Informed Search

Wednesday, 1 September 2021 7:08 AM

Logistics

- Tutorial Submission: "Your solutions for the above questions may be handwritten or typewritten, but handwritten solutions must be legible for marks to be awarded. If you are submitting handwritten solutions, please append the question paper in front of your solutions."
- Mass Consultations: Every Monday, 3-4 PM

2

Recap

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

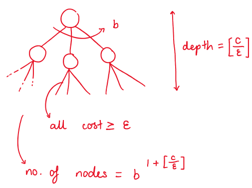
branching factor

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

Can we minimize space?

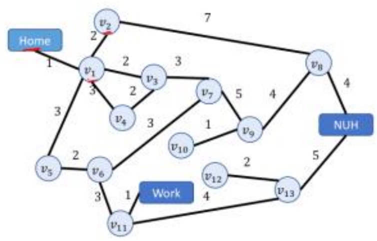
BFS - based
need to store
visited

3



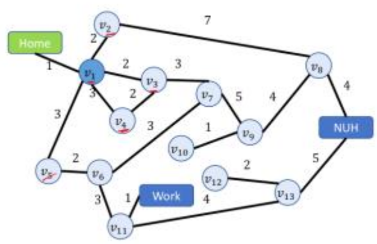
Depth First Search

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

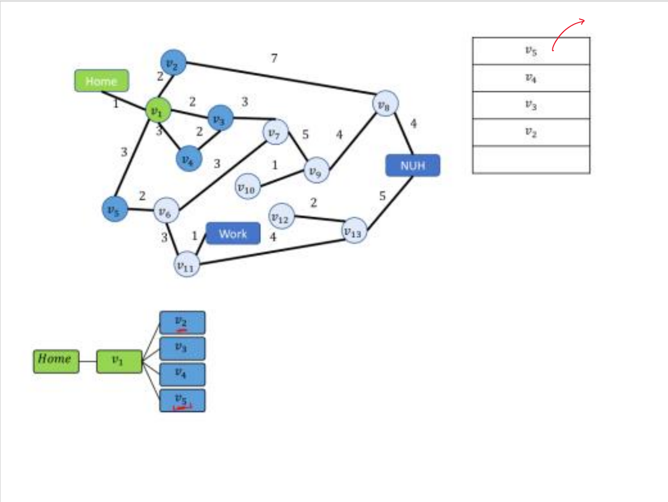


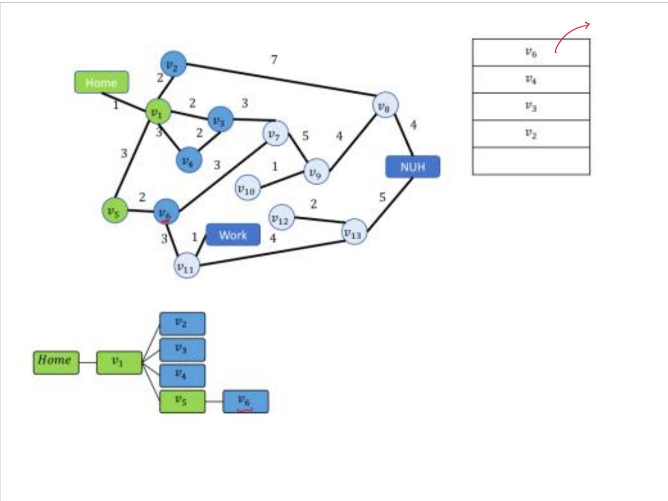
Home

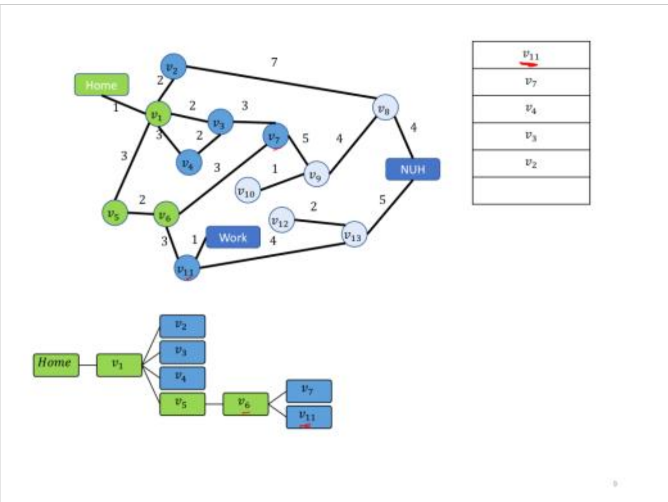
Home

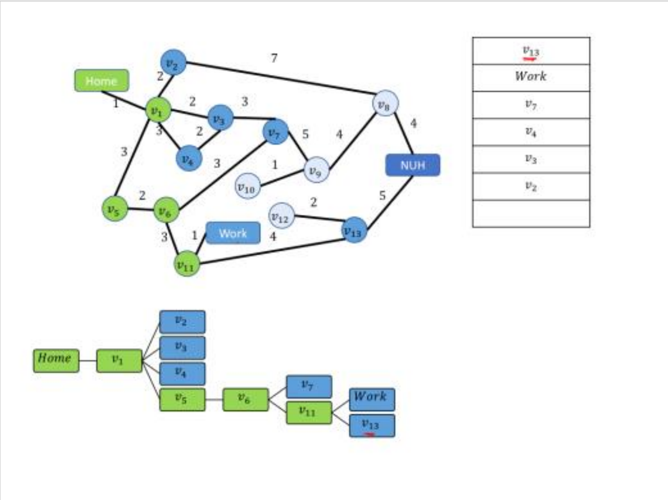


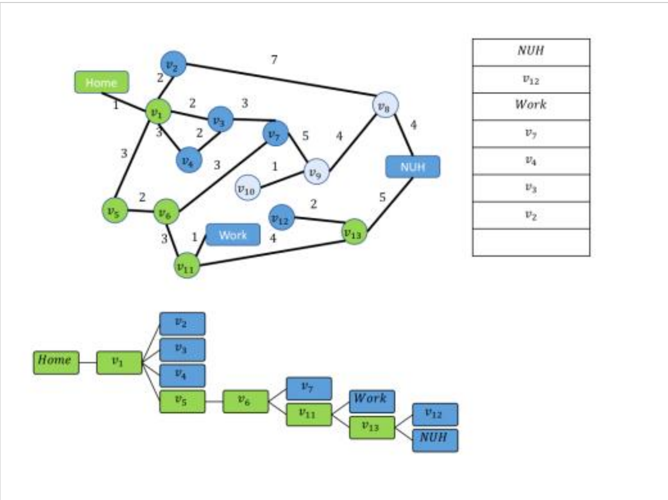
v_1

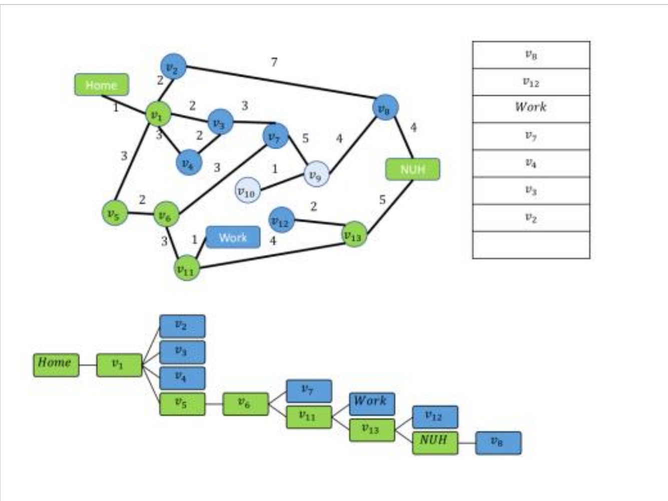

Home
NUH
Work
v_5
v_4
v_3
v_2
Home
v_1
v_2
v_3
v_4
v_5

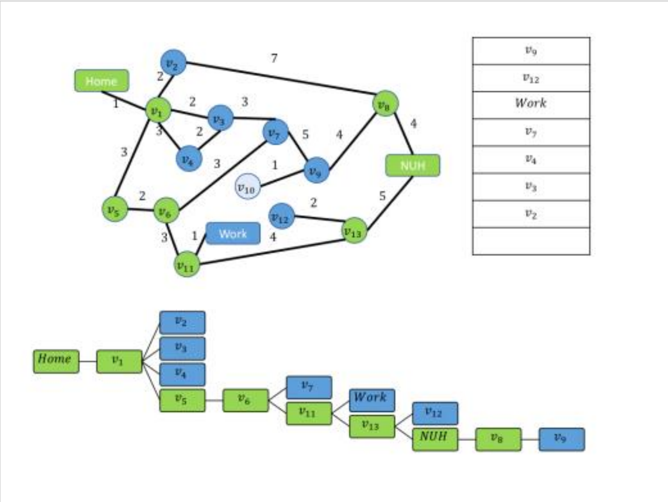

Home
NUH
Work
v_6
v_4
v_3
v_2
Home
v_1
v_2
v_3
v_4
v_5
v_6

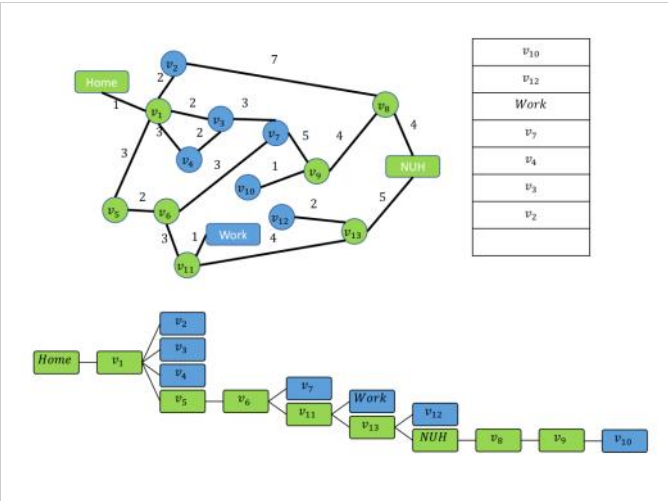

Home
NUH
Work
v_{11}
v_7
v_4
v_3
v_2
Home
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_{11}

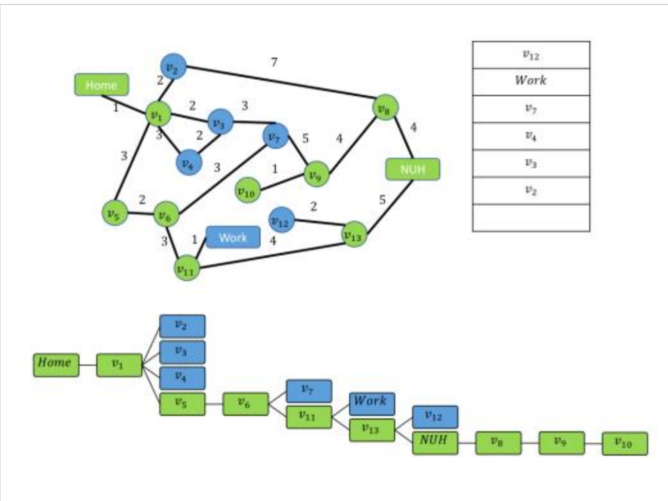

Home
Work
NUH
v_{13}
Work
v_7
v_4
v_3
v_2

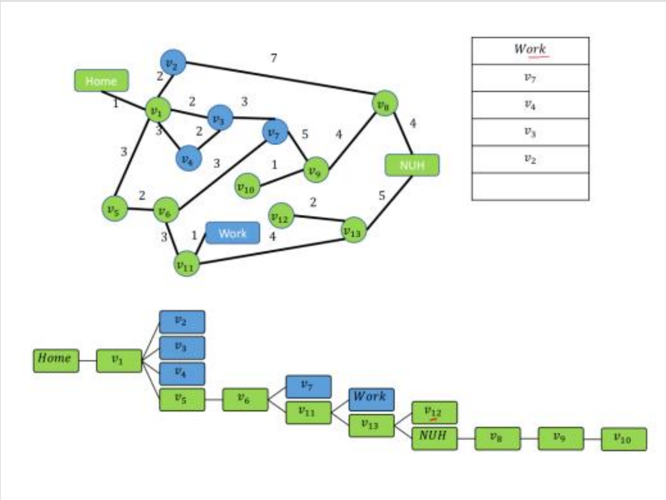

Home
Work
NUH
NUH
v_{12}
Work
v_7
v_4
v_3
v_2

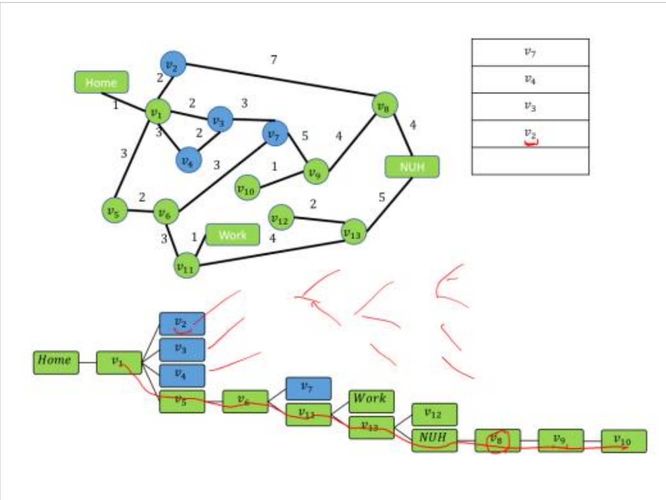

Home
Work
NUH
v_8
v_{12}
Work
v_7
v_4
v_3
v_2


Home
Work
NUH
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_{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_{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}





Depth-First Search – Graph-Search

Algorithm 4 Depth First Search(DFS): FindPathToGoal(u)

F(Frontier)$\leftarrow$ Stack(u)
E(Explored)$\leftarrow$ {} → how do we improve SPACE complexity?
while F is not empty **do**
  $u \leftarrow F$.pop()
  if GoalTest(u) **then**
    return path(u)
  E.add(u)
  if HasUnvisitedChildren(u) **then**
    for all children v of u **do**
      if v not in E **then**
        F.push(v)
return *Failure*

Depth-First Search – Tree Search

Algorithm 4 Depth First Search(DFS): FindPathToGoal(u)

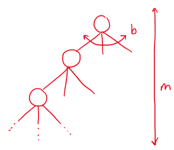
F(Frontier)$\leftarrow$ Stack(u)
~~E(Explored)$\leftarrow \{\}$~~
while F is not empty **do**
  $u \leftarrow F$.pop()
  if GoalTest(u) **then**
    return path(u)
  ~~E.add(u)~~
  if HasUnvisitedChildren(u) **then**
    for all children v of u **do**
      ~~**if** v not in E **then**~~
        F.push(v)
return *Failure*

☆ No need to maintain explored list
↳ only check stack
↳ might get stuck in an infinite loop

Depth-First Search

Property	
Complete?	No on infinite depth graphs
Optimal	No
Time	$O(b^{m+1})$ m = maximum depth
Space	$O(bm)$ → Tree Search Variant Not Explored Maintaining.

stack



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

$O(b^{m+1})$ for graph search

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

Summary

Graph Search / Tree Search

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

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

Can we do better?

Yes!

Exploit problem-specific knowledge

Obtain heuristics to guide search

22

Can we do better?

- UCS: Expand based on $\hat{g}(u)$
  - Remembering the past
- What if we do guess something about "future"?
- Evaluation function: $h(u)$
  - An estimate of distance of goal node from node u
- A natural question: how good should the estimate be?

23

A* Search

Algorithm 6 A* Algorithm: FindPathToGoal(u)

```
F(Frontier) ← PriorityQueue(u)          ▷ it should be implement with f̂ minimum
E(Explored) ← {u}
ĝ[u] ← 0
while F is not empty do
    u ← F.pop()
    if GoalTest(u) then
        return path(u)
    E.add(u)
    for all children v of u do
        if v not in E then
            if v in F then
                ĝ[v] = min(ĝ[v], ĝ[u] + c(u, v))
                f̂[v] = h[v] + ĝ[v]
            else
                F.push(v)
                ĝ[v] = ĝ[u] + c(u, v)
                f̂[v] = h[v] + ĝ[v]
return Failure
```

using $\hat{f}(u)$

$\rightarrow$ ☆optimal

$\hat{f}(u)$ must be updated when $\hat{g}(u)$ is updated

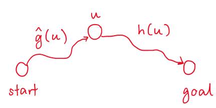
$g(u) \rightarrow$ min path cost

$\hat{g}(u) \rightarrow$ path cost so far

$h(u) \rightarrow$ estimated cost from u to goal (heuristic function)

$\hat{f}(u) \rightarrow$ evaluation function $= \hat{g}(u) + h(u)$

$f(u) \rightarrow$ optimal cost $= g(u) + h(u)$



$\hat{f}(u) = \hat{g}(u) + h(u)$

A* Search

What property of h would ensure that A* is optimal?

Thm A* with graph-search and consistent h is optimal.

25

☆ What property of $h(u)$, and hence, $\hat{f}(u)$ would make the algorithm optimal?

Refer to UCS $\rightarrow$ nodes along optimal path get popped in the right order

$\hookrightarrow g_{pop}(s_0) \le g_{pop}(s_1) \le \ldots \le g_{pop}(u)$

$\hookrightarrow \hat{g}_{pop}(s_i) = g(s_i)$

$g(s_0) \le g(s_1) \le \ldots \le g(u)$

A* uses $\hat{f}(u) \rightarrow$ prove $\hat{f}_{pop}(s_i) = f(s_i)$ Q2

Q1: $\hat{f}_{pop}(s_0) \le \hat{f}_{pop}(s_1) \le \ldots \hat{f}_{pop}(u)$

$\hookrightarrow$ bc of priority queue

$f(s_i) \le f(s_{i+1}) \longrightarrow$ if true, proves by Induction Q3

$\Leftrightarrow g(s_i) + h(s_i) \le g(s_{i+1}) + h(s_{i+1})$

$\Leftrightarrow h(s_i) \le \underbrace{g(s_{i+1}) - g(s_i)}_{c(s_i,\, s_{i+1})} + h(s_{i+1})$ Q4

$\hookrightarrow$ ☆ Consistency

logically equivalent

Q4:

$\Leftrightarrow h(s_i) \leq \underbrace{g(s_{i+1}) - g(s_i)}_{c(s_i,\, s_{i+1})} + h(s_{i+1})$ Q4 ← logically equivalent

↳ ☆ Consistency

Q4:

$h(goal) \geq 0$

$h(s_0) \leq c(s_0, s_1) + h(s_1)$

$\leq c(s_0, s_1) + c(s_1, s_2) + \ldots + c(s_k, u) + \cancel{h(u)}^{0}$

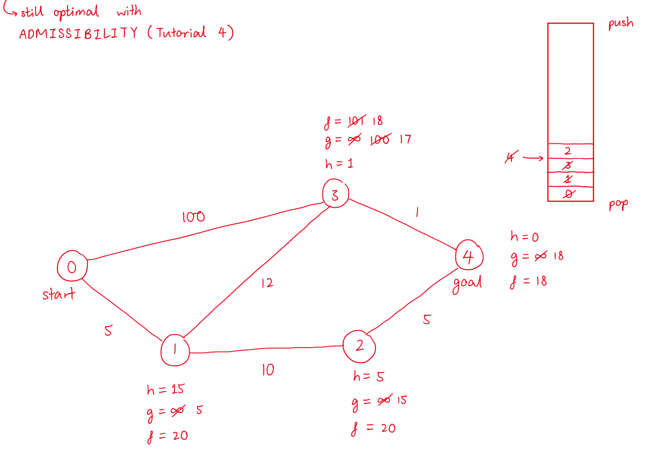
$\leq$ path cost s_0 to u

↳ ☆ Admissibility

☆ Consistency & $h(goal) \geq 0 \Rightarrow$ Admissibility → $A \not\Rightarrow C$

↳ A* OPTIMAL UNDER THESE SPECIFIC CONDITIONS

Tree search varient
no record of explored
↳ still optimal with ADMISSIBILITY (Tutorial 4)



25

Heuristic Functions

Consistent

$\forall n, n'$ $\boxed{h(n) \leq h(n') + c(n, n')}$ → Triangle Inequality

where, n' is successor of n

Admissible

$\forall n$ $h(n) \leq h^*(n)$

where $h^*(n) = g(goal) - g(n)$:= True cost to reach the goal.

↳ path cost from n to goal

26

The Power of Admissibility

Observation:

if h(goal)≥0, then consistency => admissibility

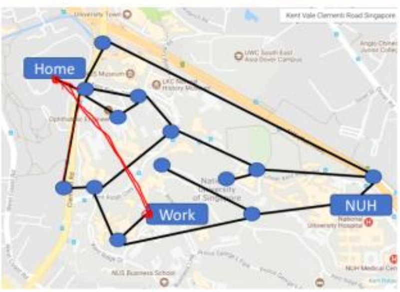
Theorem:

A* search with Tree-Search is optimal for admissible heuristics.

27

In the Search of Heuristic Functions

↳ fewer restrictions
↳ can cross over buildings



Euclidian Distance

In the Search of Heuristic Functions

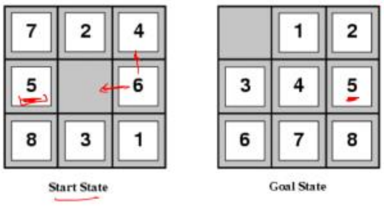
- A problem with fewer restrictions on the actions is called a relaxed problem

- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem

29

In the Search of Heuristic Functions

A tile is misplaced if it is not at the position it should be in the goal state.



30

Deriving Admissible Heuristics

- Rules of 8-puzzle:

  A tile can move from square A to square B if A is horizontally or vertically adjacent to B ~~and B is blank~~

  2 conditions: adjacent, B is blank

- We can generate two relaxed problems
  1. A tile can move from square A to square B if A is adjacent to B
     - $h_1(n)$ = number of misplaced tiles

     ↳ only 1 condition
  2. A tile can move from square A to square B
     - $h_2(n)$ = total Manhattan distance (i.e., no. of squares from desired location of each tile)

     ↳ no conditions

31

In the Search of Heuristic Functions

- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem

never OVERESTIMATES

- What about consistent heuristics?
  - We don't know of a good recipe
  - Another reason we may prefer tree-search algorithms

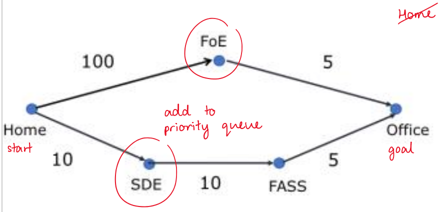
32

Greedy Best First Search

- What if we have priority queue based on h instead of $\hat{f}$
- Would this be optimal ? No. → not consistent

~~Home~~, ~~FoE~~, SDE

~~office~~ ← popped early

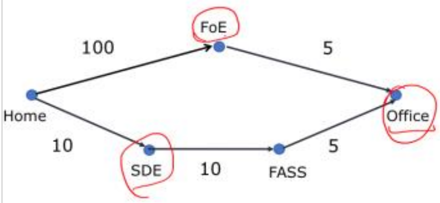


h (FoE) = 5
h(SDE) = 15
h(FASS) = 5
h (Office) = 0

33

Greedy Best First Search

- What if we have priority queue based on h instead of $\hat{f}$
- Would this be optimal ?



h (FoE) = 5
h(SDE) = 15
h(FASS) = 5
$\boxed{\text{h (Office) = 0}}$

34

In Summary

- When Space matters, use DFS (Tree search varient)

- Be Informed when you can:
  - Graph-Search: Consistent Heuristics
  - Tree-Search: Admissible Heuristics

Past & Future

$g(u)$ $f(u)$

- Trust Past than Future
  - Only Past: UCS was optimal
  - Only Future: Greedy Best First Search is not optimal