Uninformed Search

Vincent A. Cicirello, Ph.D.
Professor of Computer Science
cicirelv@stockton.edu https://www.cicirello.org/

1

Lesson 1

SEARCH PROBLEMS

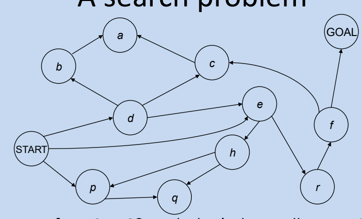
2

Overview

- Deterministic, single-agent, search problems
- Breadth First Search
- Optimality, Completeness, Time and Space complexity
- Search Trees
- Depth First Search
- Bidirectional Search
- Iterative Deepening

3

A search problem



How do we get from S to G? And what's the smallest possible number of transitions?

4

Formalizing a search problem

A search problem has five components:

Q , *S* , *G* , **succs** , **cost**

- *Q* is a finite set of states.
- $S \subseteq Q$ is a non-empty set of start states.
- $G \subseteq Q$ is a non-empty set of goal states.
- **succs** : $Q \rightarrow P(Q)$ is a function which takes a state as input and returns a set of states as output. **succs**(*s*) means "the set of states you can reach from *s* in one step".
- **cost** : *Q* , $Q \rightarrow$ *Positive Number* is a function which takes two states, *s* and *s*', as input. It returns the one-step cost of traveling from *s* to *s*'. The cost function is only defined when *s*' is a successor state of *s*.

5

Our Search Problem

Q = {START, *a* , *b* , *c* , *d* , *e* , *f* , *h* , *p* , *q* , *r* , GOAL}

S = { START }

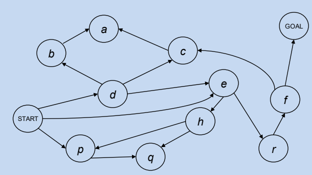
G = { GOAL }

succs(*b*) = { *a* }

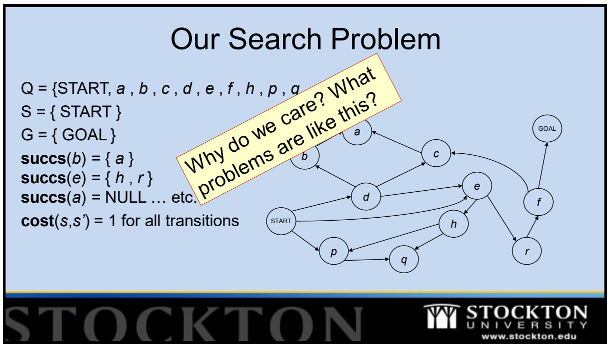
succs(*e*) = { *h* , *r* }

succs(*a*) = NULL … etc.

cost(*s*,*s*') = 1 for all transitions



6


Our Search Problem
Q = {START, a , b , c , d , e , f , h , p , q
S = { START }
G = { GOAL }
succs(b) = { a }
succs(e) = { h , r }
succs(a) = NULL … etc.
cost(s,s') = 1 for all transitions
Why do we care? What problems are like this?
GOAL
START
STOCKTON UNIVERSITY
www.stockton.edu

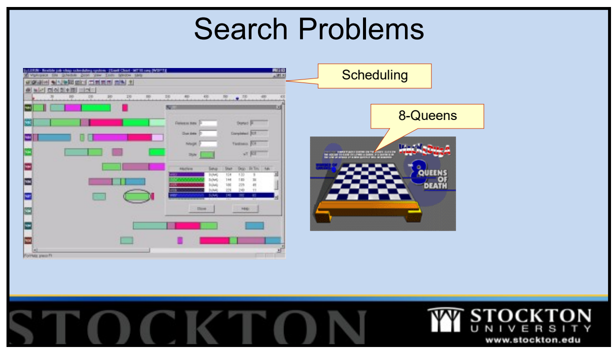
7


Search Problems
1 2 3
6 7
8 5 4
STOCKTON UNIVERSITY
www.stockton.edu

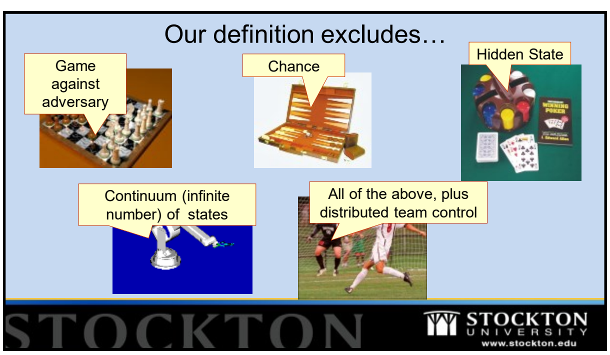
8


Search Problems
STOCKTON UNIVERSITY
www.stockton.edu

9


Search Problems
Scheduling
8-Queens
STOCKTON UNIVERSITY
www.stockton.edu

10


Our definition excludes…
Game against adversary
Chance
Hidden State
Continuum (infinite number) of states
All of the above, plus distributed team control
STOCKTON UNIVERSITY
www.stockton.edu

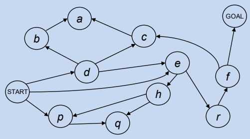
11


Lesson 2
BREADTH FIRST SEARCH (BFS)
STOCKTON UNIVERSITY
www.stockton.edu

12

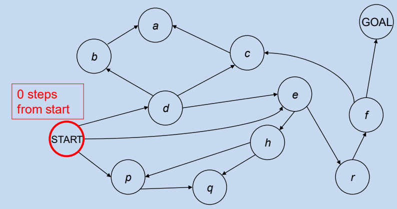
Breadth First Search

- Label all states that are reachable from S in 1 step but aren't reachable in less than 1 step.
- Then label all states that are reachable from S in 2 steps but aren't reachable in less than 2 steps.
- Then label all states that are reachable from S in 3 steps but aren't reachable in less than 3 steps.
- Etc… until Goal state reached.



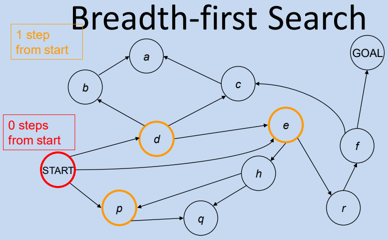
13

Breadth-first Search



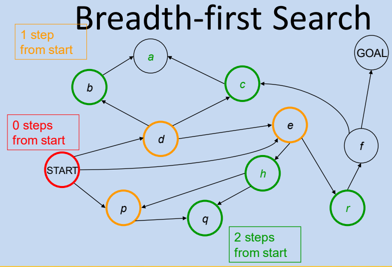
14

Breadth-first Search



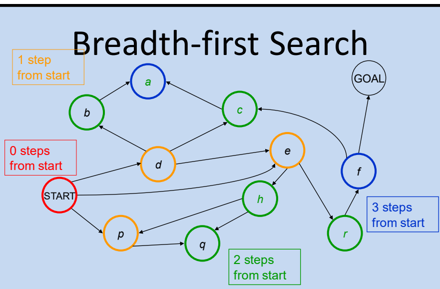
15

Breadth-first Search



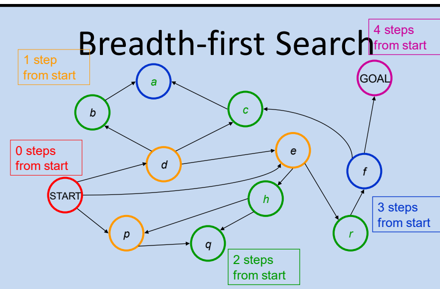
16

Breadth-first Search



17

Breadth-first Search



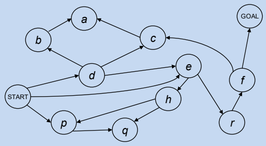
18

Remember the path!

Also, when you label a state, record the predecessor state. This record is called a *backpointer*. The history of predecessors is used to generate the solution path, once you've found the goal:

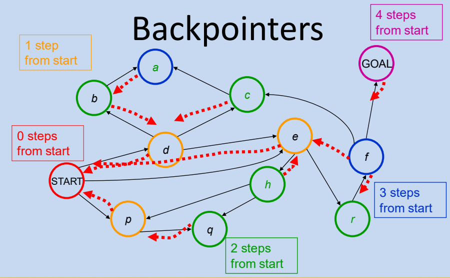
"I've got to the goal. I see I was at *f* before this. And I was at *r* before I was at *f*. And I was…

…. so solution path is S → *e* → *r* → *f* → G"



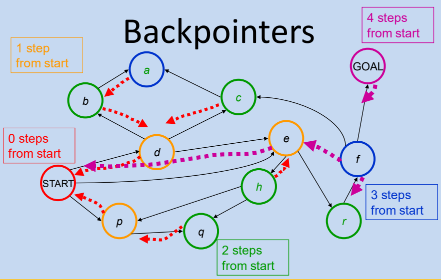
19

Backpointers



20

Backpointers



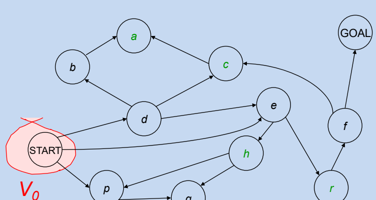
21

Starting Breadth First Search

- For any state *s* that we've labeled, we'll remember:
  - *previous*(*s*) as the previous state on a shortest path from START state to *s*.
- On the *k*th iteration of the algorithm we'll begin with V_k defined as the set of those states for which the shortest path from the start costs exactly *k steps*
- Then, during that iteration, we'll compute V_{k+1}, defined as the set of those states for which the shortest path from the start costs exactly *k+1 steps*
- We begin with $k = 0$, V_0 = {START} and we'll define, *previous*(*START*) = *NULL*
- Then we'll add in things one step from the START into V_1. And we'll keep going.

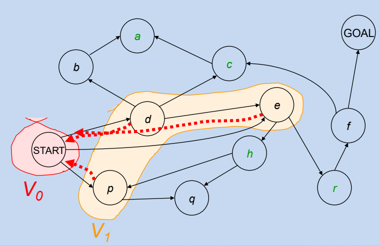
22

BFS



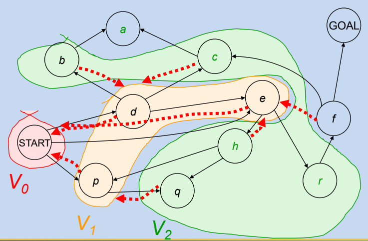
23

BFS



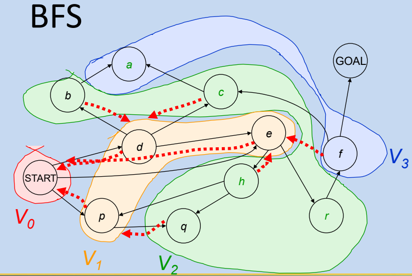
24

BFS



25

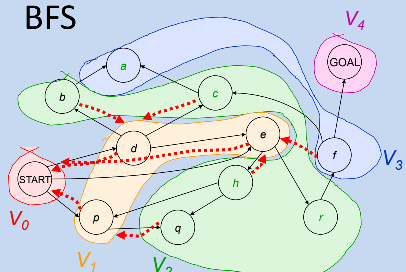
BFS



STOCKTON UNIVERSITY www.stockton.edu

26

BFS



STOCKTON UNIVERSITY www.stockton.edu

27

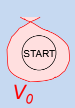
Breadth First Search

V_0 := S (the set of start states)
previous(*START*) := *NIL*
$k := 0$
while (no goal state is in V_k and V_k is not empty) **do**
    V_{k+1} := empty set
    For each state s in V_k
        For each state s' in **succs**(s)
            If s' has not already been labeled
                Set *previous*(s') := s
                Add s' into V_{k+1}
    $k := k+1$
If V_k is empty signal FAILURE
Else build the solution path thus: Let S_i be the *i*th state in the shortest path. Define S_k = GOAL, and forall $i <= k$, define S_{i-1} = *previous*(S_i).

STOCKTON UNIVERSITY www.stockton.edu

28

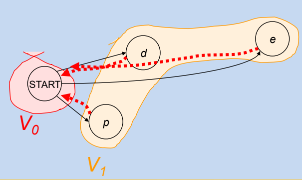
BFS



STOCKTON UNIVERSITY www.stockton.edu

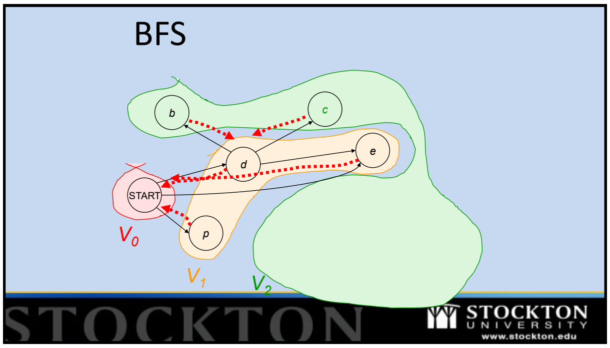
29

BFS

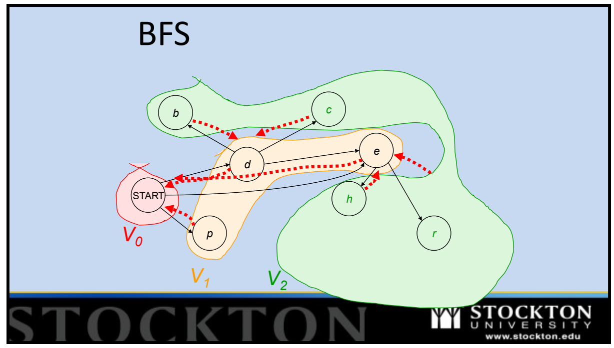


STOCKTON UNIVERSITY www.stockton.edu

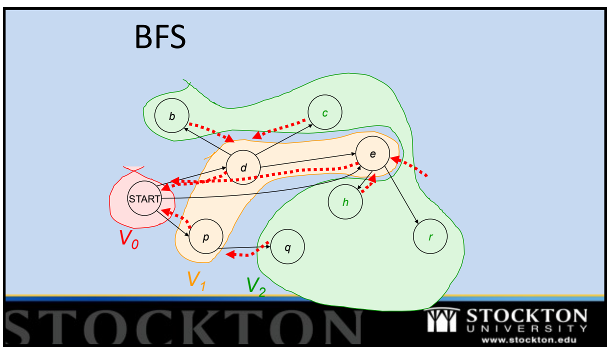
30



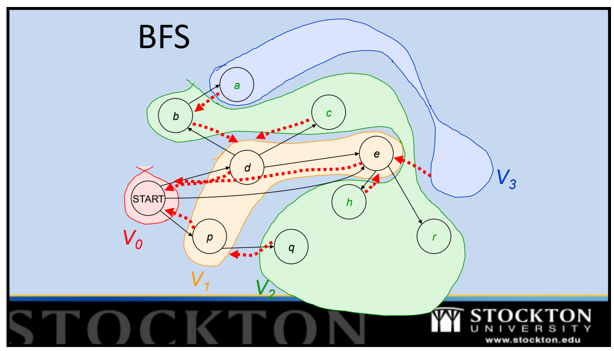
31



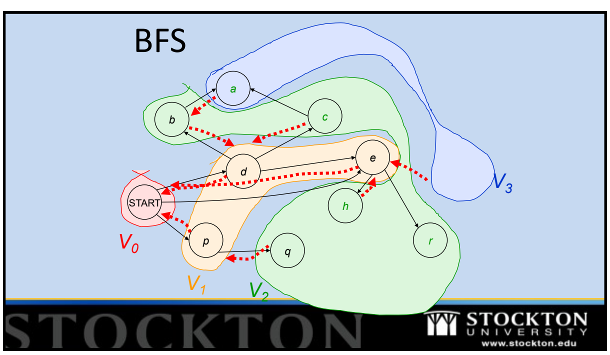
32



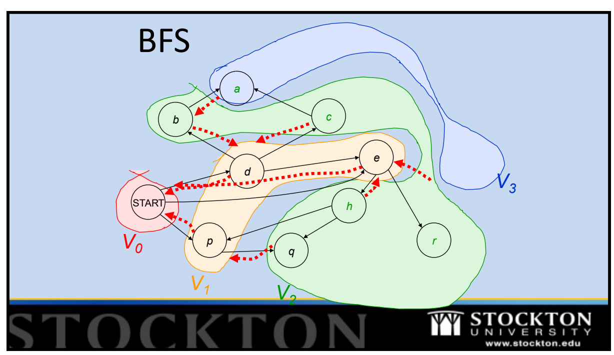
33



34

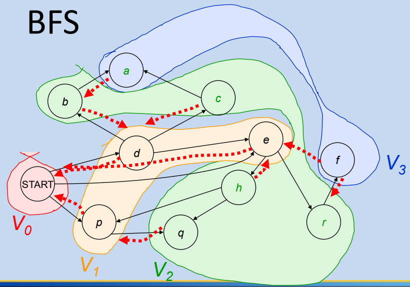


35



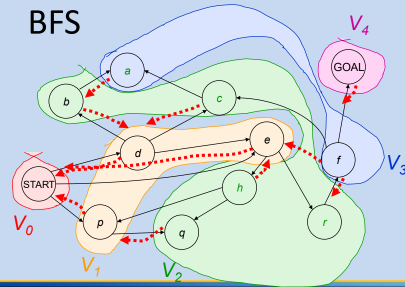
36

BFS



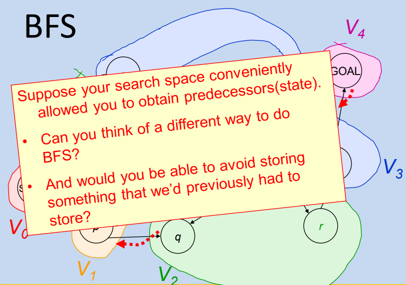
37

BFS



38

BFS



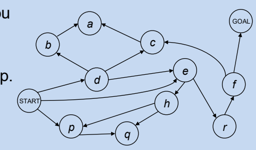
Suppose your search space conveniently allowed you to obtain predecessors(state).

- Can you think of a different way to do BFS?
- And would you be able to avoid storing something that we'd previously had to store?

39

Another way: Work backwards

- If you have a predecessors function, you can work backwards.
- Label all states that can reach G in 1 step but can't reach it in less than 1 step.
- Label all states that can reach G in 2 steps but can't reach it in less than 2 steps.
- Etc. … until start is reached.
- "number of steps to goal" labels determine the shortest path. Don't need extra bookkeeping info.



40

Breadth First Details

- It is fine for there to be more than one goal state.
- It is fine for there to be more than one start state.
- BFS usually works forwards from the start. Any algorithm which works forwards from the start is said to be *forward chaining*.
- You can also work backwards from the goal.
- Any algorithm which works backwards from the goal is said to be *backward chaining*.

41

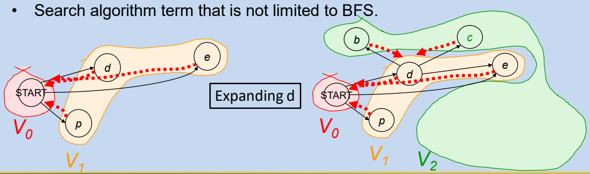
Lesson 3

SOME SEARCH TERMINOLOGY

42

Expanding

- Previously, one of the operations that we saw in BFS involved iterating over the successors of a state.
- That operation is referred to as **expanding** the state.
- Search algorithm term that is not limited to BFS.

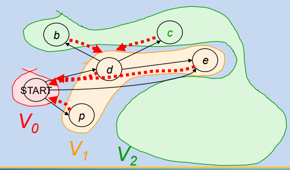


43

The Frontier

- The **frontier** is the set of states that the search has generated, but have not yet expanded.
- We know the frontier states are there, but we don't yet know what is beyond the frontier.

The frontier at this point in our BFS example consists of the states: { b, c, e, p }.



44

BFS and the Frontier

- We previously saw BFS, including pseudocode.
- There is another way to specify BFS that involves using a queue to explicitly keep track of the frontier.
  - Initialize the frontier to a queue containing the start state
  - Each iteration, poll the queue, and expand the state at its head, inserting its successors (if they aren't already labeled) into the tail of the queue.
  - Keep track of backpointers, and V labels as before.

45

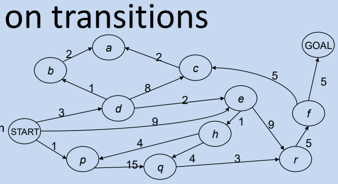
Lesson 4

LEAST COST BFS AND UNIFORM COST SEARCH (UCS)

46

Costs on transitions

- Notice that BFS finds the shortest path in terms of number of transitions. It does not find the least-cost path.
- We will quickly examine an algorithm which does find the least-cost path.
- On the *k*th iteration, for any state *S*, write $g(s)$ as the least-cost path to *S* in *k* or fewer steps.



47

Least Cost Breadth First

V_k = the set of states which can be reached in exactly *k* steps, and for which the least-cost *k*-step path is less cost than any path of length less than *k*. In other words, V_k = the set of states whose values changed on the previous iteration.

```
V0 := S (the set of start states)
previous(START) := NIL
g(START) = 0
k := 0
while (Vk is not empty) do
    Vk+1 := empty set
    For each state s in Vk
        For each state s' in succs(s)
            If s' has not already been labeled
            OR if g(s) + Cost(s,s') < g(s')
                Set previous(s') := s
                Set g(s') := g(s) + Cost(s,s')
                Add s' into Vk+1
    k := k+1
If GOAL not labeled, exit signaling FAILURE
Else build the solution path thus:  Let Sk be the kth state in the shortest path.  Define Sk = GOAL, and
forall i <= k, define Si-1 = previous(Si).
```


48

Uniform-Cost Search

- A conceptually simple BFS-like approach when there are costs on transitions
- It uses priority queues

49

Priority Queue Refresher

A priority queue is a data structure in which you can insert and retrieve (thing, value) pairs with the following operations:

Init-PriQueue(PQ)	initializes the PQ to be empty.
Insert-PriQueue(PQ, thing, value)	inserts *(thing, value)* into the queue.
Pop-least(PQ)	returns the *(thing, value)* pair with the lowest value, and removes it from the queue.

STOCKTON UNIVERSITY www.stockton.edu

50

Priority Queue Refresher

A priority queue is a data structure in which you can insert and retrieve (thing, value) pairs with the following operations:

Init-PriQueue(PQ)	initializes the PQ to be empty.
Insert-PriQueue(PQ, thing, value)	inserts *(thing, value)* into the queue.
Pop-least(PQ)	returns the *(thing, value)* pair with the lowest value, and removes it from the queue.

Priority Queues can be implemented (with a binary minheap) in such a way that the cost of the insert and pop operations are:

O(log(number of things in priority queue))

STOCKTON UNIVERSITY www.stockton.edu

51

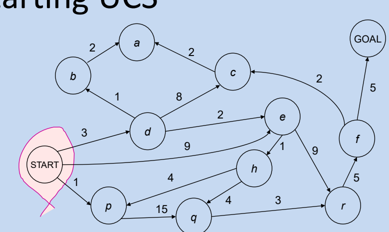
Uniform-Cost Search

- A conceptually simple BFS approach when there are costs on transitions
- It uses a priority queue
  - PQ = Set of states that have been expanded or are awaiting expansion, i.e., the frontier
  - Priority of state *s* = *g(s)* = cost of getting to *s* using path implied by backpointers.
- This is essentially Dijkstra's algorithm
  - Difference is that Dijkstra's algorithm assumes entire graph is in memory, whereas UCS does not make that assumption.

STOCKTON UNIVERSITY www.stockton.edu

52

Starting UCS

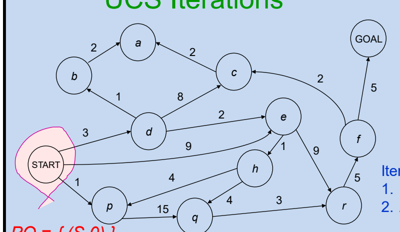


PQ = { (S,0) }

STOCKTON UNIVERSITY www.stockton.edu

53

UCS Iterations

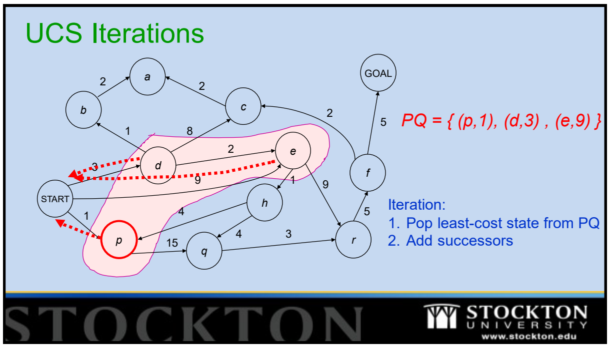


Iteration:
1. Pop least-cost state from PQ
2. Add successors

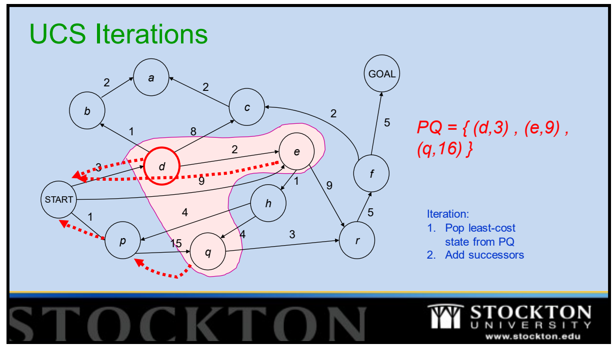
PQ = { (S,0) }

STOCKTON UNIVERSITY www.stockton.edu

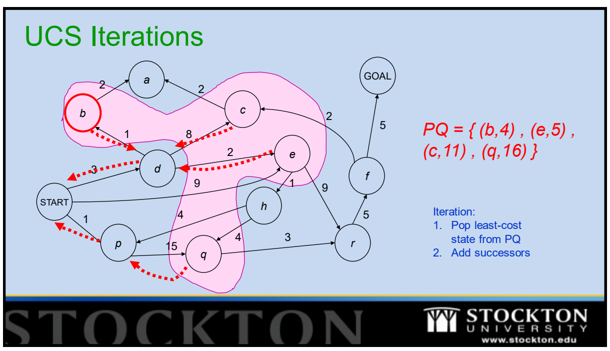
54



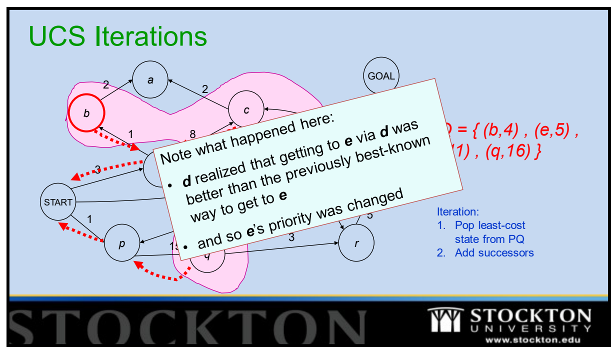
55



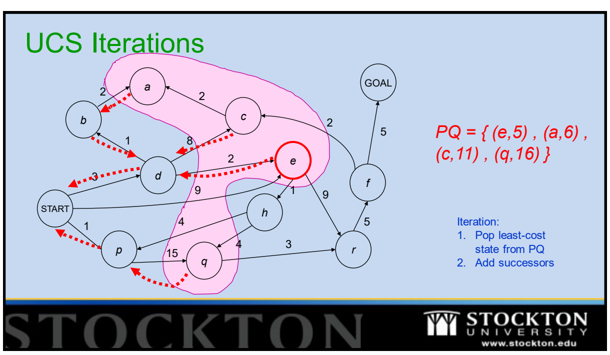
56



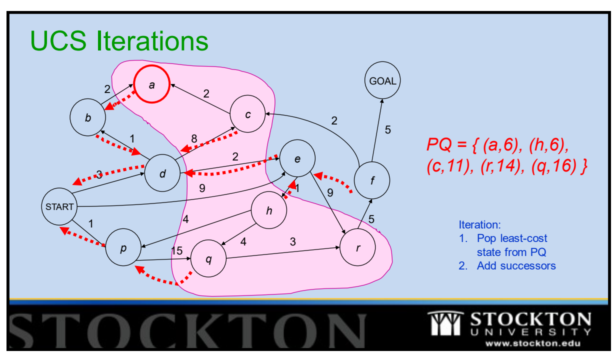
57



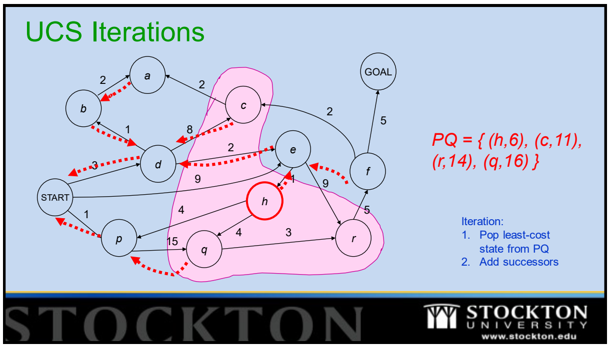
58



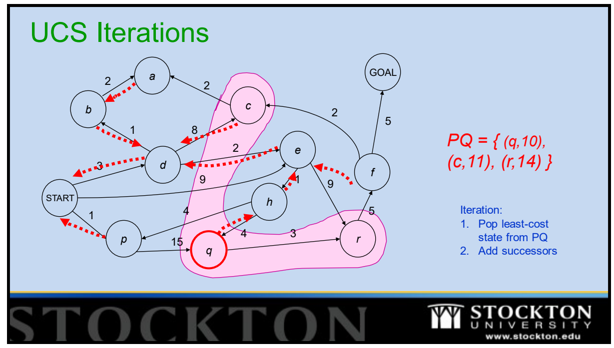
59



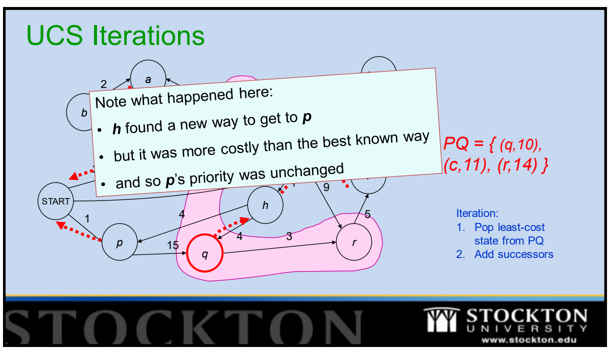
60



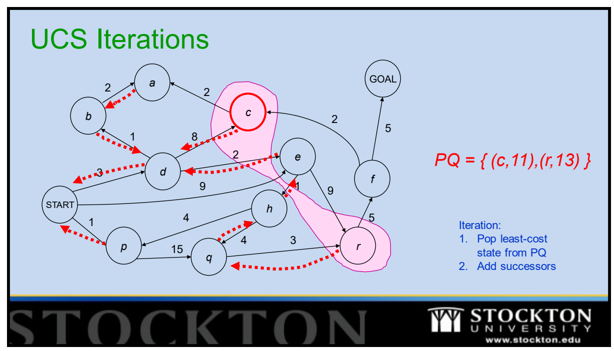
61



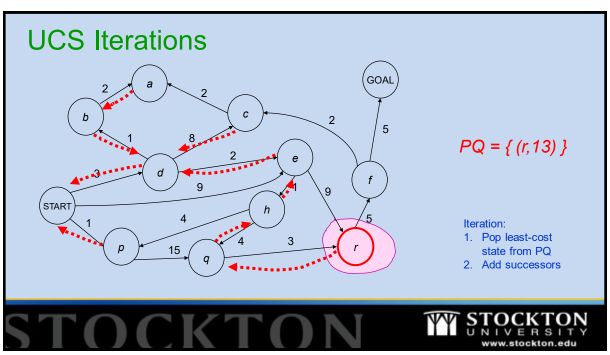
62



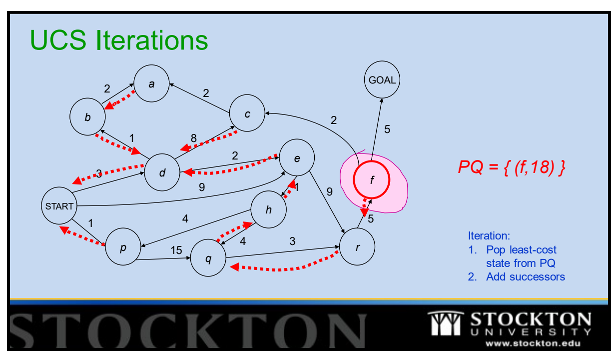
63



64

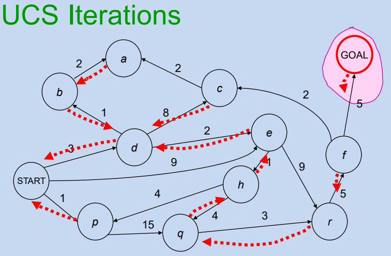


65



66

UCS Iterations

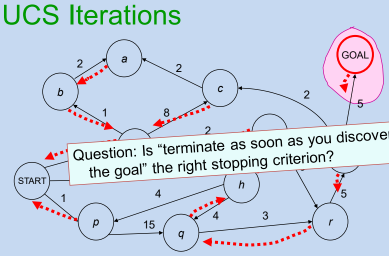


PQ = { (G,23) }

Iteration:
1. Pop least-cost state from PQ
2. Add successors

67

UCS Iterations



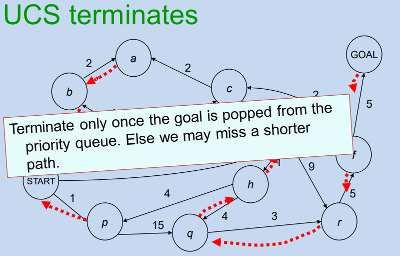
Question: Is "terminate as soon as you discover the goal" the right stopping criterion?

PQ = { (G,23) }

Iteration:
1. Pop least-cost state from PQ
2. Add successors

68

UCS terminates



Terminate only once the goal is popped from the priority queue. Else we may miss a shorter path.

PQ = { }

Iteration:
1. Pop least-cost state from PQ
2. Add successors

69

Lesson 5

JUDGING A SEARCH ALGORITHM

70

Judging a search algorithm

- Completeness: is the algorithm guaranteed to find a solution if a solution exists?
- Guaranteed to find optimal? (will it find the least cost path?)
- Algorithmic time complexity
- Space complexity (memory use)

Variables:

N	number of states in the problem
B	the average branching factor (the average number of successors) (*B*>1)
L	the length of the path from start to goal with the shortest number of steps

71

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (*B*>1)
L	the length of the path from start to goal with the shortest number of steps
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y			
LCBFS	Least Cost BFS				
UCS	Uniform Cost Search				

72

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost		
LCBFS	Least Cost BFS				
UCS	Uniform Cost Search				

STOCKTON UNIVERSITY www.stockton.edu

73

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS				
UCS	Uniform Cost Search				

STOCKTON UNIVERSITY www.stockton.edu

74

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y			
UCS	Uniform Cost Search				

STOCKTON UNIVERSITY www.stockton.edu

75

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y		
UCS	Uniform Cost Search				

STOCKTON UNIVERSITY www.stockton.edu

76

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search				

STOCKTON UNIVERSITY www.stockton.edu

77

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y			

STOCKTON UNIVERSITY www.stockton.edu

78

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
Q	the average size of the priority queue

Algorithm		Com plete	Optima l	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y		

79

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
Q	the average size of the priority queue

Algorithm		Com plete	Optima l	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$

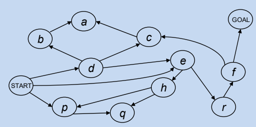
80

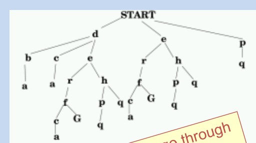
Lesson 6

DEPTH FIRST SEARCH (DFS)

81

Search Tree Representation



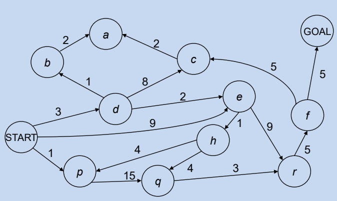


What order do we go through the search tree with BFS?

82

Depth First Search

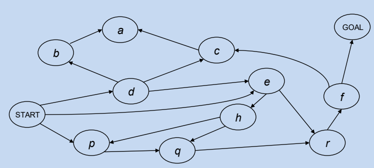
An alternative to BFS. Always expand from the most-recently-expanded node, if it has any untried successors. Else backup to the previous node on the current path.



83

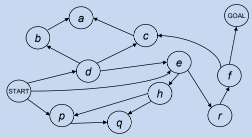
DFS in action

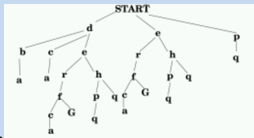
START
START *d*
START *d b*
START *d b a*
START *d c*
START *d c a*
START *d e*
START *d e r*
START *d e r f*
START *d e r f c*
START *d e r f c a*
START *d e r f* GOAL



84

DFS Search tree traversal





Can you draw in the order in which the search-tree nodes are visited?

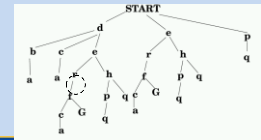
85

DFS Algorithm

We use a data structure we'll call a Path to represent the path from the START to the current state.

E.G. Path P = <START, d, e, r >

Along with each node on the path, we must remember which successors we still have available to expand. E.G. at the following point, we'll have



P = <START (expand=e , p) ,
d (expand = NULL) ,
e (expand = h) ,
r (expand = f) >

86

DFS Algorithm

Let P = <START (expand = succs(START))>
While (P not empty and top(P) not a goal)
    if expand of top(P) is empty
    then
        remove top(P) ("pop the stack")
    else
        let s be a member of expand of top(P)
        remove s from expand of top(P)
        make a new item on the top of path P:
            s (expand = succs(s))
If P is empty
    return FAILURE
Else
    return the path consisting of states in P

The path object P is essentially a stack.

Also serves as the frontier (or at least the "expand" fields in the elements of this stack do).

This algorithm can be written neatly with recursion, i.e. using the program stack to implement P.

87

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (*B*>1)
L	the length of the path from start to goal with the shortest number of steps
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$
DFS	Depth First Search	N			

88

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (*B*>1)
L	the length of the path from start to goal with the shortest number of steps
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$
DFS	Depth First Search	N	N		

89

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (*B*>1)
L	the length of the path from start to goal with the shortest number of steps
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$
DFS	Depth First Search	N	N	*N/A*	*N/A*

90

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest path from start to anywhere
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$
DFS**	Depth First Search	Y			

Assuming Acyclic Search Space

91

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest path from start to anywhere
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$
DFS**	Depth First Search	Y	N		

Assuming Acyclic Search Space

92

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest path from start to anywhere
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$
DFS**	Depth First Search	Y	N	$O(B^{LMAX})$	$O(LMAX)$

Assuming Acyclic Search Space

93

Lesson 7

A FEW NOTES ON RUNTIMES OF BFS AND DFS

94

Why are the runtimes exponential?

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest path from start to anywhere
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$
DFS**	Depth First Search	Y	N	$O(B^{LMAX})$	$O(LMAX)$

Assuming Acyclic Search Space

95

Why are the runtimes exponential?

- Wait a minute! I thought the runtimes of BFS and DFS were linear!!!
- You are correct, the runtimes are linear in the size of the graph.
- But our search problem is defined by a graph whose size is exponential in the branching factor B.

96

Why are the runtimes exponential?

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest path from start to anywhere
Q	the average size of the priority queue

- Runtime of BFS: O(V + E) where V is number of vertices and E number of edges.
  - Each problem state is a vertex in graph, so V = N.
  - There are E = BN edges.
  - O(V + E) = O(N + BN) = O(BN)
- But that still doesn't look exponential!!!!
  - Each node expansion generates B successors, each of which are expanded in turn until we reach the goal, for a cost of B * B * B * …. [L of these] = B^L
- What is that $O(min(BN,B^L))$ all about?
  - BFS remembers states it's seen before and doesn't reexpand any.

STOCKTON STOCKTON UNIVERSITY www.stockton.edu

97

What about DFS?

- Isn't the runtime of DFS O(V + E) as well, so shouldn't that same analysis for BFS lead to $O(min(BN,B^L))$?
- Why is it $O(B^{LMAX})$? Where did the min go?
- The exponent is LMAX (length of longest acyclic path to anywhere) instead of L because DFS might go down a longer path.
- The min involving BN isn't there because our DFS doesn't remember all states it has seen before.
  – If you use a version that does, then it would be $O(min(BN,B^{LMAX}))$
- *When we use DFS in AI, we'll typically do so to avoid exponential memory, which means we won't be able to keep track of all states that we visit, so runtime will be $O(B^{LMAX})$.*

STOCKTON STOCKTON UNIVERSITY www.stockton.edu

98

Lesson 8

VARIATIONS OF DFS

STOCKTON STOCKTON UNIVERSITY www.stockton.edu

99

Questions to ponder

- Removing our assumption of acyclic search space….
- How would you prevent DFS from looping?
- How could you force it to give an optimal solution?

STOCKTON STOCKTON UNIVERSITY www.stockton.edu

100

Questions to ponder

- Removing our assumption of acyclic search space….
- How would you prevent DFS from looping?
- How could you force it to give an optima

Answer 1:
PC-DFS (Path Checking DFS):

Answer 2:
MEMDFS (Memoizing DFS):

solution?

STOCKTON STOCKTON UNIVERSITY www.stockton.edu

101

Questions to ponder

- Removing our assumption of acyclic search space….
- How would you prevent DFS from looping?
- How could you force it to give an optima

Answer 1:
PC-DFS (Path Checking DFS):
Don't recurse on a state if that state is already in the current path

Answer 2:
MEMDFS (Memoizing DFS):
Remember all states expanded so far. Never expand anything twice.

solution?

STOCKTON STOCKTON UNIVERSITY www.stockton.edu

102

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest **cycle-free** path from start to anywhere
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$
PCDFS	Path Check DFS	Y			
MEMDFS	Memoizing DFS	Y			

103

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest **cycle-free** path from start to anywhere
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$
PCDFS	Path Check DFS	Y	N		
MEMDFS	Memoizing DFS	Y	N		

104

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest **cycle-free** path from start to anywhere
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$
PCDFS	Path Check DFS	Y	N	$O(B^{LMAX})$	
MEMDFS	Memoizing DFS	Y	N	$O(min(BN,B^{LMAX}))$	

105

Judging a search algorithm

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest **cycle-free** path from start to anywhere
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$
PCDFS	Path Check DFS	Y	N	$O(B^{LMAX})$	$O(LMAX)$
MEMDFS	Memoizing DFS	Y	N	$O(min(BN,B^{LMAX}))$	$O(min(N,B^{LMAX}))$

106

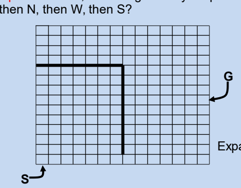
Lesson 9

ANIMATIONS OF BFS AND DFS

107

Maze example

Imagine states are cells in a maze, you can move N, E, S, W. What would plain DFS do, assuming it always expanded the E successor first, then N, then W, then S?

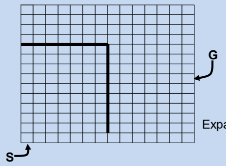


Expansion order E, N, W, S

108

Maze example (BFS)

Imagine states are cells in a maze, you can move N, E, S, W. What would plain BFS do, assuming it always expanded the E successor first, then N, then W, then S?

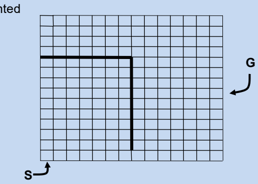


109

Another DFS examples

Order: N, E, S, W

with loops prevented



STOCKTON UNIVERSITY www.stockton.edu

110

Lesson 10

BIDIRECTIONAL BFS

STOCKTON UNIVERSITY www.stockton.edu

111

Invent An Algorithm Time!

- Here's a way to dramatically decrease costs sometimes. Bidirectional Search. Can you guess what this algorithm is, and why it can be a huge cost-saver?
- Run two BFS, one forward chaining from start, and the other backward chaining from the goal.
- Interleave their execution so that forward search does goes one full level away from start and then backward search does one full level backwards.
- Terminate when searches meet in middle.

STOCKTON UNIVERSITY www.stockton.edu

112

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest **cycle-free** path from start to anywhere
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	*O(min(BN,B^L))*	*O(min(N,B^L))*
LCBFS	Least Cost BFS	Y	Y	*O(min(BN,B^L))*	*O(min(N,B^L))*
UCS	Uniform Cost Search	Y	Y	*O(log(Q)*min(BN,B^L))*	*O(min(N,B^L))*
PCDFS	Path Check DFS	Y	N	*O(B^LMAX)*	*O(LMAX)*
MEMDFS	Memoizing DFS	Y	N	*O(min(BN,B^LMAX))*	*O(min(N,B^LMAX))*
BIBFS	Bidirection BF Search	Y			

STOCKTON UNIVERSITY www.stockton.edu

113

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest **cycle-free** path from start to anywhere
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	*O(min(BN,B^L))*	*O(min(N,B^L))*
LCBFS	Least Cost BFS	Y	Y	*O(min(BN,B^L))*	*O(min(N,B^L))*
UCS	Uniform Cost Search	Y	Y	*O(log(Q)*min(BN,B^L))*	*O(min(N,B^L))*
PCDFS	Path Check DFS	Y	N	*O(B^LMAX)*	*O(LMAX)*
MEMDFS	Memoizing DFS	Y	N	*O(min(BN,B^LMAX))*	*O(min(N,B^LMAX))*
BIBFS	Bidirection BF Search	Y	All trans same cost		

STOCKTON UNIVERSITY www.stockton.edu

114

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest cycle-free path from start to anywhere
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$
PCDFS	Path Check DFS	Y	N	$O(B^{LMAX})$	$O(LMAX)$
MEMDFS	Memoizing DFS	Y	N	$O(min(BN,B^{LMAX}))$	$O(min(N,B^{LMAX}))$
BIBFS	Bidirection BF Search	Y	All trans same cost	$O(min(BN,2B^{L/2}))$	$O(min(N,2B^{L/2}))$

115

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest cycle-free path from start to anywhere
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$
PCDFS	Path Check DFS	Y	N	$O(B^{LMAX})$	$O(LMAX)$
MEMDFS	Memoizing DFS	Y	N	$O(min(BN,B^{LMAX}))$	$O(min(N,B^{LMAX}))$
BIBFS	Bidirection BF Search	Y	All trans same cost	$O(min(BN,B^{L/2}))$	$O(min(N,B^{L/2}))$

116

Lesson 11

ITERATIVE DEEPENING

117

Depth-limited DFS

- Depth-limited DFS is like DFS, but it limits the search to go no deeper than k steps from start.
- It is not complete (since if shortest path to goal is longer than k steps, you'll miss it).
- Just like DFS, it is also not optimal.
- We'll use it as a subroutine in an algorithm called Iterative Deepening.

118

Iterative Deepening

Iterative deepening is a simple algorithm which uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up any path of length 2)
2. If "1" failed, do a DFS which only searches paths of length 2 or less.
3. If "2" failed, do a DFS which only searches paths of length 3 or less.
   ….and so on until success

Cost is

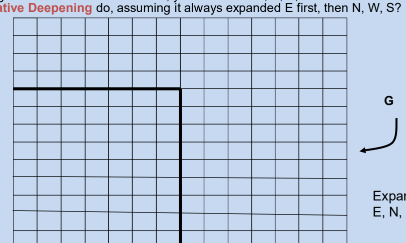
$O(b^1 + b^2 + b^3 + b^4 \ldots + b^L) = O(b^L)$

Can be much better than regular DFS. But cost can be much greater than the number of states.

119

Maze example

Imagine states are cells in a maze, you can move N, E, S, W. What would **Iterative Deepening** do, assuming it always expanded E first, then N, W, S?

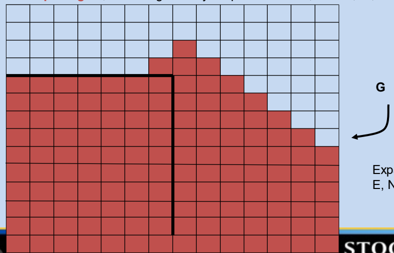


Expansion order
E, N, W, S

120

SKIPPING AHEAD A BIT…

Imagine states are cells in a maze, you can move N, E, S, W. What would **Iterative Deepening** do, assuming it always expanded E first, then N, W, S?



Expansion order
E, N, W, S

121

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest **cycle-free** path from start to anywhere
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$
PCDFS	Path Check DFS	Y	N	$O(B^{LMAX})$	$O(LMAX)$
MEMDFS	Memoizing DFS	Y	N	$O(min(BN,B^{LMAX}))$	$O(min(N,B^{LMAX}))$
BIBFS	Bidirection BF Search	Y	All trans same cost	$O(min(BN,B^{L/2}))$	$O(min(N,B^{L/2}))$
ID	Iterative Deepening	Y			

122

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest **cycle-free** path from start to anywhere
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$
PCDFS	Path Check DFS	Y	N	$O(B^{LMAX})$	$O(LMAX)$
MEMDFS	Memoizing DFS	Y	N	$O(min(BN,B^{LMAX}))$	$O(min(N,B^{LMAX}))$
BIBFS	Bidirection BF Search	Y	All trans same cost	$O(min(BN,B^{L/2}))$	$O(min(N,B^{L/2}))$
ID	Iterative Deepening	Y	if all transitions same cost		

123

N	number of states in the problem
B	the average branching factor (the average number of successors) (B>1)
L	the length of the path from start to goal with the shortest number of steps
LMAX	Length of longest **cycle-free** path from start to anywhere
Q	the average size of the priority queue

Algorithm		Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	if all transitions same cost	$O(min(BN,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	Y	Y	$O(min(BN,B^L))$	$O(min(N,B^L))$
UCS	Uniform Cost Search	Y	Y	$O(log(Q)*min(BN,B^L))$	$O(min(N,B^L))$
PCDFS	Path Check DFS	Y	N	$O(B^{LMAX})$	$O(LMAX)$
MEMDFS	Memoizing DFS	Y	N	$O(min(BN,B^{LMAX}))$	$O(min(N,B^{LMAX}))$
BIBFS	Bidirection BF Search	Y	All trans same cost	$O(min(BN,B^{L/2}))$	$O(min(N,B^{L/2}))$
ID	Iterative Deepening	Y	if all transitions same cost	$O(B^L)$	$O(L)$

124