Assignment Lecture

14. September 2022

Content

- Assignment 1
  - Solution
- Assignment 2
  - Representing graphs
  - BFS
  - DFS
  - Topological ordering
  -
  - Tema Tema

Task 1a

a) Explaining stable matching

Explain in your own words, what it means for a matching to be *stable*.

Solution:
A matching is considered to be stable when there does *not* exist a pairing (m, w) where m and w prefer each other to their current matching.

1b

b) Consider a version of the *SMP* in which we are attempting to match schools with students. Prove the following statement:

Consider the case in which we have a high school student and a university that have each other highest on their respective preference lists. i.e the student s prefers university u the most. And the university u prefers student s the most. Then in every S of this occurrence, the pair (s, u) belongs to S.

Solution:

This statement goes hand in hand with the concept of stability: if we propose a case where the student s is matched up with some other university u'. And the university u is matched up with some other student s'. This would lead to the matchings not being stable, as u and s would gladly leave their pairings to match up with each other, proving the statement.

2a

Task 2 Asymptotic growth rate

a) Sort the following list of functions in ascending order of growth rate. Meaning if function $f(n)$ follows function $g(n)$, then $f(n)$ is $O(g(n))$, explain your answer.

$$g_1(n) = n^2 \log n$$
$$g_2(n) = n!$$
$$g_3(n) = 400$$
$$g_4(n) = \log n$$
$$g_5(n) = 3n$$
$$g_6(n) = 5n^3$$

2a

Task 2 Asymptotic growth rate

a) Sort the following list of functions in ascending order of growth rate. Meaning if function $f(n)$ follows function $g(n)$, then g(n) = O(f(n) **, explain your answer.**

$$g_1(n) = n^2 \log n$$
$$g_2(n) = n!$$
$$g_3(n) = 400$$
$$g_4(n) = \log n$$
$$g_5(n) = 3n$$
$$g_6(n) = 5n^3$$

2a

Remove noise and add asymptotic notation

$$g_1 = \Theta(n^2 \log n)$$

$$g_2 = \Theta(n!)$$

g_3 will be simplified to $\Theta(1)$ as it runs in constant time.

$$g_4 = \Theta(\log n)$$

Then we remove the constants from g_5 and g_6 :

$$g_5 = \Theta(n)$$

$$g_6 = \Theta(n^3)$$

Sort

$$g_3 < g_4 < g_5 < g_1 < g_6 < g_2$$

2b

b) Match the following expressions so that there is an $f_i(n) = \Theta(g_i(n))$:

$$f_1(n) = n^2 + 12n + 4$$
$$f_2(n) = 4(\lg n)$$
$$f_3(n) = n^2 \cdot 42n$$
$$f_4(n) = n + 2$$
$$g_1(n) = \lg n + 45$$
$$g_2(n) = 5n - 3$$
$$g_3(n) = 3n^3$$
$$g_4(n) = (n + 5)^2$$

2b

Again: remove noise and add asymptotic notation

$$f_1(n) = n^2 + 12n + 4$$
$$f_2(n) = 4(\lg n)$$
$$f_3(n) = n^2 \cdot 42n$$
$$f_4(n) = n + 2$$
$$g_1(n) = \lg n + 45$$
$$g_2(n) = 5n - 3$$
$$g_3(n) = 3n^3$$
$$g_4(n) = (n + 5)^2$$

$$f_1 = \Theta(n^2)$$
$$f_2 = \Theta(\log n)$$
$$f_3 = \Theta(n^3)$$
$$f_4 = \Theta(n)$$
$$g_1 = \Theta(\log n)$$
$$g_2 = \Theta(n)$$
$$g_3 = \Theta(n^3)$$
$$g_4 = \Theta(n^2)$$

2b

Again: remove noise and add asymptotic notation

$$f_1(n) = n^2 + 12n + 4 \qquad f_1 = \Theta(n^2)$$
$$f_2(n) = 4(\lg n) \qquad f_2 = \Theta(\log n)$$
$$f_3(n) = n^2 \cdot 42n \qquad f_3 = \Theta(n^3)$$
$$f_4(n) = n + 2 \qquad f_4 = \Theta(n)$$
$$g_1(n) = \lg n + 45 \qquad g_1 = \Theta(\log n)$$
$$g_2(n) = 5n - 3 \qquad g_2 = \Theta(n)$$
$$g_3(n) = 3n^3 \qquad g_3 = \Theta(n^3)$$
$$g_4(n) = (n+5)^2 \qquad g_4 = \Theta(n^2)$$

Resulting match:

$$[(f_1, g_4), (f_2, g_1), (f_3, g_3), (f_4, g_2)]$$

2c

c) Consider the following expressions:

$$f(n) = \lg(n^{\lg 7})$$
$$g(n) = \lg(7^{\lg n})$$

Which of the listed asymptotic growth rates could represent the relationship between the functions, explain why or why not for each:

$$f(n) = \Omega(g(n))$$
$$f(n) = O(g(n))$$
$$f(n) = \Theta(g(n))$$

2c

Rewrite the expression

We have that $f(n) = \lg(n^{\lg 7})$

2c

Rewrite the expression

We have that $f(n) = \lg(n^{\lg 7})$

$$\lg 7 \lg n = \lg 7^{\lg n} = g(n).$$

2c

Rewrite the expression

We have that $f(n) = \lg(n^{\lg 7})$

$$\lg 7 \lg n = \lg 7^{\lg n} = g(n).$$

$$f(n) = \Theta(g(n))$$

2d

d) Simplify the following asymptotic expression, without the loss of precision:

$$\Theta(n^2) + O(n)$$

2d

d) Simplify the following asymptotic expression, without the loss of precision:

$$\Theta(n^2) + O(n)$$

Asymptotically tight

2d

d) Simplify the following asymptotic expression, without the loss of precision:

$$\Theta(n^2) + O(n)$$

Asymptotically tight	Upper bound

2d

d) Simplify the following asymptotic expression, without the loss of precision:

$$\Theta(n^2) + O(n)$$

Asymptotically tight	Upper bound

2d

d) Simplify the following asymptotic expression, without the loss of precision:

$$\Theta(n^2)$$

3a

```
ArraySum(List):
    n = len(List)
    currentSum = 0
    for i in range(n):
        currentSum = List[i]
        for j in range (i+1, n):
            currentSum = CurrentSum + List[j]
            Sum[i][j] = currentSum
```

Analyse the runtime of the algorithm **arraySum**

3a

```
ArraySum(List):
    n = len(List)                                    Constant time
    currentSum = 0
    for i in range(n):
        currentSum = List[i]
        for j in range (i+1, n):
            currentSum = CurrentSum + List[j]
            Sum[i][j] = currentSum
```

Analyse the runtime of the algorithm **arraySum**

3a

```
ArraySum(List):
    n = len(List)                                    Constant time
    currentSum = 0                                   Constant time
    for i in range(n):
        currentSum = List[i]
        for j in range (i+1, n):
            currentSum = CurrentSum + List[j]
            Sum[i][j] = currentSum
```

Analyse the runtime of the algorithm **arraySum**

3a

```
ArraySum(List):
    n = len(List)                                   Constant time
    currentSum = 0                                  Constant time
    for i in range(n):                              Runs n times
        currentSum = List[i]
        for j in range (i+1, n):
            currentSum = CurrentSum + List[j]
            Sum[i][j] = currentSum
```

Analyse the runtime of the algorithm `arraySum`

3a

```
ArraySum(List):
    n = len(List)                            Constant time
    currentSum = 0                           Constant time
    for i in range(n):                       Runs n times
        currentSum = List[i]                 Constant time
        for j in range (i+1, n):
            currentSum = CurrentSum + List[j]
            Sum[i][j] = currentSum
```

Analyse the runtime of the algorithm **arraySum**

3a

```
ArraySum(List):
    n = len(List)
    currentSum = 0
    for i in range(n):
        currentSum = List[i]
        for j in range (i+1, n):
            currentSum = CurrentSum + List[j]
            Sum[i][j] = currentSum
```

- `n = len(List)` — Constant time
- `currentSum = 0` — Constant time
- `for i in range(n):` — Runs n times
- `currentSum = List[i]` — Constant time
- `for j in range (i+1, n):` — Runs n-i times

Analyse the runtime of the algorithm **arraySum**

3a

```
ArraySum(List):
    n = len(List)                                   Constant time
    currentSum = 0                                  Constant time
    for i in range(n):                              Runs n times
        currentSum = List[i]                            Constant time
        for j in range (i+1, n):                        Runs n-i times
            currentSum = CurrentSum + List[j]           Constant time
            Sum[i][j] = currentSum                      Constant time
```

Analyse the runtime of the algorithm **arraySum**

3a

```
ArraySum(List):
    n = len(List)                                   Constant time
    currentSum = 0                                  Constant time
    for i in range(n):                              Runs n times
        currentSum = List[i]                        Constant time
        for j in range (i+1, n):                    Runs n-i times
            currentSum = CurrentSum + List[j]       Constant time
            Sum[i][j] = currentSum                  Constant time
```

Analyse the runtime of the algorithm **`arraySum`**

3a

```
for i in range(n):
```

This runs n times

```
for j in range (i+1, n):
```

For every time the above runs, this line runs (n-i) times

3a

```
for i in range(n):
```

```
for j in range (i+1, n):
```

$$i = 0 \rightarrow 1 \ldots (n-1) = (n-1)\ iterations$$
$$i = 1 \rightarrow 2 \ldots (n-1) = (n-2)\ iterations$$
$$\vdots$$
$$i = n-1 \rightarrow (n-1) \ldots (n-1) = 1\ iteration$$

3a

```
for i in range(n):
```

```
for j in range (i+1, n):
```

$$
\left.\begin{aligned}
&i = 0 \to 1 \ldots (n-1) = (n-1)\ \textit{iterations} \\
&i = 1 \to 2 \ldots (n-1) = (n-2)\ \textit{iterations} \\
&\vdots \\
&i = n-1 \to (n-1) \ldots (n-1) = 1\ \textit{iteration}
\end{aligned}\right\}
\quad (n-1) + (n-2) + \cdots + 2 + 1 =
$$

3a

```
for i in range(n):
```

```
for j in range (i+1, n):
```

$i = 0 \rightarrow 1 \ldots (n-1) = (n-1)\ iterations$
$i = 1 \rightarrow 2 \ldots (n-1) = (n-2)\ iterations$
$\vdots$
$i = n-1 \rightarrow (n-1) \ldots (n-1) = 1\ iteration$

$$(n-1) + (n-2) + \cdots + 2 + 1 =$$
$$(n-1+1) + (n-2+2) + \cdots + \left(n - \frac{n}{2} + \frac{n}{2}\right) =$$

3a

```
for i in range(n):
```

```
for j in range (i+1, n):
```

$$
\left.\begin{array}{l}
i = 0 \rightarrow 1 \ldots (n-1) = (n-1)\ iterations \\
i = 1 \rightarrow 2 \ldots (n-1) = (n-2)\ iterations \\
\vdots \\
i = n-1 \rightarrow (n-1) \ldots (n-1) = 1\ iteration
\end{array}\right\}
\begin{array}{l}
(n-1) + (n-2) + \cdots + 2 + 1 = \\
(n-1+1) + (n-2+2) + \cdots + \left(n - \frac{n}{2} + \frac{n}{2}\right) = \\
n + n + \cdots + n =
\end{array}
$$

3a

```
for i in range(n):
```

```
for j in range (i+1, n):
```

$$
\left.
\begin{array}{l}
i = 0 \rightarrow 1 \ldots (n-1) = (n-1)\ iterations \\
i = 1 \rightarrow 2 \ldots (n-1) = (n-2)\ iterations \\
\vdots \\
i = n-1 \rightarrow (n-1) \ldots (n-1) = 1\ iteration
\end{array}
\right\}
\begin{array}{l}
(n-1) + (n-2) + \cdots + 2 + 1 = \\
(n-1+1) + (n-2+2) + \cdots + \left(n - \frac{n}{2} + \frac{n}{2}\right) = \\
n + n + \cdots + n = \\
n * \frac{n-1}{2}
\end{array}
$$

3a

$$n * \frac{n-1}{2} = \frac{n^2 - n}{2} = \frac{1}{2}(n^2 - n) = O(n^2 - n) = O(n^2)$$

3b

The following made-up algorithm `SLY` has a runtime of $\Theta(n^2)$, it includes the algorithm `BENTLEY`, which has a runtime of $\Theta(n^2)$. Given this information, what can we say about the runtime of `MURRAY`?

```
SLY(A):
    n = len(A)
    BENTLEY(n)
    for i in range (n):
        MURRAY(A, i)
```

3b

The following made-up algorithm **SLY** has a runtime of $\Theta(n^2)$, it includes the algorithm **BENTLEY**, which has a runtime of $\Theta(n^2)$. Given this information, what can we say about the runtime of **MURRAY**?

```
SLY(A):
    n = len(A)
    BENTLEY(n)
    for i in range (n):
        MURRAY(A, i)
```

Runs n*n times

Runs n*n times

Can not run more than n*n times

3b

The following made-up algorithm **SLY** has a runtime of $\Theta(n^2)$, it includes the algorithm **BENTLEY**, which has a runtime of $\Theta(n^2)$. Given this information, what can we say about the runtime of **MURRAY**?

```
SLY(A):
    n = len(A)
    BENTLEY(n)
    for i in range (n):
        MURRAY(A, i)
```

Runs n*n times

Runs n*n times

Can not run more than n*n times

Since the for-loop runs n times, MURRAY must have a runtime of O(n)

4a

Imagine that we want to modify the stable matching problem to allow same-sex marriages. Explain how this changes the problem and provide a draft for an algorithm that solves it (the answer does not have to be code, you can explain it with words, drawings, figures, etc.).

Solution:
Stable matching with same-sex marriage is commonly known as the stable roommates problem. Here you were supposed to recognize that when there are no separate groups to match, and they all have preferences within the same group, there isn't always a stable matching. So the algorithm first needs to check if a stable matching exists before finding said matching.

Simplify the following asymptotic expression, and explain your steps:

$$\frac{\Omega(n^4)}{O(n^2)} + \frac{\Theta(n^7)}{\Theta(n^3)} + \frac{O(n^7)}{\Omega(n^4)}$$

Simplify the following asymptotic expression, and explain your steps:

$$\frac{\Omega(n^4)}{O(n^2)} + \frac{\Theta(n^7)}{\Theta(n^3)} + \frac{O(n^7)}{\Omega(n^4)}$$

Solution:

$$\frac{\Omega(n^4)}{O(n^2)} + \frac{\Theta(n^7)}{\Theta(n^3)} + \frac{O(n^7)}{\Omega(n^4)}$$

$$\Omega(n^2) + \Theta(n^4) + O(n^3)$$

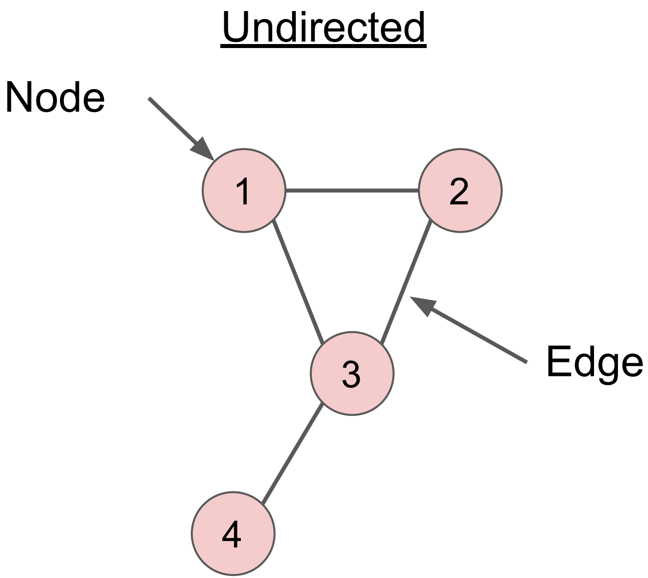
$$\Omega(n^4)$$

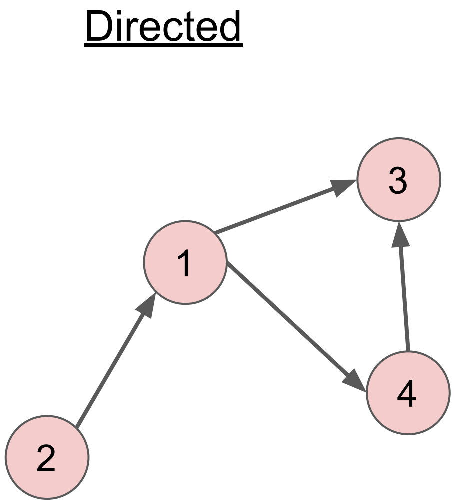
Assignment 2

Graph G = (V, E)

V = nodes

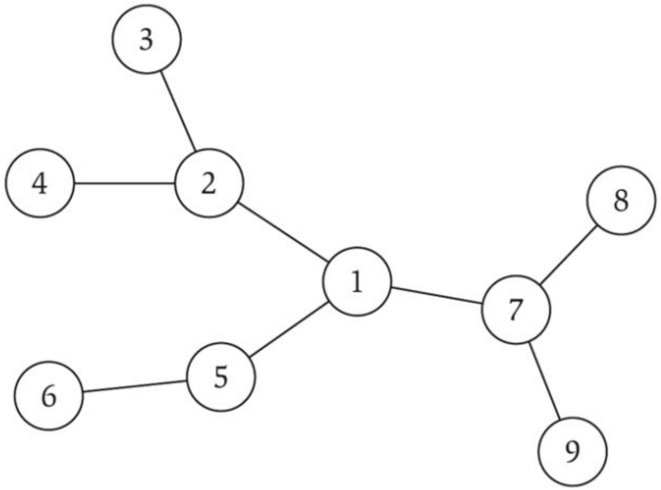
E = Edge - connects a pair of nodes





Trees

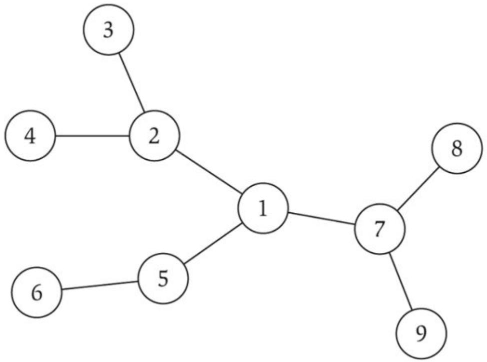
An undirected graph is a **tree** if it is connected and does not contain a cycle



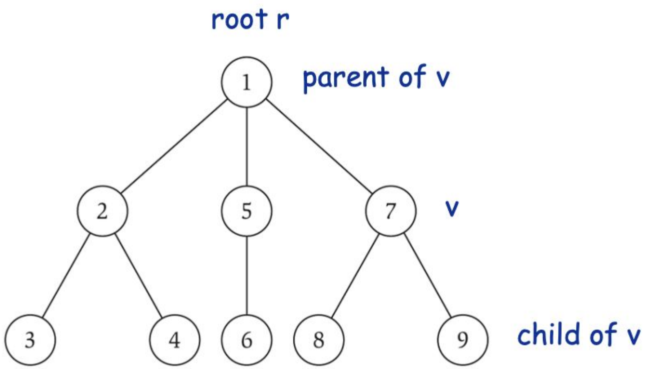
Rooted trees

Given a tree T, choose a root node r and orient each edge away from r

Models hierarchical structure



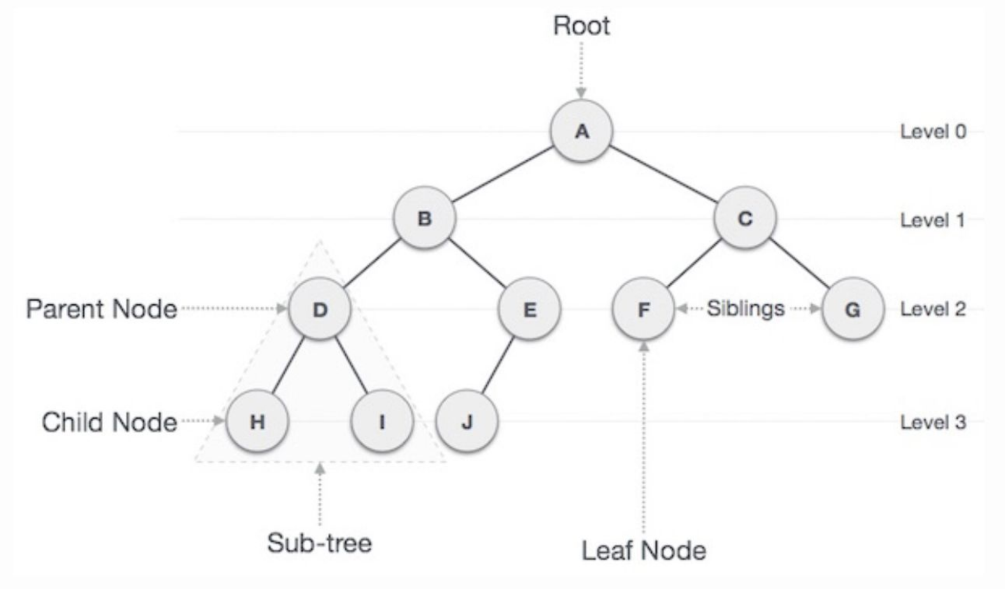
a tree



the same tree, rooted at 1

Binary trees

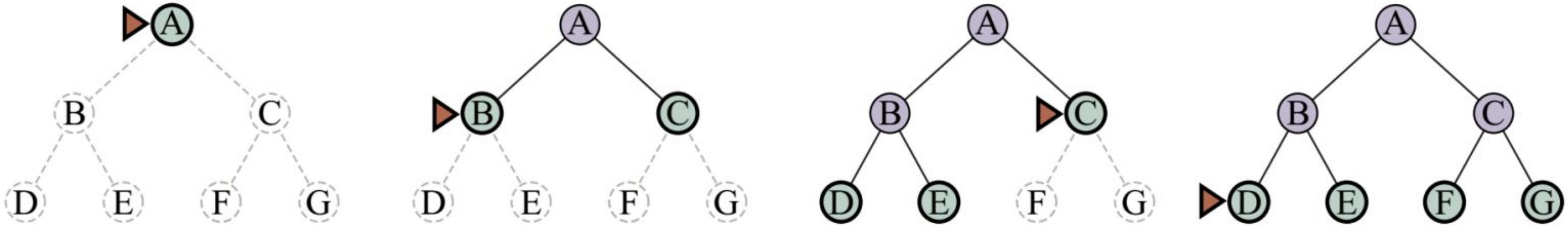
Binary tree has a maximum of 2 child nodes from each node



Graph traversal

BFS - Breadth First Search

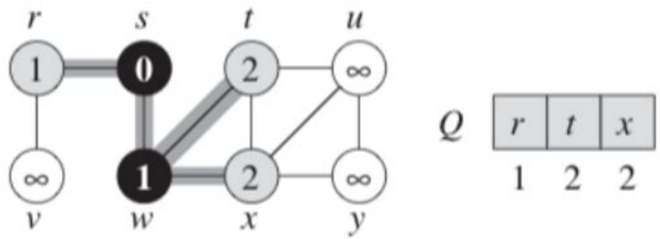
- Implements a FIFO-list
  - First-in-first-out
- Expands nodes with least debth first



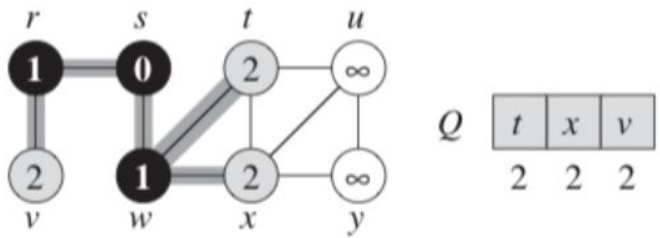
Example

- White nodes = undiscovered
- Gray nodes = discovered
- Black nodes = all neighboring nodes detected

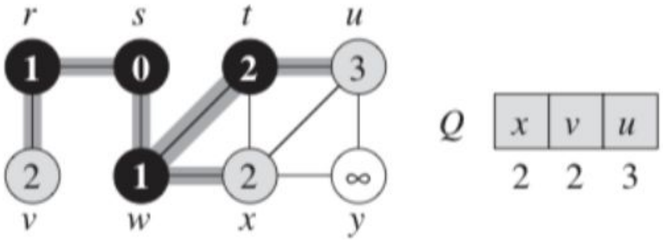
(c)



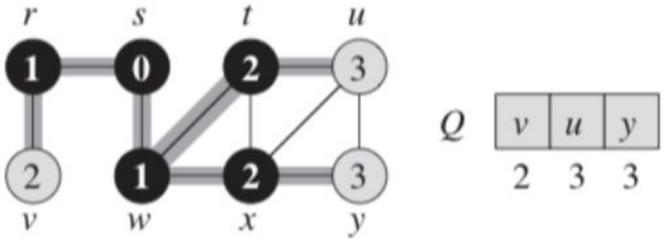
(d)

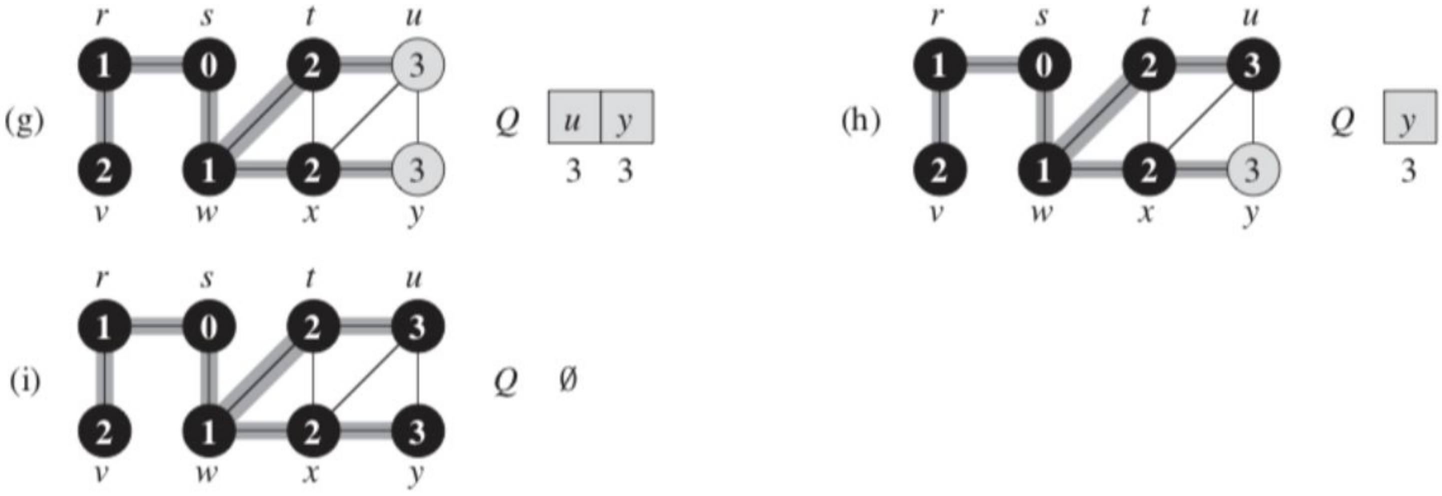


(e)



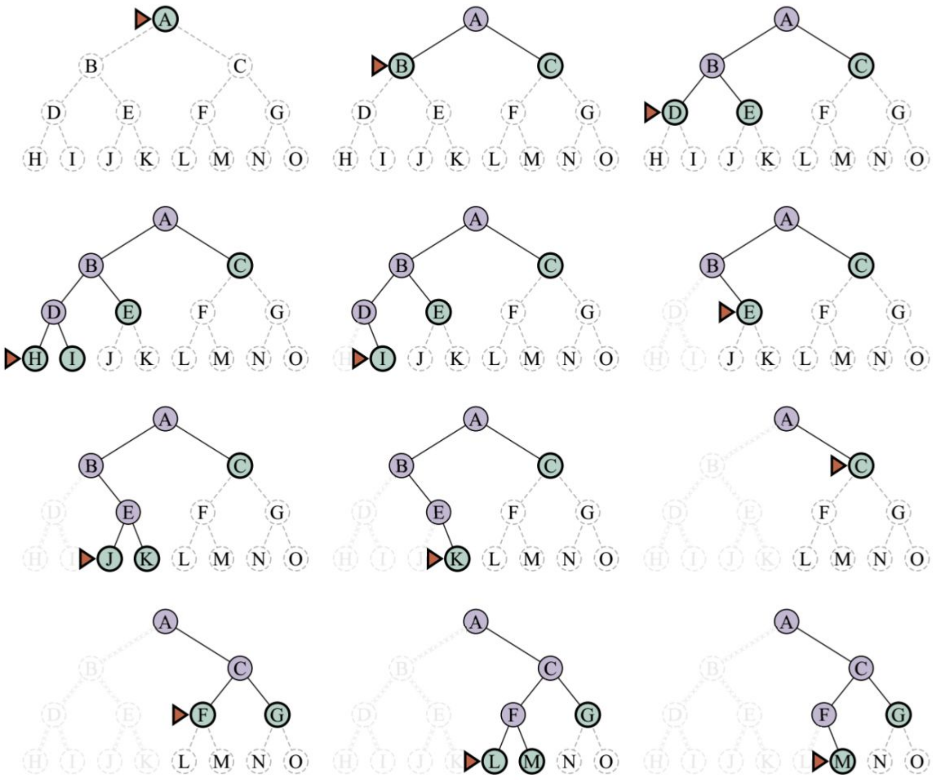
(f)



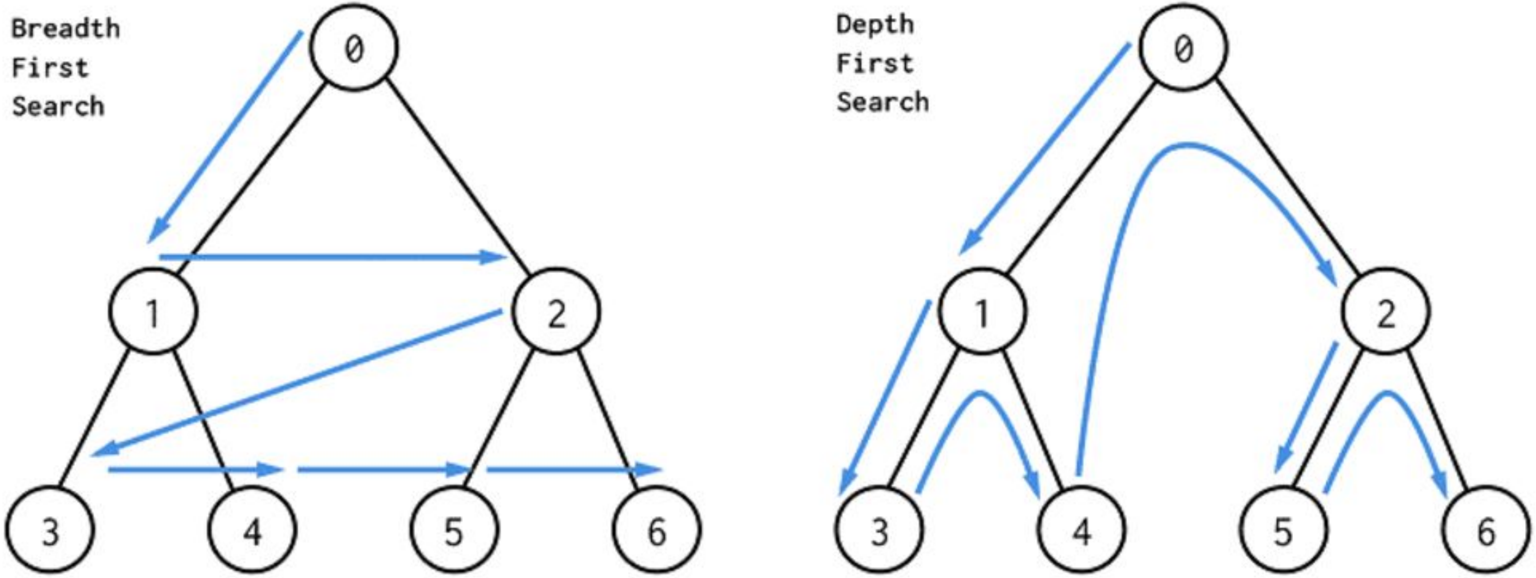

(g)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q u y
3 3
(h)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q y
3
(i)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q ∅

DFS - Depth-First Search

- Implements LIFO list
  - Last-in-First-out
- Expands all nodes along one path, before expanding any nodes on the next path

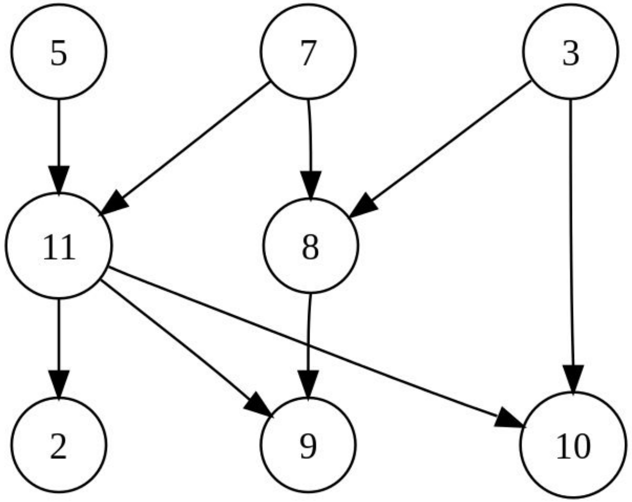


Comparison



Topoligical sorting

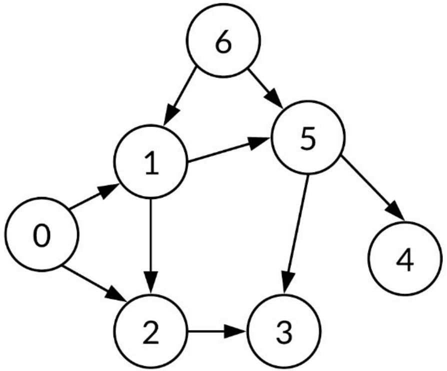
Linear ordering of it's vertices, such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering



The graph shown to the left has many valid topological sorts, including:

- 5, 7, 3, 11, 8, 2, 9, 10 (visual top-to-bottom, left-to-right)
- 3, 5, 7, 8, 11, 2, 9, 10 (smallest-numbered available vertex first)
- 5, 7, 3, 8, 11, 10, 9, 2 (fewest edges first)
- 7, 5, 11, 3, 10, 8, 9, 2 (largest-numbered available vertex first)
- 5, 7, 11, 2, 3, 8, 9, 10 (attempting top-to-bottom, left-to-right)
- 3, 7, 8, 5, 11, 10, 2, 9 (arbitrary)

Unsorted graph



Topologically
sorted graph