# CS 5100

## Search Algorithms

Assessing an algorithm
- Completeness
- Optimality
- time complexity
- Space complexity

Breadth First Search (FIFO)
- yes complete
- not optimal
- $O(b^d)$ time
- $O(b^d)$ space

b: # nodes branched out from one node

d: depth of the shallowest good node

Depth First Search (LIFO)
- not complete (infinite # of states)
- not optimal
- $O(b^d)$ time
- we could write DFS in a way to get $O(m)$

m: max depth of the tree

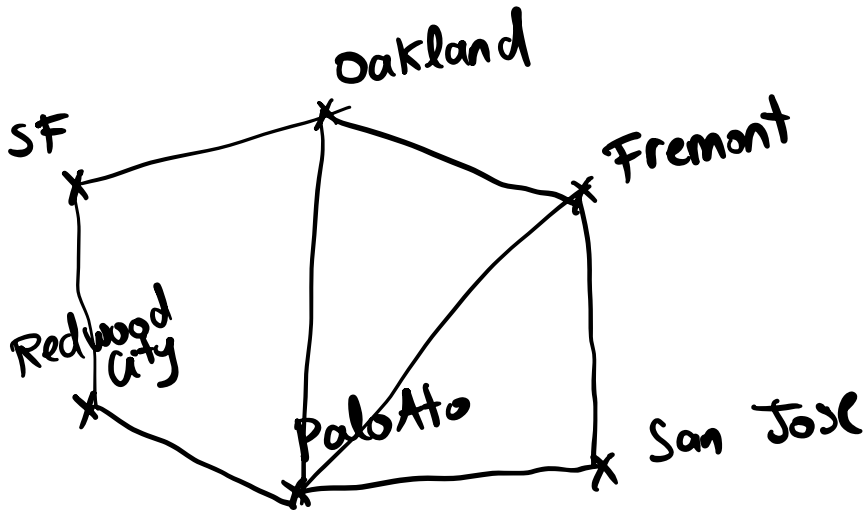
we could limit the depth of the tree we reach. What depth limit $l$ to choose?

If we choose $l < d \Rightarrow$ not complete

If we choose $l \geq d \Rightarrow$ complete

Depth limited search

resolve completeness but not optimality




Knowledge of the problem can help

6 cities

$\ell = 5$

(could also choose a smaller value $\ell = 3$ for instance)

the smallest value we can choose is known as the diameter

What if I don't have any useful information to help me select $\ell$?

Iterative deepening search

We can try different values of $\ell$, we iterate from 0, then 1 then 2 until a goal node is found

$\Downarrow$

$\ell = d$ ( depth of the shallowest goal node)

How much time complexity will it cost me?

$l=0$

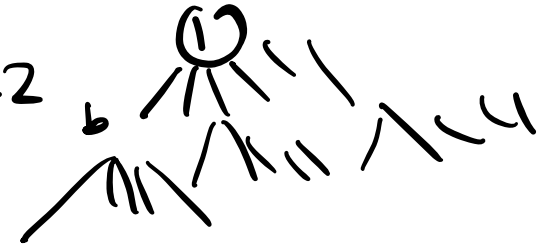
$l=1$


1 node

$1+b$ nodes

$1+b+b^2$ nodes

$l=2$




$1+b+b^2+b^3$ nodes

$l=3$

$\vdots$

$l=d$

$1+b+b^2+b^3+\dots+b^d$

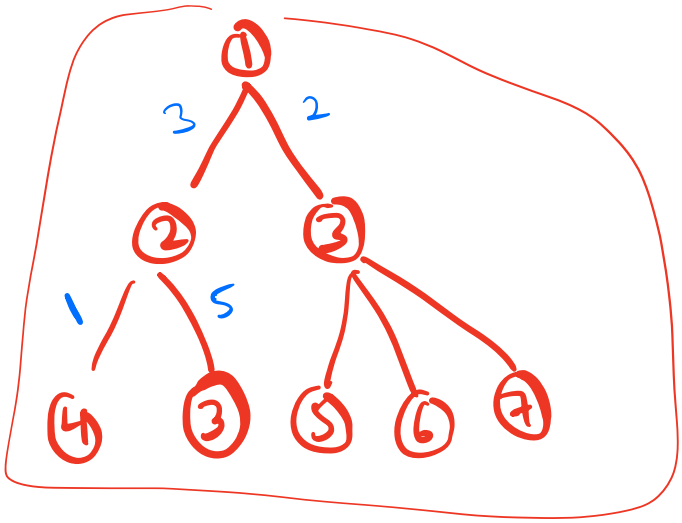
$$
\begin{aligned}
&1 \\
&+\ 1+b \\
&+\ 1+b+b^2 \\
&+\ 1+b+b^2+b^3 \\
&+\ \vdots \\
&+\ 1+b+b^2+b^3+\dots+b^d
\end{aligned}
$$

$$
\begin{aligned}
&= 1(d+1) + bd \\
&\quad + b^2(d-1) \\
&\quad + \dots + b^{d-1} \cdot 2 \\
&\quad + b^d \cdot 1
\end{aligned}
$$

$O(b^d)$

it will cost a bit more than the regular DFS but it still has the same performance in terms of big O notation (asymptotically same as DFS)

Yay for completeness




Goal node is 3

$l = 2$

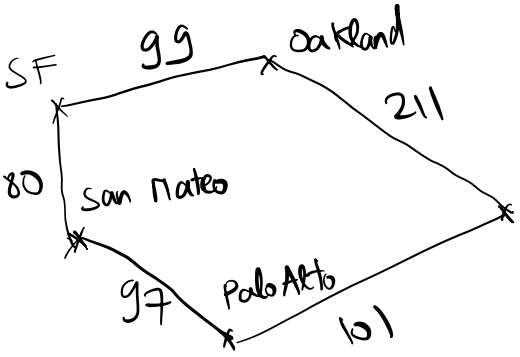
we still have the issue of optimality

Let's explore a search algorithm that can find the optimal solution

## Uniform Cost Search

the node chosen to be expanded will have the lowest path cost

* goal test is applied when it is selected for expansion (rather than when it is generated)
* we'll add another test in case a better path is found to a node that is currently in frontier




**function** UNIFORM-COST-SEARCH(*problem*) **returns** a solution, or failure

*node* ← a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0
*frontier* ← a priority queue ordered by PATH-COST, with *node* as the only element
*explored* ← an empty set
**loop do**
  **if** EMPTY?(*frontier*) **then return** failure
  *node* ← POP(*frontier*) /* chooses the lowest-cost node in *frontier* */
  **if** *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)
  add *node*.STATE to *explored*
  **for each** *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
    *child* ← CHILD-NODE(*problem*, *node*, *action*)
    **if** *child*.STATE is not in *explored* or *frontier* **then**
      *frontier* ← INSERT(*child*, *frontier*)
    **else if** *child*.STATE is in *frontier* with higher PATH-COST **then**
      replace that *frontier* node with *child*

Goal SF to San Jose

node = SF
Frontier = { SF } 0
explored = { }

loop
node popped = SF (not goal)
explored = {SF}

loop actions

child = San Mateo
frontier = { San Mateo } 80

child = Oakland
frontier = { San Mateo 80, Oakland 99 }

node popped = San Mateo (not goal)
explored = {SF, San Mateo}

loop actions

child = Palo Alto
frontier = { Oakland 99, Palo Alto 80+97 = 177 }

node popped = Oakland (not goal)
explored = {SF, San Mateo, Oakland}

loop actions

child = San Jose
frontier = { Palo Alto 177, San Jose 99+211 = 310 }

node popped = Palo Alto (not goal)

explored = {SF, San Mateo, Oakland, Palo Alto}

loop actions

child = San Jose (already in frontier but with a higher path cost ⇒ will be replaced)

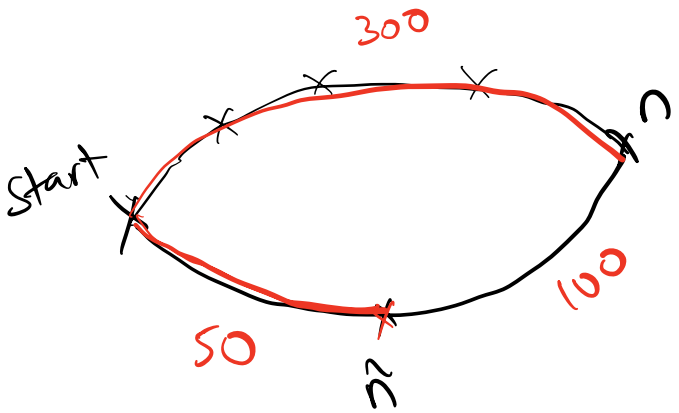
Frontier = {San Jose} 177 + 101 = 278

node popped = San Jose (Yes, it is the goal node)

return San Jose with path cost of 278

## Why is it the optimal path?

* Whenever UCS selects some node $n$ for expansion, the optimal path to that node has been found




Assume path in red was not the optimal one

there should have been another node $\hat{n}$ on the optimal path from start to $n$ with $\hat{n}$ having lower cost than $n$ (requiring it to get selected first)

* Assume that the step costs are non-negative (paths never get less costly as nodes are added)

$\Rightarrow$ UCS expands nodes in order of their optimal path cost.

* Cost to the optimal solution is $C^*$
* You had edges with costs $\varepsilon$ (a very small positive quantity)

$$O\left(b^{\left\lfloor \frac{C^*}{\varepsilon} \right\rfloor}\right)$$

could become very costly especially if $C^*$ is really large and $\varepsilon$ is really small

We will try to speed up the process by using some approximations / heuristics in the algorithm.

$\Rightarrow$ informed search strategies