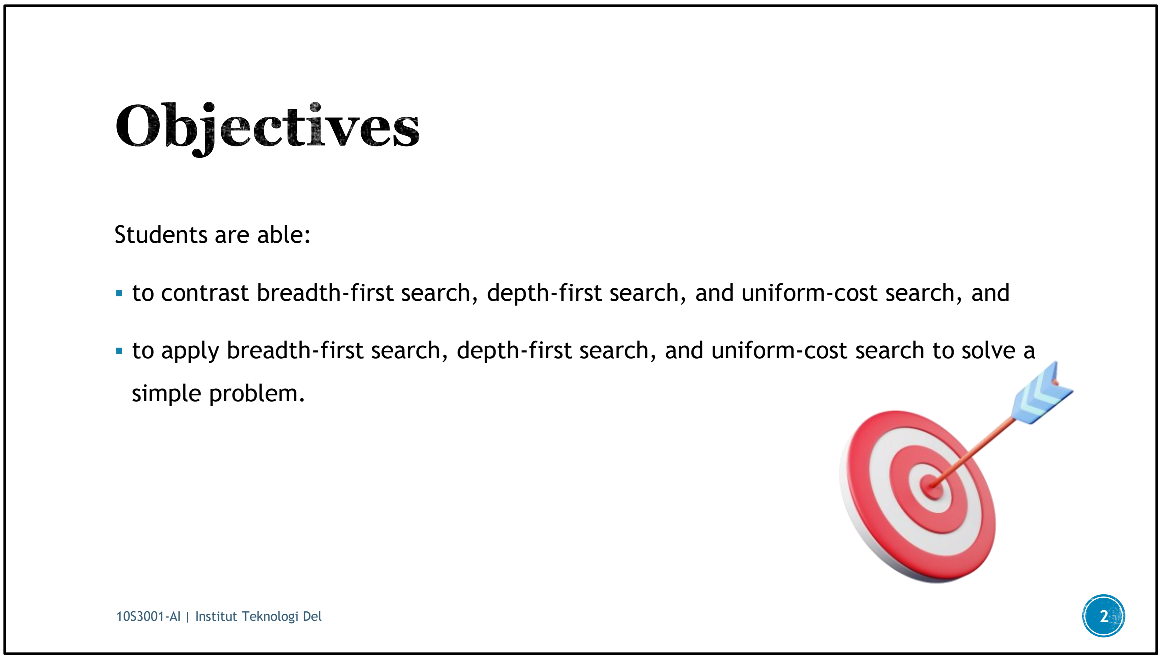
Uninformed Search

10S3001 – Artificial Intelligence

Samuel I. G. Situmeang

Faculty of Informatics and Electrical Engineering



Objectives

Students are able:

- to contrast breadth-first search, depth-first search, and uniform-cost search, and
- to apply breadth-first search, depth-first search, and uniform-cost search to solve a simple problem.

Siswa mampu:

- untuk membedakan *breadth-first search*, *depth-first search*, dan *uniform-cost search*, serta
- untuk menerapkan *breadth-first search*, *depth-first search*, dan *uniform-cost search* untuk menyelesaikan persoalan sederhana.

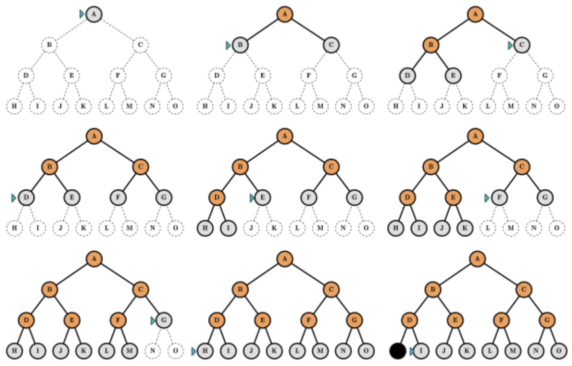
Uninformed Search

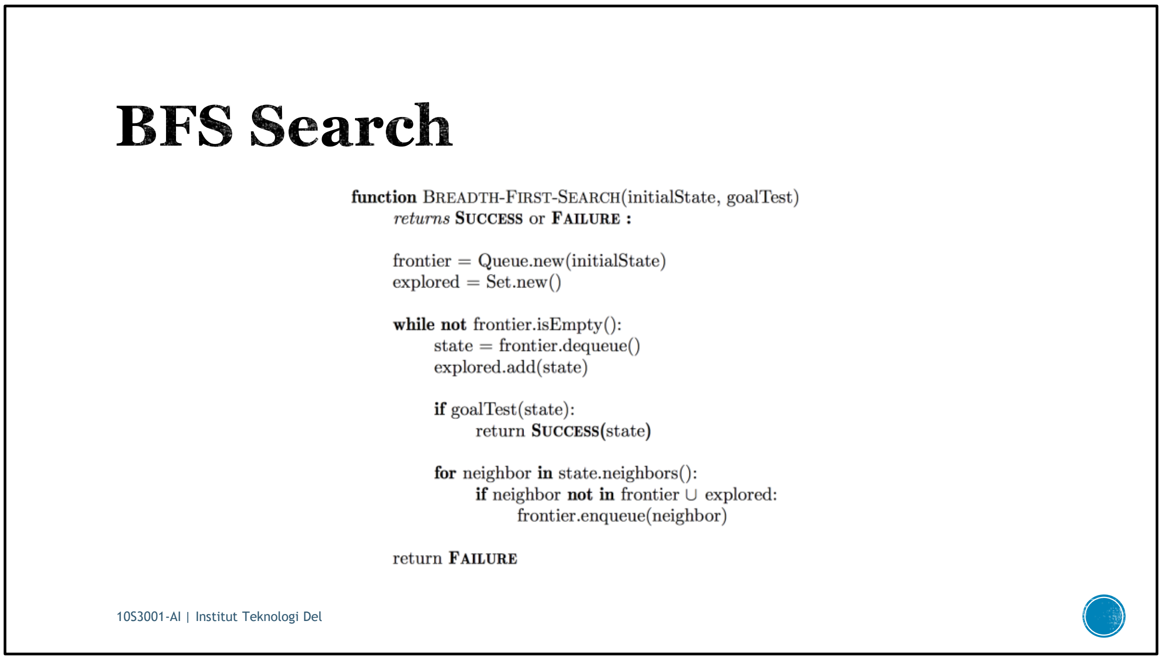
- **Use no domain knowledge!**

- **Strategies:**
  1. Breadth-first search (BFS): Expand shallowest node
  2. Depth-first search (DFS): Expand deepest node
  3. Uniform-cost search (UCS): Expand least cost node

Breadth-first Search (BFS)

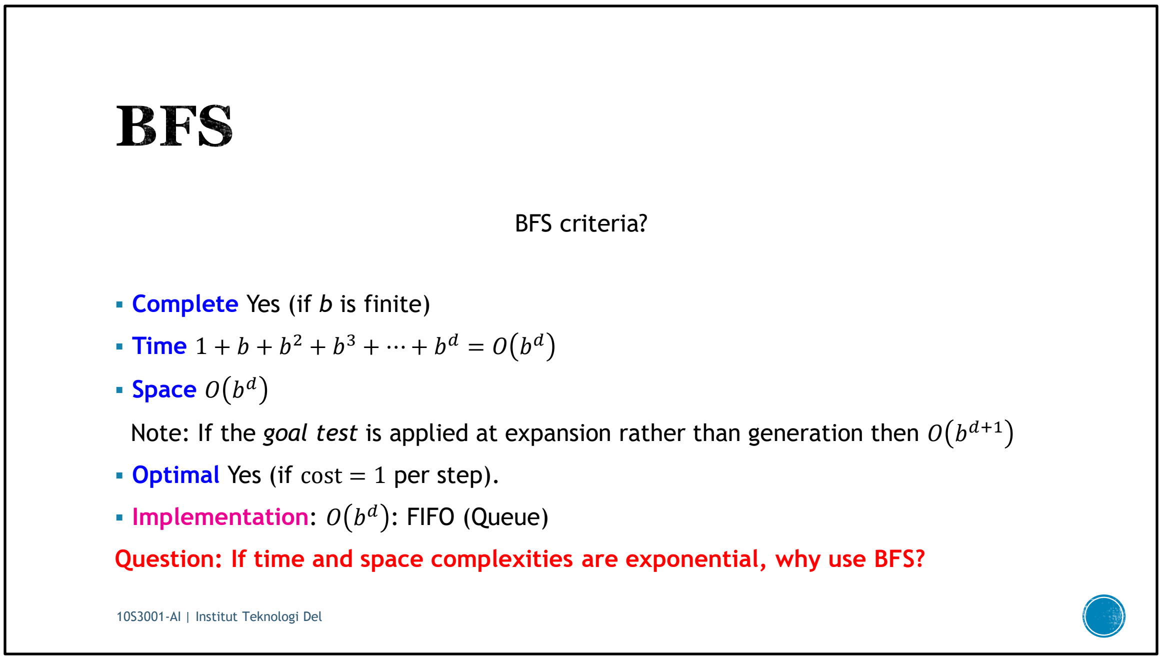
- BFS: Expand shallowest first.





BFS Search

```
function BREADTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Queue.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.dequeue()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.enqueue(neighbor)

    return FAILURE
```



BFS

BFS criteria?

- **Complete** Yes (if *b* is finite)
- **Time** $1 + b + b^2 + b^3 + \cdots + b^d = O(b^d)$
- **Space** $O(b^d)$

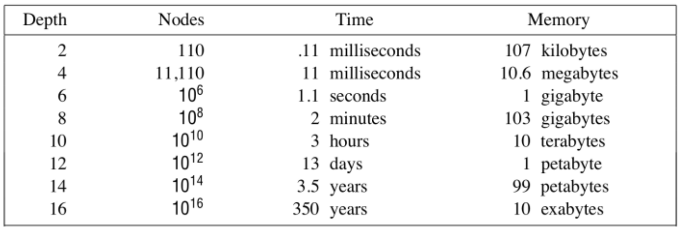
  Note: If the *goal test* is applied at expansion rather than generation then $O(b^{d+1})$
- **Optimal** Yes (if $\text{cost} = 1$ per step).
- **Implementation**: $O(b^d)$: FIFO (Queue)

Question: If time and space complexities are exponential, why use BFS?

If B is finite, which means that the maximum branching factor is finite, we don't have an infinite space to explore, the process or the search strategy is complete because we are going to find the solution, if it exists.
the time and space complexity can be measured in terms of number of nodes.
It depends on the problem. For some problems, the solution is so shallow that it is much better to do a BFS, you get it sooner and you won’t get really this exponential complexity in time and space.

BFS

How bad is BFS?



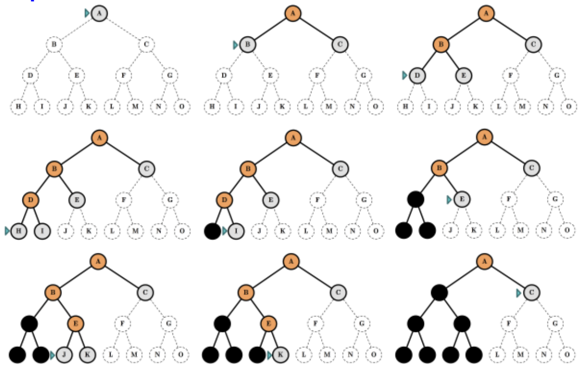
Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

- Time and Memory requirements for breadth-first search for a branching factor $b = 10$; 1 million nodes per second; 1,000 bytes per node.

Memory requirement + exponential time complexity are the biggest handicaps of BFS!

DFS

- DFS: Expand deepest first.



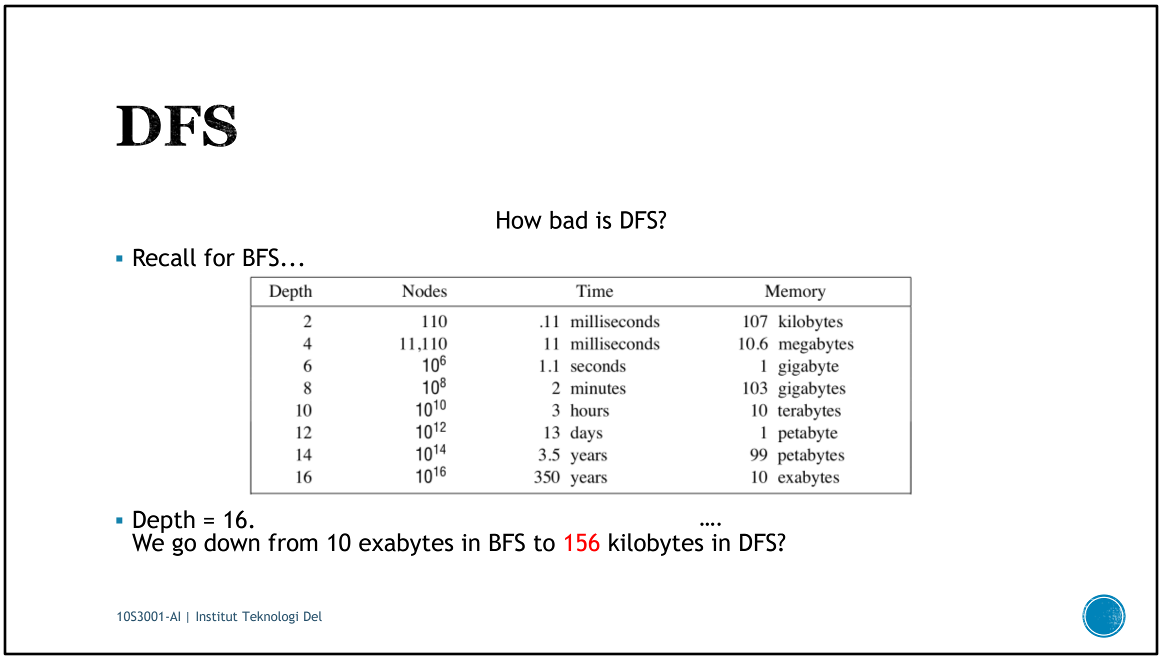
DFS Search

```
function DEPTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Stack.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.pop()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier $\cup$ explored:
                frontier.push(neighbor)

    return FAILURE
```

DFS

DFS criteria?

- **Complete** No: fails in infinite-depth spaces, spaces with loops
  Modify to avoid repeated states along path.
  $\Longrightarrow$ complete in finite spaces
- **Time** $O(b^m)$: $1 + b + b^2 + b^3 + \cdots + b^m = O(b^m)$
  bad if m is much larger than d
  but if solutions are dense, may be much faster than BFS.
- **Space** $O(bm)$ linear space complexity! (needs to store only a single path from the root to a leaf node, **along with the remaining unexpanded sibling nodes for each node on the path, hence the m factor.**)
- **Optimal** No
- **Implementation**: fringe: LIFO (Stack)



DFS

How bad is DFS?

- Recall for BFS...

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

- Depth = 16. ....
  We go down from 10 exabytes in BFS to 156 kilobytes in DFS?

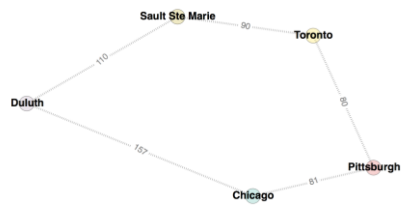
Uniform-cost Search

- The arcs in the search graph may have weights (different cost attached). How to leverage this information?
- BFS will find the shortest path which may be costly.
- We want the cheapest not shallowest solution.
- Modify BFS: Prioritize by cost not depth → **Expand node n with the lowest path cost $g(n)$.**
- Explores increasing costs.

UCS

```
function UNIFORM-COST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost $f(n) = g(n)$ */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

Uniform-cost Search



- Go from Chicago to Sault Ste Marie. Using BFS, we would find Chicago-Duluth-Sault Ste Marie. However, using UCS, we would find Chicago-Pittsburgh-Toronto-Sault Ste Marie, which is actually the shortest path!

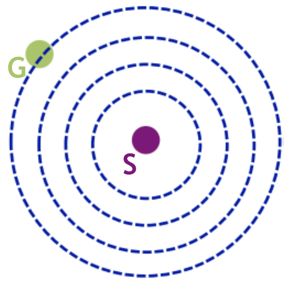
Uniform-cost Search

UCS criteria?

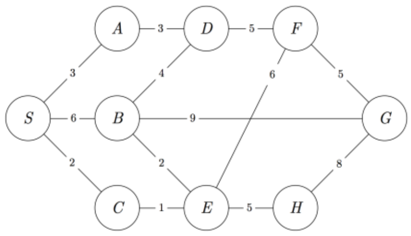
- **Complete** Yes, if solution has a finite cost.
- **Time**
  - Suppose C^*: cost of the optimal solution
  - Every action costs at least ϵ (bound on the cost)
  - The effective depth is roughly C^*/ϵ (how deep the *cheapest* solution could be).
  - $O\left(b^{C^*/\epsilon}\right)$
- **Space** # of nodes with $g \leq$ cost of optimal solution, $O\left(b^{C^*/\epsilon}\right)$
- **Optimal** Yes
- **Implementation**: fringe = queue ordered by path cost $g(n)$, lowest first = Heap!

Uniform-cost Search

- While complete and optimal, UCS explores the space in every direction because no information is provided about the goal!

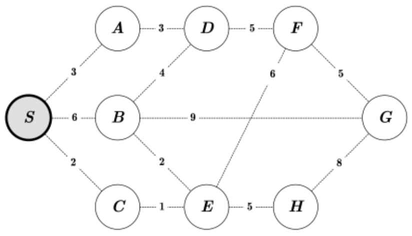


Example: Toy Example



- Question: What is the **order of visits of the nodes** and the **path** returned by BFS, DFS and UCS?

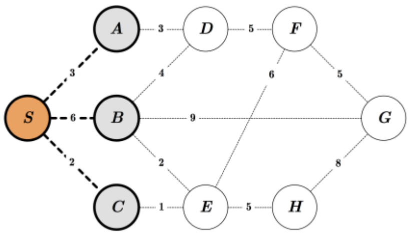
Example: Toy Example with BFS



Queue:

Order of Visit:

Example: Toy Example with BFS



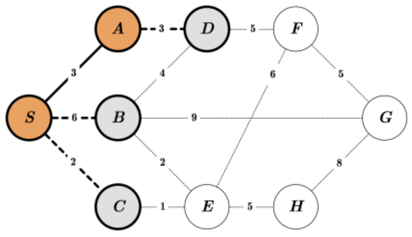
Queue:

S	*A*	*B*	*C*

Order of Visit:

S

Example: Toy Example with BFS



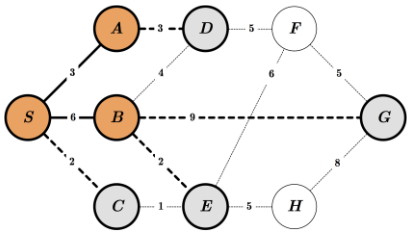
Queue:

S	A	B	C	D

Order of Visit:

S A

Example: Toy Example with BFS



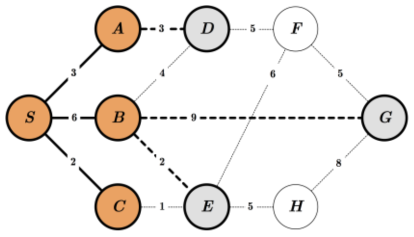
Queue:

S	*A*	*B*	*C*	*D*	*E*	*G*

Order of Visit:

S *A* *B*

Example: Toy Example with BFS



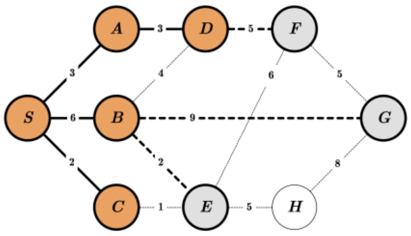
Queue:

S	A	B	C	D	E	G

Order of Visit:

S A B C

Example: Toy Example with BFS



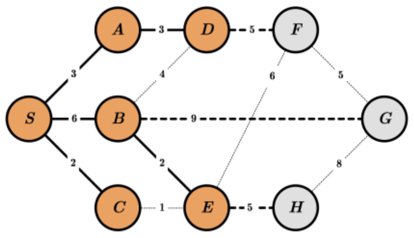
Queue:

S	*A*	*B*	*C*	*D*	*E*	*G*	*F*

Order of Visit:

S *A* *B* *C* *D*

Example: Toy Example with BFS



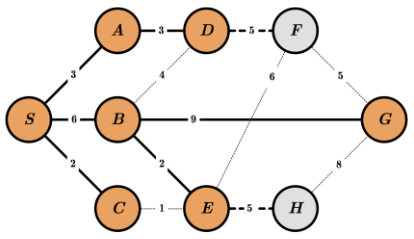
Queue:

S	*A*	*B*	*C*	*D*	*E*	*G*	*F*	*H*

Order of Visit:

S *A* *B* *C* *D* *E*

Example: Toy Example with BFS



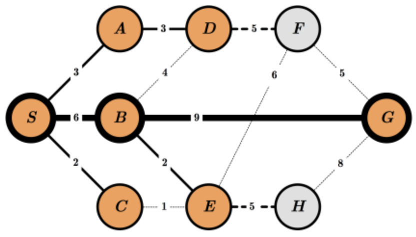
Queue:

S	A	B	C	D	E	G	F	H

Order of Visit:

$S \quad A \quad B \quad C \quad D \quad E \quad G$

Example: Toy Example with BFS



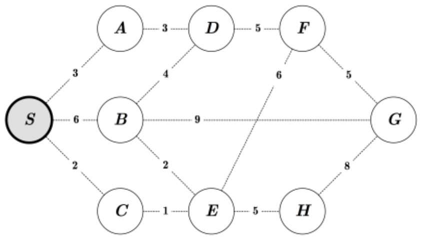
Queue:

S	A	B	C	D	E	G	F	H

Order of Visit:

$S \quad A \quad B \quad C \quad D \quad E \quad G$

Example: Toy Example with DFS

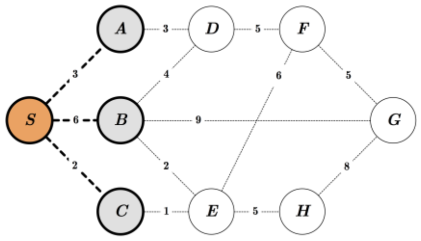


Stack:

$\boxed{S}$

Order of Visit:

Example: Toy Example with DFS



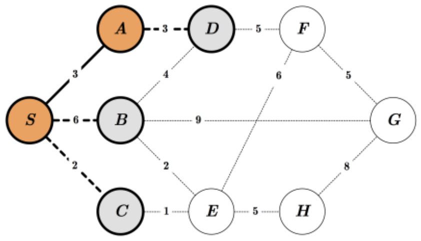
Stack:

S	*C*	*B*	*A*

Order of Visit:

S

Example: Toy Example with DFS



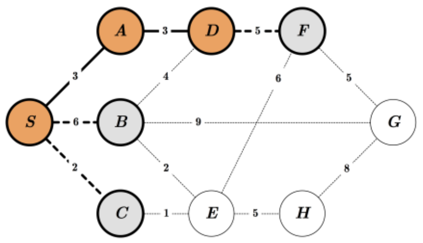
Stack:

S	*C*	*B*	*A*	*D*

Order of Visit:

S *A*

Example: Toy Example with DFS



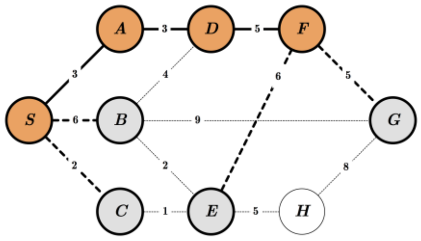
Stack:

S	C	B	A	D	F

Order of Visit:

S A D

Example: Toy Example with DFS



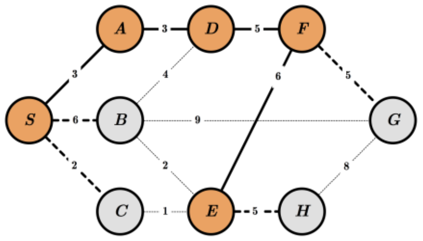
Stack:

S	*C*	*B*	*A*	*D*	*F*	*G*	*E*

Order of Visit:

S *A* *D* *F*

Example: Toy Example with DFS



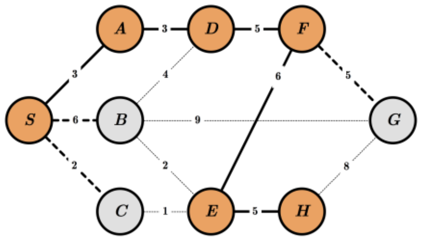
Stack:

S	*C*	*B*	*A*	*D*	*F*	*G*	*E*	*H*

Order of Visit:

S *A* *D* *F* *E*

Example: Toy Example with DFS



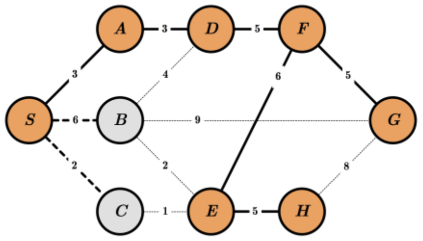
Stack:

S	*C*	*B*	*A*	*D*	*F*	*G*	*E*	*H*

Order of Visit:

S *A* *D* *F* *E* *H*

Example: Toy Example with DFS



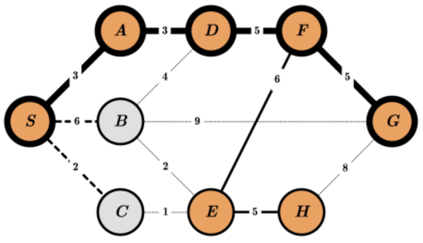
Stack:

S	*C*	*B*	*A*	*D*	*F*	*G*	*E*	*H*

Order of Visit:

S *A* *D* *F* *E* *H* *G*

Example: Toy Example with DFS



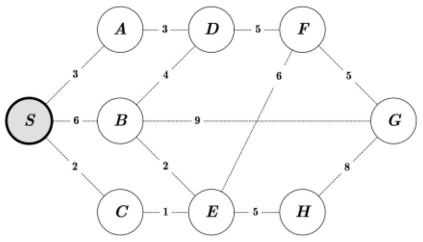
Stack:

S	***C***	***B***	*A*	*D*	*F*	*G*	*E*	*H*

Order of Visit:

S ***A*** ***D*** ***F*** ***E*** ***H*** ***G***

Example: Toy Example with UCS

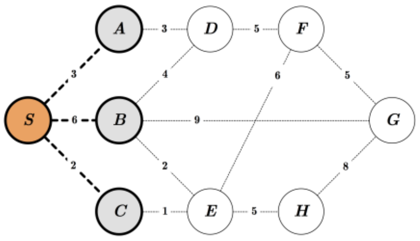


Priority Queue:

S_0

Order of Visit:

Example: Toy Example with UCS



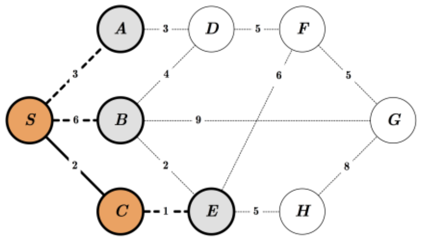
Priority Queue:

S_0	C_2	A_3	B_6

Order of Visit:

S

Example: Toy Example with UCS



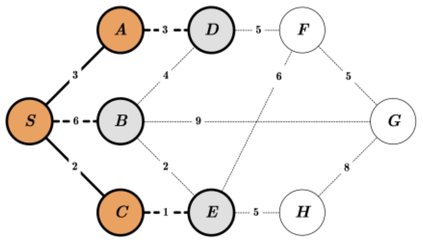
Priority Queue:

S_0	C_2	A_3	E_3	B_6

Order of Visit:

$S \quad C$

Example: Toy Example with UCS



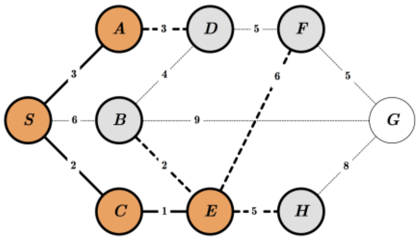
Priority Queue:

S_0	C_2	A_3	E_3	B_6	D_6

Order of Visit:

$S \quad C \quad A$

Example: Toy Example with UCS



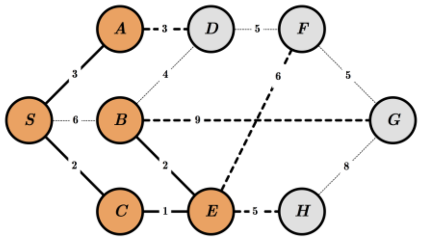
Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9

Order of Visit:

S *C* *A* *E*

Example: Toy Example with UCS



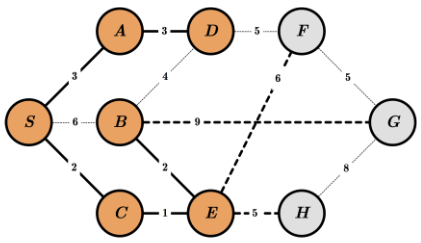
Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}

Order of Visit:

$S \quad C \quad A \quad E \quad B$

Example: Toy Example with UCS



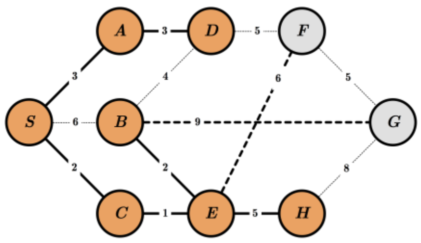
Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}

Order of Visit:

S *C* *A* *E* *B* *D*

Example: Toy Example with UCS



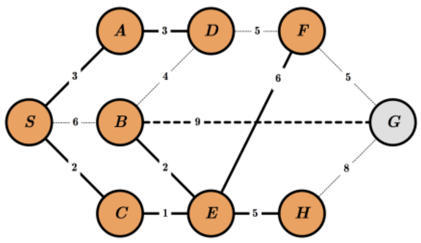
Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}

Order of Visit:

S *C* *A* *E* *B* *D* *H*

Example: Toy Example with UCS



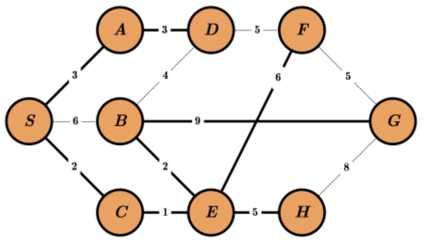
Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}

Order of Visit:

S C A E B D H F

Example: Toy Example with UCS

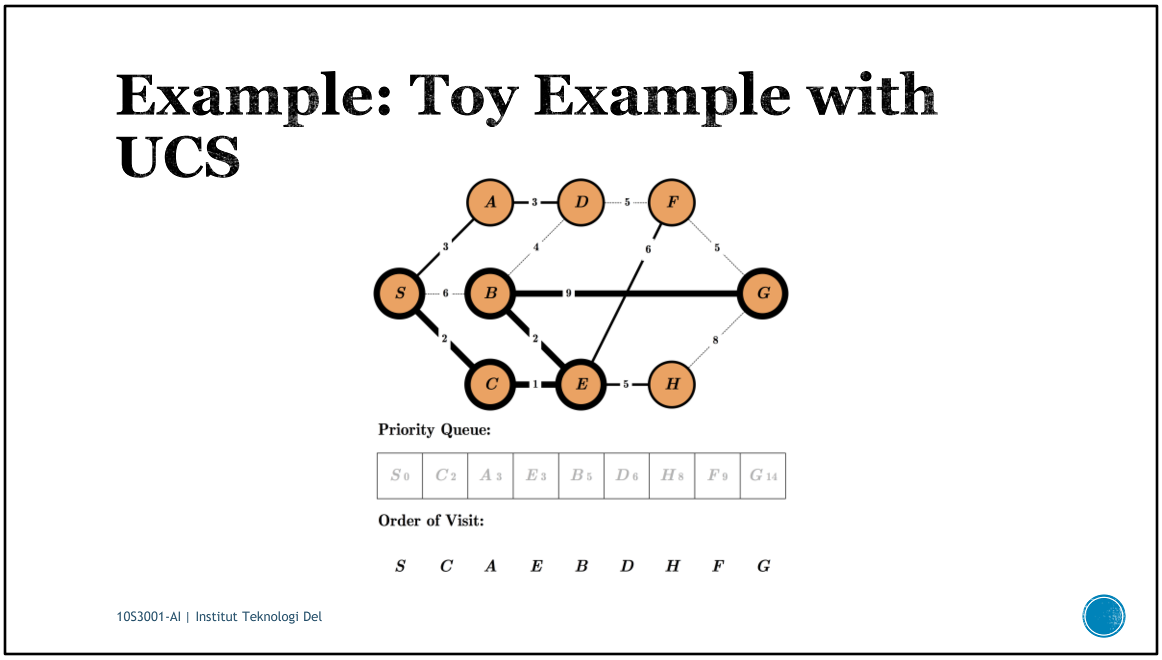


Priority Queue:

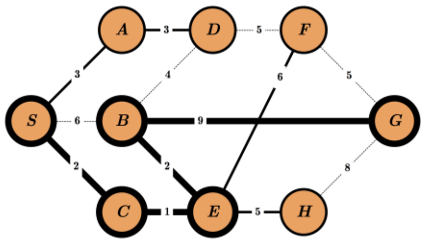
S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}

Order of Visit:

$S \quad C \quad A \quad E \quad B \quad D \quad H \quad F \quad G$



Example: Toy Example with UCS



Priority Queue:

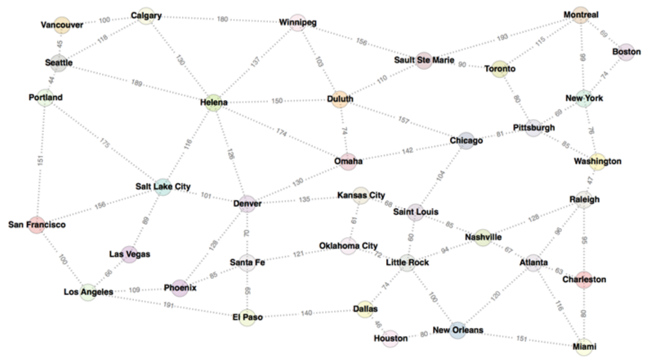
S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}

Order of Visit:

S *C* *A* *E* *B* *D* *H* *F* *G*

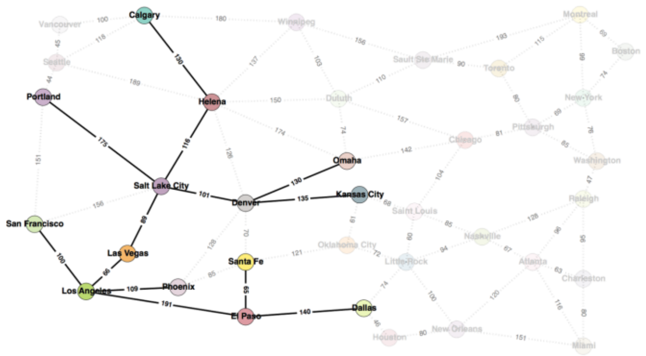
Example: Map Example

Start: Las Vegas
Goal: Calgary



Example: Map Example with BFS

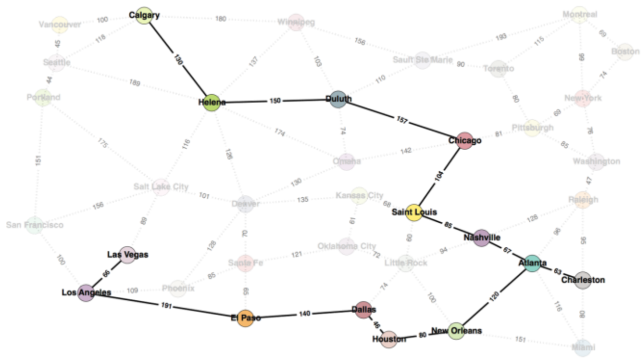
Start: Las Vegas
Goal: Calgary



Order of Visit: Las Vegas, Los Angeles, Salt Lake City, El Paso, Phoenix, San Francisco, Denver, Helena, Portland, Dallas, Santa Fe, Kansas City, Omaha, Calgary.

Example: Map Example with DFS

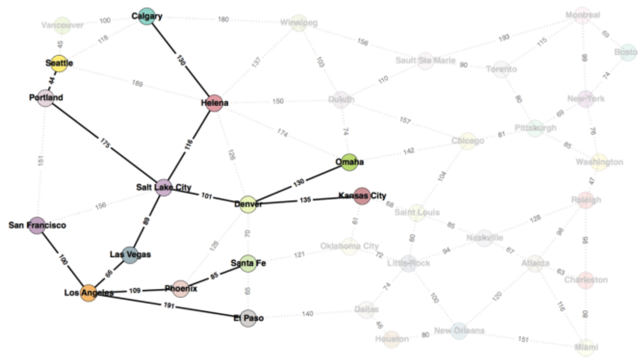
Start: Las Vegas
Goal: Calgary



Order of Visit: Las Vegas, Los Angeles, El Paso, Dallas, Houston, New Orleans, Atlanta, Charleston, Nashville, Saint Louis, Chicago, Duluth, Helena, Calgary.

Example: Map Example with UCS

Start: Las Vegas
Goal: Calgary



Order of Visit: Las Vegas, Los Angeles, Salt Lake City, San Francisco, Phoenix, Denver, Helena, El Paso, Santa Fe, Portland, Seattle, Omaha, Kansas City, Calgary.

References

- S. J. Russell and P. Borvig, *Artificial Intelligence: A Modern Approach (4th Edition)*, Prentice Hall International, 2020.

eof