HY387 – Recitation 2

ΓΙΑΚΟΥΜΆΚΗΣ ΘΕΌΔΩΡΟΣ-ΜΆΡΙΟΣ

9/10/2020

Search Algorithms

Uninformed:

1. Breadth-first search
2. Uniform-cost search
3. Depth-first search
4. Depth-limited search
5. Iterative-deepening depth-first search
6. Bidirectional search

Search Algorithms

Informed:

1. Greedy best-first search
   - f(n) = h(n)
2. A* search
   - f(n) = g(n) + h(n)

g(n): the cost from the initial node to node n

h(n): the heuristic function which estimates the cost from n to the goal

The heuristic is called **admissible** if it never overestimates the cost to the goal.

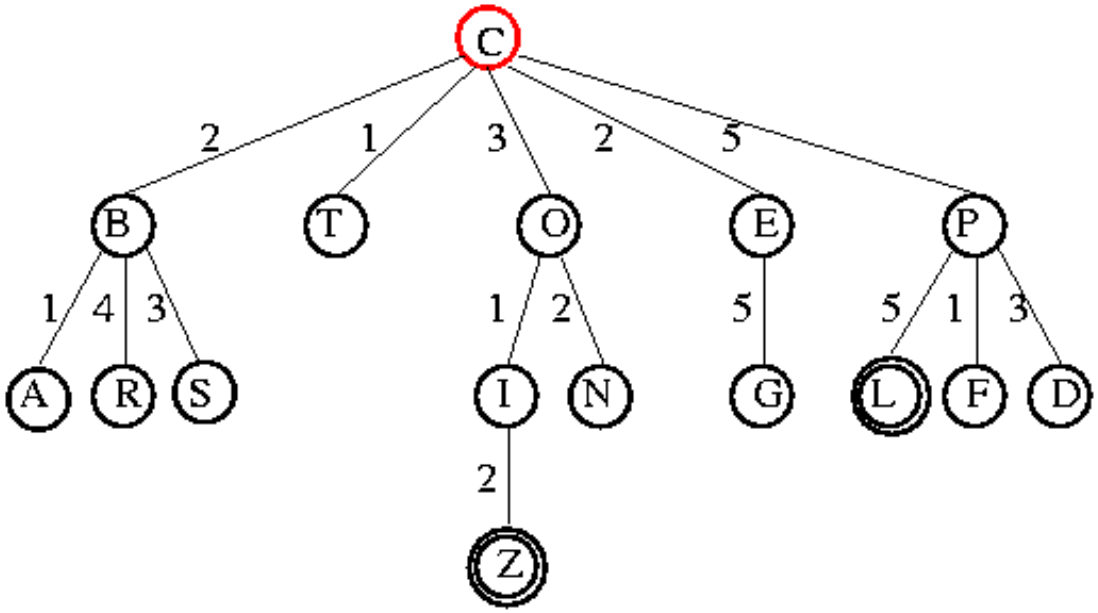
Search Algorithms

Local:

1. Hill - climbing
2. Simulated Annealing
3. Genetic algorithms

Example
Greedy best-first search

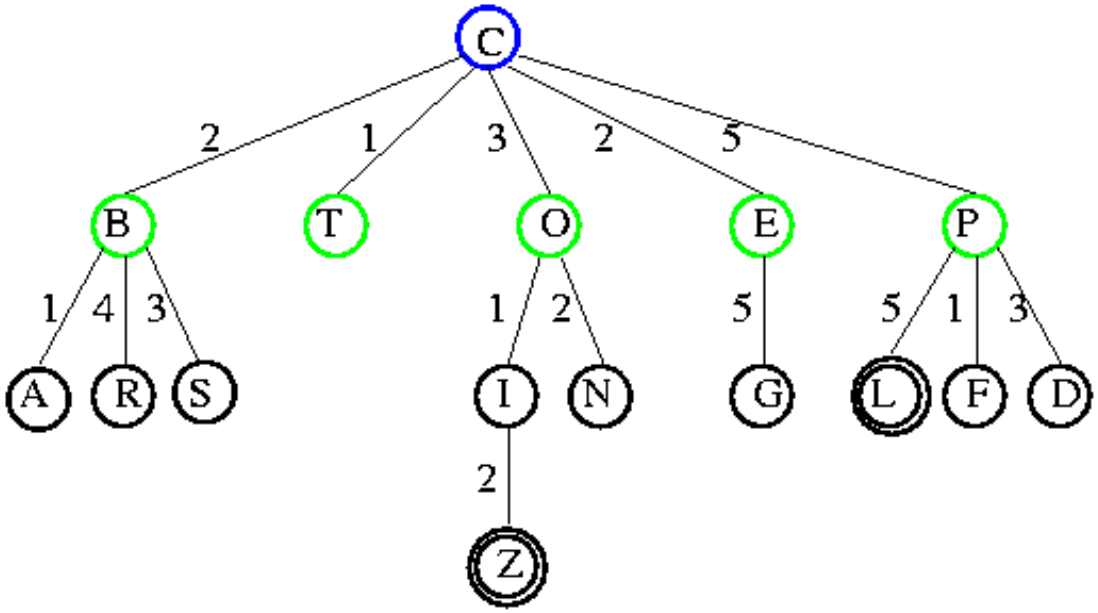
Given a heuristic that calculates cost to next node as such:



Resolve ties alphabetically and based on depth.

Example
Greedy best-first search

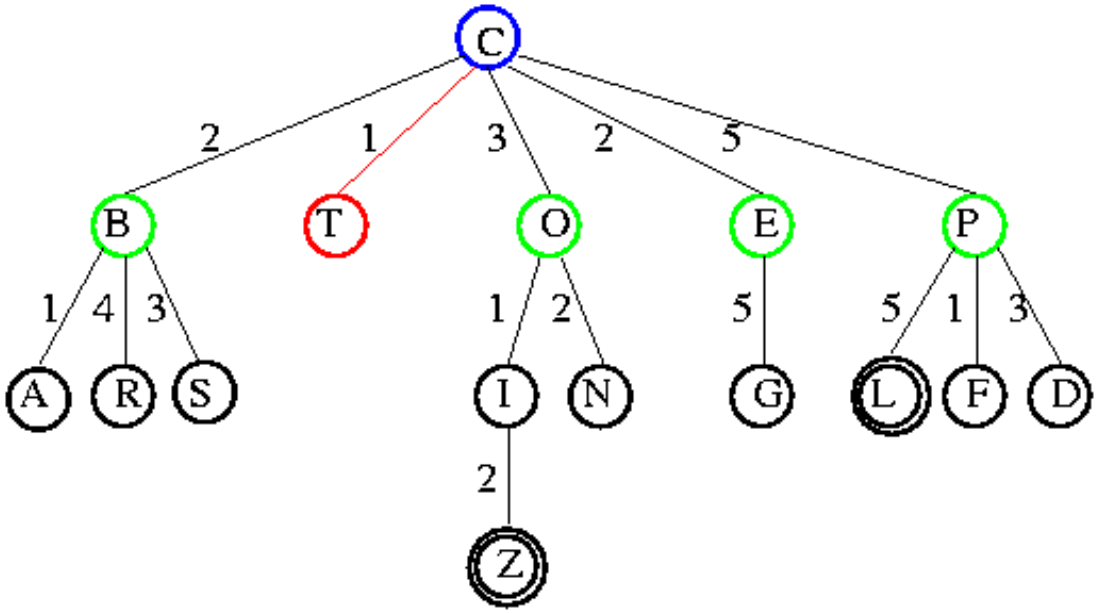
Given a heuristic that calculates cost to next node as such:



Resolve ties alphabetically and based on depth.

Example
Greedy best-first search

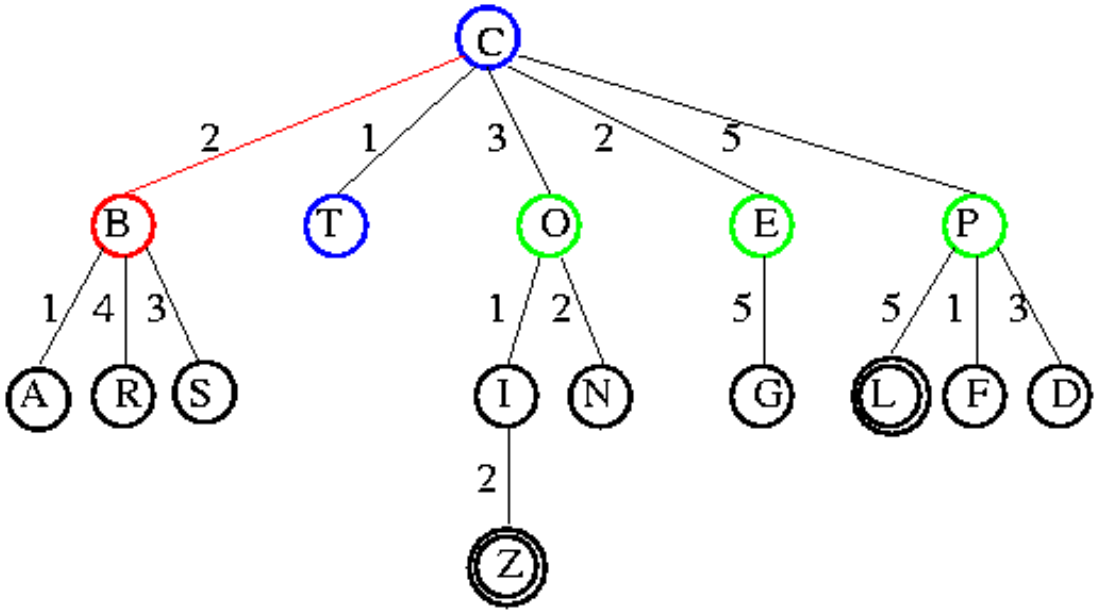
Given a heuristic that calculates cost to next node as such:



Resolve ties alphabetically and based on depth.

Example
Greedy best-first search

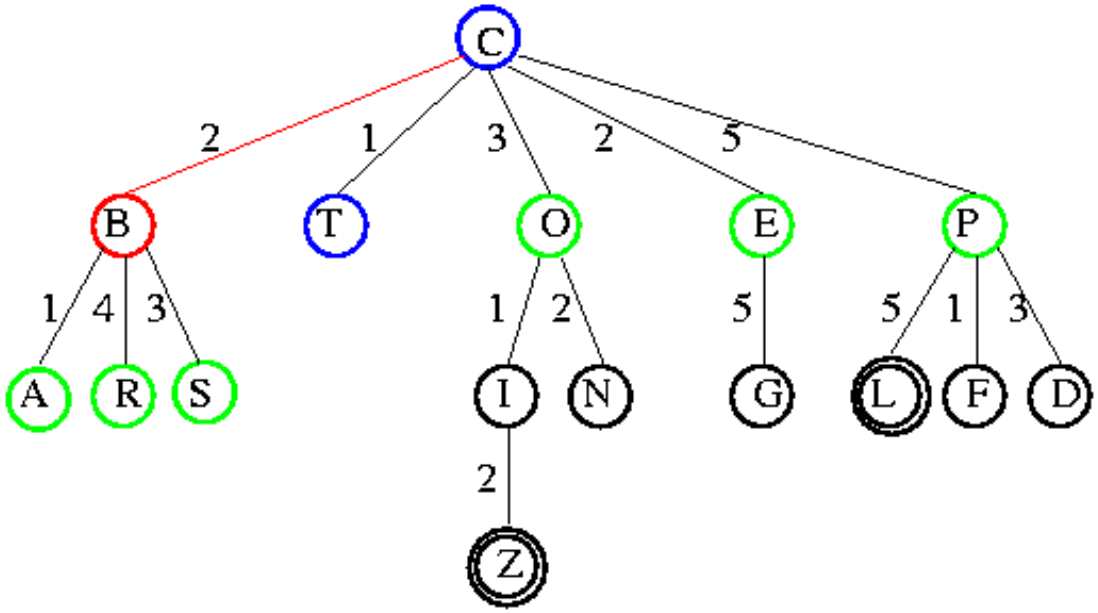
Given a heuristic that calculates cost to next node as such:



Resolve ties alphabetically and based on depth.

Example
Greedy best-first search

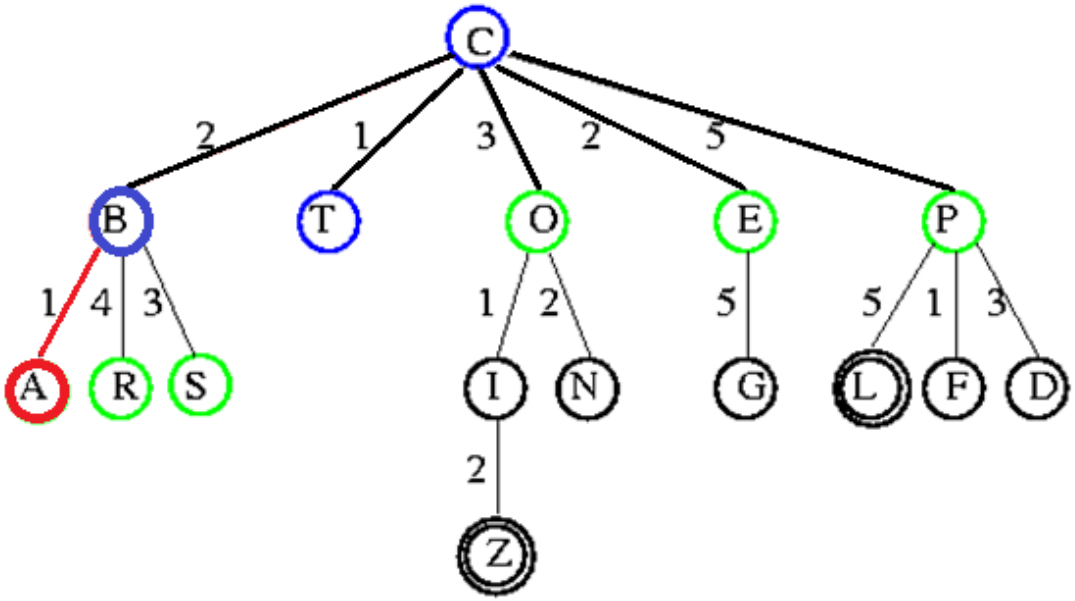
Given a heuristic that calculates cost to next node as such:



Resolve ties alphabetically and based on depth.

Example
Greedy best-first search

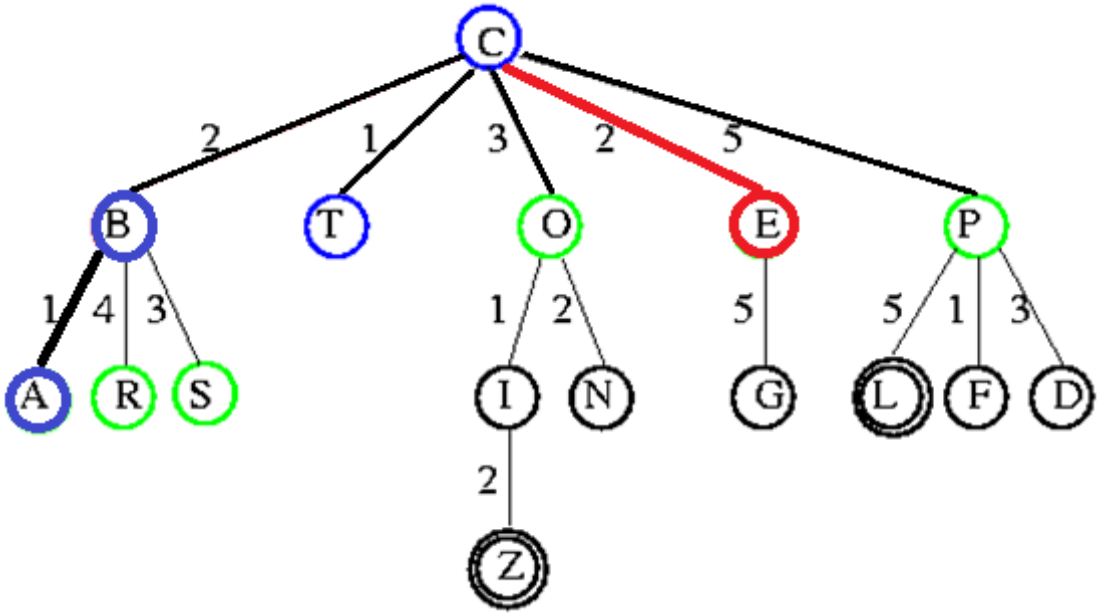
Given a heuristic that calculates cost to next node as such:



Resolve ties alphabetically and based on depth.

Example
Greedy best-first search

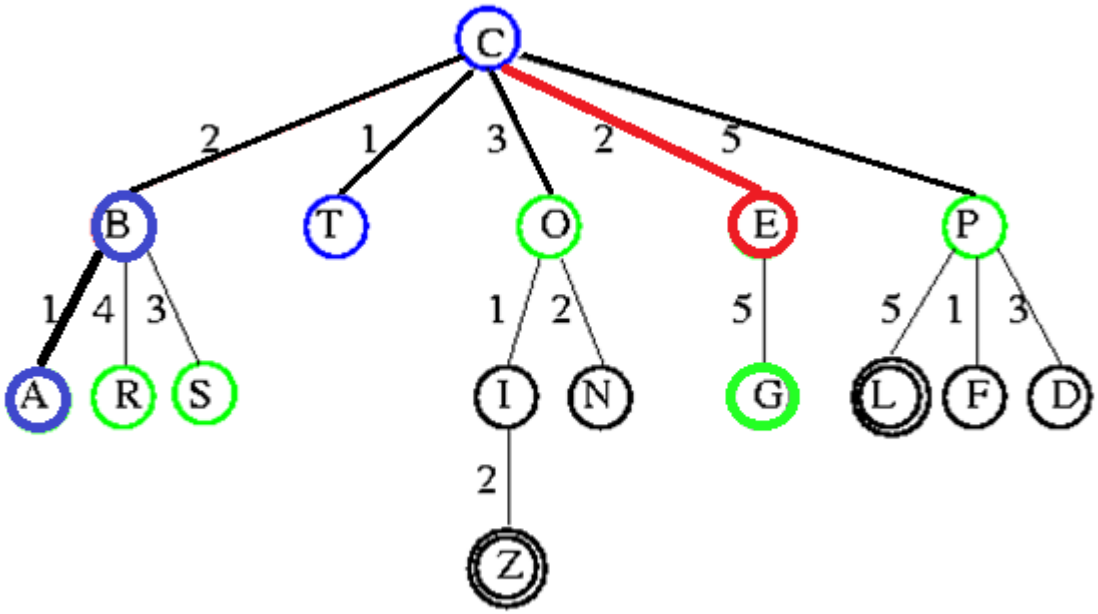
Given a heuristic that calculates cost to next node as such:



Resolve ties alphabetically and based on depth.

Example
Greedy best-first search

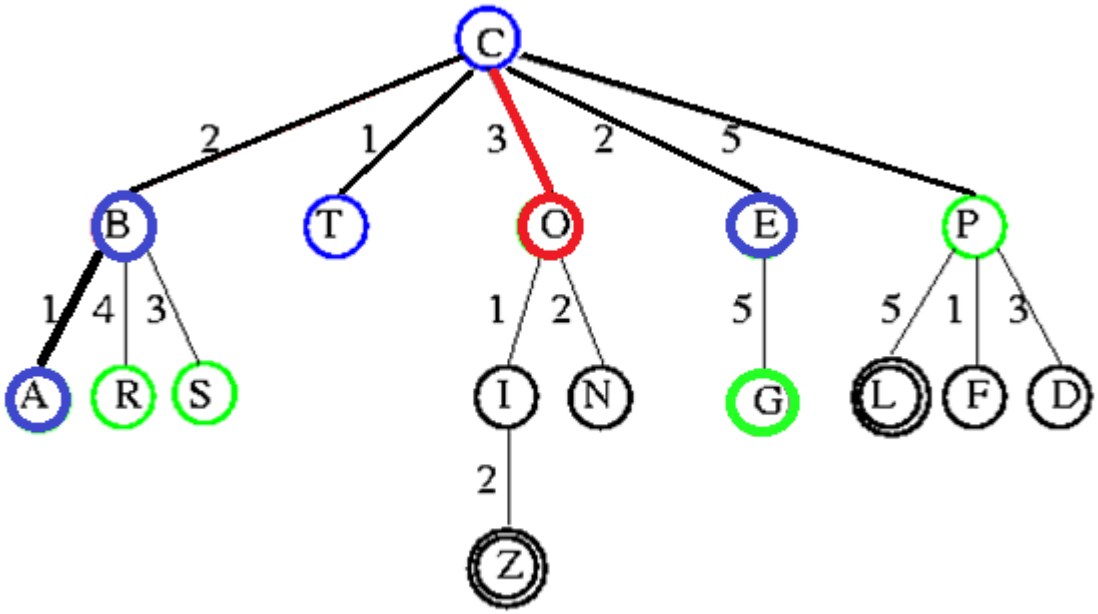
Given a heuristic that calculates cost to next node as such:



Resolve ties alphabetically and based on depth.

Example
Greedy best-first search

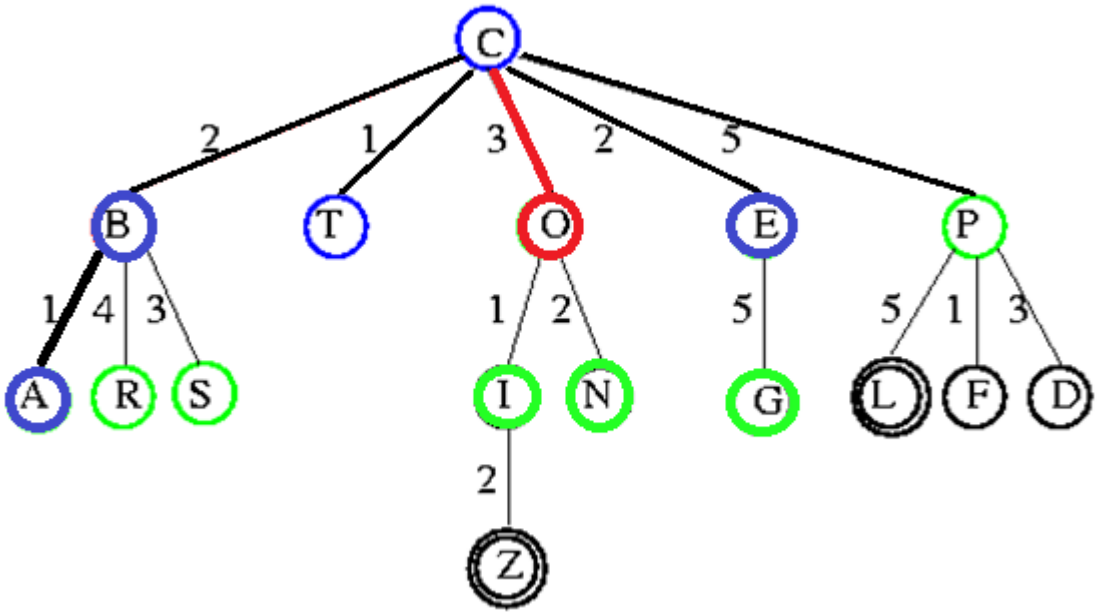
Given a heuristic that calculates cost to next node as such:



Resolve ties alphabetically and based on depth.

Example
Greedy best-first search

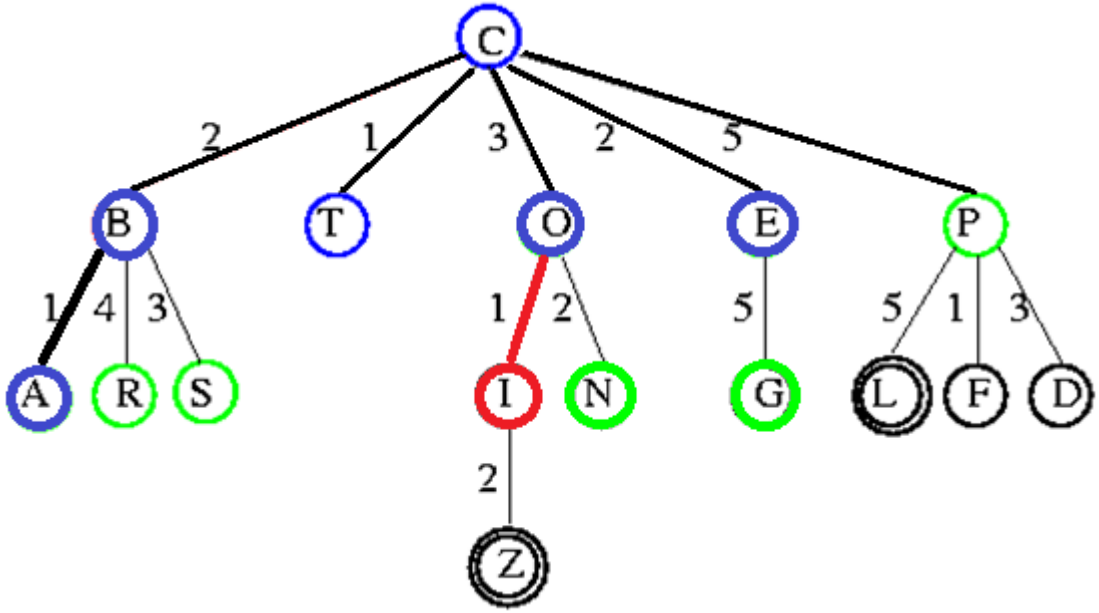
Given a heuristic that calculates cost to next node as such:



Resolve ties alphabetically and based on depth.

Example
Greedy best-first search

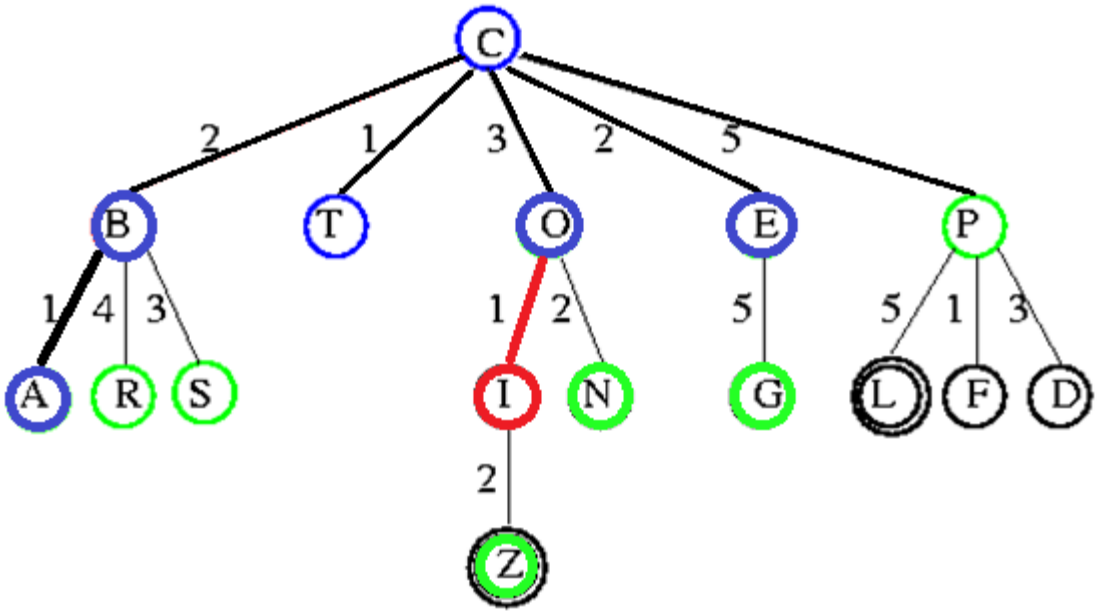
Given a heuristic that calculates cost to next node as such:



Resolve ties alphabetically and based on depth.

Example
Greedy best-first search

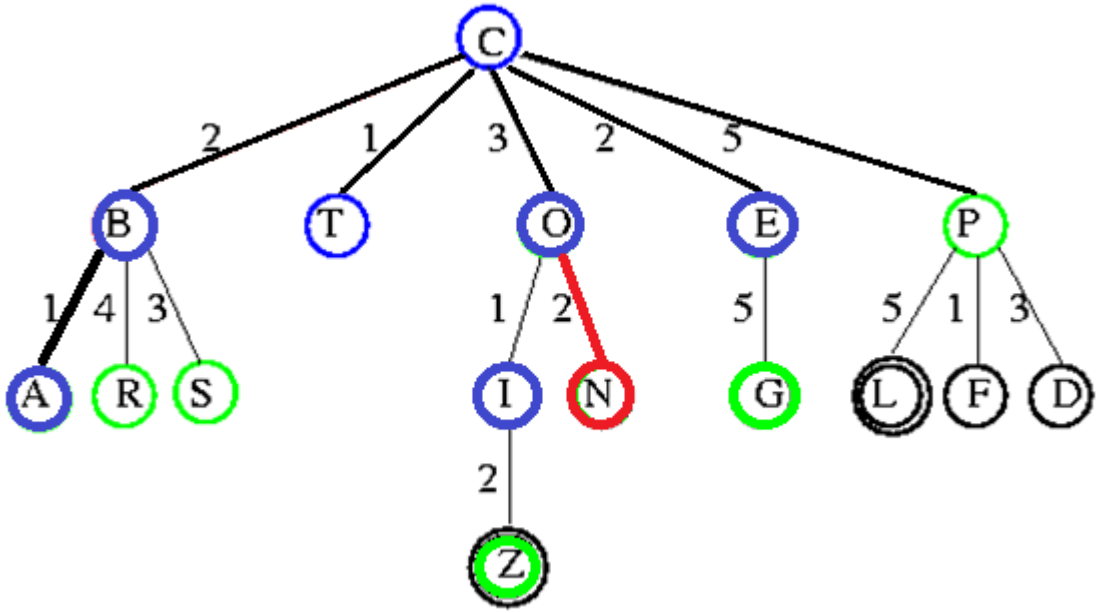
Given a heuristic that calculates cost to next node as such:



Resolve ties alphabetically and based on depth.

Example
Greedy best-first search

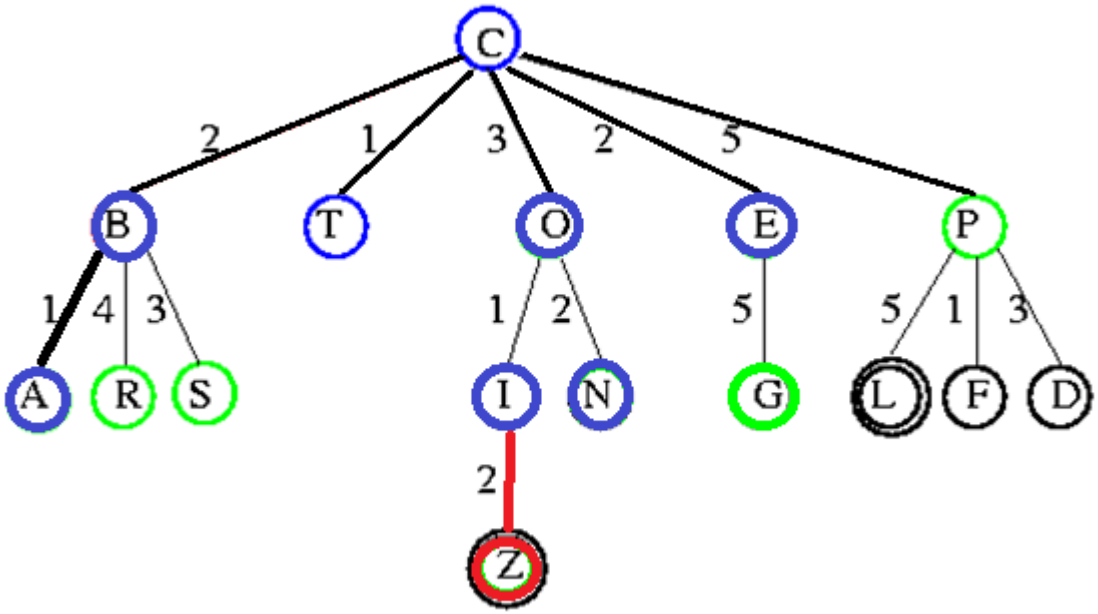
Given a heuristic that calculates cost to next node as such:



Resolve ties alphabetically and based on depth.

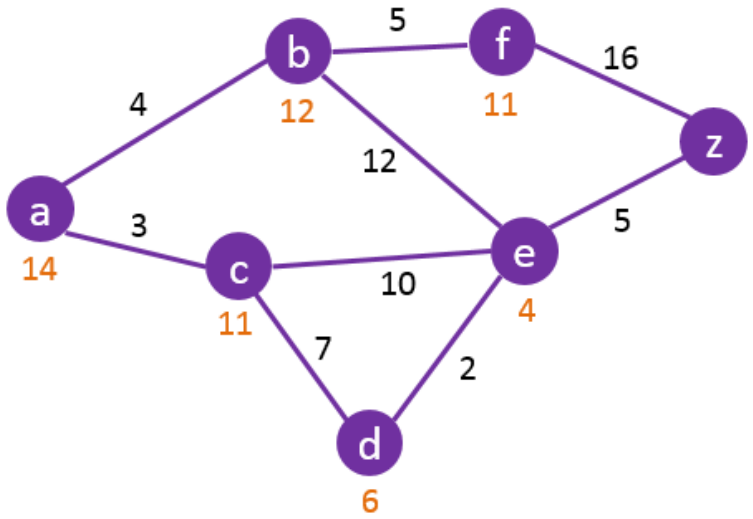
Example
Greedy best-first search

Given a heuristic that calculates cost to next node as such:



Resolve ties alphabetically and based on depth.

Example
A*

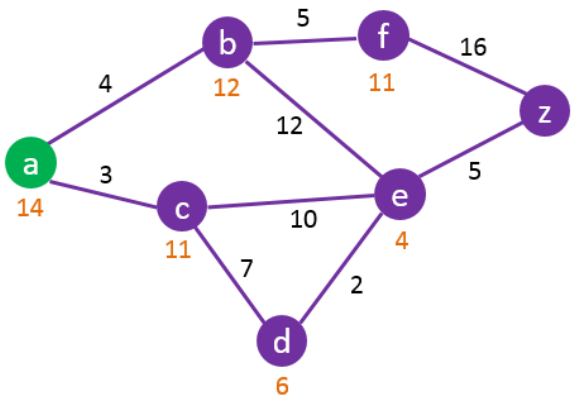


A* Search Algorithm

What is the shortest path to travel from A to Z?

Numbers in orange are the heuristic values, distances in a straight line (as the crow flies) from a node to node Z.

Example
A*

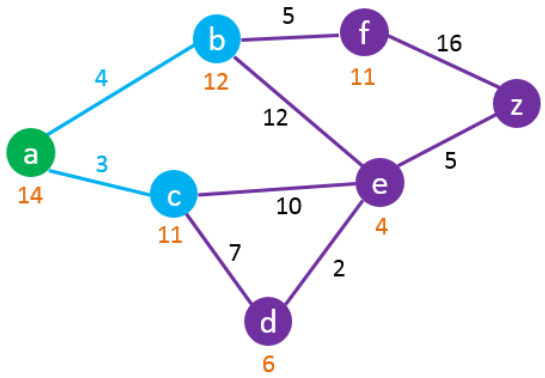


Node	Status	Shortest Distance From A	Heuristic Distance to Z	Total Distance*	Previous Node
A	Current	0	14	14	
B		∞	12		
C		∞	11		
D		∞	6		
E		∞	4		
F		∞	11		
Z		∞	0		

* Total Distance = Shortest Distance from A + Heuristic Distance to Z

Start by setting the starting node (A) as the current node.

Example
A*

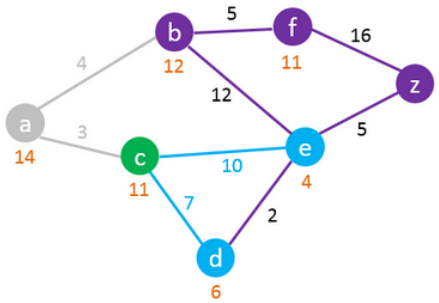


Node	Status	Shortest Distance From A	Heuristic Distance to Z	Total Distance*	Previous Node
A	Current	0	14	14	
B		~~∞~~ 4	12	16	A
C		~~∞~~ 3	11	14	A
D		∞	6		
E		∞	4		
F		∞	11		
Z		∞	0		

* Total Distance = Shortest Distance from A + Heuristic Distance to Z

Check all the nodes connected to A and update their **"Shortest Distance from A"** and set their **"previous node"** to "A". Update their total distance by adding the shortest distance from A and the heuristic distance to Z.

Example
A*



Node	Status	Shortest Distance From A	Heuristic Distance to Z	Total Distance*	Previous Node
A	Visited	0	14	14	
B		4	12	16	A
C	Current	3	11	14	A
D		∞ 3+7=10	6	16	C
E		∞ 3+10=13	4	17	C
F		∞	11		
Z		∞	0		

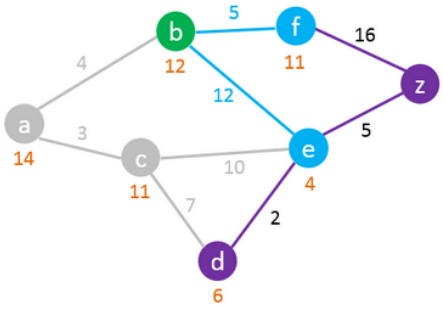
* Total Distance = Shortest Distance from A + Heuristic Distance to Z

Set the current node (A) to **"visited"** and use the unvisited node with the smallest total distance as the **current node** (e.g. in this case: Node C).
Check all unvisited nodes connected to the current node and add the distance from A to C to all distances from the connected nodes. Replace their values only if the new distance is lower than the previous one.

C -> D: 3 + 7 = 10 < ∞ – Change Node D
C -> E: 3 + 10 = 13 < ∞ – Change Node E

The next current node (unvisited node with the shortest total distance) could be either node B or node D. Let's use node B.

Example
A*



Node	Status	Shortest Distance From A	Heuristic Distance to Z	Total Distance*	Previous Node
A	Visited	0	14	14	
B	Current	4	12	16	A
C	Visited	3	11	14	A
D		10	6	16	C
E		13 4+12=~~16~~	4	17	C
F		∞ 4+5=9	11	20	B
Z		∞	0		

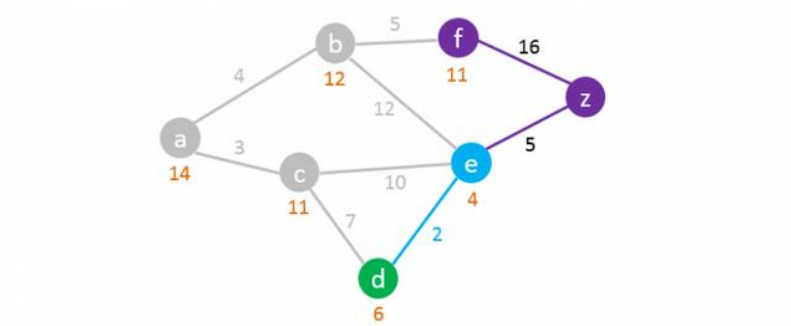
* Total Distance = Shortest Distance from A + Heuristic Distance to Z

Check all unvisited nodes connected to the current node (B) and add the distance from A to B to all distances from the connected nodes. Replace their values only if the new distance is lower than the previous one.

B -> E: 4 + 12 = 16 > 13 – Do not change Node E
B -> F: 4 + 5 = 9 < ∞ – Change Node F

The next current node (unvisited node with the shortest total distance) is D.

Example
A*



Node	Status	Shortest Distance From A	Heuristic Distance to Z	Total Distance*	Previous Node
A	Visited	0	14	14	
B	Visited	4	12	16	A
C	Visited	3	11	14	A
D	Current	10	6	16	C
E		~~13~~ 10+2=12	4	16	D
F		9	11	20	B
Z		∞	0		

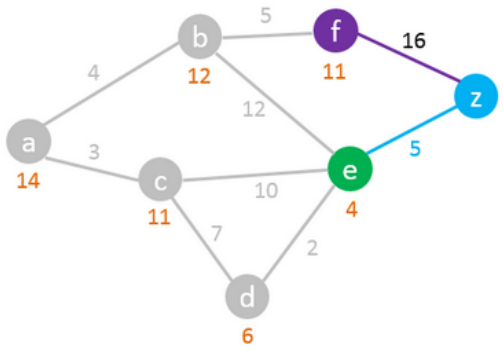
* Total Distance = Shortest Distance from A + Heuristic Distance to Z

Check all unvisited nodes connected to the current node (D) and add the distance from A to D to all distances from the connected nodes. Replace their values only if the new distance is lower than the previous one.

D -> E: 10 + 2 = 12 < 13 – Change Node E

The next current node (unvisited node with the shortest total distance) is E.

Example
A*



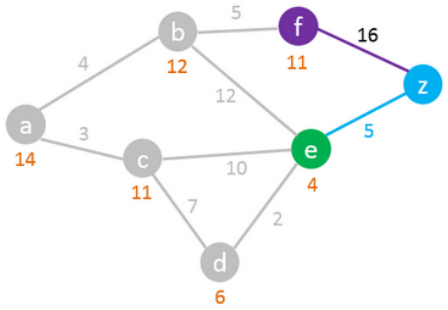
Node	Status	Shortest Distance From A	Heuristic Distance to Z	Total Distance*	Previous Node
A	Visited	0	14	14	
B	Visited	4	12	16	A
C	Visited	3	11	14	A
D	Visited	10	6	16	C
E	Current	12	4	16	D
F		9	11	20	B
Z		∞ 12+5=17	0	17	E

* Total Distance = Shortest Distance from A + Heuristic Distance to Z

Check all unvisited nodes connected to the current node (E) and add the distance from A to E to all distances from the connected nodes. Replace their values only if the new distance is lower than the previous one.

E -> Z: 12 + 5 = 17 < ∞ – Change Node Z

Example
A*



Node	Status	Shortest Distance From A	Heuristic Distance to Z	Total Distance*	Previous Node
A	Visited	0	14	14	
B	Visited	4	12	16	A
C	Visited	3	11	14	A
D	Visited	10	6	16	C
E	Visited	12	4	16	D
F		9	11	20	B
Z	Current	17	0	17	E

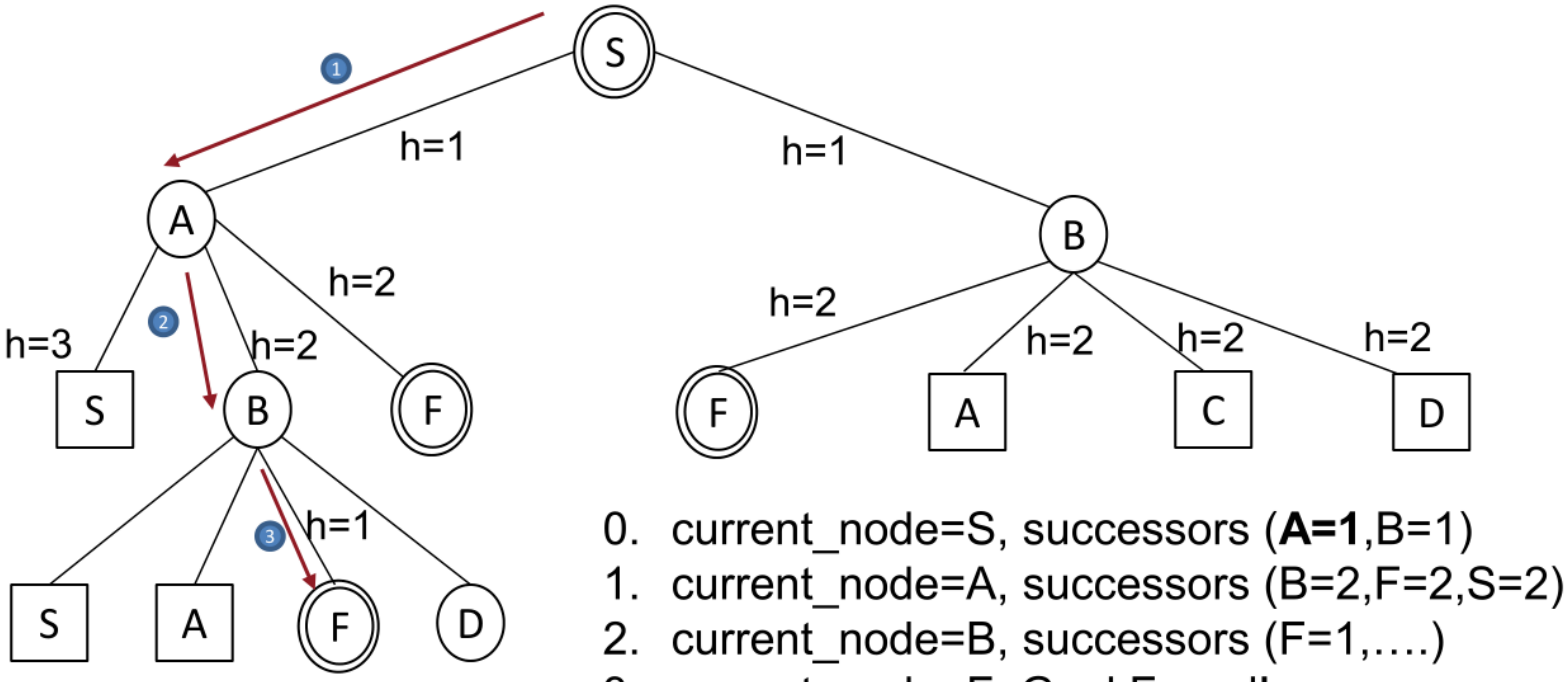
* Total Distance = Shortest Distance from A + Heuristic Distance to Z

We found a path from A to Z, but is it the shortest one?

Check all unvisited nodes. In this example, there is only one unvisited node (F). However its total distance (20) is already greater than the distance we have from A to Z (17) so there is no need to visit node F as it will not lead to a shorter path.

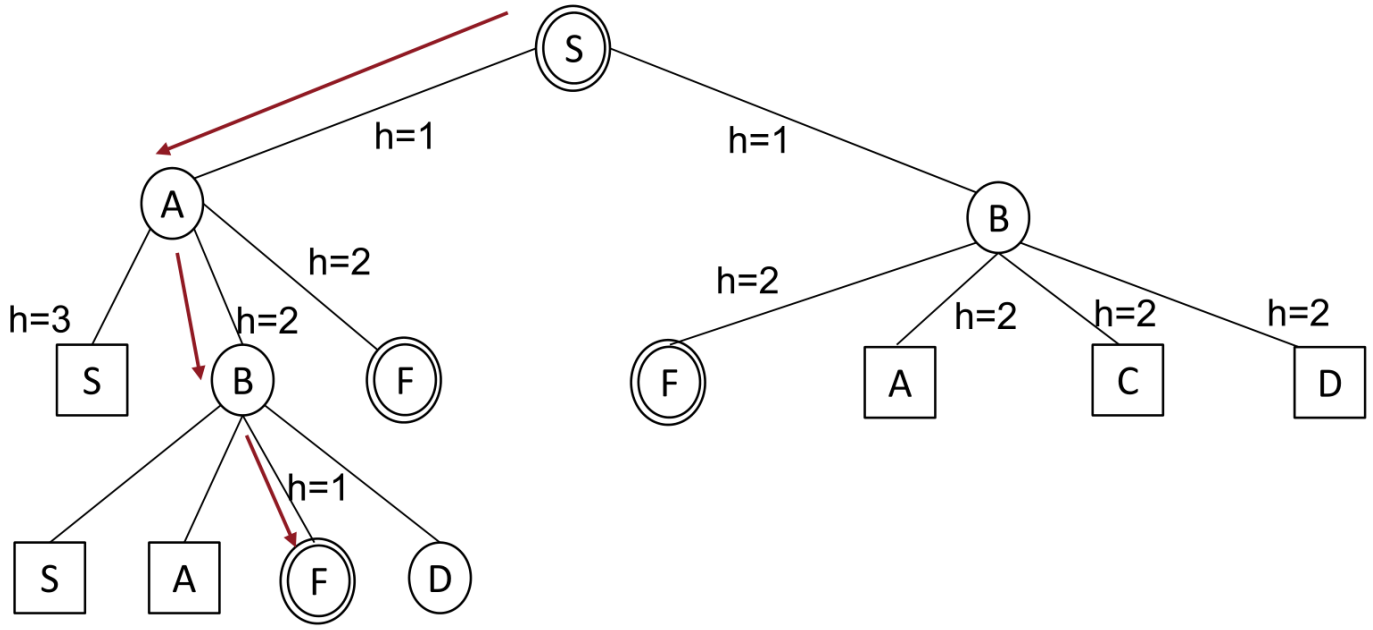
We found the shortest path from A to Z.
Read the path from Z to A using the previous node column:
Z > E > D > C > A
So the Shortest Path is:
A – C – D – E – Z with a length of 17

Example
Hill Climbing



0. current_node=S, successors (**A=1**,B=1)
1. current_node=A, successors (B=2,F=2,S=2)
2. current_node=B, successors (F=1,….)
3. current_node=F: Goal Found!

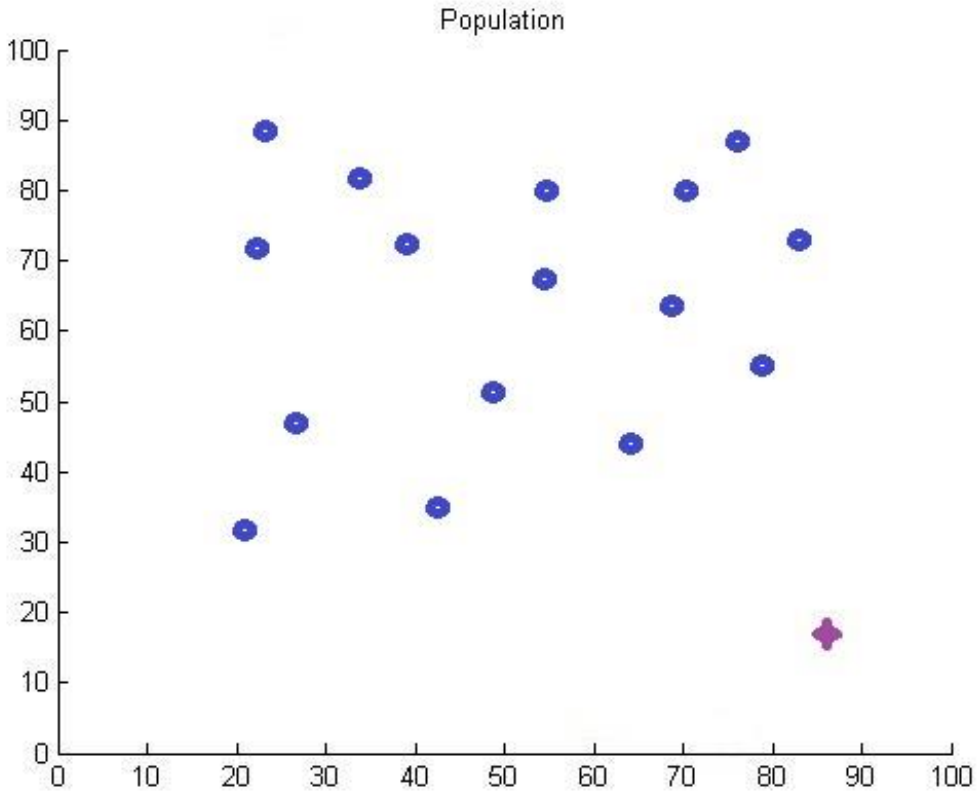
Example
Hill Climbing



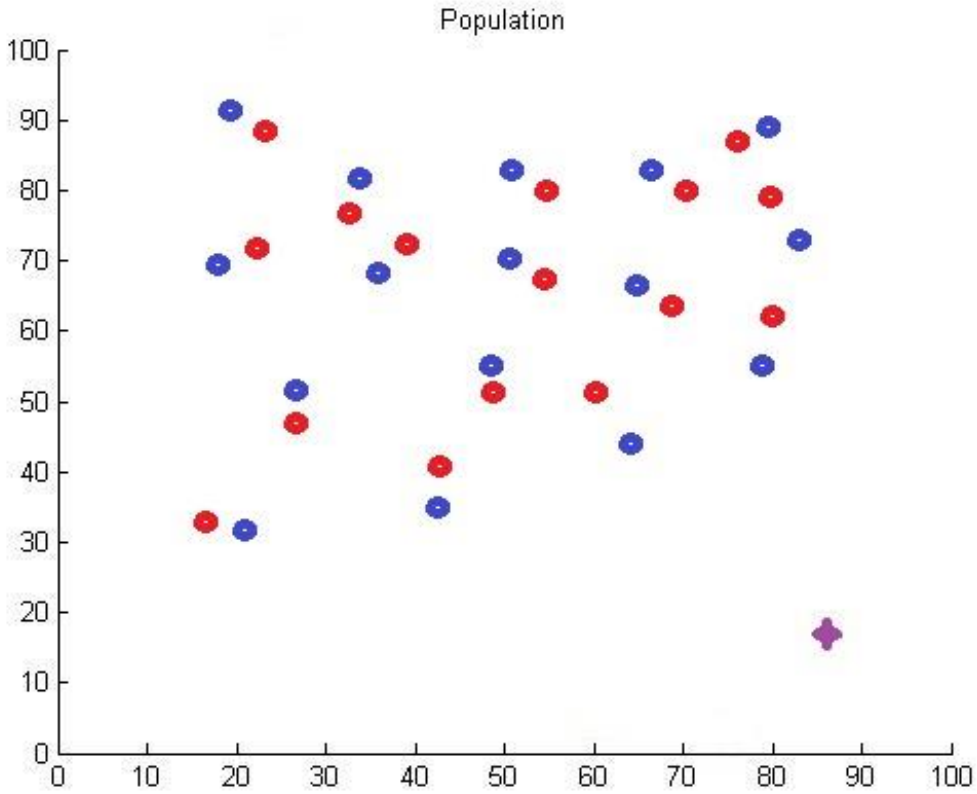
Result is: S->A->B->F!

Not optimal, if at step 1 h(S)=2 we would have completed without funding a solution

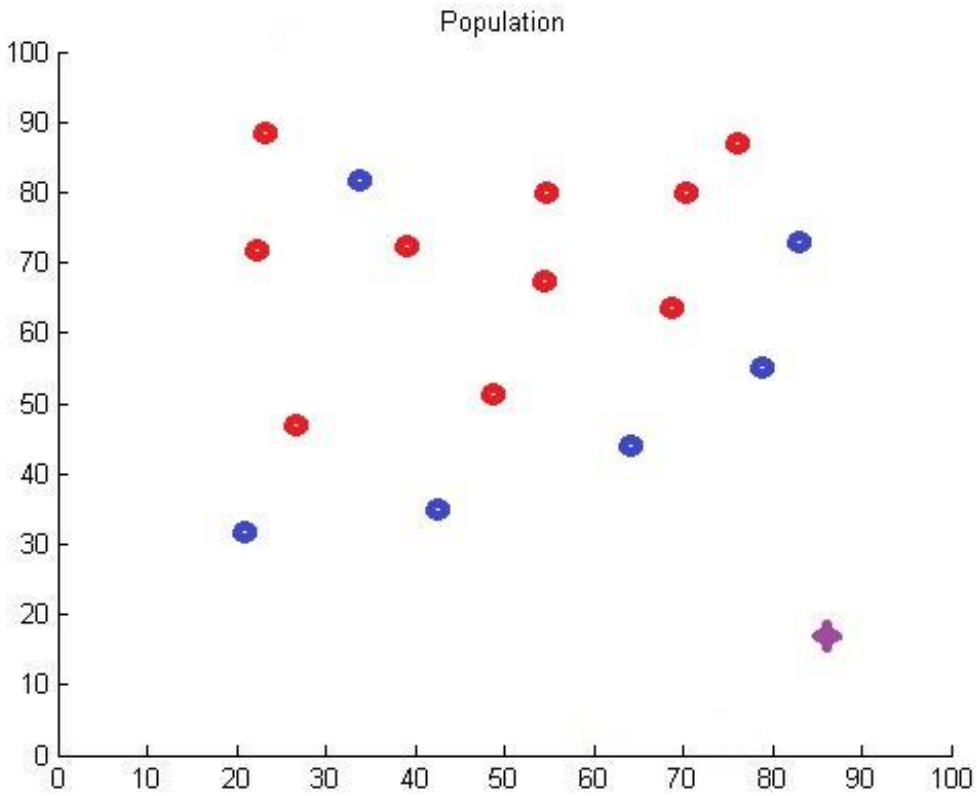
Example
Simulated Annealing



Example
Simulated Annealing



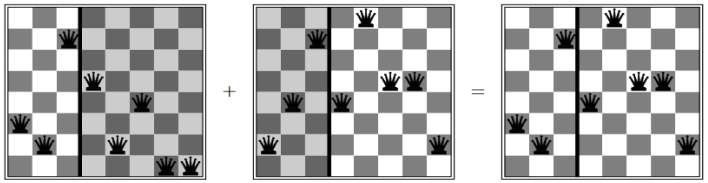
Example
Simulated Annealing

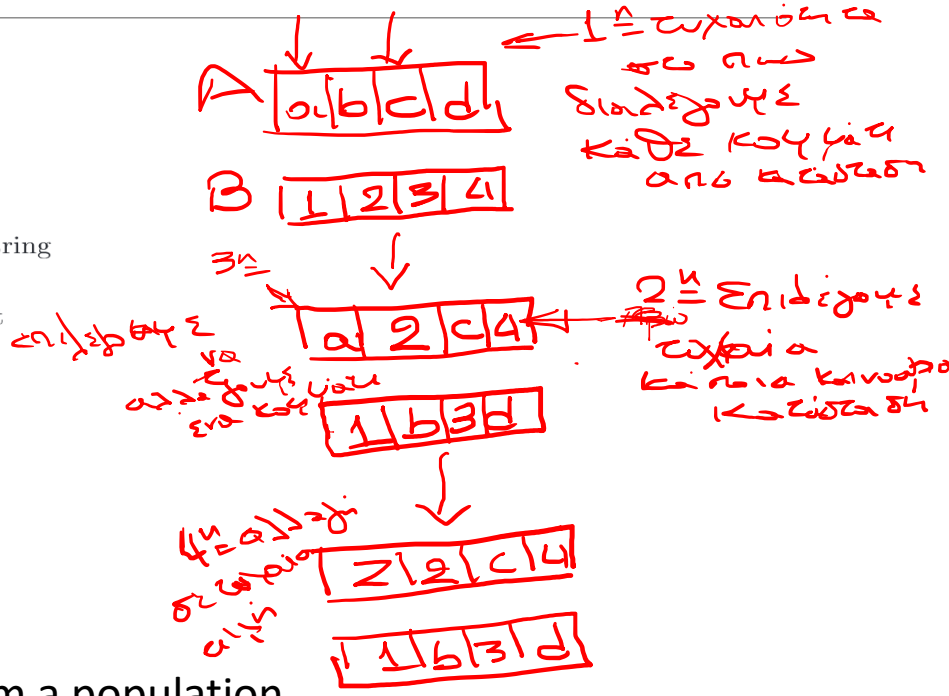


Example
Genetic Algorithms

Genetic algorithms

- Basic concept: combines two (parent) states
- Mechanism:
  Starts with k random states (population)
  Encodes individuals in a compact representation (e.g., a string in an alphabet)
  Combines partial solutions to generate new solutions (next generation)





Stochasticity added at:

a) Randomly choose 2 “parents” from a population
b) Split parents at random points
c) Randomly choose an offspring
d) Perform or not mutation based on a probability
e) Mutation can be based on a heuristic or simply a random value

Exercise 4.2

The **heuristic path algorithm** is a best-first search in which the objective function is $f(n) = (2-w) \cdot g(n) + w \cdot h(n)$.

1. For what values of w is this algorithm guaranteed to be optimal? (assume h is admissible)
2. What kind of search does this perform when:
   a) w = 0?
   b) w = 1?
   c) w = 2?

Exercise 4.2

For what values of w is this algorithm guaranteed to be optimal? (assume h is admissible)

$$f(n) = (2-w) \cdot g(n) + w \cdot h(n) \text{ ?}$$

$$f(n) = (2-w) \cdot \left(g(n) + \frac{w}{2-w} \cdot h(n)\right)$$

$$\frac{w}{2-w} \leq 1 \text{ (h(n) admissible)} \Leftrightarrow$$

$$w \leq 2-w \Leftrightarrow$$

$$w \leq 1$$

For $w \leq 1$, $\frac{w}{2-w} \cdot h(n)$ is always less than h(n) hence admissible, provided h(n) is itself admissible

Exercise 4.2

What kind of search does this perform when:

a) w = 0?
b) w = 1?
c) w = 2?

Exercise 4.2

What kind of search does this perform when:

a) w = 0? $f(n) = 2 \cdot g(n)$ ➔ Uniform-cost search
b) w = 1?
c) w = 2?

Exercise 4.2

What kind of search does this perform when:

a) w = 0? $f(n) = 2 \cdot g(n)$ ➔ Uniform-cost search
b) w = 1? $f(n) = g(n) + h(n)$ ➔ A* search
c) w = 2?

Exercise 4.2

What kind of search does this perform when:

a) $w = 0$? $f(n) = 2 \cdot g(n)$ ➔ Uniform-cost search
b) $w = 1$? $f(n) = g(n) + h(n)$ ➔ A* search
c) $w = 2$? $f(n) = 2 \cdot h(n)$ ➔ Greedy best-first search

Exercise 4.3

Prove each of the following statements:

1. Breadth-first search, depth-first search, and uniform-cost search are special cases of best-first search.
2. Uniform-cost search is a special case of A* search.

Exercise 4.3

Breadth-first search, depth-first search, and uniform-cost search are special cases of best-first search.

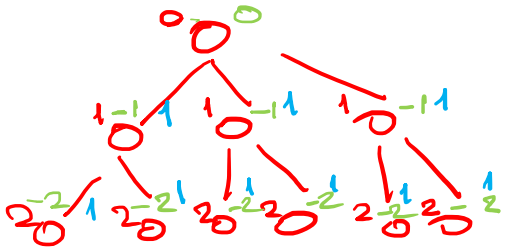
Breadth-first search: f(n) = ?

Depth-first search: f(n) = ?

Uniform-cost search: f(n) = ?

Exercise 4.3

Breadth-first search, depth-first search, and uniform-cost search are special cases of best-first search.



Breadth-first search: f(n) = depth(n)

Depth-first search: f(n) = -depth(n)

Uniform-cost search: f(n) = g(n)

Exercise 4.3

Uniform-cost search is a special case of A search.*

f(n) = ?

Exercise 4.3

Uniform-cost search is a special case of A search.*

f(n) = g(n) (no heuristic function)

Exercise 4.8

The traveling salesman problem (TSP) can be solved via the minimum spanning tree (MST) heuristic, which is used to estimate the cost of completing a tour, given that a partial tour has already been constructed. The MST cost of a set of cities is the smallest sum of the link costs of any tree that connects all cities.

1. Show how this heuristic can be derived from a relaxed version of the TSP.
2. Show that the MST heuristic dominates straight-line distance.

Exercise 4.8

Show how this heuristic can be derived from a relaxed version of the TSP.

An MST is a spanning tree with the minimal total length. A spanning tree connects all vertices of a graph, but is not necessarily a path which passes through each vertex.

It is admissible, as it is always shorter or equal to the to the optimal path.

The TSP problem is to find a minimal (total length) path through the cities that forms a closed loop. MST is a relaxed version of that because it asks for a minimal (total length) graph that need not be a closed loop—it can be any fully-connected graph. As a heuristic, MST is admissible—it is always shorter than or equal to a closed loop.

Exercise 4.8

Show that the MST heuristic dominates straight-line distance (SLD).

The straight-line distance back to the start city is a rather weak heuristic—it vastly underestimates when there are many cities. In the later stage of a search when there are only a few cities left it is not so bad. To say that MST dominates straight-line distance is to say that MST always gives a higher value. This is obviously true because a MST that includes the goal node and the current node must either be the straight line between them, or it must include two or more lines that add up to more. (This all assumes the triangle inequality.)

This is true because the MST is either a straight line to the goal (equal with SLD) or includes two or more lines (higher than SLD), assuming the triangle inequality holds.