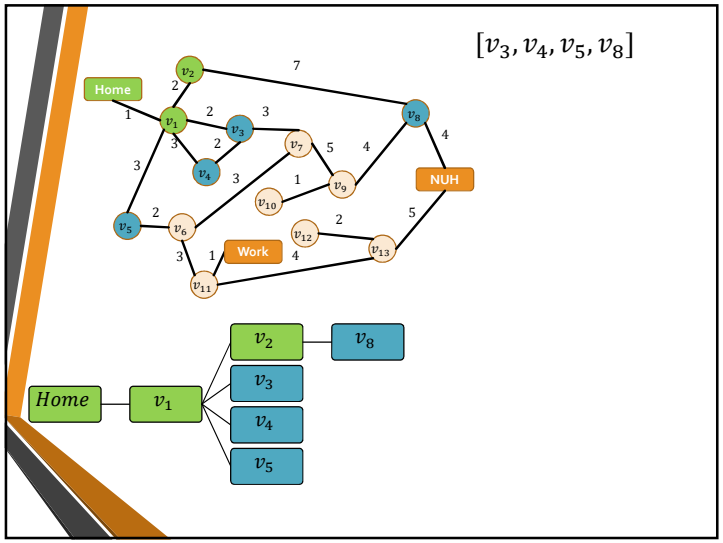
Breadth-First Search (BFS)

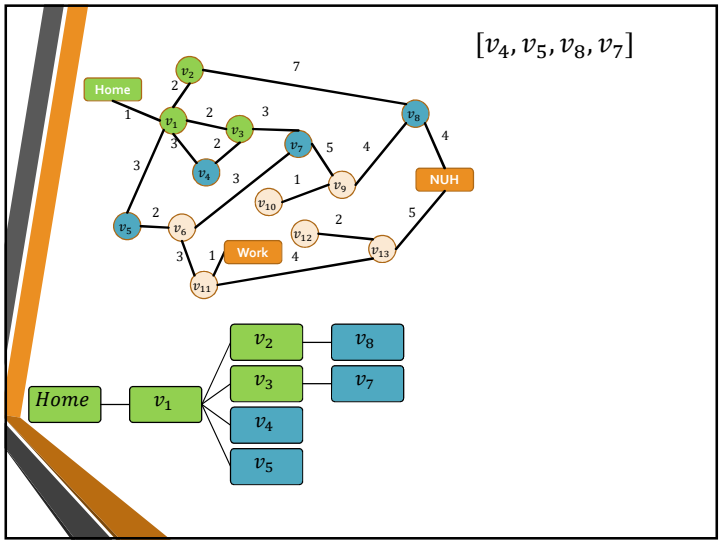
- Idea: Expand shallowest unexpanded node
- Implementation: Frontier is a FIFO queue, i.e., insert new successors at the end

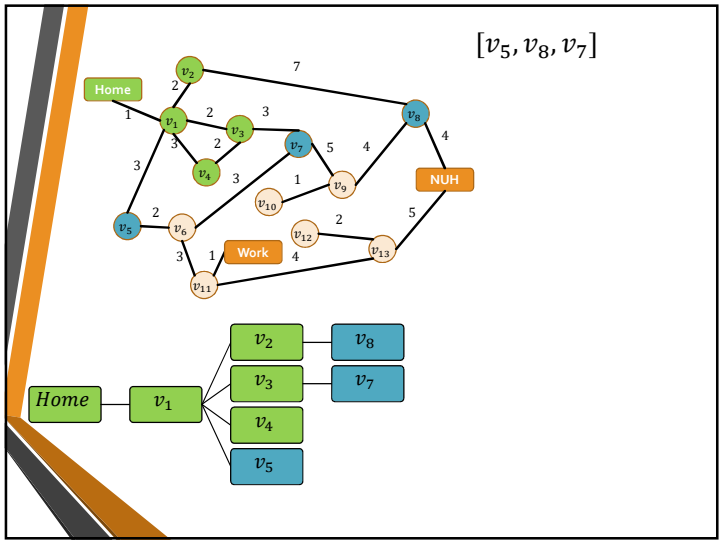


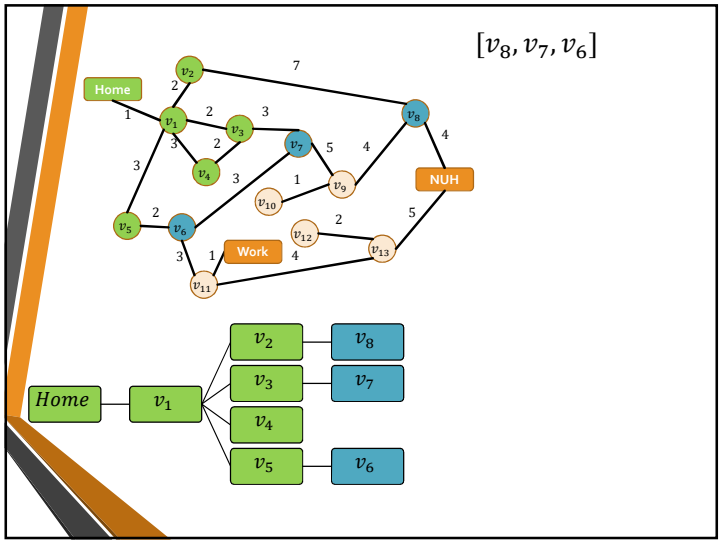


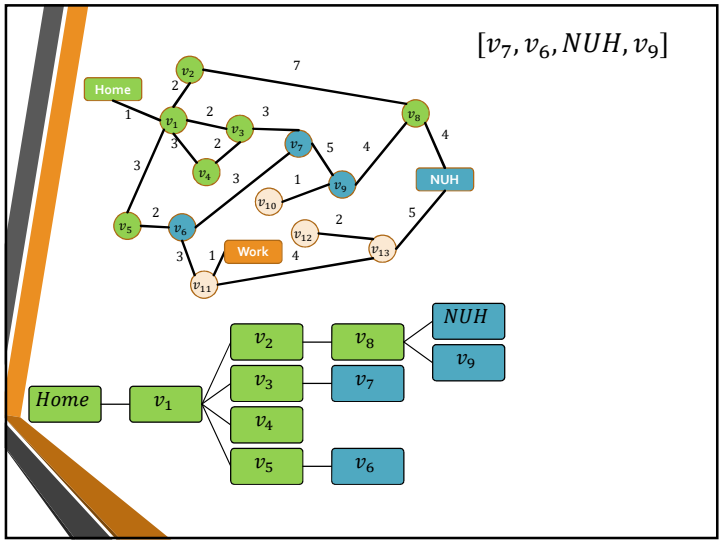

$[v_2, v_3, v_4, v_5]$
Home
NUH
Work
Home
v_1
v_2
v_3
v_4
v_5

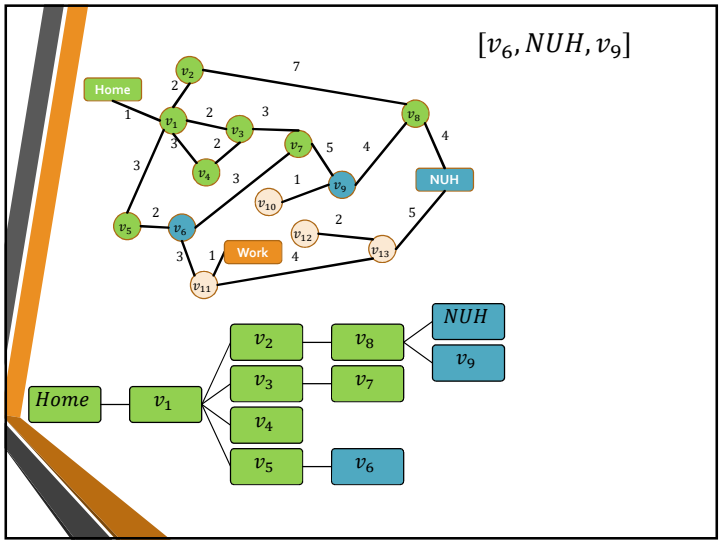

$[v_3, v_4, v_5, v_8]$
Home
NUH
Work
Home
v_1
v_2
v_8
v_3
v_4
v_5

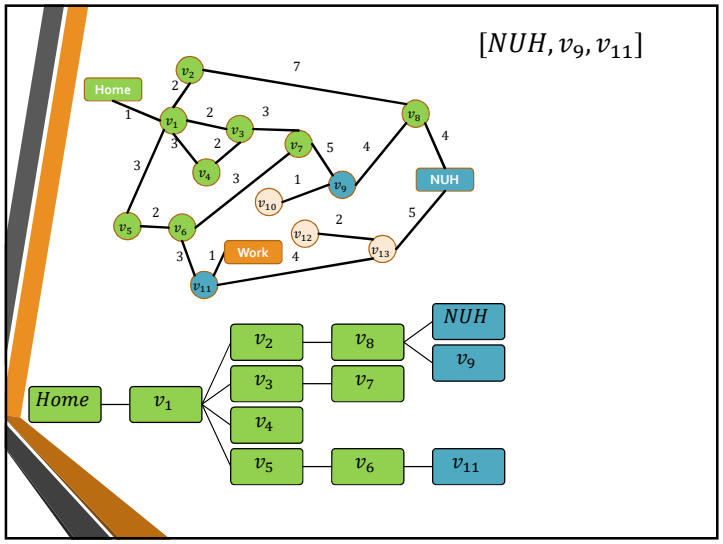

$[v_4, v_5, v_8, v_7]$
Home
NUH
Work
Home
v_1
v_2
v_8
v_3
v_7
v_4
v_5

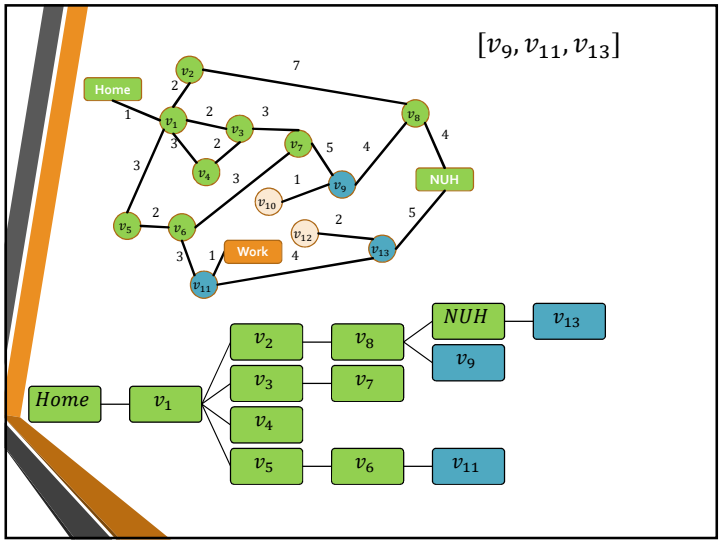

$[v_5, v_8, v_7]$
Home
NUH
Work
Home
v_1
v_2
v_8
v_3
v_7
v_4
v_5

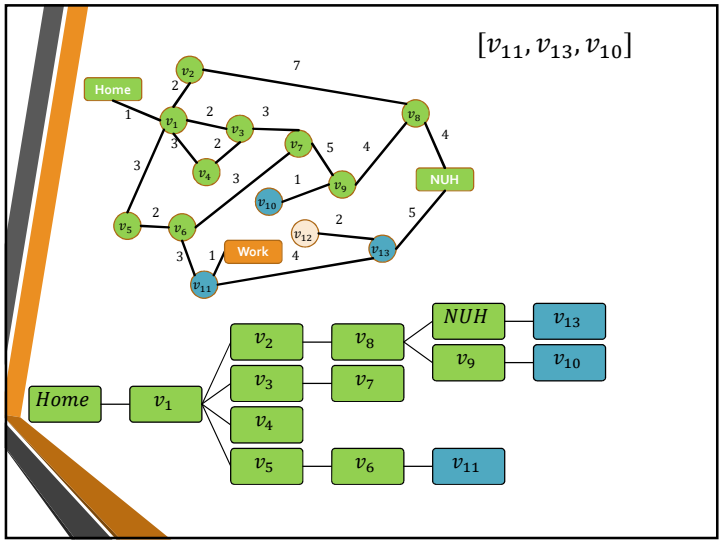

$[v_8, v_7, v_6]$
Home
NUH
Work
Home
v_1
v_2
v_3
v_4
v_5
v_8
v_7
v_6

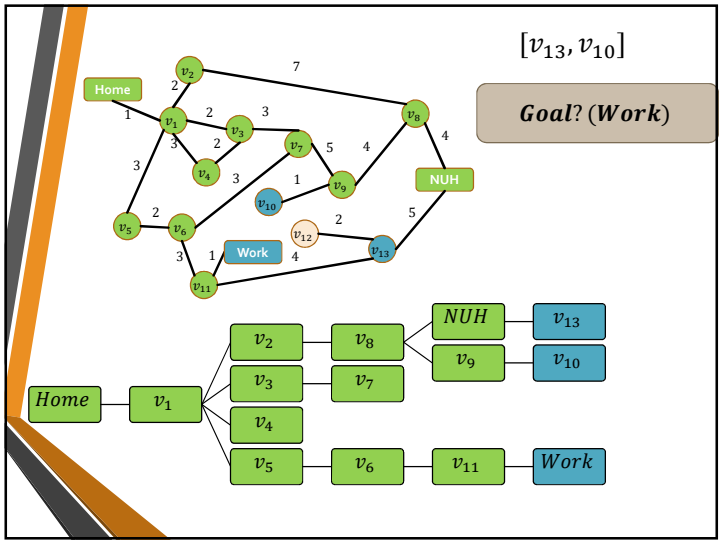

$[v_7, v_6, NUH, v_9]$
Home
NUH
Work
Home
v_1
v_2
v_3
v_4
v_5
v_8
v_7
v_6
NUH
v_9


$[v_6, NUH, v_9]$
Home
NUH
Work
Home
v_1
v_2
v_3
v_4
v_5
v_8
v_7
v_6
NUH
v_9


$[NUH, v_9, v_{11}]$
Home
NUH
Work
Home
v_1
v_2
v_3
v_4
v_5
v_8
v_7
v_6
NUH
v_9
v_{11}







Properties of BFS

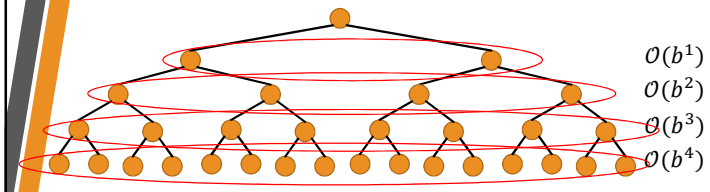
Property	
Complete?	Yes (if b is finite)
Optimal	No (unless step cost $= 1$)
Time	$\mathcal{O}(b) + \mathcal{O}(b^2) + \cdots + \mathcal{O}(b^d) = \mathcal{O}(b^d)$
Space	Max size of frontier $\mathcal{O}(b^d)$

Space is the bigger problem (more than time)

completeness: always find a solution if exists
optimality: find a least-cost solution
time complexity: number of nodes generated
space complexity: max. number of nodes in memory

Properties of BFS

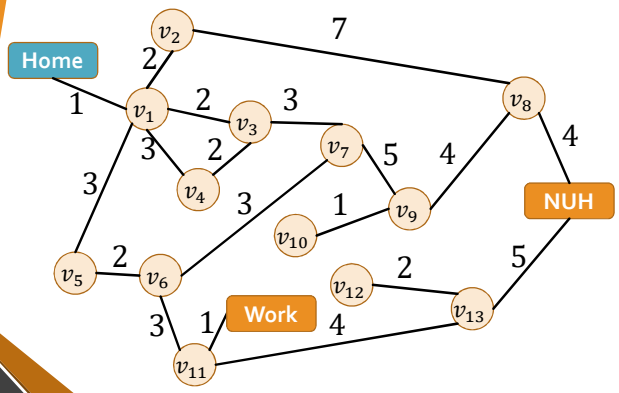
$b = 2$



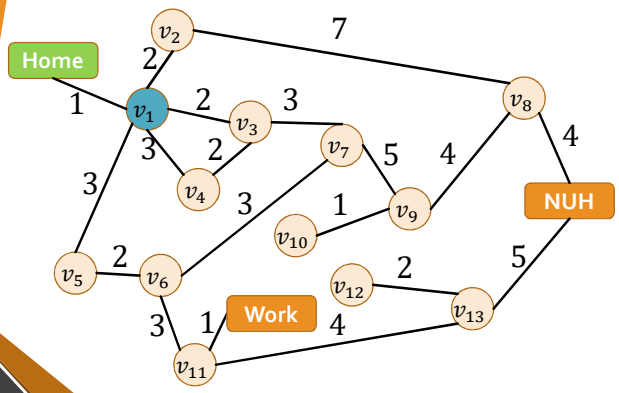
Uniform-Cost Search (UCS)

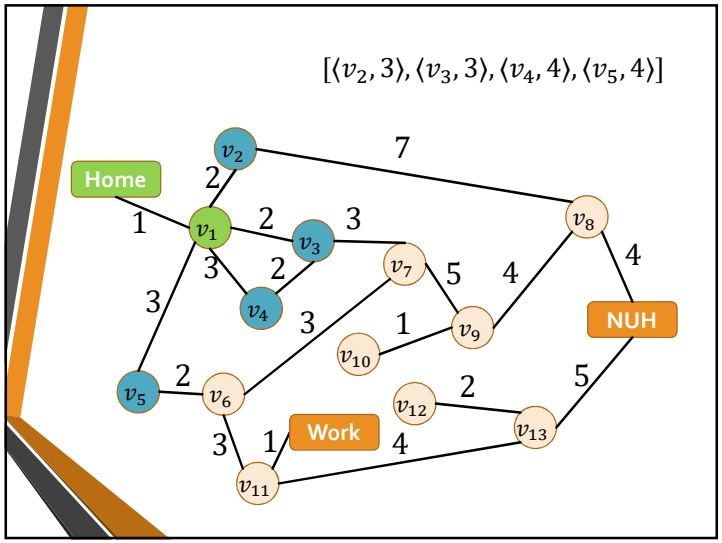
- Idea: expand least-path-cost unexpanded node
- Frontier: priority queue ordered by path cost g
- Equivalent to BFS if all step costs are equal

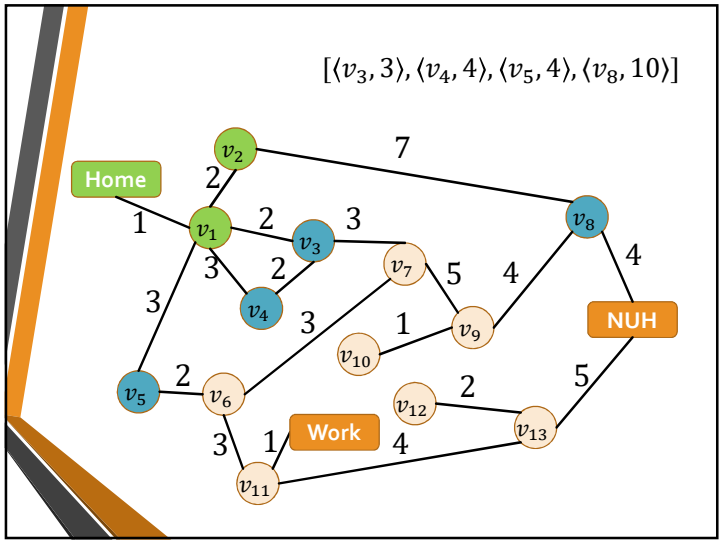
$[\langle Home, 0 \rangle]$

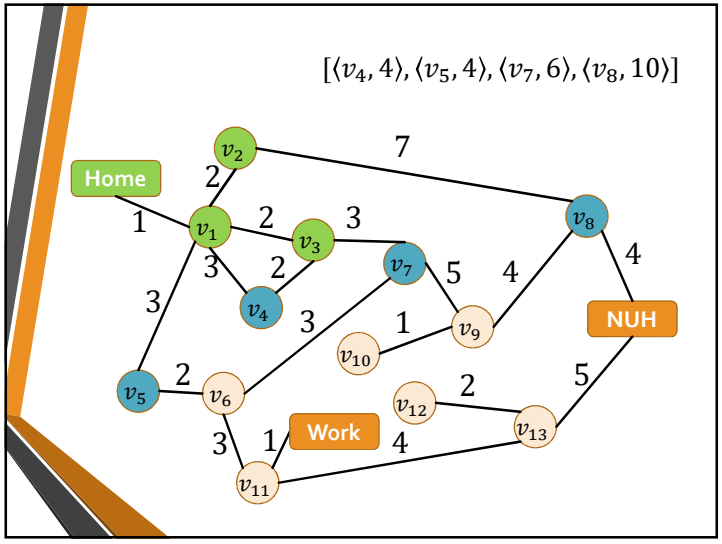


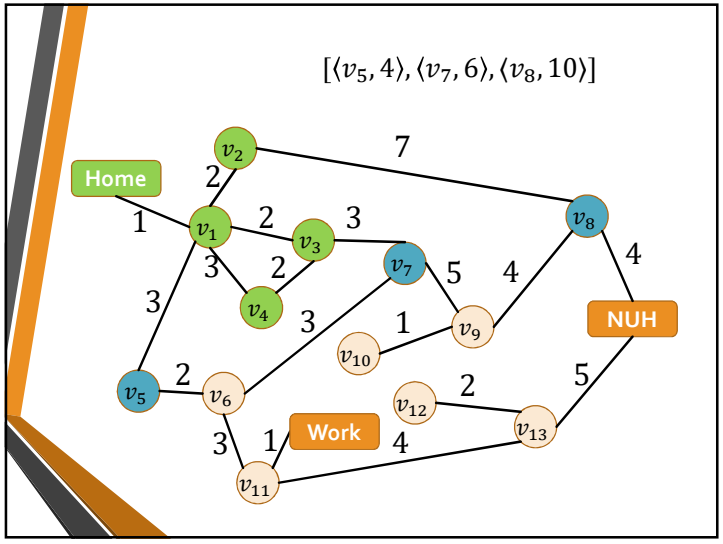
$[\langle v_1, 1 \rangle]$

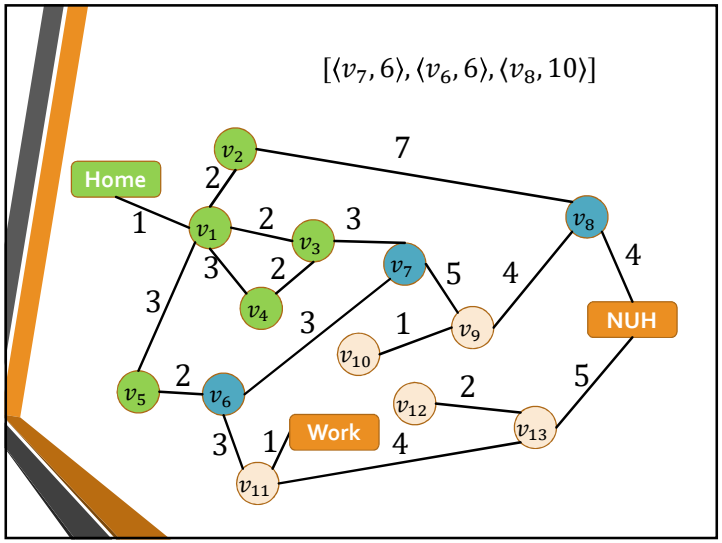


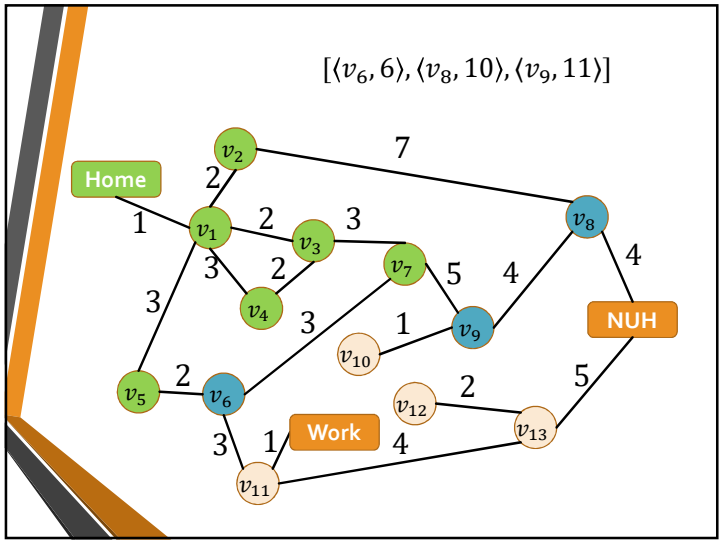

$[\langle v_2, 3\rangle, \langle v_3, 3\rangle, \langle v_4, 4\rangle, \langle v_5, 4\rangle]$
Home
NUH
Work
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}

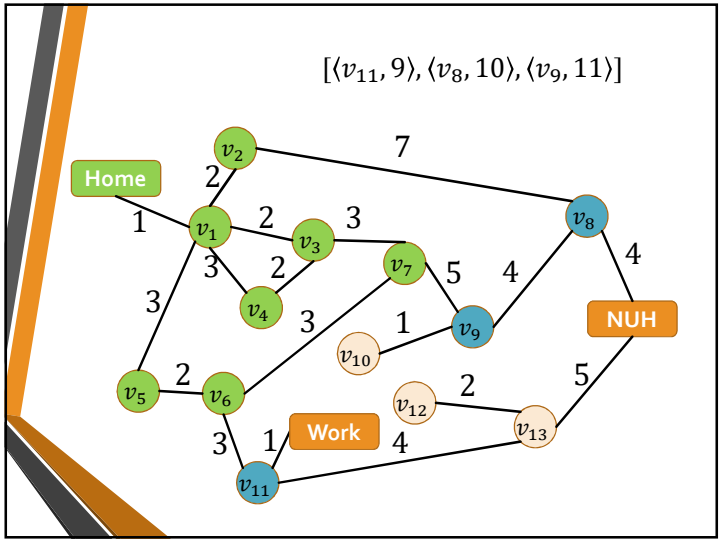

$[\langle v_3, 3\rangle, \langle v_4, 4\rangle, \langle v_5, 4\rangle, \langle v_8, 10\rangle]$
Home
NUH
Work
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}

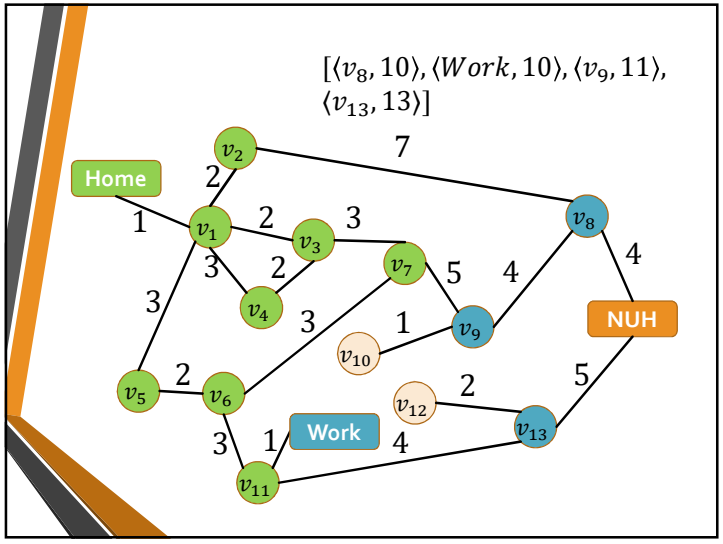

$[\langle v_4, 4\rangle, \langle v_5, 4\rangle, \langle v_7, 6\rangle, \langle v_8, 10\rangle]$
Home
NUH
Work
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}

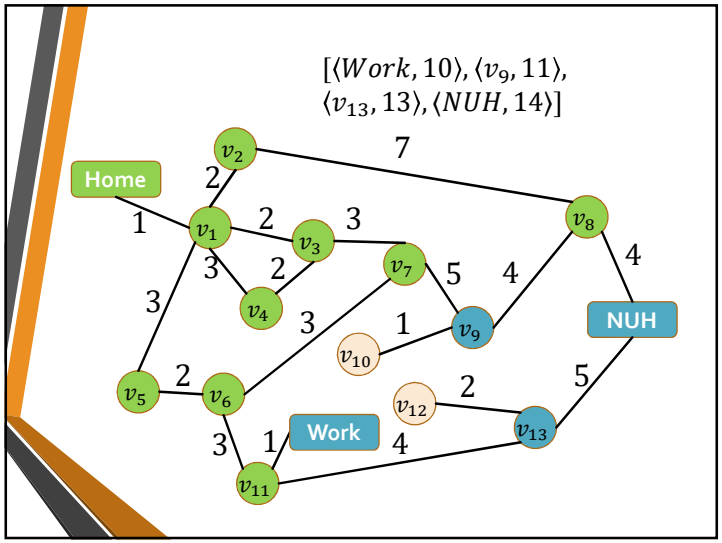

$[\langle v_5, 4\rangle, \langle v_7, 6\rangle, \langle v_8, 10\rangle]$
Home
NUH
Work
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}

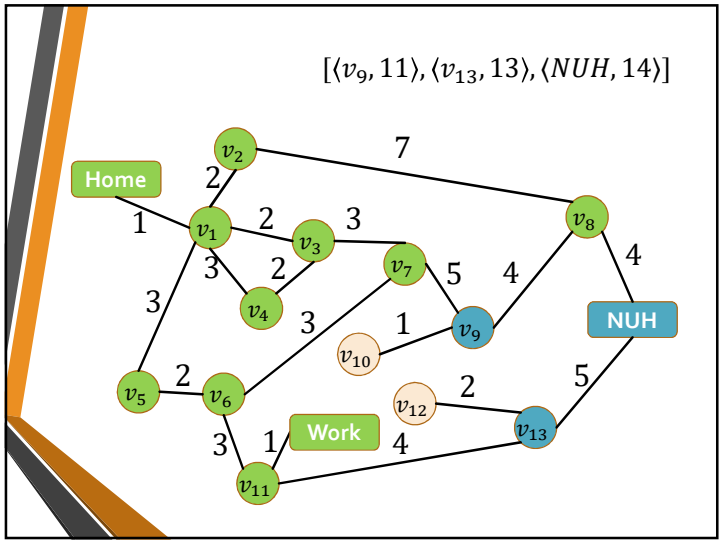

$[\langle v_7, 6\rangle, \langle v_6, 6\rangle, \langle v_8, 10\rangle]$
Home
NUH
Work
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}


$[\langle v_6, 6\rangle, \langle v_8, 10\rangle, \langle v_9, 11\rangle]$
Home
NUH
Work
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}


$[\langle v_{11}, 9\rangle, \langle v_8, 10\rangle, \langle v_9, 11\rangle]$
Home
NUH
Work
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}


$[\langle v_8, 10\rangle, \langle Work, 10\rangle, \langle v_9, 11\rangle, \langle v_{13}, 13\rangle]$
Home
NUH
Work
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}





Properties of Uniform Cost Search

Property	
Complete?	Yes (if all step costs are $\geq \varepsilon$)
Optimal	Yes (shortest path nodes expanded first)
Time	$\mathcal{O}\left(b^{1+\left\lfloor \frac{C^*}{\varepsilon} \right\rfloor}\right)$ where C^* is the optimal cost.
Space	$\mathcal{O}\left(b^{1+\left\lfloor \frac{C^*}{\varepsilon} \right\rfloor}\right)$

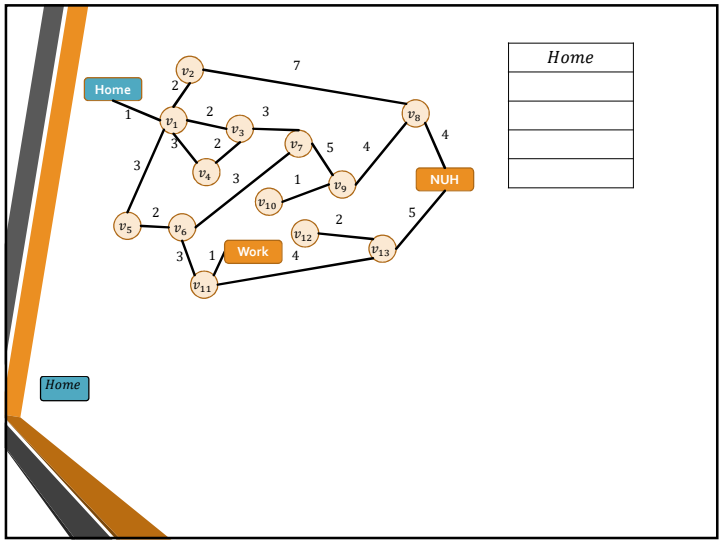
completeness: always find a solution if exists
optimality: find a least-cost solution
time complexity: number of nodes generated
space complexity: max. number of nodes in memory

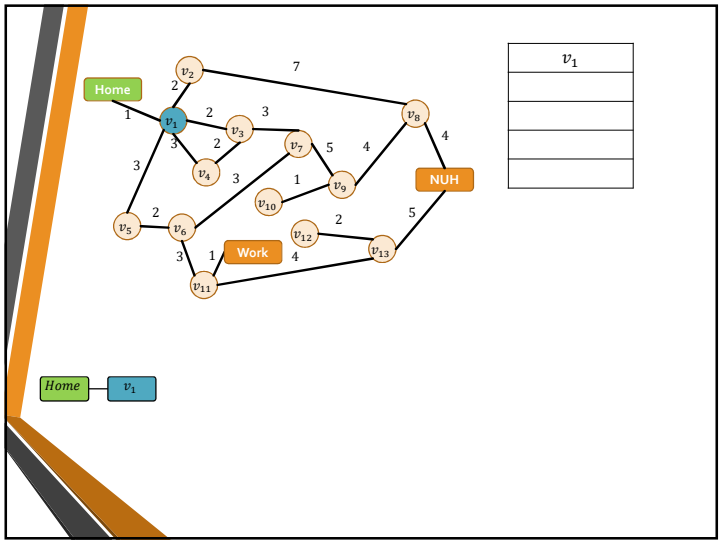
Uniform Cost Search

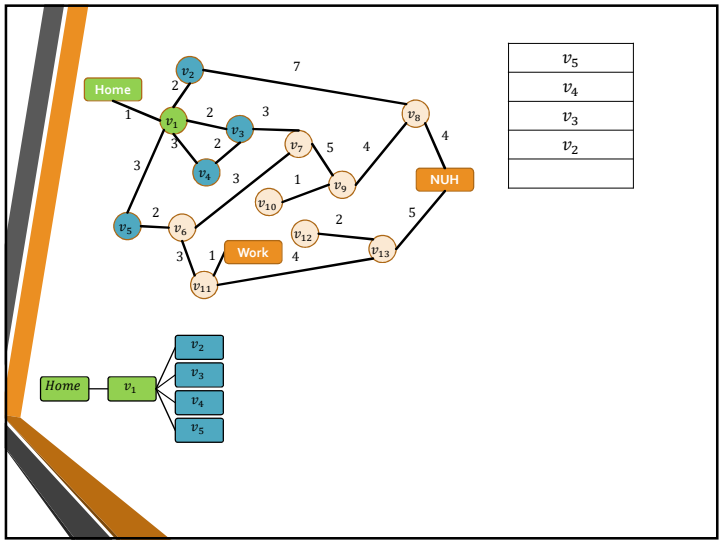
- At every round we get at least a distance of ε closer to the goal.
- Reach nodes at distance $0, \varepsilon, 2\varepsilon, \ldots, \left\lfloor \frac{C^*}{\varepsilon} \right\rfloor \varepsilon$ of goal; total $\left\lfloor \frac{C^*}{\varepsilon} \right\rfloor + 1$ steps.
- At step k (depth k at most): keep $\leq b^k$ nodes in frontier.

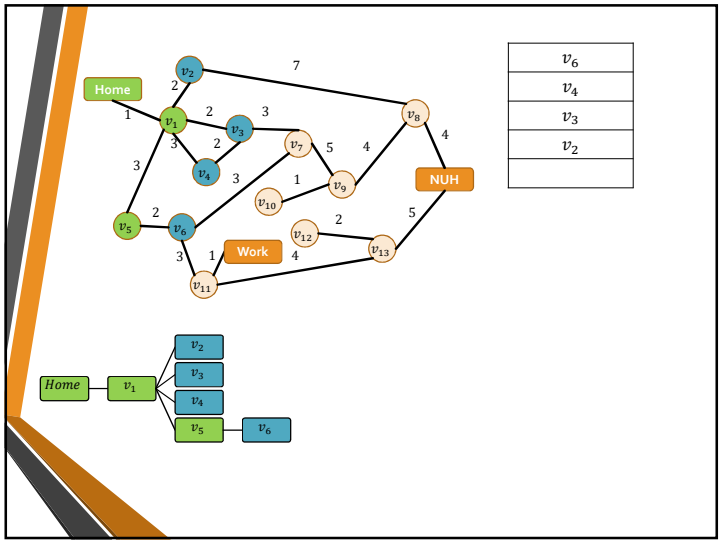
Depth-First Search

- Idea: Expand deepest unexpanded node
- Implementation: Frontier = LIFO stack, i.e., insert successors at the front

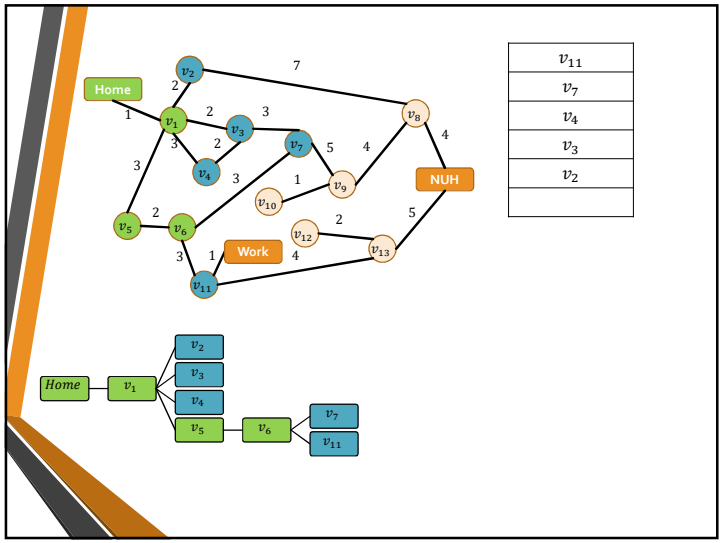




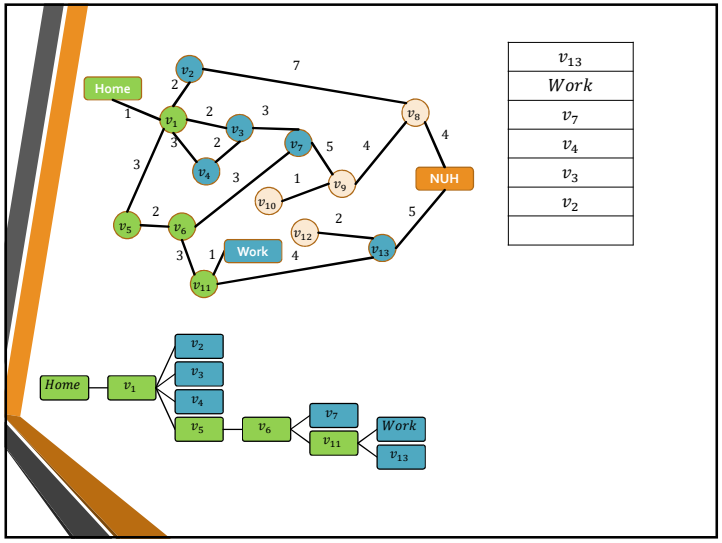




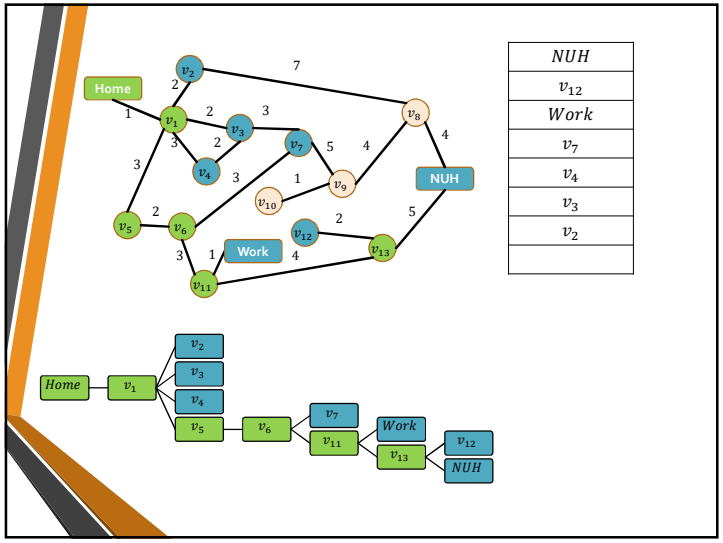
v_6
v_4
v_3
v_2



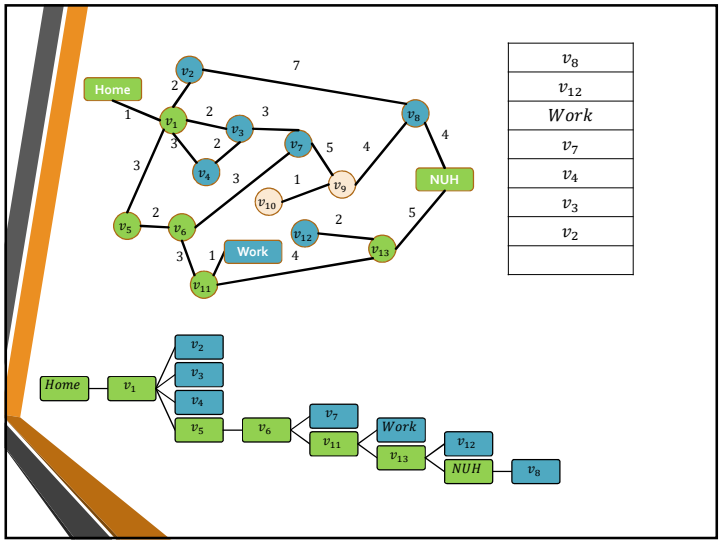
v_{11}
v_7
v_4
v_3
v_2

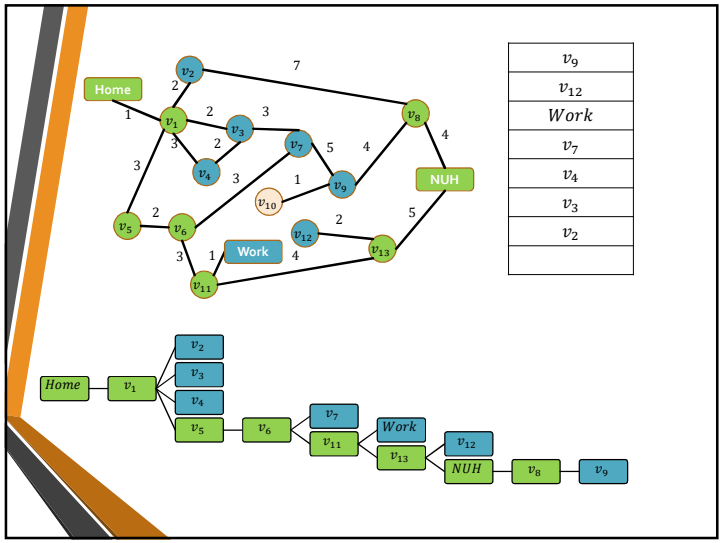


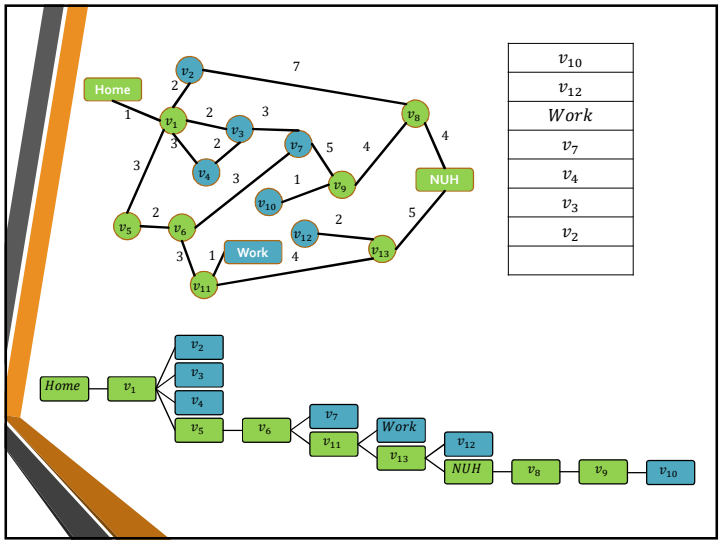
v_{13}
$Work$
v_7
v_4
v_3
v_2

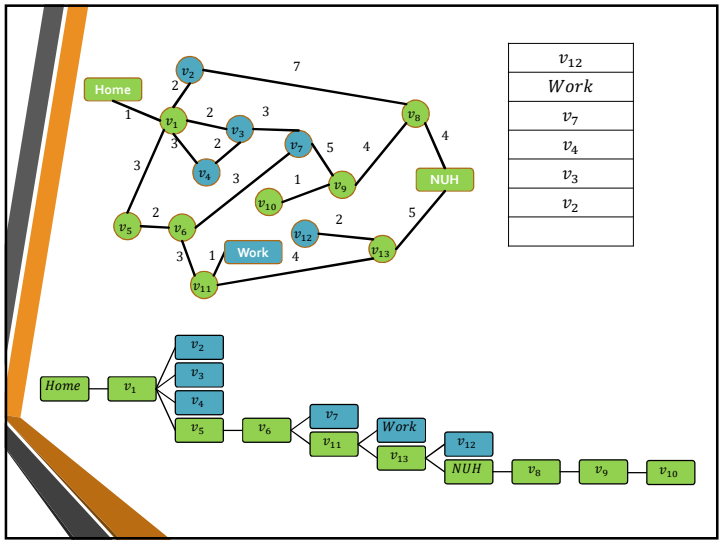


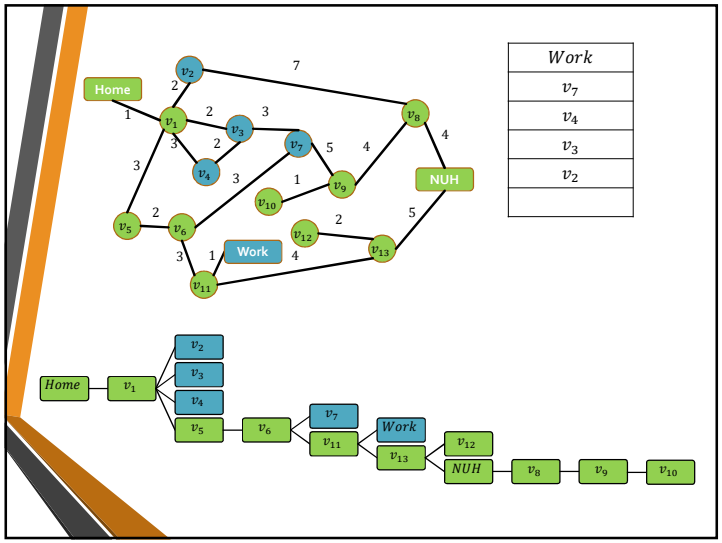
NUH
v_{12}
$Work$
v_7
v_4
v_3
v_2

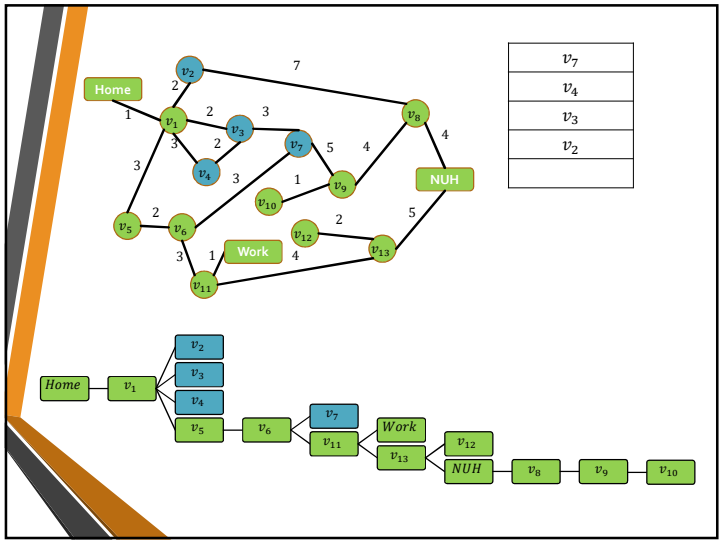

Home
Work
NUH
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}
v_8
v_{12}
$Work$
v_7
v_4
v_3
v_2
Home v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_{11} $Work$ v_{13} v_{12} NUH v_8


Home
Work
NUH
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}
v_9
v_{12}
$Work$
v_7
v_4
v_3
v_2
Home v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_{11} $Work$ v_{13} v_{12} NUH v_8 v_9


Home
Work
NUH
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}
v_{10}
v_{12}
$Work$
v_7
v_4
v_3
v_2
Home v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_{11} $Work$ v_{13} v_{12} NUH v_8 v_9 v_{10}


Home
Work
NUH
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}
v_{12}
$Work$
v_7
v_4
v_3
v_2
Home v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_{11} $Work$ v_{13} v_{12} NUH v_8 v_9 v_{10}





Properties of DFS

Property	
Complete?	No on infinite depth graphs
Optimal	No
Time	$\mathcal{O}(b^m)$
Space	$\mathcal{O}(bm)$ (can be $\mathcal{O}(m)$)

completeness: always find a solution if exists
optimality: find a least-cost solution
time complexity: number of nodes generated
space complexity: max. number of nodes in memory

Depth-First Search

- When checking a node v, we push at most b descendants to stack.
- We do so at most m times ⇒ $\mathcal{O}(bm)$ space.
- Do we really need to push **all** b descendants to the stack?

Depth-Limited Search (DLS)

- Idea: run DFS with depth limit ℓ, i.e., do not search at depth greater than ℓ.
- Same guarantees as DFS, with ℓ instead of m ($\mathcal{O}(b^{\ell})$ time; $\mathcal{O}(b\ell)$ space)

Iterative Deepening Search (IDS)

- Idea: Perform DLSs with increasing depth limit until goal node is found
- Better if state space is large and depth of solution is unknown
- Implementation:

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    for depth = 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
```

How Wasteful is IDS?

- Number of nodes generated in DLS to depth ℓ with branching factor b:

$$\mathcal{O}(b^0) + \mathcal{O}(b^1) + \mathcal{O}(b^2) + \cdots + \mathcal{O}(b^{\ell-2}) + \mathcal{O}(b^{\ell-1}) + \mathcal{O}(b^{\ell})$$

Number of nodes generated in IDS to depth d with branching factor b:

$$(d+1)\mathcal{O}(b^0) + d\mathcal{O}(b^1) + (d-1)\mathcal{O}(b^2) + \cdots 3\mathcal{O}(b^{d-2}) + 2\mathcal{O}(b^{d-1}) + \mathcal{O}(b^{d})$$

- For $b = 10, d = 5$,
  - $N_{DLS} = 1 + 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}111$
  - $N_{IDS} = 6 + 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}456$
- Overhead $= \frac{123{,}456 - 111{,}111}{111{,}111} = 11\%$

Properties of IDS

Property	
Complete?	Yes (if b is finite)
Optimal	No (unless step cost is 1)
Time	$\mathcal{O}(b^{d})$
Space	$\mathcal{O}(bd)$ (can be $\mathcal{O}(d)$)

completeness: always find a solution if exists
optimality: find a least-cost solution
time complexity: number of nodes generated
space complexity: max. number of nodes in memory

All in

Property	BFS	UCS	DFS	DLS	IDS
Complete	Yes[1]	Yes[2]	No	No	Yes[1]
Optimal	No[3]	Yes	No	No	No[3]
Time	$\mathcal{O}(b^d)$	$\mathcal{O}\left(b^{1+\left\lfloor \frac{C^*}{\varepsilon} \right\rfloor}\right)$	$\mathcal{O}(b^m)$	$\mathcal{O}(b^\ell)$	$\mathcal{O}(b^d)$
Space	$\mathcal{O}(b^d)$	$\mathcal{O}\left(b^{1+\left\lfloor \frac{C^*}{\varepsilon} \right\rfloor}\right)$	$\mathcal{O}(bm)$	$\mathcal{O}(b\ell)$	$\mathcal{O}(bd)$

1. BFS and IDS are complete if b is finite.
2. UCS is complete if b is finite and step cost $\geq \varepsilon$
3. BFS and IDS are optimal if all step costs are identical

Choosing a Search Strategy

- Depends on the problem:
  - Finite/infinite depth of search tree
  - Known/unknown solution depth
  - Repeated states
  - Identical/non-identical step costs
  - Completeness and optimality needed?
  - Resource constraints (e.g., time, space)

83

Can We Do Better?

- Yes! Exploit problem-specific knowledge; obtain heuristics to guide search
- How?
  - Informed (heuristic) search
  - Expand "more promising" nodes.

84

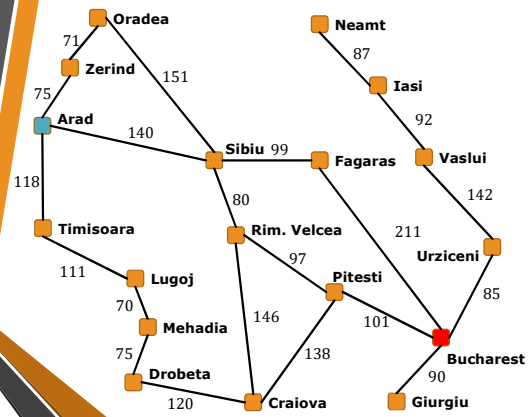
Best-First Search

- Idea: use an evaluation function $f(n)$ for each node n
  - Cost estimate → Expand node with lowest evaluation/cost first
- Implementation:

  Frontier = priority queue ordered by nondecreasing cost f
- Special cases (different choices of f):
  - Greedy best-first search
  - A* search

85

Romania with Step Costs (km)

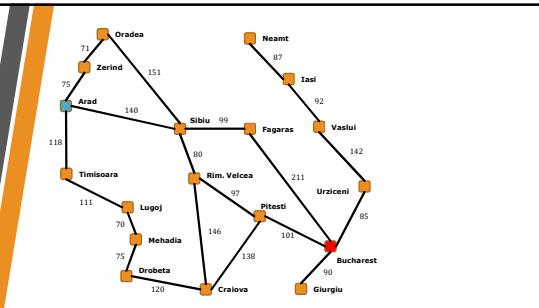


h_{SLD} to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Fagaras	176
Giurgiu	77
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
R. Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

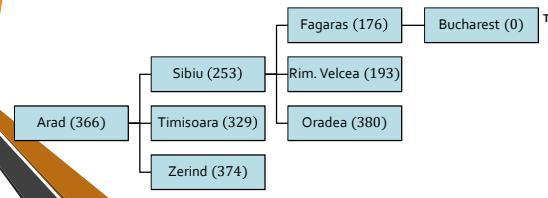
Greedy Best-First Search

- Evaluation function $f(n) = h(n)$ (heuristic function) = estimated cost of cheapest path from n to goal
- e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal

87



h_{SLD} to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Fagaras	176
Giurgiu	77
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
R. Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



88

Properties of Greedy Best-First Search

Property	
Complete?	Yes (if b is finite)
Optimal	No (shortest path to Bucharest: $418km$)
Time	$\mathcal{O}(b^m)$, but a good heuristic can reduce complexity substantially
Space	Max size of frontier $\mathcal{O}(b^m)$

What Important Information Does the Algorithm Ignore?

89