Uninformed Search

Wednesday, 18 August 2021 9:54 AM

Logistics

- Every tutorial will be composed of two parts:
  - Questions discussed in the tutorial (Discussed in Week X)
  - Questions to be submitted (Due in Week X+1)
    - Ensures you attempt questions after you have had time to absorb the material properly.
- Welcome Quiz
  - If your score is below 30, you will find module very hard.
- All private queries: cs3243private@googlegroups.com
  - Read by selected TAs and me

2

Different Kinds of Agent Programs

Type	Characteristics	Analogy
Simple Reflex	Action depends only on percept	**Infant** • If Hungry then Cry • If Happy then Sleep
Model-based Reflex	Action depends on internal state (based on percept history), model of the world, and percept	**Kid** If Want candy & Didn't Receive candy then Ask for candy
Goal-based	Action depends on current state, percepts, model of the world Action plan to achieve desired goal	**Teenager** Goal: Want to get into NUS Plan: Study and Playing based on the Goal.
Utility-based	Useful for multiple (possible) conflicting goals A weighted combination of goals	**Adult** α*Job+ β*Partner+ γ*Health

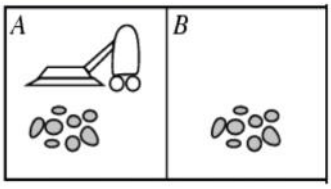
3

Goal-Based Agents

- Deterministic
- Fully Observable

4

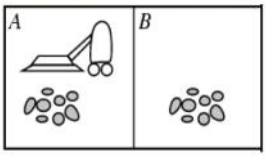
The Life of Our MopBot



Modeling is the key

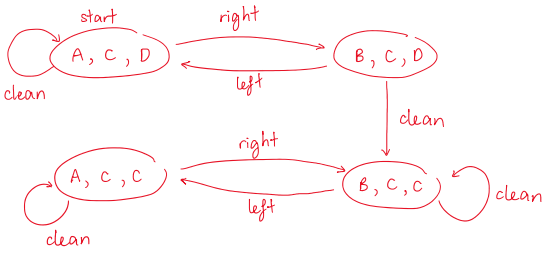
5

Abstraction

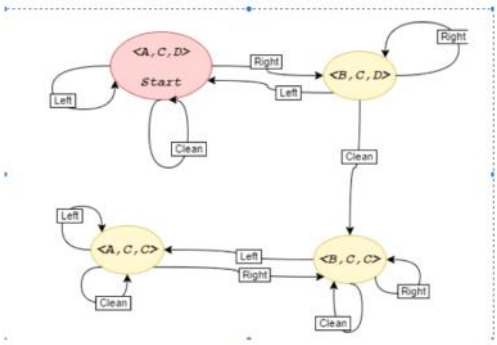


1. State (of Environment): < Location, Status(A), Status(B) >
2. Actions: {Left,Right,Clean,Idle}
3. Transition Model: $g: State \times Action \rightarrow State$
4. Performance Measure: $h: State \times Action \rightarrow \mathbb{R}^+$ → cost / benefit
5. Goal State(s): {State1, State2, . . .}
6. Start State: e.g < Room A, Clean, Dirty >

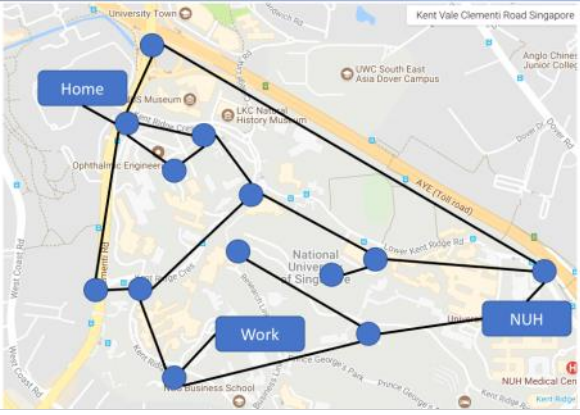
6



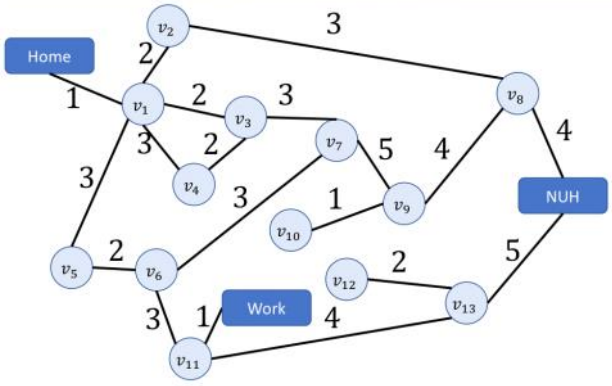
Abstraction



Route Finding



Route Finding



Searching for Solutions

Solution: Sequence of actions leading from initial state to goal state

Goal Test

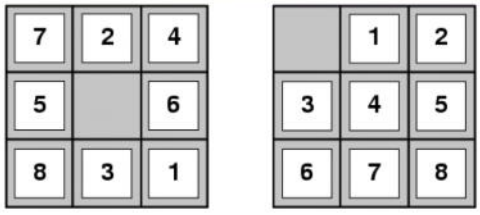
- Is the state s equal the goal state?
- Explicit set of goal states, e.g., $\{In(Work)\}$
- Implicit function, e.g., $IsCheckmate(s)$

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

10

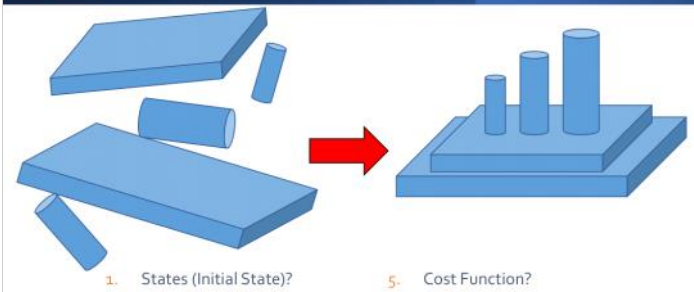
Example: 8 Puzzle



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

11

Example: Item Assembly



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

12

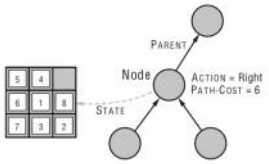
All about Solutions

- Solution: Sequence of actions leading from initial state to goal state
- Properties that we care about:
  - Is the solution unique?
  - Is it optimal?
  - Can it be efficiently found?

13

Implementation Details: 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



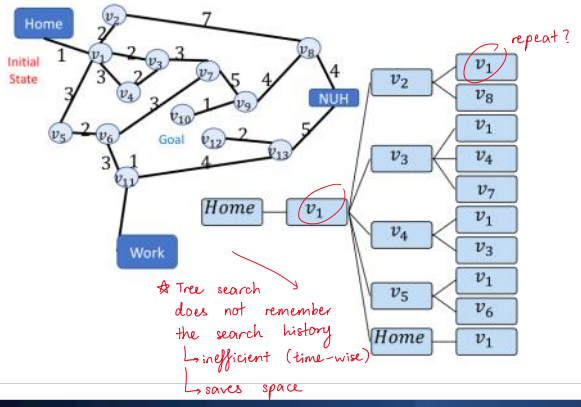
14

Tree Search Algorithms
(Learnings from CS1231)

- Start at the initial node, keep searching until we reach a goal node.
- 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.

15

Tree Search



☆ Tree search does not remember the search history
↳ inefficient (time-wise)
↳ saves space

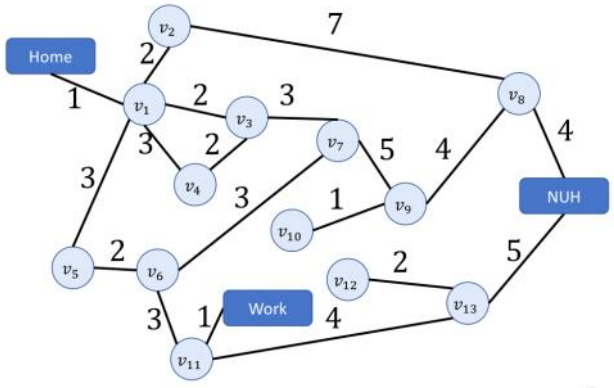
16

Traversal Algorithms

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

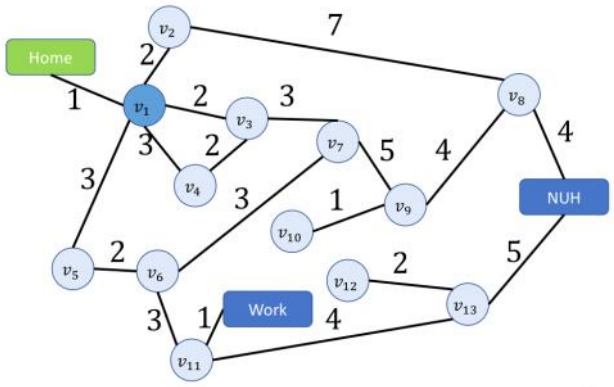
- *Graph Search:* A node that's been explored once will not be revisited.

17

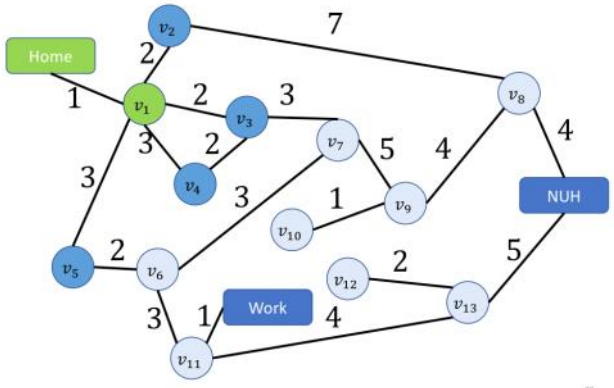


18

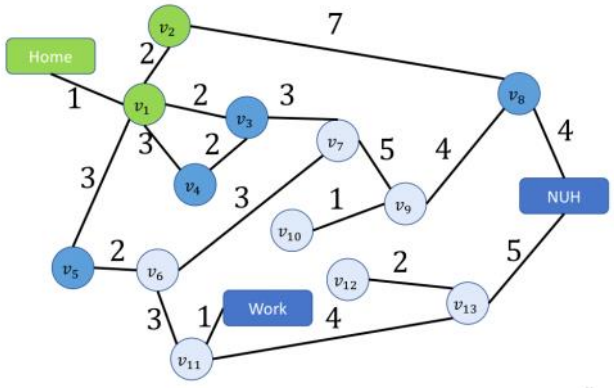
$\langle Home, cost = 0, parent = nil \rangle$



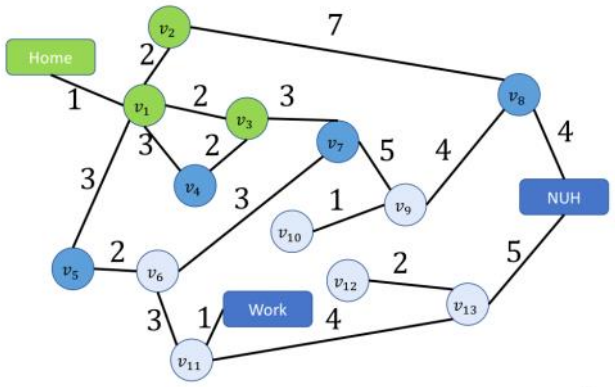
$\langle Home, g = 0, p = nil \rangle$
$\langle v_1, g = 1, p = Home \rangle$



$\langle Home, g = 0, p = nil \rangle$
$\langle v_1, g = 1, p = Home \rangle$
$\langle v_2, g = 3, p = v_1 \rangle$

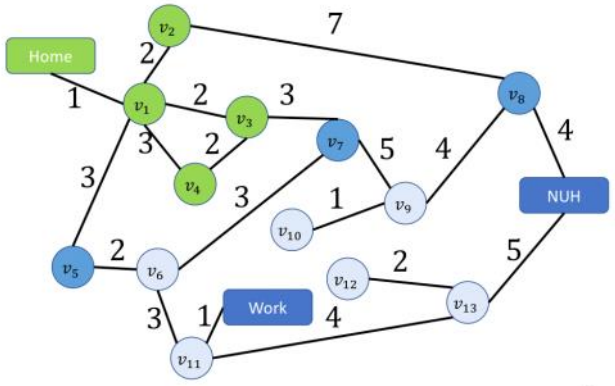


$\langle Home, g = 0, p = nil \rangle$
$\langle v_1, g = 1, p = Home \rangle$
$\langle v_2, g = 3, p = v_1 \rangle$
$\langle v_3, g = 3, p = v_1 \rangle$



22

$\langle Home, g = 0, p = nil \rangle$
$\langle v_1, g = 1, p = Home \rangle$
$\langle v_2, g = 3, p = v_1 \rangle$
$\langle v_3, g = 3, p = v_1 \rangle$
$\langle v_4, g = 4, p = v_1 \rangle$

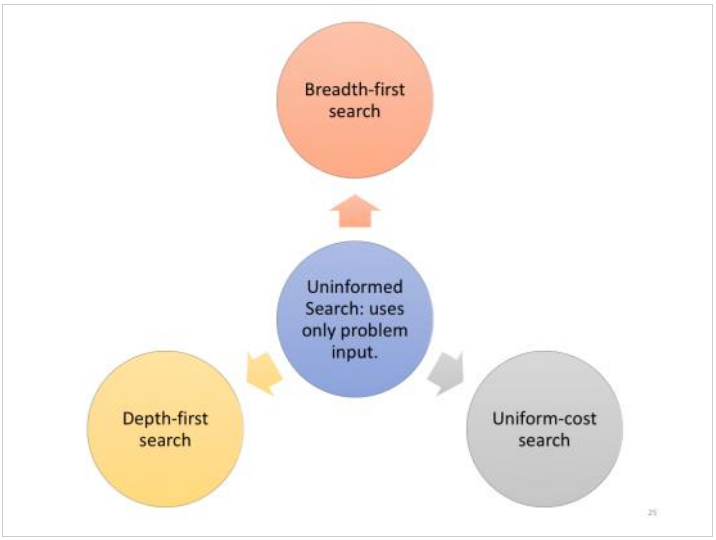


23

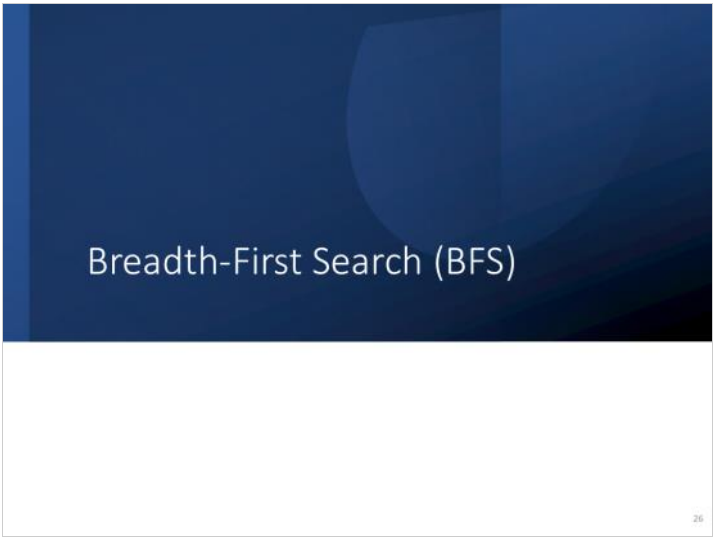
How do we measure?

- 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 ∞) ← branching factor
  - d: depth of shallowest **goal** node → smallest solution path weight
  - m: maximum depth of a node from start node → furthest node

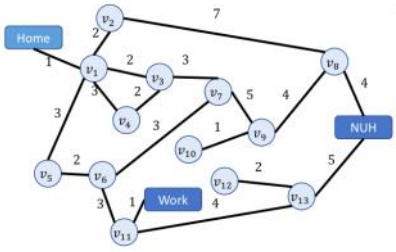
24



Breadth-First Search (BFS)

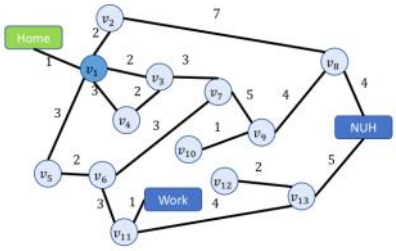


$[Home]$



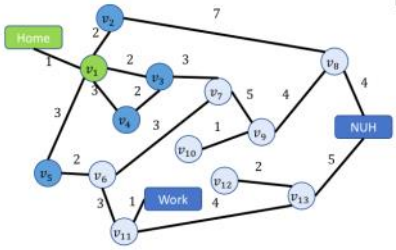
Home

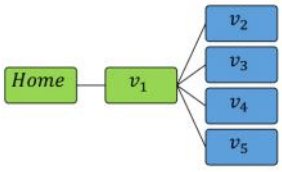
$[v_1]$



28

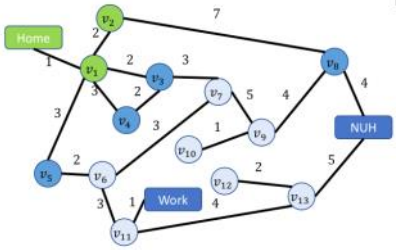
$[v_2, v_3, v_4, v_5]$

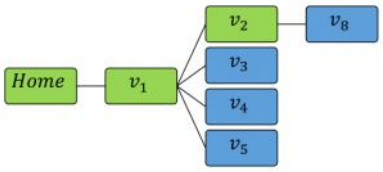




29

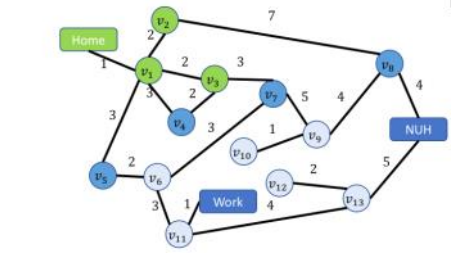
$[v_3, v_4, v_5, v_8]$

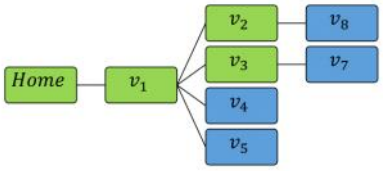




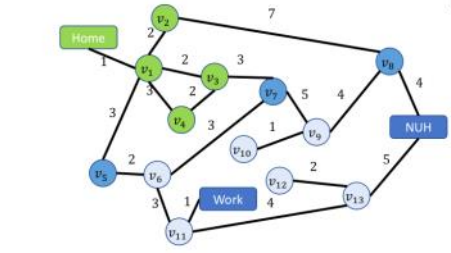
30

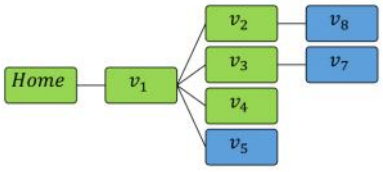
$[v_4, v_5, v_8, v_7]$



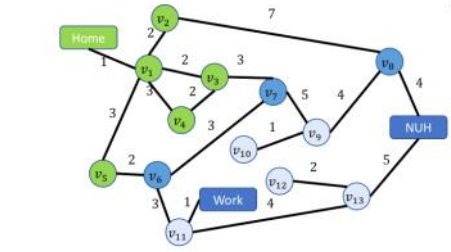


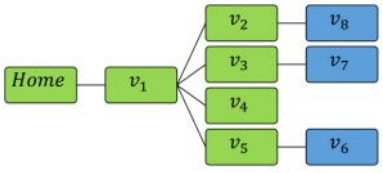
$[v_5, v_8, v_7]$

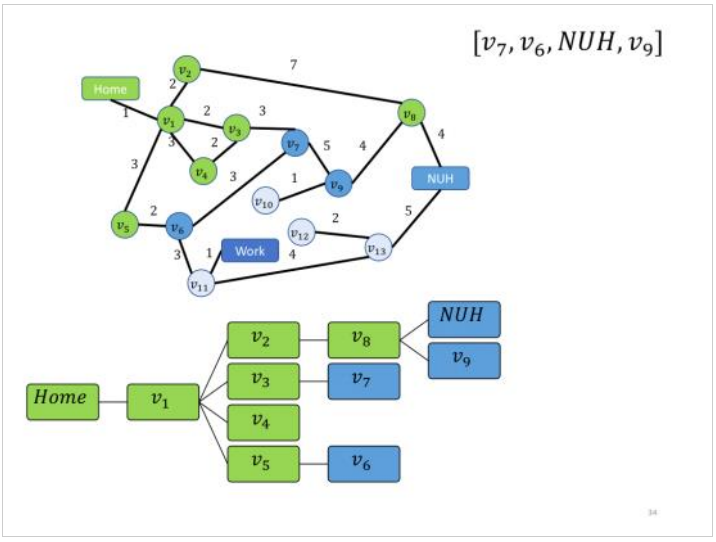


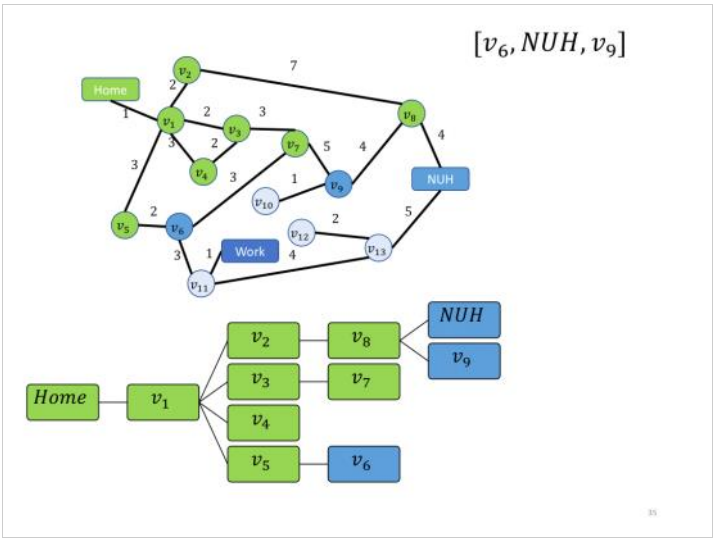


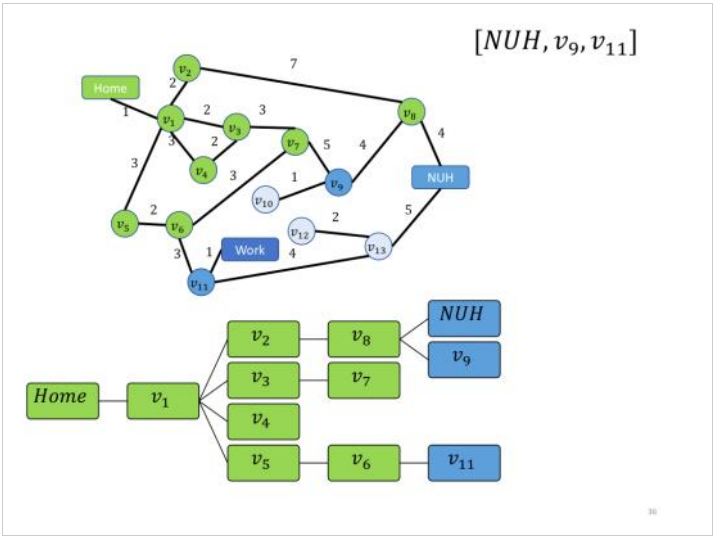
$[v_8, v_7, v_6]$

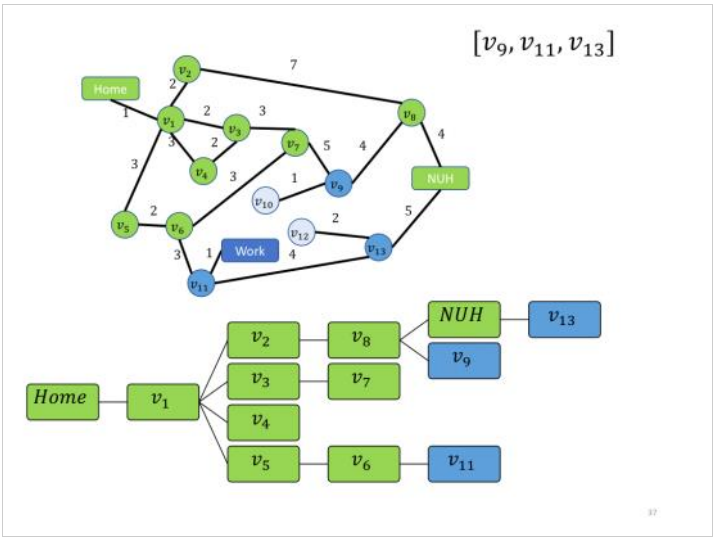


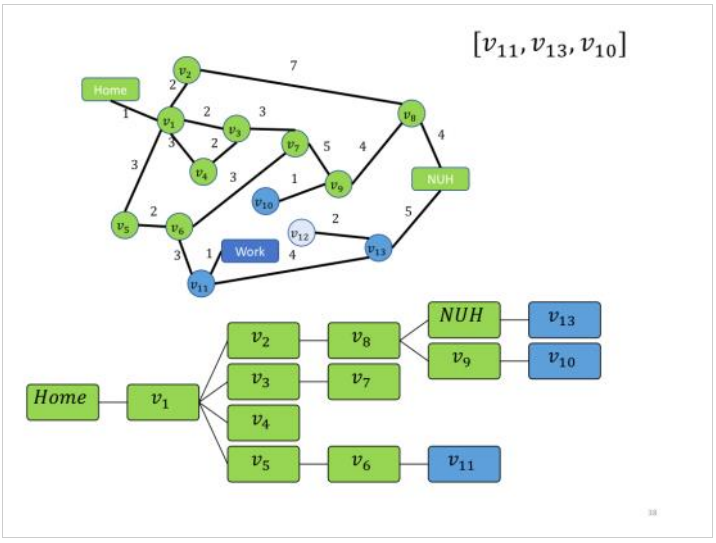


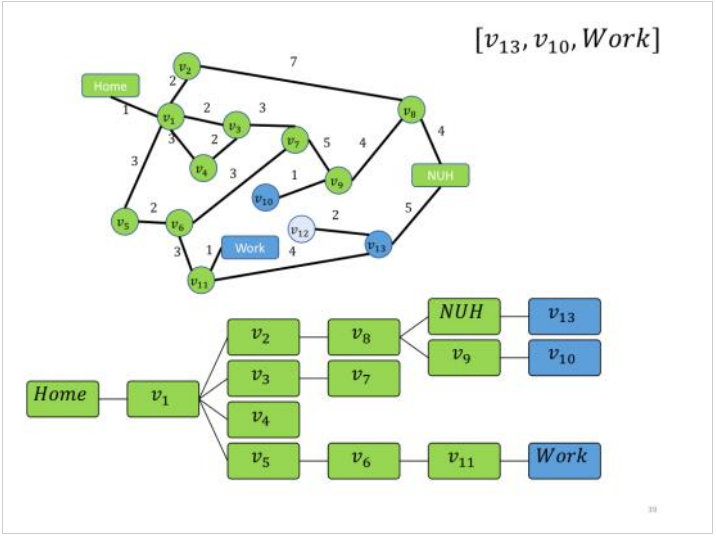

$[v_7, v_6, NUH, v_9]$
Home
Work
NUH
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
v_9
v_{10}
v_{11}
v_{12}
v_{13}
34

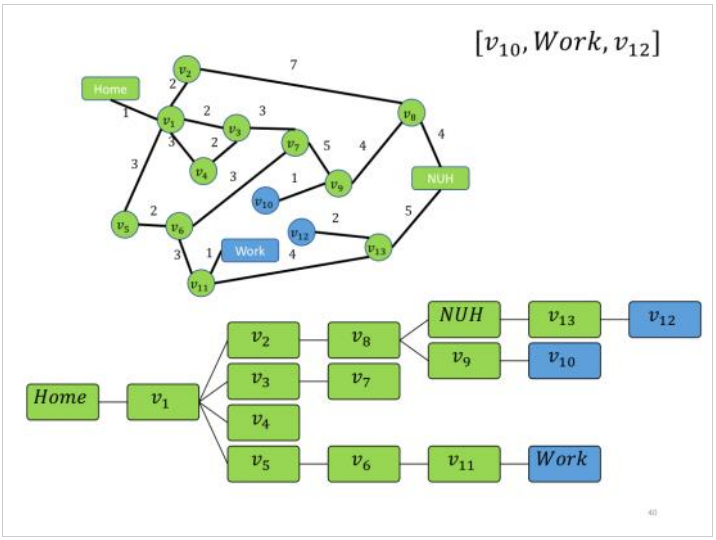

$[v_6, NUH, v_9]$
Home
Work
NUH
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
v_9
v_{10}
v_{11}
v_{12}
v_{13}
35

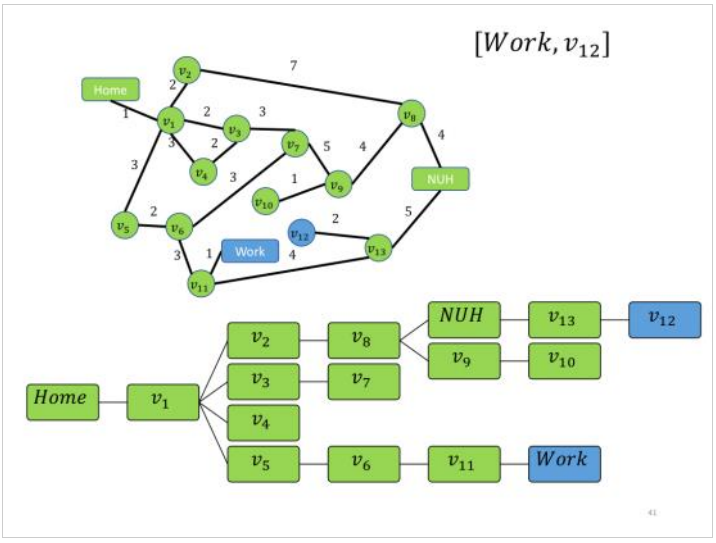

$[NUH, v_9, v_{11}]$
Home
Work
NUH
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
v_9
v_{10}
v_{11}
v_{12}
v_{13}
36

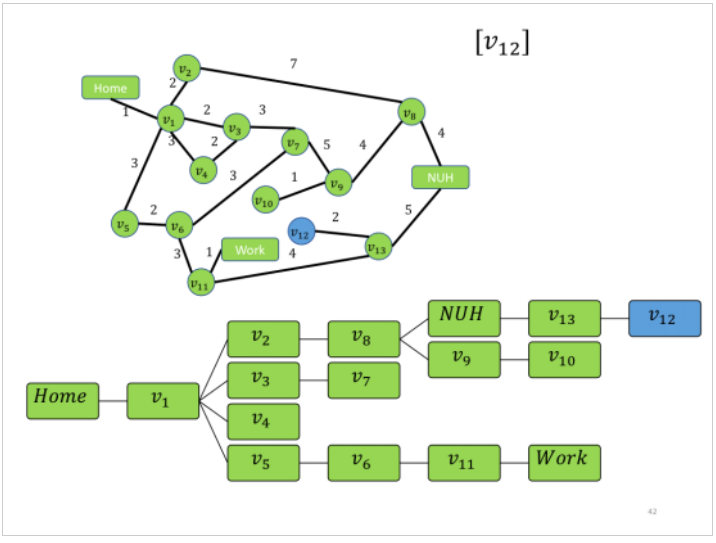

$[v_9, v_{11}, v_{13}]$
Home
NUH
Work
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}
Home
NUH
NUH
v_{13}
v_9
v_{11}


$[v_{11}, v_{13}, v_{10}]$
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}
Home
NUH
NUH
v_{13}
v_9
v_{10}
v_{11}


$[v_{13}, v_{10}, Work]$
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}
Home
NUH
NUH
v_{13}
v_9
v_{10}
v_{11}
Work


$[v_{10}, Work, v_{12}]$
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}
Home v_1
v_2 v_3 v_4 v_5
v_8 v_7 v_6
NUH v_9
v_{13} v_{10}
v_{12}
v_{11}
Work
40


$[Work, v_{12}]$
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}
Home v_1
v_2 v_3 v_4 v_5
v_8 v_7 v_6
NUH v_9
v_{13} v_{10}
v_{12}
v_{11}
Work
41


$[v_{12}]$
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}
Home v_1
v_2 v_3 v_4 v_5
v_8 v_7 v_6
NUH v_9
v_{13} v_{10}
v_{12}
v_{11}
Work
42

BFS

Algorithm 2 Breadth First Search: FindPathToGoal(u)

```
1: F(Frontier)← Queue(u)
2: E(Explored)← {u}
3: while F is not empty do
4:     u ← F.pop()
5:     for all children v of u do
6:         if GoalTest(v) then return path(v)
7:         else
8:             if v not in E then
9:                 E.add(v)
10:                F.push(v)
11: return Failure
```

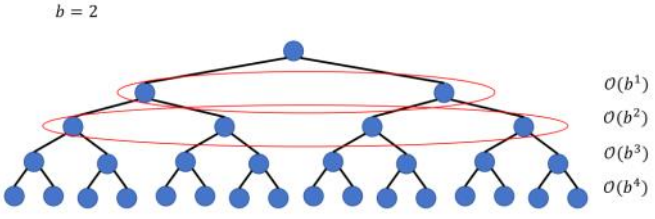
43

Properties of BFS

Property	
Complete?	Yes
Time	$O(b) + O(b^2) \ldots + O(b^d) = O(b^d)$ / $O(V+E)$
Space	$O(b^d)$
Optimal	

44

Properties of BFS

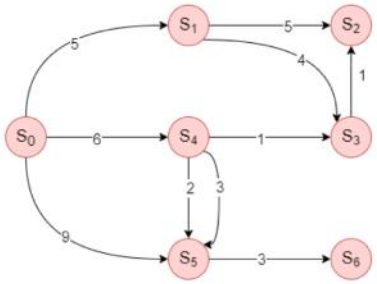


45

Properties of BFS

Property	
Complete?	Yes
Time	$\mathcal{O}(b) + \mathcal{O}(b^2) + \cdots + \mathcal{O}(b^d) = \mathcal{O}(b^d)$
Space	$\mathcal{O}(b^d)$
Optimal	

46



What would be the trace of BFS?

Start: S_0 Goal: S_6

S_0, S_5, S_6

47

BFS with Priority Queue

Algorithm 3 Breadth First Search with Priority Queue: FindPathToGoal(u)

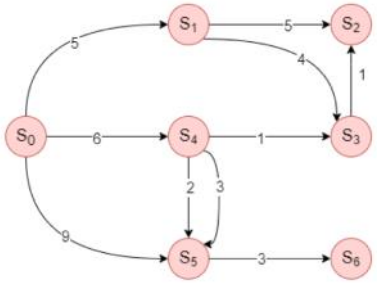
```
1:  F(Frontier)← PriorityQueue(u)                          ▷
2:  E(Explored)← {u}
3:  while F is not empty do
4:      u ← F.pop()
5:      for all children v of u do
6:          if GoalTest(v) then return path(v)
7:          else
8:              if v not in E then
9:                  E.add(v)
10:                 F.push(v)
11: return Failure
```

goal too soon

check after popped from queue

added too soon

48



What would be the trace of BFS with priority queue?

A. Start: S_0 Goal: S_6 S_0 S_5 S_6

B. Start: S_0 Goal: S_2

49

Diagnosis

- GoalTest is being invoked too early

50

UCS vs Dijstrak's

1 source, 1 destination, 1 shortest path

1 source, multiple destinations, shortest path to all.

Uniform Cost Search

Based on path cost

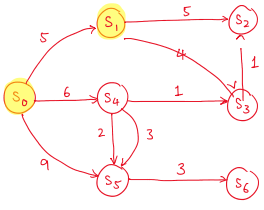
Algorithm 4 Uniform Cost Search(UCS): FindPathToGoal(u)

```
1:  F(Frontier)← PriorityQueue(u)
2:  E(Explored)← {u}
3:  ĝ[u] ← 0
4:  while F is not empty do
5:      u ← F.pop()
6:      if GoalTest(u) then
7:          return path(u)
8:      E.add(u)
9:      for all children v of u do
10:         if v not in E then
11:             if v in F then
12:                 ĝ[v] = min(ĝ[v], ĝ[u] + c(u, v))
13:             else
14:                 F.push(v)
15:                 ĝ[v] = ĝ[u] + c(u, v)
16: return Failure
```

↳ g(u) = path cost to get to u from source

51

1)

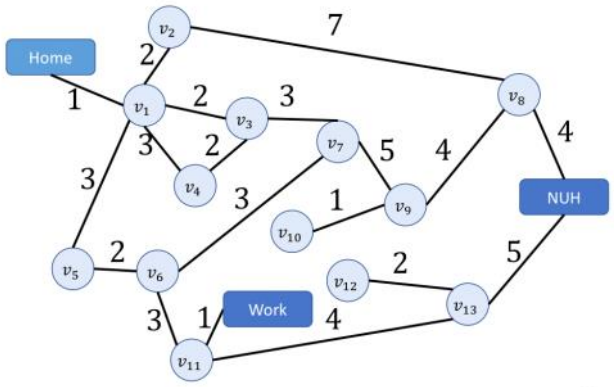


g | 0 | 5 | | | 6 | 9 | |

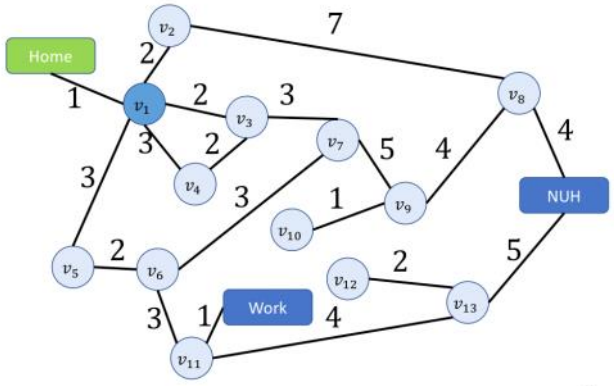
F : 1 4 5 3 2

↙ pop

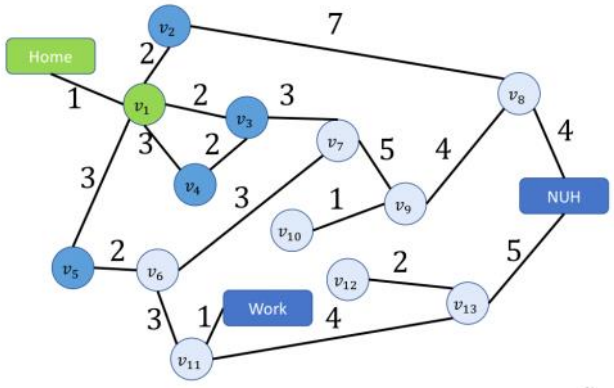
$[\langle Home, 0\rangle]$



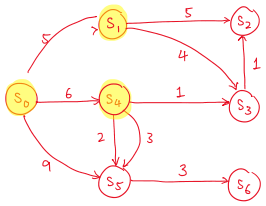
$[\langle v_1, 1\rangle]$

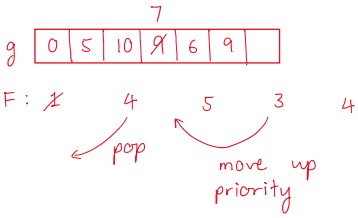


$[\langle v_2, 3\rangle, \langle v_3, 3\rangle, \langle v_4, 4\rangle, \langle v_5, 4\rangle]$

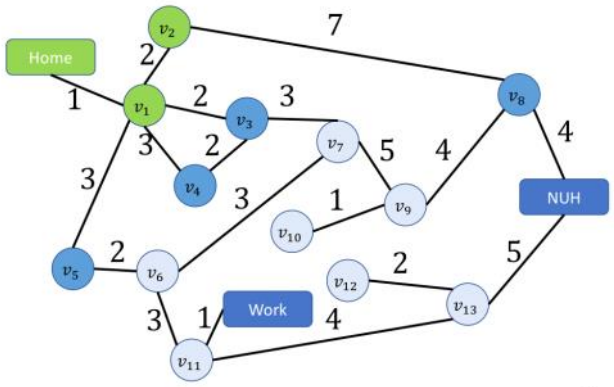


2)

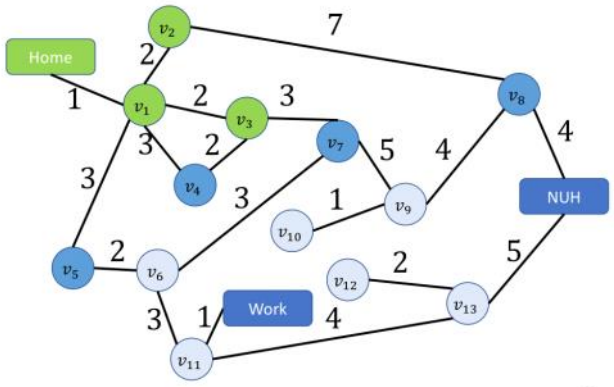




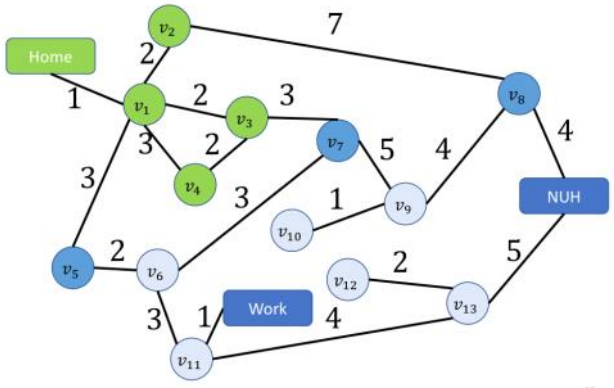
$[\langle v_3, 3\rangle, \langle v_4, 4\rangle, \langle v_5, 4\rangle, \langle v_8, 10\rangle]$



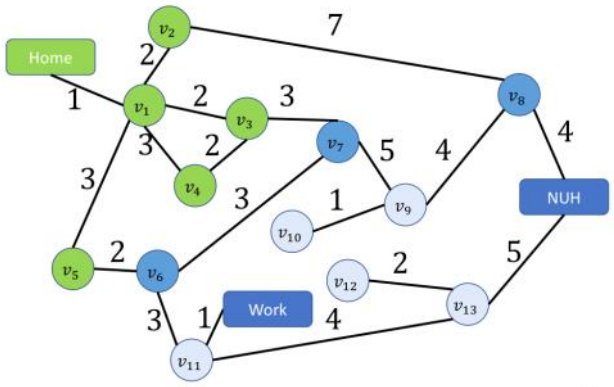
$[\langle v_4, 4\rangle, \langle v_5, 4\rangle, \langle v_7, 6\rangle, \langle v_8, 10\rangle]$



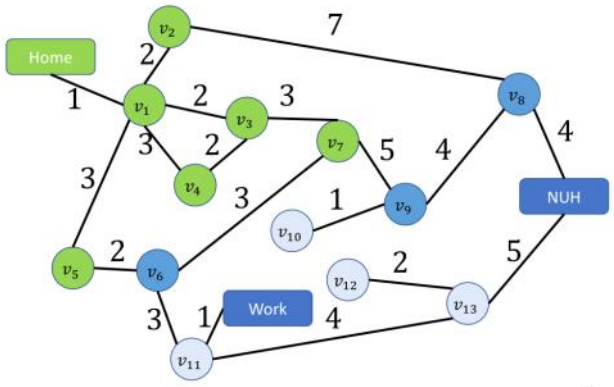
$[\langle v_5, 4\rangle, \langle v_7, 6\rangle, \langle v_8, 10\rangle]$



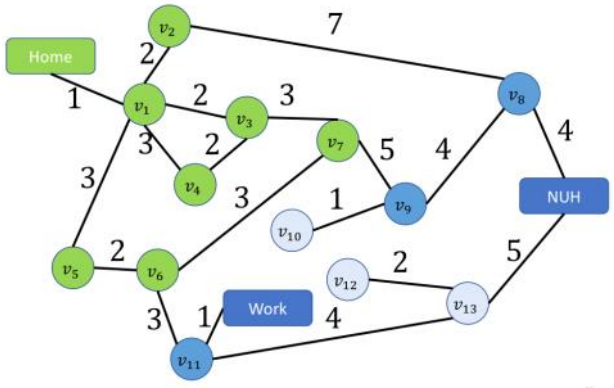
$[\langle v_7, 6\rangle, \langle v_6, 6\rangle, \langle v_8, 10\rangle]$



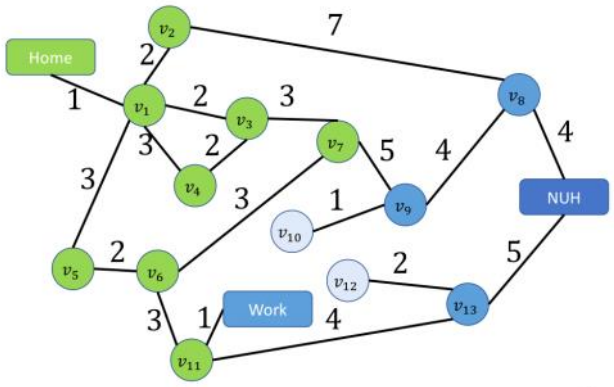
$[\langle v_6, 6\rangle, \langle v_8, 10\rangle, \langle v_9, 11\rangle]$



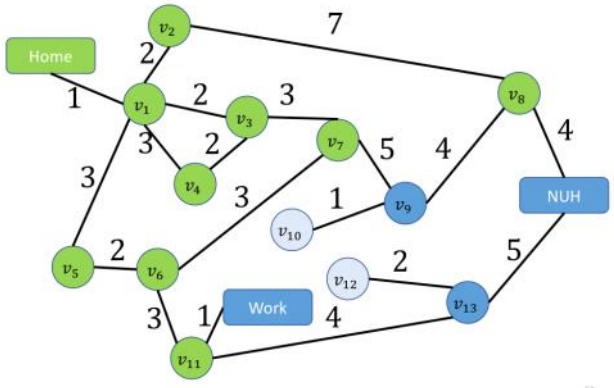
$[\langle v_{11}, 9\rangle, \langle v_8, 10\rangle, \langle v_9, 11\rangle]$



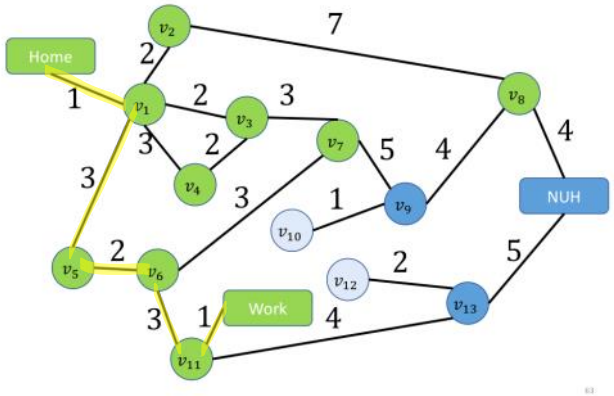
$[\langle v_8, 10\rangle, \langle Work, 10\rangle, \langle v_9, 11\rangle, \langle v_{13}, 13\rangle]$



$[\langle Work, 10\rangle, \langle v_9, 11\rangle, \langle v_{13}, 13\rangle, \langle NUH, 14\rangle]$

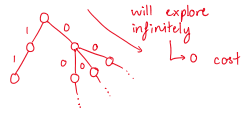


$[\langle v_9, 11\rangle, \langle v_{13}, 13\rangle, \langle NUH, 14\rangle]$



Uniform Cost Search

Property	
Complete?	Yes (if all step costs are $\geq \varepsilon$)
Optimal	
Time	
Space	



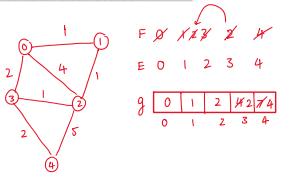
0 / 1

64

Uniform Cost Search

Algorithm 4 Uniform Cost Search(UCS): FindPathToGoal(u)

```
1:  F(Frontier)← PriorityQueue(u)
2:  E(Explored)← {u}
3:  ĝ[u] ← 0
4:  while F is not empty do
5:      u ← F.pop()
6:      if GoalTest(u) then
7:          return path(u)
8:      E.add(u)
9:      for all children v of u do
10:         if v not in E then
11:             if v in F then
12:                 ĝ[v] = min(ĝ[v], ĝ[u] + c(u, v))
13:             else
14:                 F.push(v)
15:                 ĝ[v] = ĝ[u] + c(u, v)
16: return Failure
```



65

Proof of Optimality

Theorem: When we pop u from F, we have found optimal path to u from the start node (say, S_0)

Notations:

- $g(u)$: Minimum distance from S_0 to u
- $\hat{g}_{pop}(u)$: The value of $\hat{g}$ when u is popped

Formally, we want to prove $\hat{g}_{pop}(u) = g(u)$

66

Uniform Cost Search

Property	
Complete?	Yes (if all step costs are $\geq \varepsilon$)
Optimal	Yes
Time	
Space	

☆ NOT the same as Dijstra
↳ Dijstra is too ALL nodes from root
↳ Uniform cost is only start to goal

Proof by induction:

① Assume optimal path to u:
$S_0, S_1, \ldots S_k, u$

② Base case:
$g_{pop}(S_0) = g(S_0) = 0$

③ Assume for S_0 to S_k
$g_{pop}(S_k) = g(S_k)$

④ $g(S_1) \leq g(S_2) \leq \ldots \leq g(u)$ ☆ $\varepsilon \geq 0$

⑤ $g_{pop}(u) \geq g(u)$
↳ cannot be less

After popping S_k,

$g(u) = \min(\hat{g}(u), g_{pop}(S_k) + c(S_i, u))$
$= g(S_k) + c(S_i, u)$
$= g_{pop}(u)$

} no other path will produce a better $g(u)$
↳ u eventually popped

Uniform Cost Search

Property	
Complete?	Yes (if all step costs are $\geq \varepsilon$)
Optimal	Yes
Time	
Space	

67

Time and Space Complexity

- 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.

68

Uniform Cost Search

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

no. of nodes

69

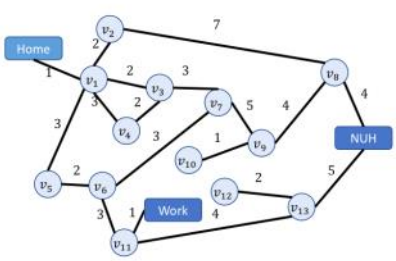
= g pop (u)

path will produce a better g(u) → u eventually popped

Depth First Search

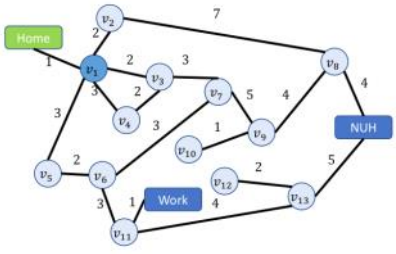
- Idea: Expand deepest unexpanded node
- Implementation: Frontier = LIFO stack, i.e., insert successors at the front

70



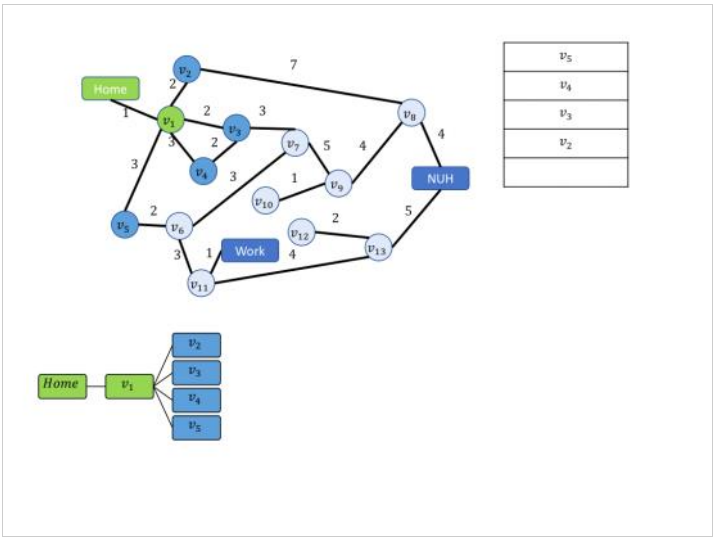
Home

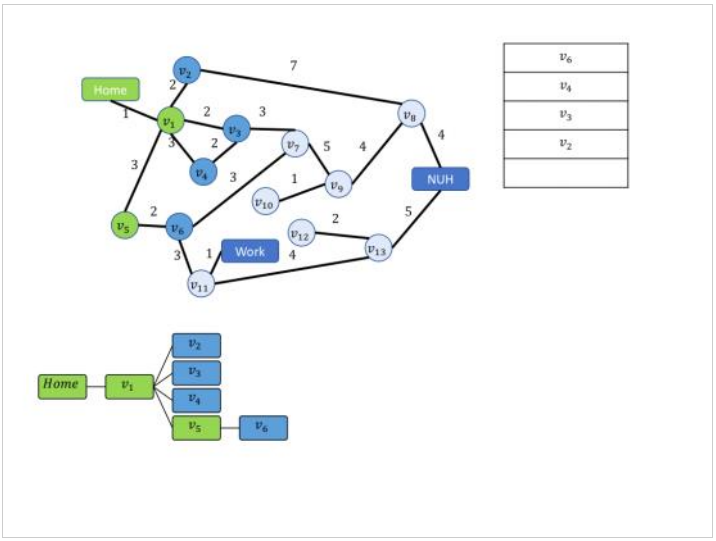
Home

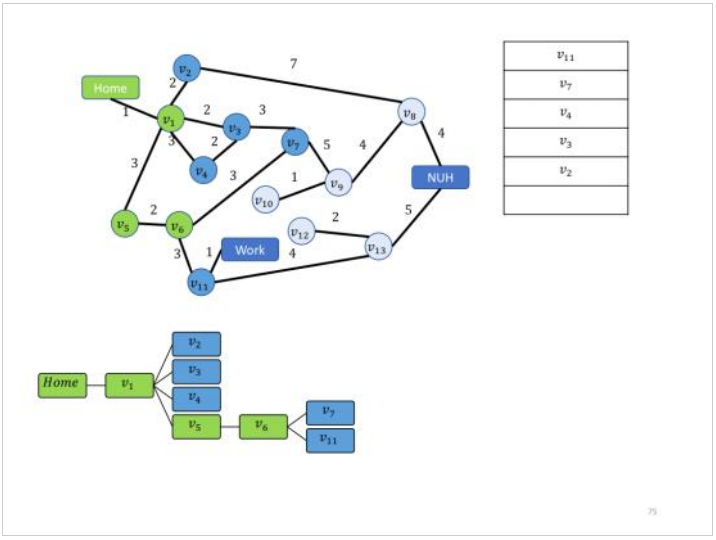


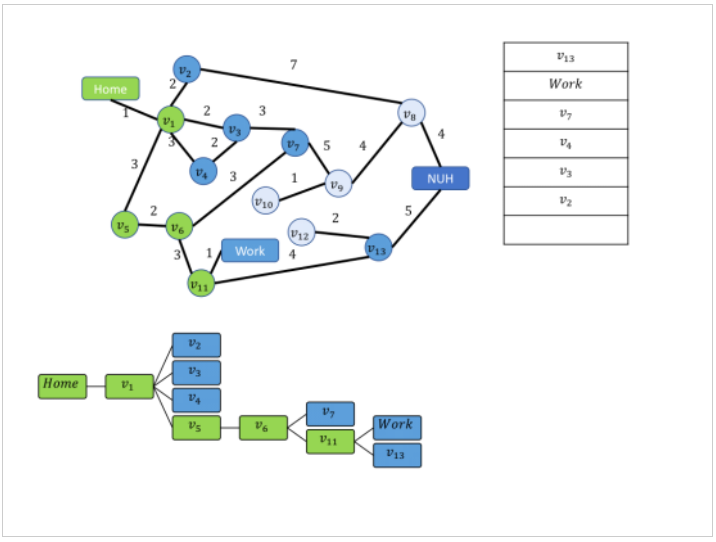
v_1

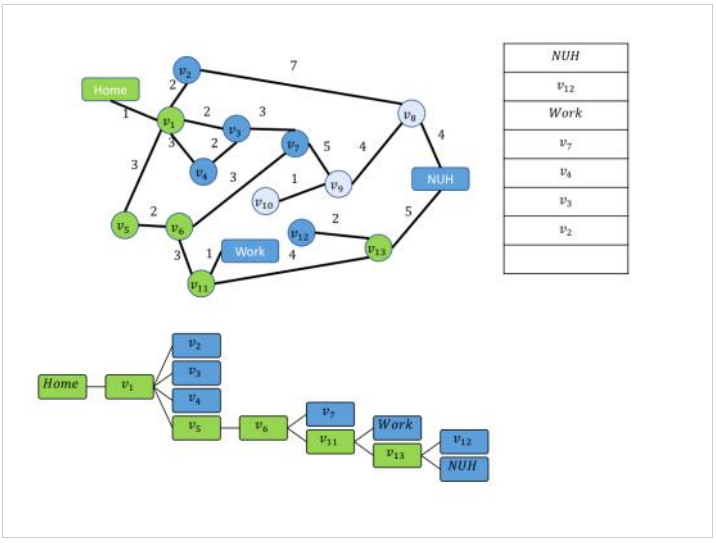
Home — v_1

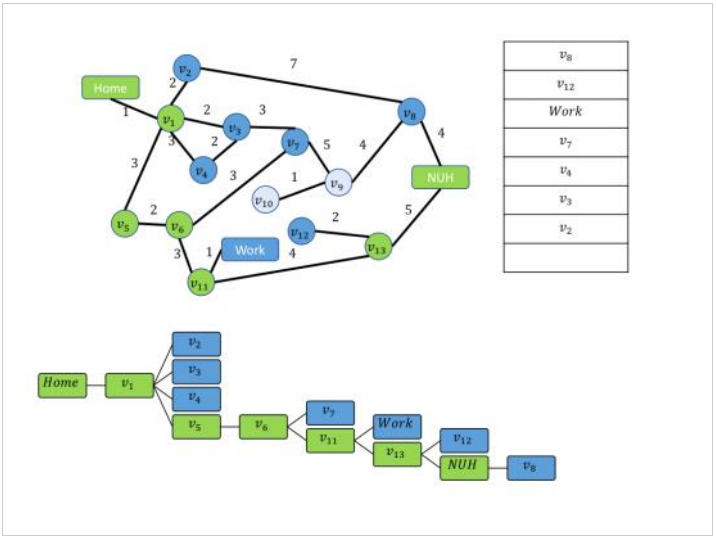

Home
Work
NUH
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}
v_5
v_4
v_3
v_2
Home v_1 v_2 v_3 v_4 v_5

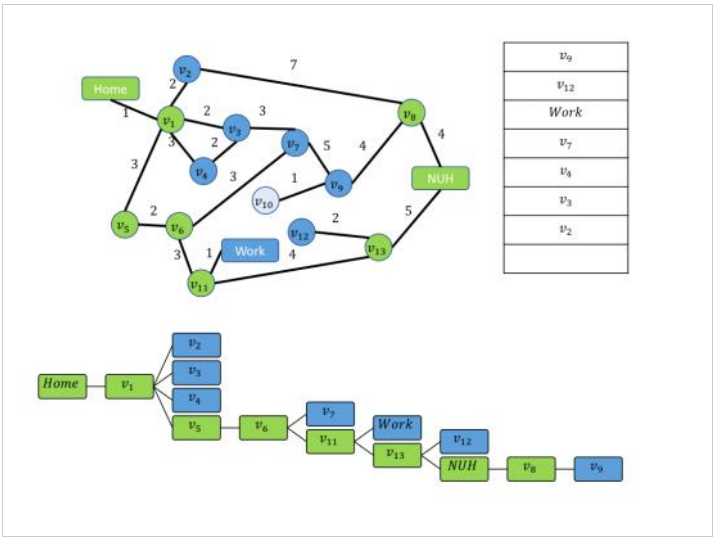

Home
Work
NUH
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}
v_6
v_4
v_3
v_2
Home v_1 v_2 v_3 v_4 v_5 v_6

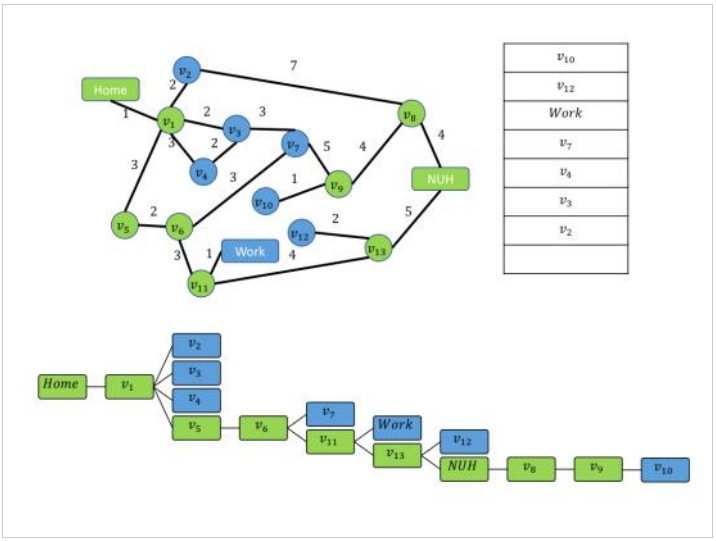

Home
Work
NUH
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}
v_{11}
v_7
v_4
v_3
v_2
Home v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_{11}

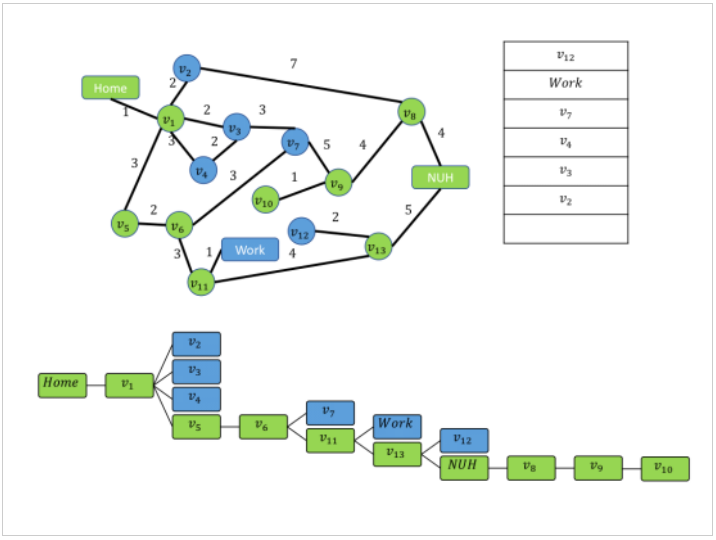

Home
Work
NUH
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}
v_{13}
Work
v_7
v_4
v_3
v_2

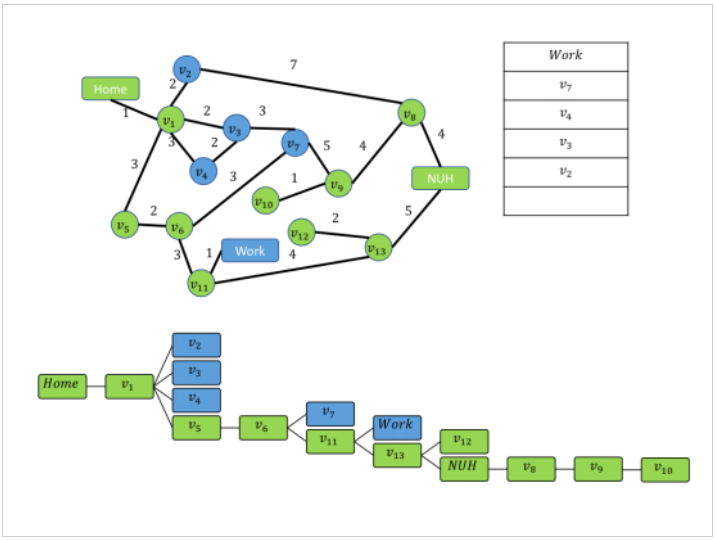

Home
Work
NUH
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}
NUH
v_{12}
Work
v_7
v_4
v_3
v_2

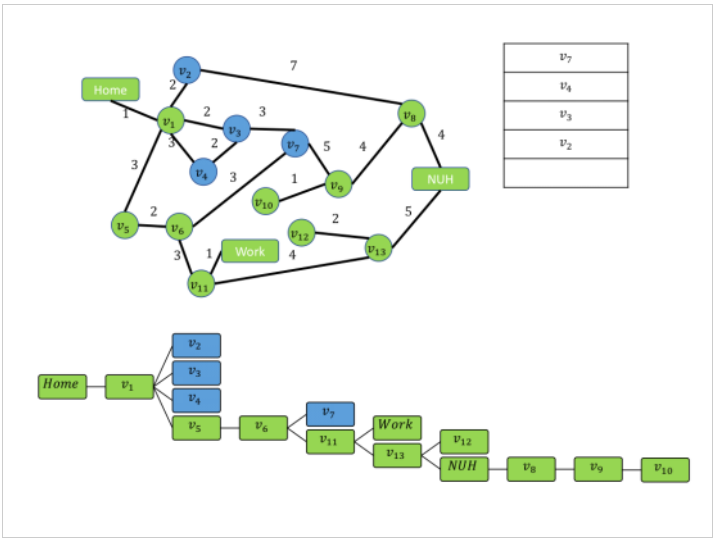

Home
Work
NUH
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13}
v_8
v_{12}
Work
v_7
v_4
v_3
v_2


Home
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
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
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





Depth-First Search

Algorithm 5 Depth First Search(DFS): FindPathToGoal(u)

```
1: F(Frontier)← Stack(u)
2: E(Explored)← {}
3: while F is not empty do
4:     u ← F.peek()
5:     if GoalTest(u) then
6:         return path(u)
7:     if HasUnvisitedChildren(u) then
8:         for all children v of u do
9:             if v not in E then
10:                F.push(v)
11:                E.add(v)
12:    else
13:        F.pop()
14:        E.add(u)
15: return Failure
```

84

Depth-First Search

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

When checking a node v, we push at most b descendants to stack.

We do so at most m times $\Rightarrow \mathcal{O}(bm)$ space.

85

Summary

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

1. if b is finite.
2. f b is finite and step cost $\geq \varepsilon$

86