Graph Search (6A)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice and Octave.

Graph Traversal

graph traversal (**graph search**) refers to
the process of visiting (checking and/or updating)
each **vertex** in a graph.

Such traversals are classified
by the order in which the vertices are visited.

Tree traversal is a special case of **graph traversal**.

https://en.wikipedia.org/wiki/Graph_traversal

DFS

A depth-first search (**DFS**)
is an algorithm for traversing a finite graph.

DFS visits the **child vertices**
before visiting the **sibling vertices**;

that is, it traverses the **depth** of any particular path
before exploring its **breadth**.

A **stack** (often the program's call stack via recursion) is
generally used when implementing the algorithm.

https://en.wikipedia.org/wiki/Graph_traversal

DFS Backtrack

The algorithm begins with a chosen "**root**" vertex;

it then iteratively transitions from the **current** vertex to an **adjacent**, **unvisited** vertex, until it can no longer find an **unexplored vertex** to transition to from its current location.

The algorithm then **backtracks** along previously **visited vertices**, until it finds a vertex connected to yet more uncharted territory.

It will then proceed down the **new path** as it had before, **backtracking** as it encounters **dead-ends**, and ending only when the algorithm has backtracked past the original "root" vertex from the very first step.

https://en.wikipedia.org/wiki/Graph_traversal

BFS

A breadth-first search (**BFS**) is
another technique for traversing a finite graph.

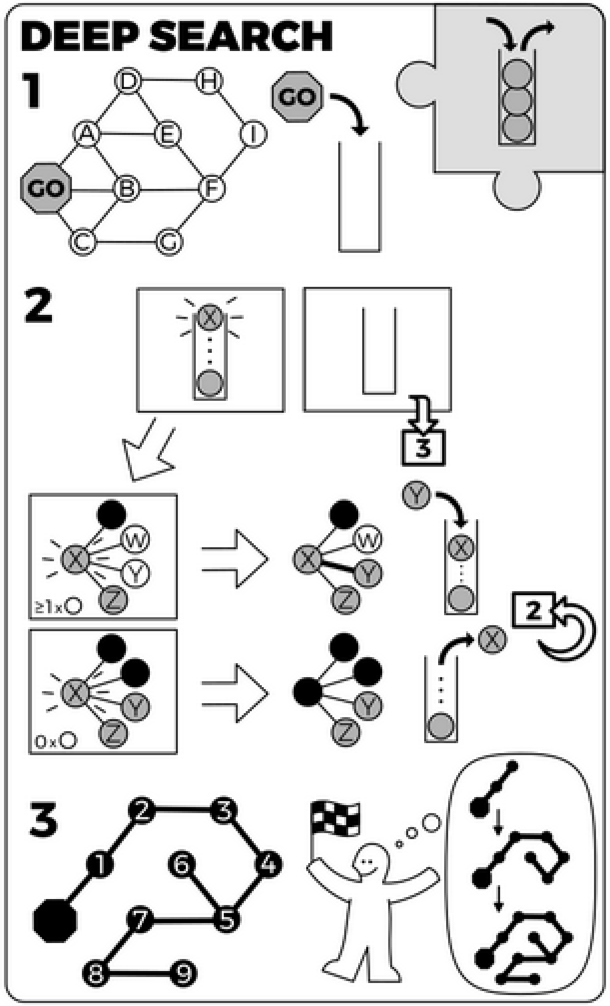
BFS visits the **neighbor** vertices
before visiting the **child** vertices

a **queue** is used in the search process

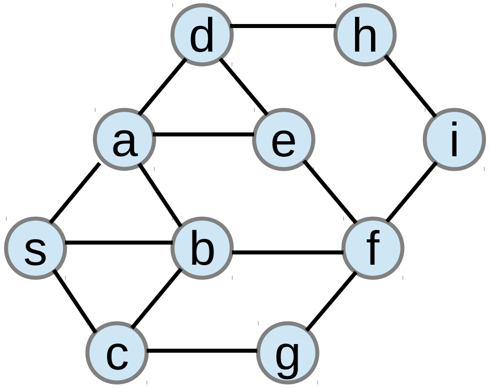
This algorithm is often used to find
the **shortest path** from one vertex to another.

https://en.wikipedia.org/wiki/Graph_traversal

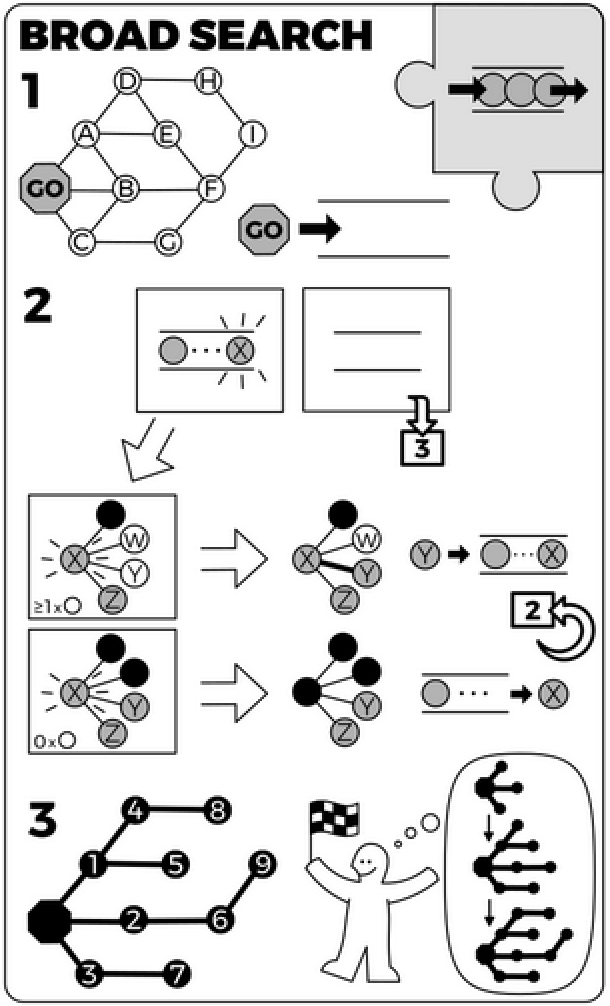
Depth First Search Example

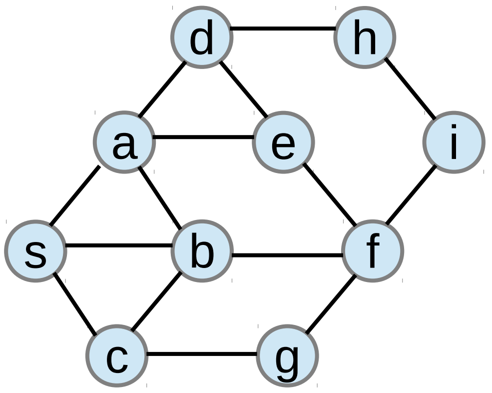


https://en.wikipedia.org/wiki/Graph_traversal



Breadth First Search Example





https://en.wikipedia.org/wiki/Graph_traversal

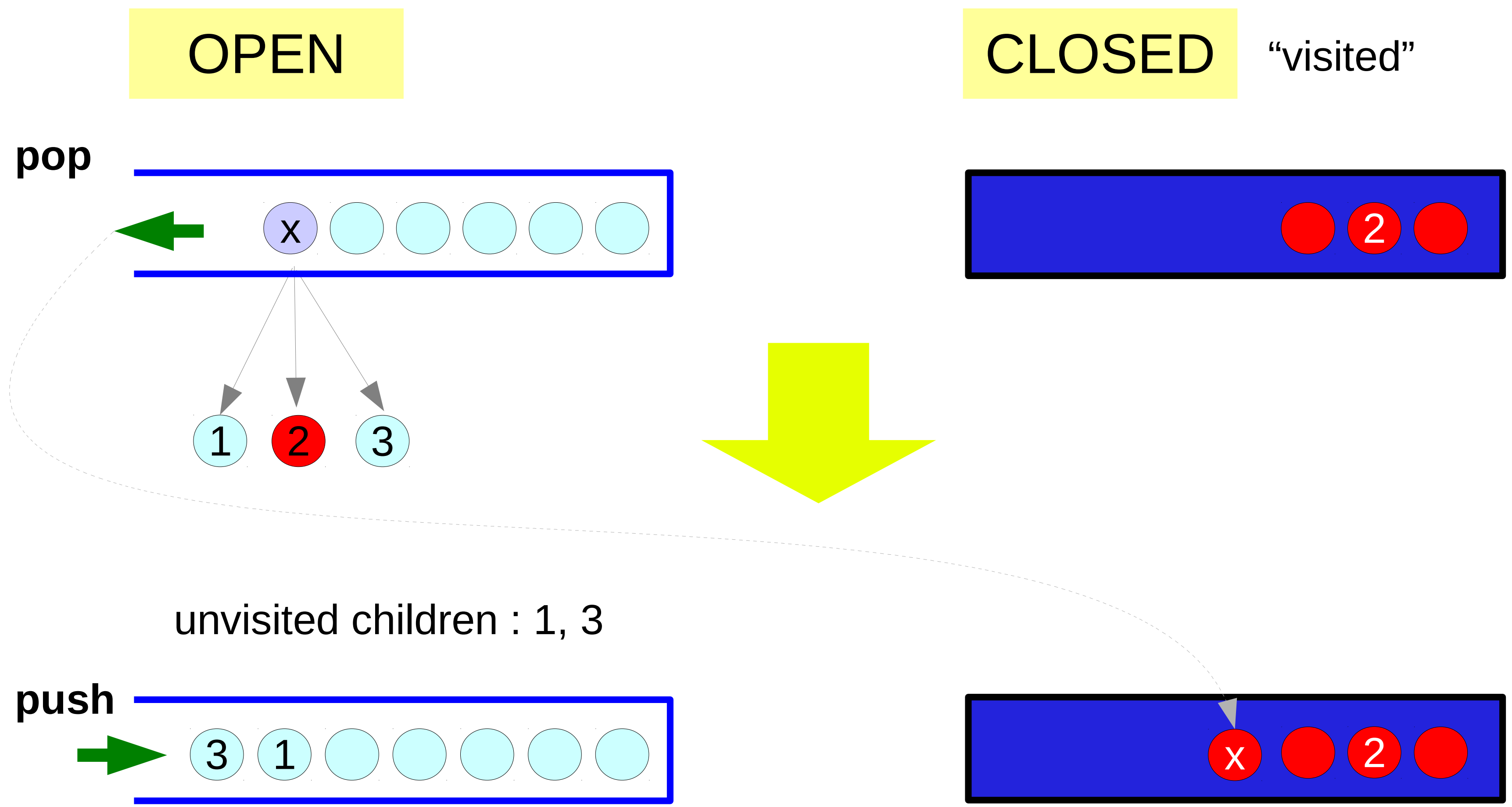
General Graph Search Algorithm – 1

```
Search(Start, isGoal, Criteria)
    insert(Start, Open);
    repeat
        if (empty(Open)) then return fail;
        select node from Open using Criteria;
        mark node as visited;
        if (isGoal(node)) then return node;
        for each child of node do
            if (child not already visited)
                then insert(child, Open);
```

https://courses.cs.washington.edu/courses/cse326/08wi/a/lectures/lecture13.pdf

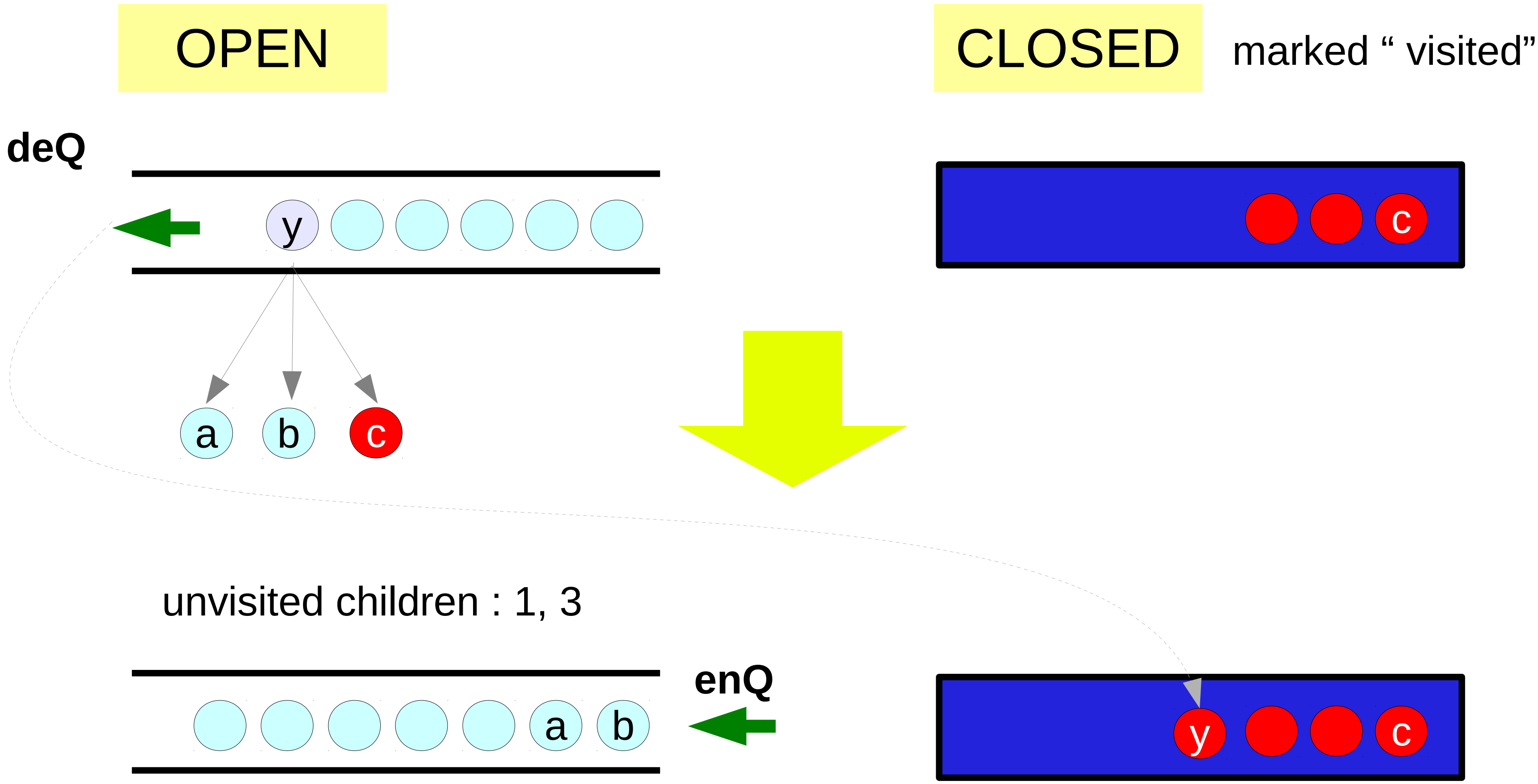
DFS

https://en.wikiversity.org/wiki/Artificial_intelligence/Lecture_aid



BFS

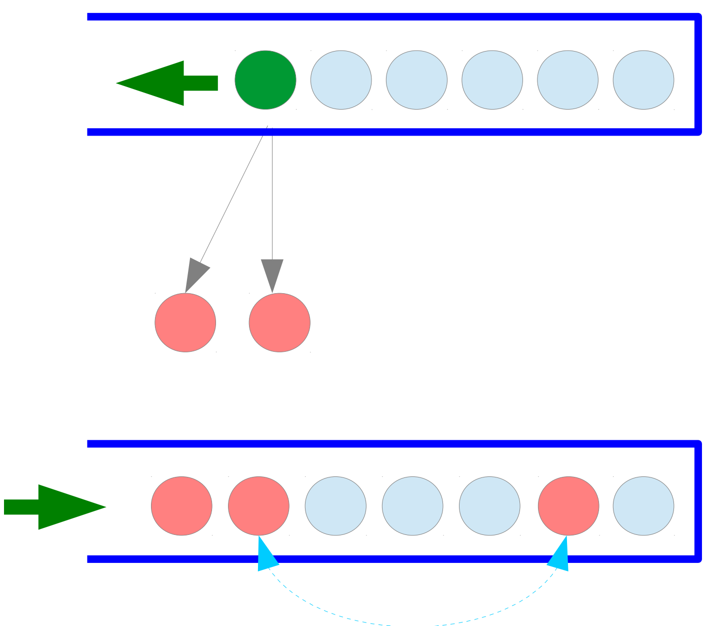
https://en.wikiversity.org/wiki/Artificial_intelligence/Lecture_aid



Possible duplication

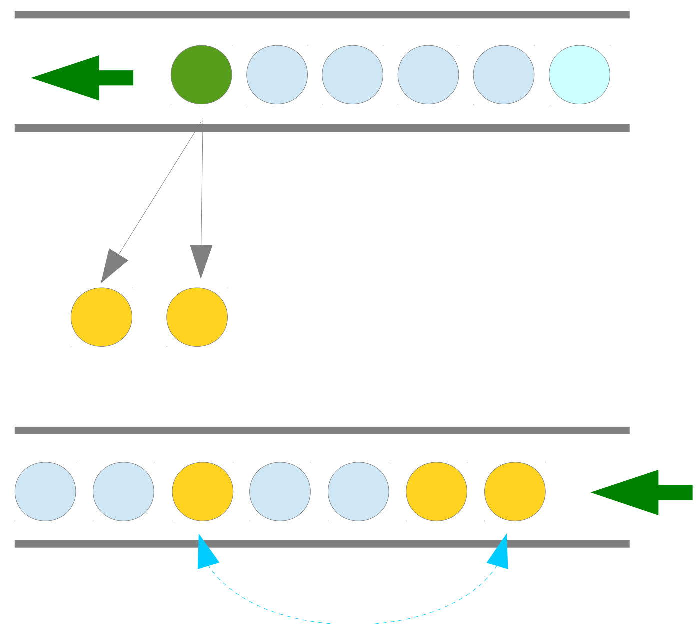
https://en.wikiversity.org/wiki/Artificial_intelligence/Lecture_aid

DFS ***Stack***



possible duplication
- not yet expanded

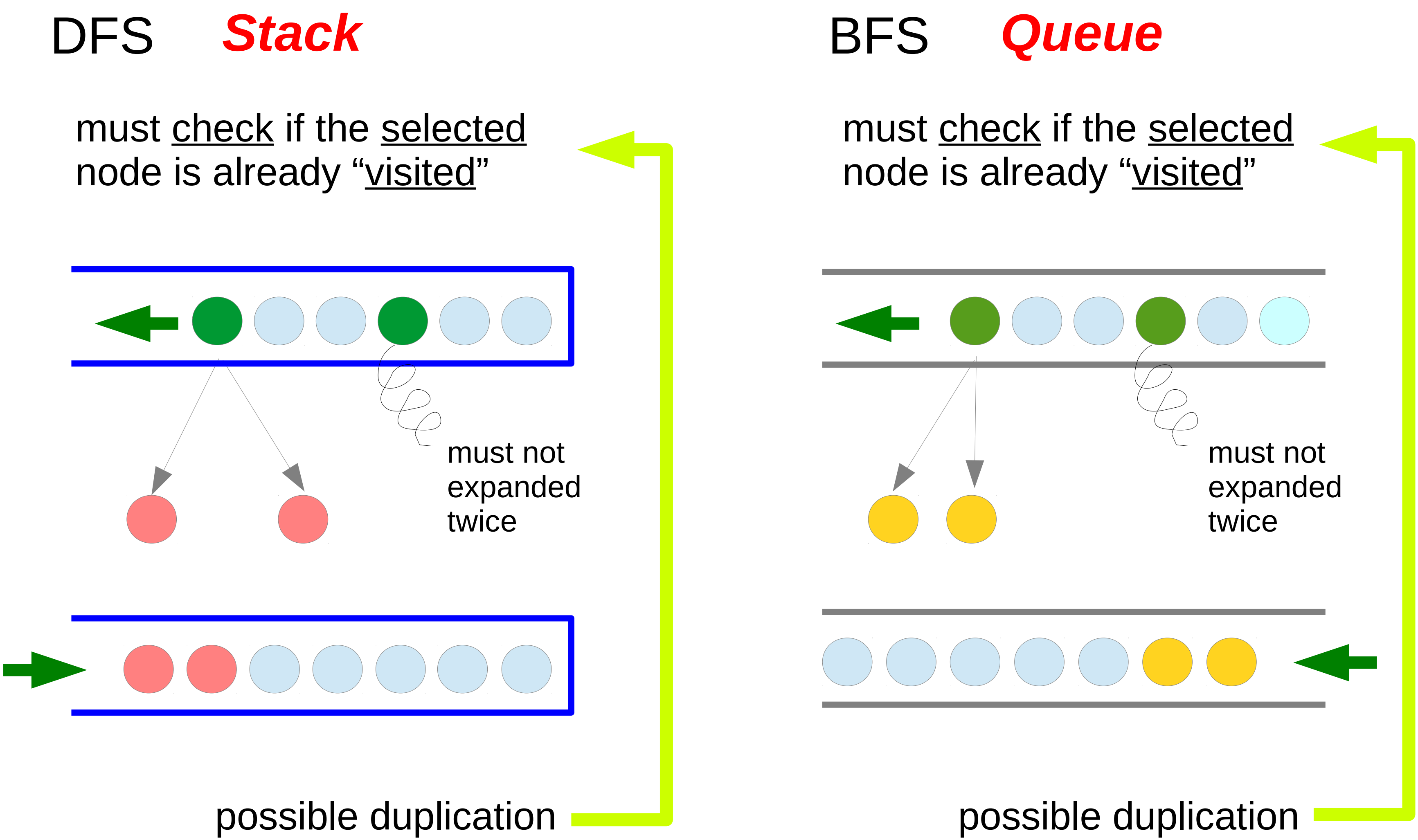
BFS ***Queue***



possible duplication
- not yet expanded

Must check before expansion

https://en.wikiversity.org/wiki/Artificial_intelligence/Lecture_aid



General Graph Search Algorithm – 1

```
Search(Start, isGoal, Criteria)
    insert(Start, Open);
    repeat
        if (empty(Open)) then return fail;
        select node from Open using Criteria;
        mark node as visited;
        if (isGoal(node)) then return node;
        for each child of node do
            if (child not already visited)
                then insert(child, Open);
```

Remedy 1:
check if visited when selecting

Remedy 2:
check redundant nodes

https://courses.cs.washington.edu/courses/cse326/08wi/a/lectures/lecture13.pdf

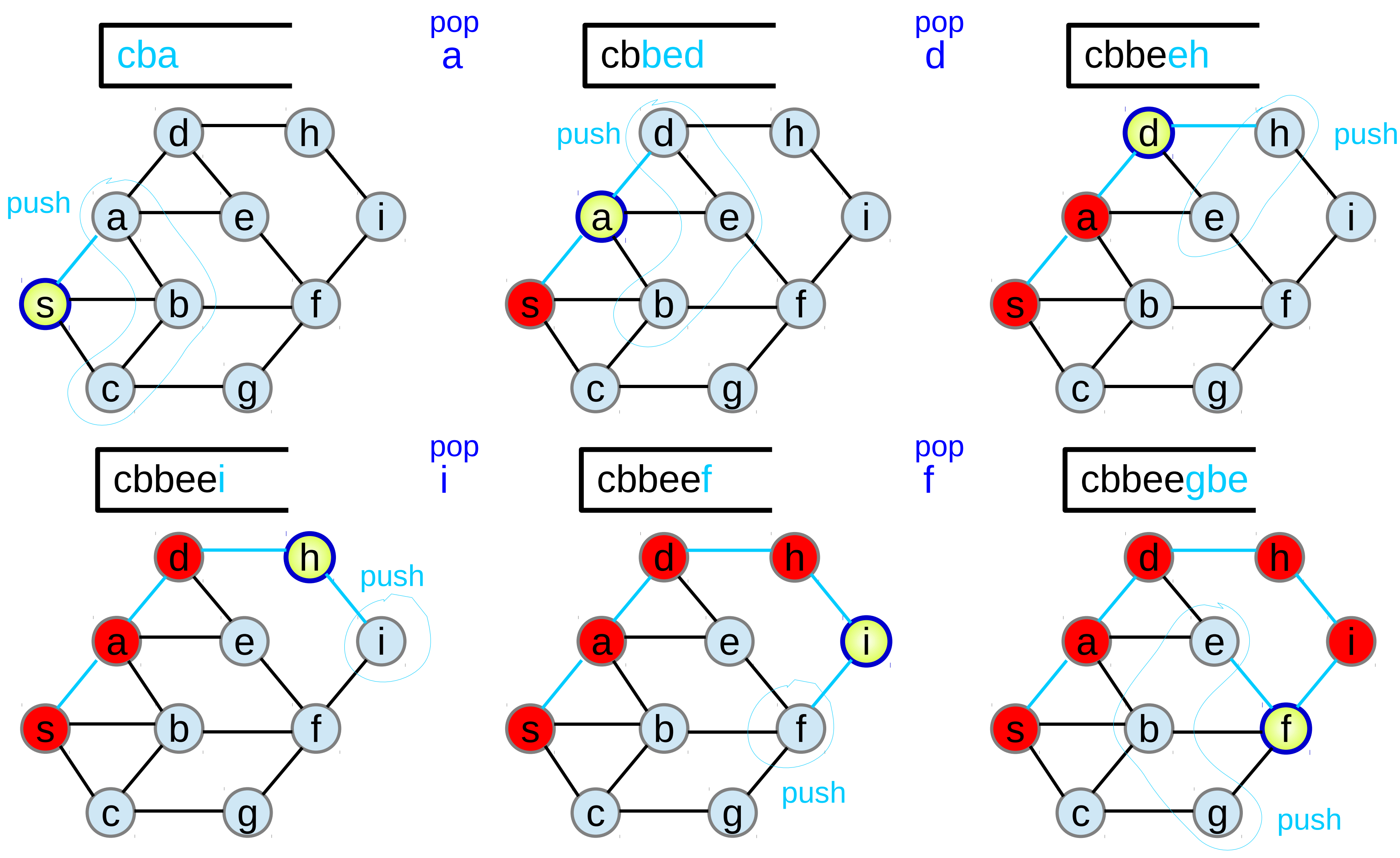
DFS-1 (Depth First Search)

Open list – use a stack
Select with Criteria – **pop**

```
DFS(Start, isGoal)
    push(Start, Open);                          // push
    repeat
        if (empty(Open)) then return fail;
        node := pop(Open);                      // pop
        mark node as visited;
        if (isGoal(node)) then return node;
        for each child of node do
            if (child not already visited) then
                push(child, Open);              // push
```

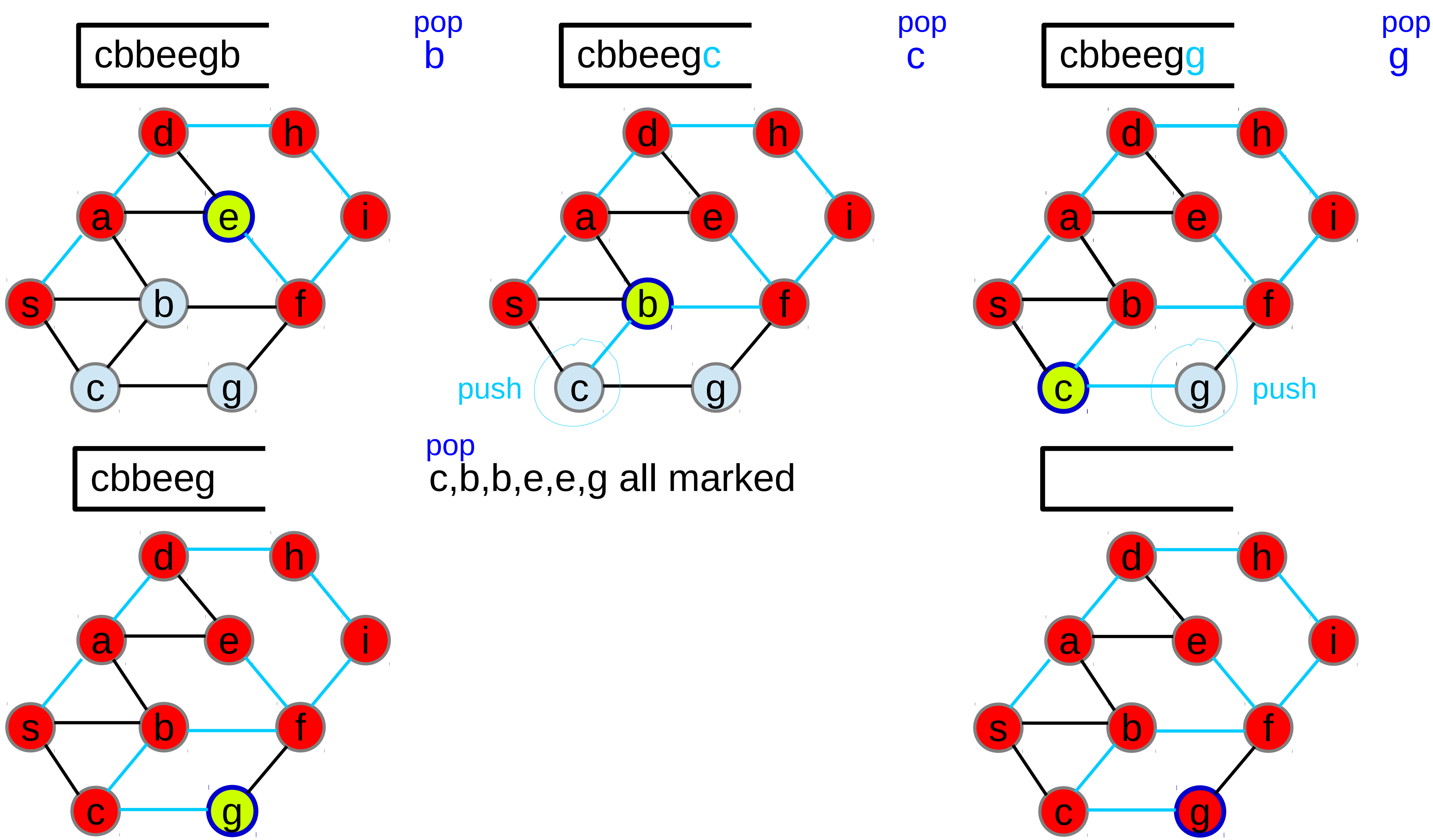
https://courses.cs.washington.edu/courses/cse326/08wi/a/lectures/lecture13.pdf

DFS-1 Example (1)



https://en.wikipedia.org/wiki/Graph_traversal

DFS-1 Example (2)



https://en.wikipedia.org/wiki/Graph_traversal

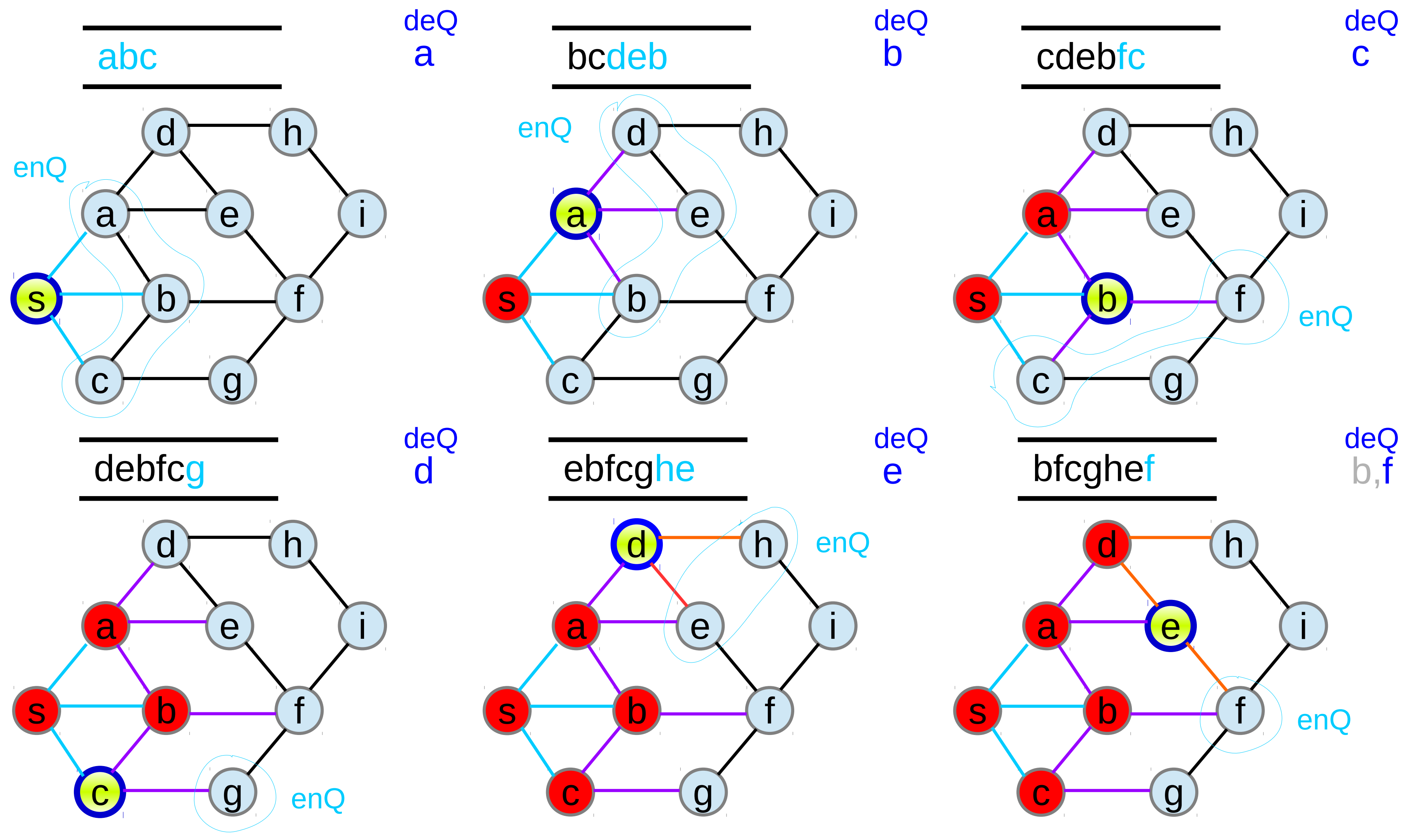
BFS-1 (Breadth First Search)

Open list – use a FIFO
Select with Criteria – **dequeue**

```
BFS(Start, isGoal)
    enqueue(Start, Open);                          // enqueue
    repeat
        if (empty(Open)) then return fail;
        node := dequeue(Open);                     // dequeue
        mark node as visited;
        if (isGoal(node)) then return node;
        for each child of node do
            if (child not already visited) then
                enqueue(child, Open);              // enqueue
```

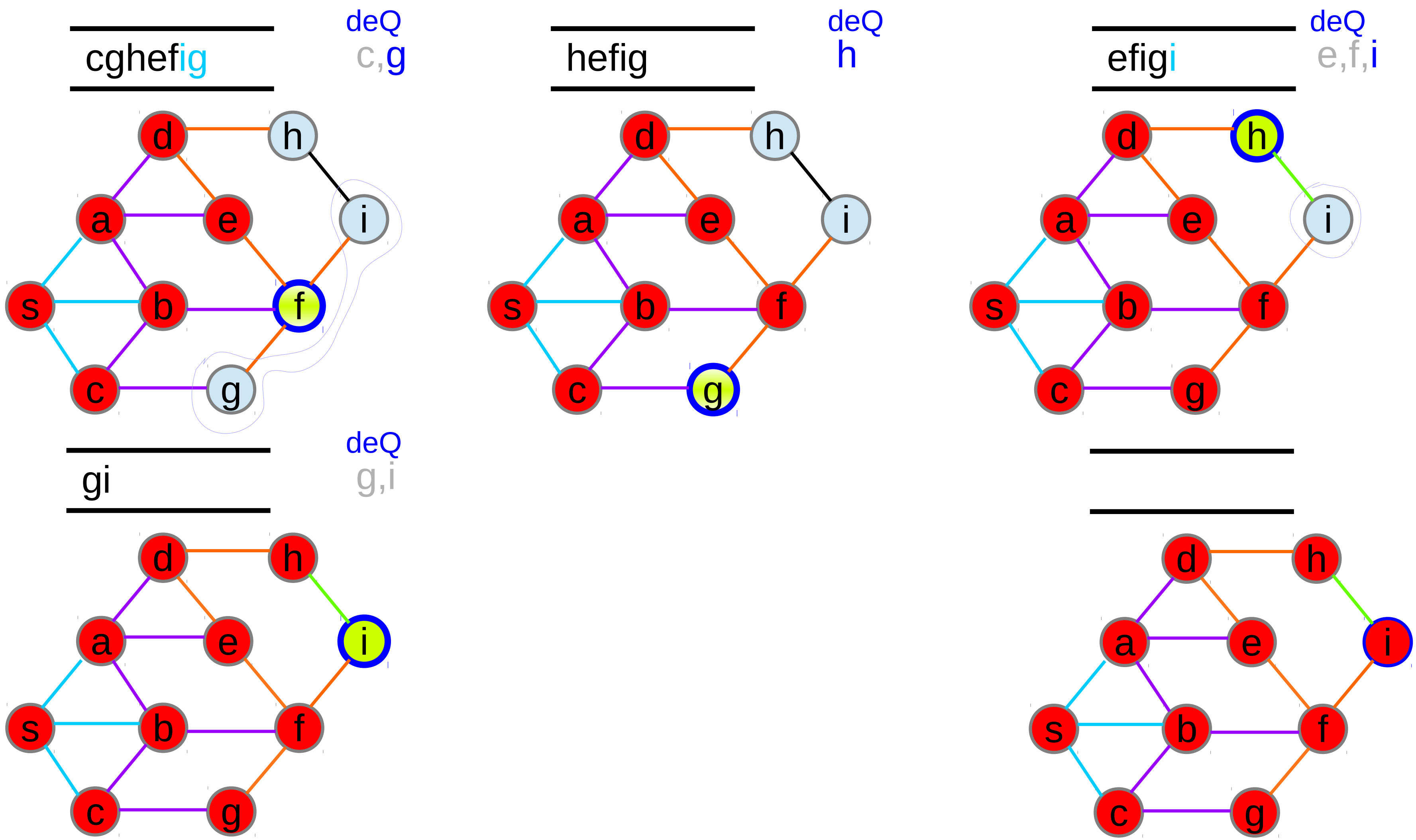
https://courses.cs.washington.edu/courses/cse326/08wi/a/lectures/lecture13.pdf

BFS-1 Example (1)



https://en.wikipedia.org/wiki/Graph_traversal

BFS-1 Example (2)



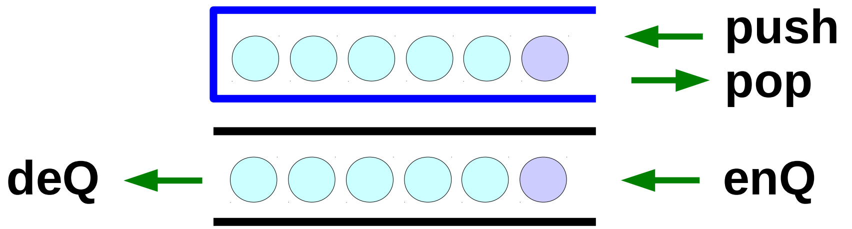
https://en.wikipedia.org/wiki/Graph_traversal

General Graph Search Algorithm – 2

```
Initialize as follows:
    unmark all nodes in N;
    mark node s;
    pred(s) = 0;   {that is, it has no predecessor}
    LIST = {s}
while LIST ≠ ø do
    select a node i in LIST;
    if node j is incident to an admissible arc (i,j) then
        mark node j;
        pred(j) := i;
        add node j to the end of LIST;
    else
        delete node i from LIST
```

DFS : select the last node i in LIST;

BFS : select the first node i in LIST;



https://ocw.mit.edu/courses/sloan-school-of-management/15-082j-network-optimization-fall-2010/lecture-notes/MIT15_082JF10_lec03.pdf

Admissible arc

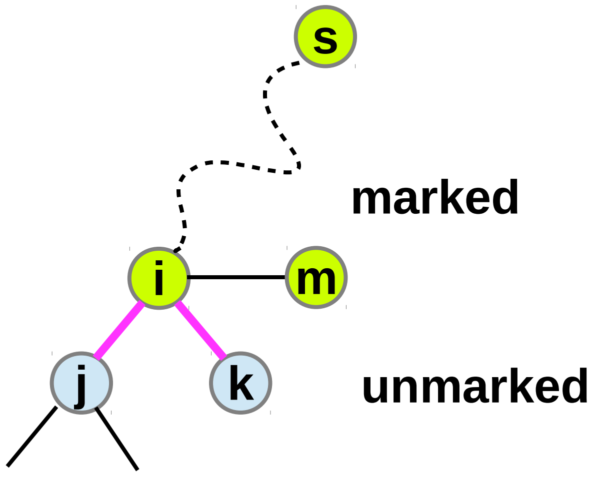
pred(**j**) is a node that **precedes j** on some path from s;

A node is either **marked** or **unmarked**.

Initially only node s is **marked**.

If a node is **marked**, it is **reachable** from node s.

An arc (**i**,**j**) ∈ A is **admissible**
if node **i** is marked and **j** is not.



https://ocw.mit.edu/courses/sloan-school-of-management/15-082j-network-optimization-fall-2010/lecture-notes/MIT15_082JF10_lec03.pdf

LIST

Before a node is <u>added</u> into LIST,
the node is **marked**

LIST contains only the **marked** nodes

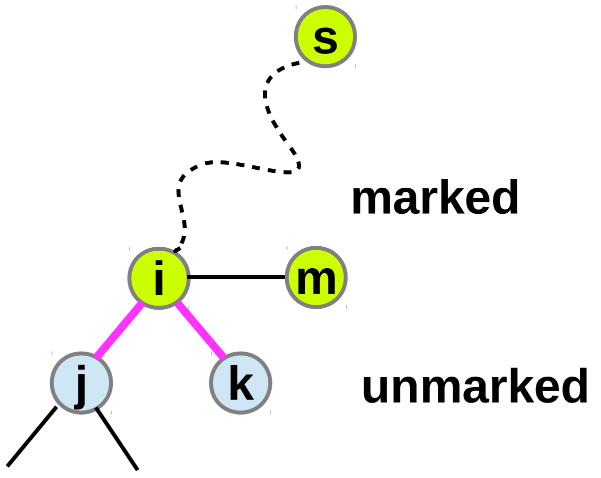
thus, the <u>selected</u> node **i** is **marked** already

The node **j** incident to the **admissible** arc(**i**,**j**)
must be **unmarked**

This node **j** is **marked** and <u>added</u> into LIST

In this way, LIST contains
only **marked** and **non-repeating** nodes

Check before inserting



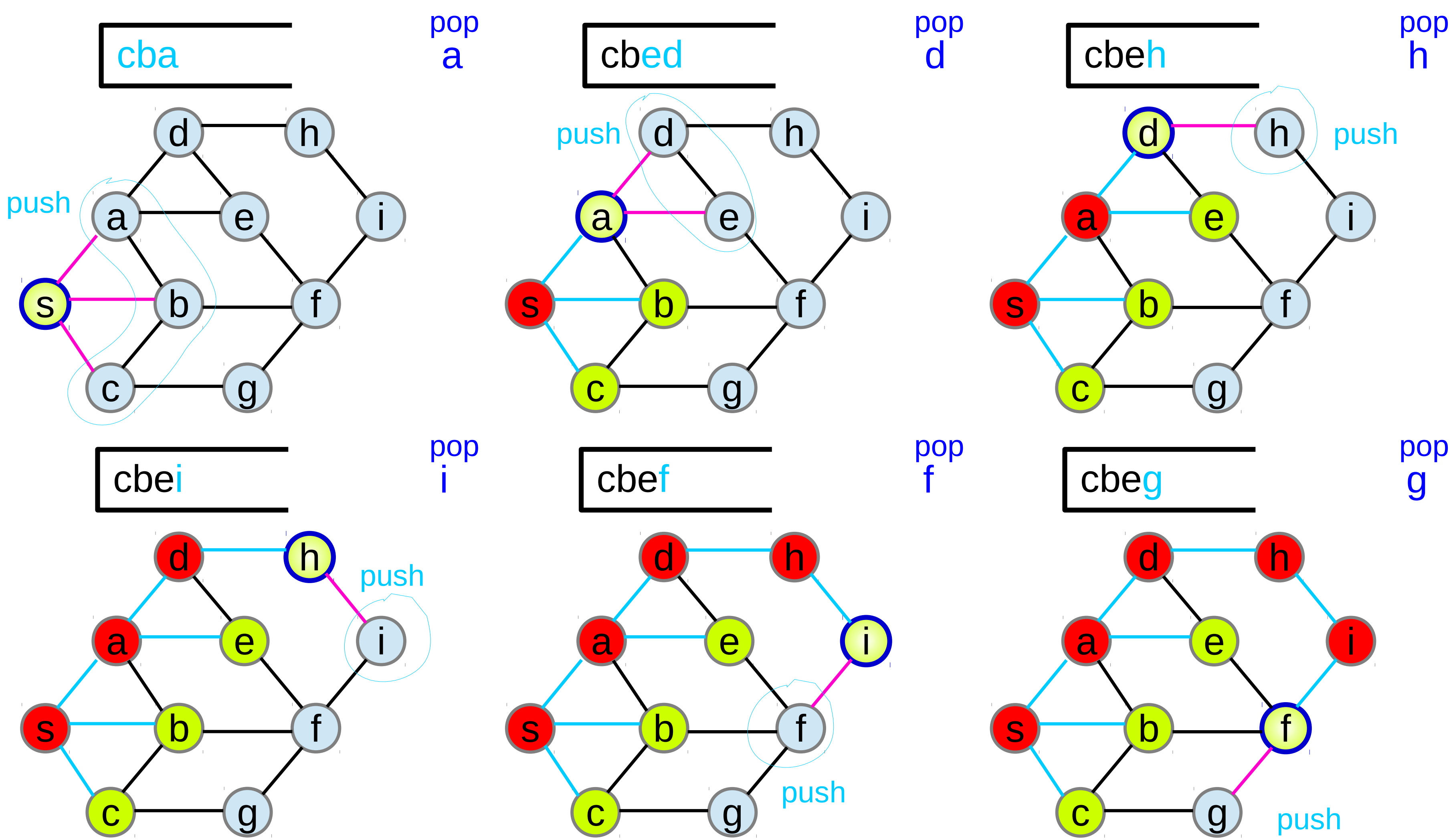
https://ocw.mit.edu/courses/sloan-school-of-management/15-082j-network-optimization-fall-2010/lecture-notes/MIT15_082JF10_lec03.pdf

DFS-2

```
Initialize as follows:
    unmark all nodes in N;
    mark node s;
    pred(s) = 0;   {that is, it has no predecessor}
    push s onto LIST
while LIST ≠ ø do
    pop a node i from LIST;
    if node j is incident to an admissible arc (i,j) then
        mark node j;
        pred(j) := i;
        push(node j) onto LIST;
    else
        delete node i from LIST
```

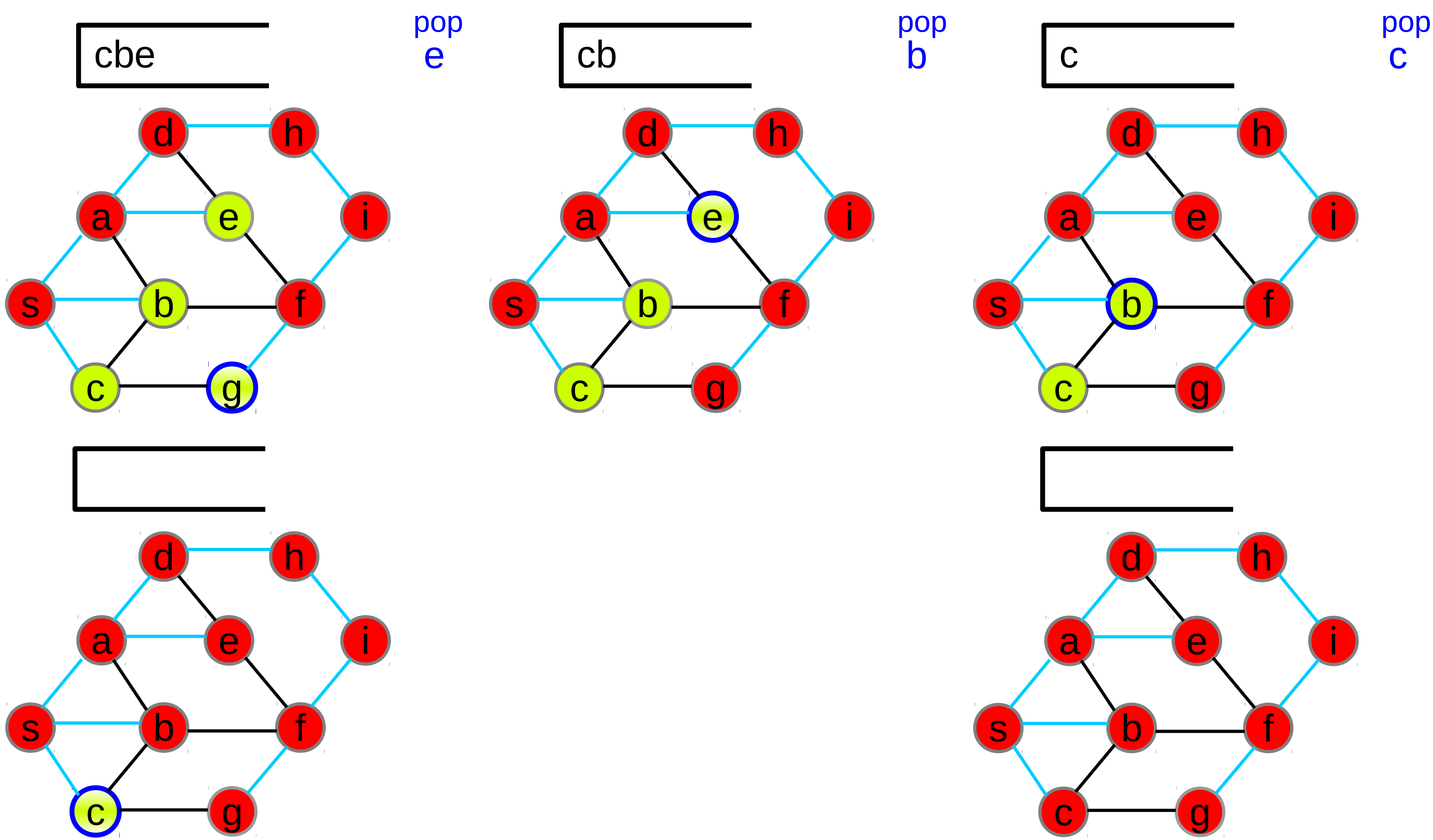
https://ocw.mit.edu/courses/sloan-school-of-management/15-082j-network-optimization-fall-2010/lecture-notes/MIT15_082JF10_lec03.pdf

DFS-2 Example (1)



https://en.wikipedia.org/wiki/Graph_traversal

DFS-2 Example (2)



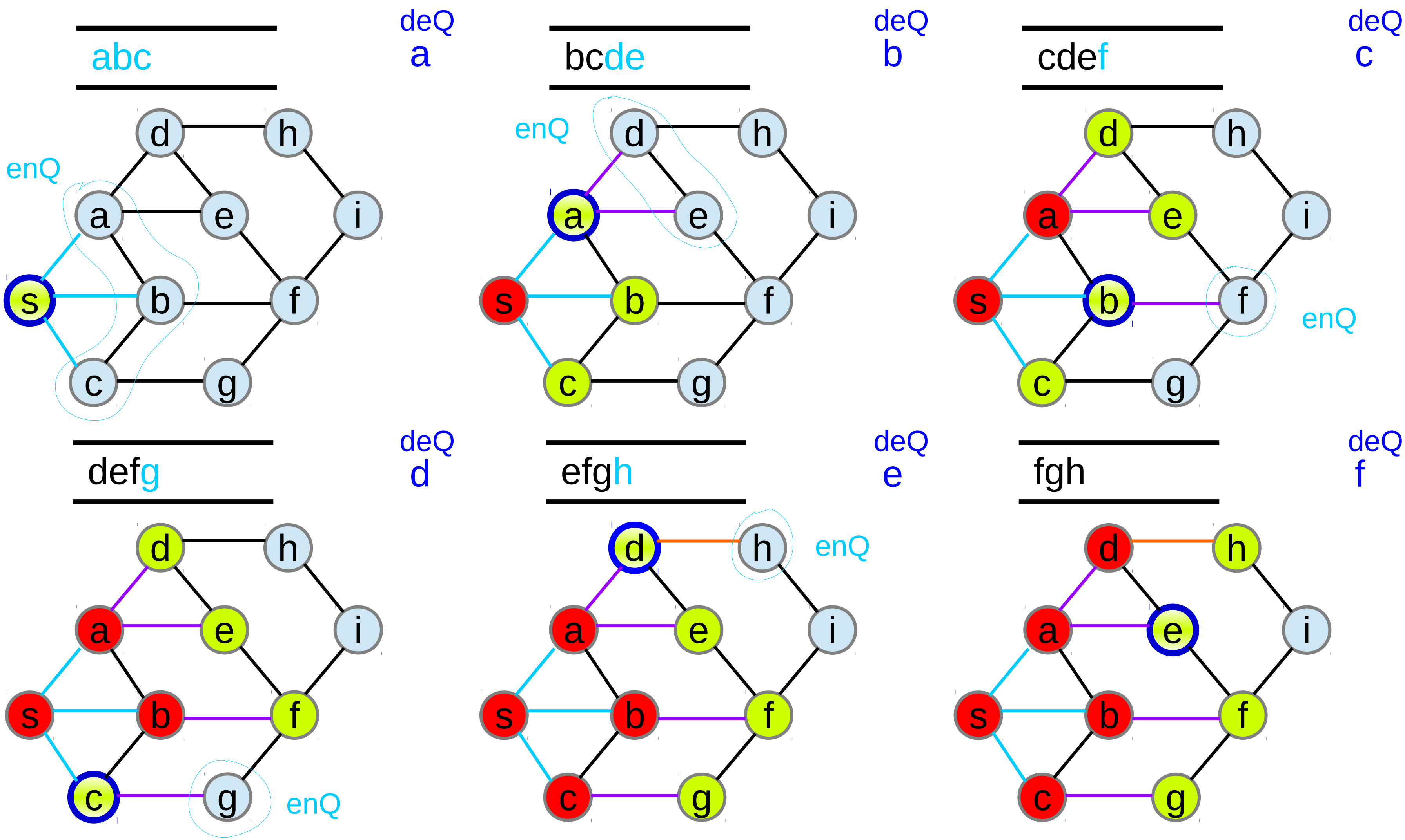
https://en.wikipedia.org/wiki/Graph_traversal

BFS-2

```
Initialize as follows:
    unmark all nodes in N;
    mark node s;
    pred(s) = 0;   {that is, it has no predecessor}
    enqueue s onto LIST
while LIST ≠ ø do
    dequeue node i from LIST;
    if node j is incident to an admissible arc (i,j) then
        mark node j;
        pred(j) := i;
        enqueue node j onto LIST;
    else
        delete node i from LIST
```

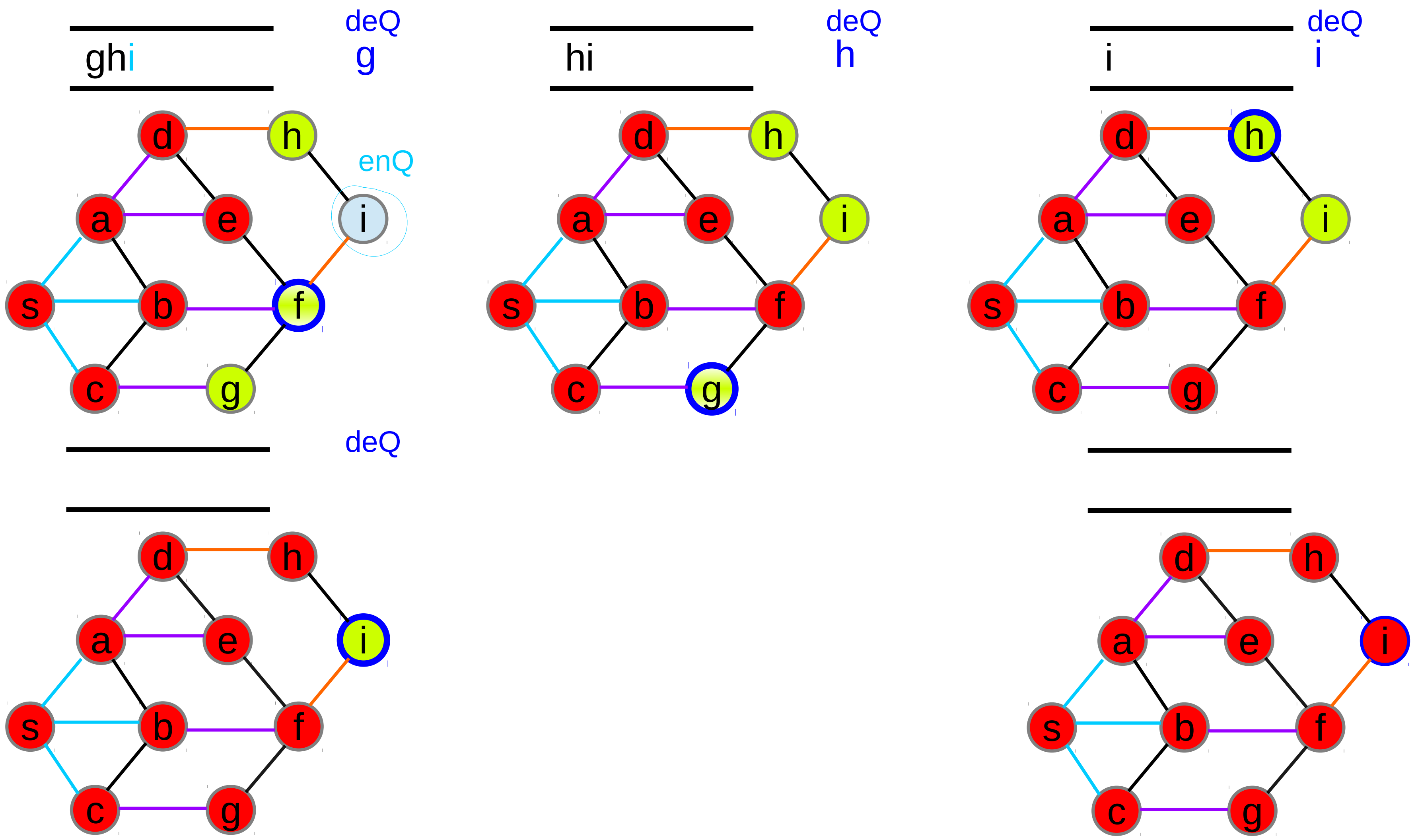
https://ocw.mit.edu/courses/sloan-school-of-management/15-082j-network-optimization-fall-2010/lecture-notes/MIT15_082JF10_lec03.pdf

BFS-2 Example (1)



https://en.wikipedia.org/wiki/Graph_traversal

BFS-2 Example (2)



https://en.wikipedia.org/wiki/Graph_traversal

DFS Pseudocode

```
procedure DFS(G, v):
    label v as explored
    for all edges e in G.incidentEdges(v) do
        if edge e is unexplored then
            w ← G.adjacentVertex(v, e)
            if vertex w is unexplored then
                label e as a discovered edge
                recursively call DFS(G, w)
            else
                label e as a back edge
```

https://en.wikipedia.org/wiki/Graph_traversal

BFS Pseudocode

```
procedure BFS(G, v):
    create a queue Q
    enqueue v onto Q
    mark v
    while Q is not empty:
        t ← Q.dequeue()
        if t is what we are looking for:
            return t
        for all edges e in G.adjacentEdges(t) do
            o ← G.adjacentVertex(t, e)
            if o is not marked:
                mark o
                enqueue o onto Q
    return null
```

https://en.wikipedia.org/wiki/Graph_traversal

References

[1] http://en.wikipedia.org/
[2]