Algorithms Lab

BFS/DFS and Greedy Algorithms

Outline

- Exercise: Shelves
- Exercise: Even Matrices
- Graph traversals - (very) short reminder
    - Exercise: Race Tracks
- Greedy Algorithms
    - Exercise: Interval Scheduling

Exercise: Shelves

Goal: Find $a, b \in \mathbb{N}$ such that $am + bn \leq l$ and minimize $l - am - bn$ with b as high as possible

Exercise: Shelves

Goal: Find $a, b \in \mathbb{N}$ such that $am + bn \leq l$ and minimize $l - am - bn$ with b as high as possible

Naive solution 1:

```
for (int b = l / n; b>=0; b--)
  a = (l - b * n) / m;
  if (l - a * m - b * n < best)
    best = l - a * m - b * n;
    store (a, b);
```

Exercise: Shelves

Goal: Find $a, b \in \mathbb{N}$ such that $am + bn \leq l$ and minimize $l - am - bn$ with b as high as possible

Naive solution 1:

```
for (int b = l / n; b>=0; b--)
  a = (l - b * n) / m;
  if (l - a * m - b * n < best)
    best = l - a * m - b * n;
    store (a, b);
```

- We have l/n iterations.
- If n is "large", the algorithm is fast.
- But, if n is "small" the algorithm is slow(er)

Exercise: Shelves

Goal: Find $a, b \in \mathbb{N}$ such that $am + bn \leq l$ and minimize $l - am - bn$ with b as high as possible

Naive solution 2:

```
for (int a = 0; a <= l / m; a++)
  b = (l  - a * m) / n;
  if (l - a * m - b * n < best)
    best = l - a * m - b * n;
    store (a, b)
```

Exercise: Shelves

Goal: Find $a, b \in \mathbb{N}$ such that $am + bn \leq l$ and minimize $l - am - bn$ with b as high as possible

Naive solution 2:

```
for (int a = 0; a <= l / m; a++)
  b = (l  - a * m) / n;
  if (l - a * m - b * n < best)
    best = l - a * m - b * n;
    store (a, b)
```

- We have l/m iterations.
- Do we really need to go through all of them?

Exercise: Shelves

Goal: Find $a, b \in \mathbb{N}$ such that $am + bn \leq l$ and minimize $l - am - bn$ with b as high as possible

Naive solution 2:

```
for (int a = 0; a <= l / m; a++)
  b = (l  - a * m) / n;
  if (l - a * m - b * n < best)
    best = l - a * m - b * n;
    store (a, b)
```

- We have l/m iterations.
- Do we really need to go through all of them?

- What if $a = n + x$, for some $x > 0$.

Exercise: Shelves

Goal: Find $a, b \in \mathbb{N}$ such that $am + bn \leq l$ and minimize $l - am - bn$ with b as high as possible

Naive solution 2:

```
for (int a = 0; a <= l / m; a++)
  b = (l  - a * m) / n;
  if (l - a * m - b * n < best)
    best = l - a * m - b * n;
    store (a, b)
```

- We have l/m iterations.
- Do we really need to go through all of them?

- What if $a = n + x$, for some $x > 0$.

Note: $(n + x)m + bn$

Exercise: Shelves

Goal: Find $a, b \in \mathbb{N}$ such that $am + bn \leq l$ and minimize $l - am - bn$ with b as high as possible

Naive solution 2:

```
for (int a = 0; a <= l / m; a++)
  b = (l  - a * m) / n;
  if (l - a * m - b * n < best)
    best = l - a * m - b * n;
    store (a, b)
```

- We have l/m iterations.
- Do we really need to go through all of them?

- What if $a = n + x$, for some $x > 0$.

Note: $(n+x)m + bn = xm + (b+m)n$

Exercise: Shelves

Goal: Find $a, b \in \mathbb{N}$ such that $am + bn \leq l$ and minimize $l - am - bn$ with b as high as possible

Naive solution 2:

```
for (int a = 0; a <= l / m; a++)
  b = (l  - a * m) / n;
  if (l - a * m - b * n < best)
    best = l - a * m - b * n;
    store (a, b)
```

- We
- Do

- Wh

$(x, b + m)$ is better than $(n + x, b)$

Note: $(n + x)m + bn = xm + (b + m)n$

Exercise: Shelves

Goal: Find $a, b \in \mathbb{N}$ such that $am + bn \leq l$ and minimize $l - am - bn$ with b as high as possible

Conclusion

Naive solution 2 can iterate only until a = n.

```
for (int a = 0; a <= l / m; a++)
  b = (l  - a * m) / n;
  if (l - a * m - b * n < best)
    best = l - a * m - b * n;
    store (a, b)
```

Exercise: Shelves

Goal: Find $a, b \in \mathbb{N}$ such that $am + bn \leq l$ and minimize $l - am - bn$ with b as high as possible

Conclusion

Naive solution 2 can iterate only until a = n.

```
for (int a = 0; a <= l / m; a++)
   b = (l  - a * m) / n;
   if (l - a * m - b * n < best)
      best = l - a * m - b * n;
      store (a, b)
```

Remember!

We said Naive solution 1 does l/n iterations and is fast when n is large.

```
for (int b = l / n; b>=0; b--)
   a = (l - b * n) / m;
   if (l - a * m - b * n < best)
      best = l - a * m - b * n;
      store (a, b);
```

Exercise: Shelves

Goal: Find $a, b \in \mathbb{N}$ such that $am + bn \leq l$ and minimize $l - am - bn$ with b as high as possible

if $n \leq \sqrt{l}$

Naive solution 2 does at most $\sqrt{l}$ iterations!

```
for (int a = 0; a <= l / m; a++)
  b = (l  - a * m) / n;
  if (l - a * m - b * n < best)
    best = l - a * m - b * n;
    store (a, b)
```

```
for (int b = l / n; b>=0; b--)
  a = (l - b * n) / m;
  if (l - a * m - b * n < best)
    best = l - a * m - b * n;
    store (a, b);
```

Exercise: Shelves

Goal: Find $a, b \in \mathbb{N}$ such that $am + bn \leq l$ and minimize $l - am - bn$ with b as high as possible

if $n \leq \sqrt{l}$

Naive solution 2 does at most $\sqrt{l}$ iterations!

```
for (int a = 0; a <= l / m; a++)
  b = (l  - a * m) / n;
  if (l - a * m - b * n < best)
    best = l - a * m - b * n;
    store (a, b)
```

else if $n > \sqrt{l}$

Naive solution 1 does $l/n < \sqrt{l}$ iterations!

```
for (int b = l / n; b>=0; b--)
  a = (l - b * n) / m;
  if (l - a * m - b * n < best)
    best = l - a * m - b * n;
    store (a, b);
```

Exercise: Shelves

Goal: Find $a, b \in \mathbb{N}$ such that $am + bn \leq l$ and minimize $l - am - bn$ with b as high as possible

if $n < \sqrt{l}$

Naive solution
$\sqrt{l}$ iterations!

```
n; a++)
best)
* n;
```

Total complexity: $\mathcal{O}(\sqrt{l})$

else if n

Naive solution I does $l/n < \sqrt{l}$ iterations!

```
for (int b = l / n; b>=0; b--)
  a = (l - b * n) / m;
  if (l - a * m - b * n < best)
    best = l - a * m - b * n;
    store (a, b);
```

Exercise: Even Matrices

Goal: Find the number of (i_1, i_2, j_1, j_2) , $i_1 \leq i_2, j_1 \leq j_2$

such that $\displaystyle\sum_{i'=i_1}^{i_2} \sum_{j'=j_1}^{j_2} x_{i',j'}$ is even.

Exercise: Even Matrices

Goal: Find the number of (i_1, i_2, j_1, j_2) , $i_1 \le i_2, j_1 \le j_2$

such that $\sum_{i'=i_1}^{i_2} \sum_{j'=j_1}^{j_2} x_{i',j'}$ is even.

		i_1		i_2	
	1	0	1	1	0
	1	0	1	0	1
j_1	0	1	1	0	1
	0	0	1	0	0
	1	0	1	0	1
j_2	0	1	0	1	0

- Geometry - rectangles

Exercise: Even Matrices

Goal: Find the number of (i_1, i_2, j_1, j_2) , $i_1 \le i_2, j_1 \le j_2$

such that $\displaystyle\sum_{i'=i_1}^{i_2} \sum_{j'=j_1}^{j_2} x_{i',j'}$ is even.

- Geometry - rectangles

		i_1		i_2	
	1	0	1	1	0
	1	0	1	0	1
j_1	0	1	1	0	1
	0	0	1	0	0
	1	0	1	0	1
j_2	0	1	0	1	0

Exercise: Even Matrices

Goal: Find the number of (i_1, i_2, j_1, j_2) , $i_1 \leq i_2, j_1 \leq j_2$

such that $\sum_{i'=i_1}^{i_2} \sum_{j'=j_1}^{j_2} x_{i',j'}$ is even.

- Simple solution :
  for each quadruple, calculate the sum by iterating over all elements of the rectangle

- $\mathcal{O}(n^6)$ - 20 points

Exercise: Even Matrices

Goal: Find the number of (i_1, i_2, j_1, j_2) , $i_1 \le i_2, j_1 \le j_2$

such that $\sum_{i'=i_1}^{i_2} \sum_{j'=j_1}^{j_2} x_{i',j'}$ is even.

1	0	1	1	0
1	0	1	0	1
0	1	1	0	1
0	0	1	0	0
1	0	1	0	1
0	1	0	1	0

- Improved solution
  - Let $S_{i,j} = \sum_{i'=1}^{i} \sum_{j'=1}^{j} x_{i',j'}$

Exercise: Even Matrices

Goal: Find the number of (i_1, i_2, j_1, j_2) , $i_1 \leq i_2, j_1 \leq j_2$

such that $\displaystyle\sum_{i'=i_1}^{i_2} \sum_{j'=j_1}^{j_2} x_{i',j'}$ is even.

- Improved solution
  - Let $\displaystyle S_{i,j} = \sum_{i'=1}^{i} \sum_{j'=1}^{j} x_{i',j'}$

1	0	1	1	0
1	0	1	0	1
0	1	1	0	1
0	0	1	0	0
1	0	1	0	1
0	1	0	1	0

Exercise: Even Matrices

Goal: Find the number of (i_1, i_2, j_1, j_2) , $i_1 \le i_2, j_1 \le j_2$

such that $\sum_{i'=i_1}^{i_2} \sum_{j'=j_1}^{j_2} x_{i',j'}$ is even.

- Improved solution
  - Let $S_{i,j} = \sum_{i'=1}^{i} \sum_{j'=1}^{j} x_{i',j'}$

		i_1		i_2	
	1	0	1	1	0
	1	0	1	0	1
j_1	0	1	1	0	1
	0	0	1	0	0
	1	0	1	0	1
j_2	0	1	0	1	0

Exercise: Even Matrices

Goal: Find the number of (i_1, i_2, j_1, j_2) , $i_1 \le i_2, j_1 \le j_2$ such that $\sum_{i'=i_1}^{i_2} \sum_{j'=j_1}^{j_2} x_{i',j'}$ is even.

- Improved solution
  - Let $S_{i,j} = \sum_{i'=1}^{i} \sum_{j'=1}^{j} x_{i',j'}$
  - S_{i_2,j_2}

$$
\begin{array}{c|ccccc|}
 & & i_1 & & i_2 & \\
\hline
 & 1 & 0 & 1 & 1 & 0 \\
 & 1 & 0 & 1 & 0 & 1 \\
j_1 & 0 & 1 & 1 & 0 & 1 \\
 & 0 & 0 & 1 & 0 & 0 \\
 & 1 & 0 & 1 & 0 & 1 \\
j_2 & 0 & 1 & 0 & 1 & 0 \\
\hline
\end{array}
$$

Exercise: Even Matrices

Goal: Find the number of (i_1, i_2, j_1, j_2) , $i_1 \leq i_2, j_1 \leq j_2$

such that $\sum_{i'=i_1}^{i_2} \sum_{j'=j_1}^{j_2} x_{i',j'}$ is even.

		i_1		i_2	
	1	0	1	1	0
	1	0	1	0	1
j_1	0	1	1	0	1
	0	0	1	0	0
	1	0	1	0	1
j_2	0	1	0	1	0

- Improved solution
  - Let $S_{i,j} = \sum_{i'=1}^{i} \sum_{j'=1}^{j} x_{i',j'}$
  - $S_{i_2,j_2} - S_{i_1-1,j_2}$

Exercise: Even Matrices

Goal: Find the number of (i_1, i_2, j_1, j_2) , $i_1 \le i_2, j_1 \le j_2$

such that $\sum_{i'=i_1}^{i_2} \sum_{j'=j_1}^{j_2} x_{i',j'}$ is even.

- Improved solution
  - Let $S_{i,j} = \sum_{i'=1}^{i} \sum_{j'=1}^{j} x_{i',j'}$
  - $S_{i_2,j_2} - S_{i_1-1,j_2}$
    $-S_{i_2,j_1-1}$

		i_1		i_2	
	1	0	1	1	0
	1	0	1	0	1
j_1	0	1	1	0	1
	0	0	1	0	0
	1	0	1	0	1
j_2	0	1	0	1	0

Exercise: Even Matrices

Goal: Find the number of (i_1, i_2, j_1, j_2) , $i_1 \leq i_2, j_1 \leq j_2$

such that $\sum_{i'=i_1}^{i_2} \sum_{j'=j_1}^{j_2} x_{i',j'}$ is even.

- Improved solution
  - Let $S_{i,j} = \sum_{i'=1}^{i} \sum_{j'=1}^{j} x_{i',j'}$
  - $S_{i_2,j_2} - S_{i_1-1,j_2}$
    $- S_{i_2,j_1-1} + S_{i_1-1,j_1-1}$

		i_1		i_2	
	1	0	1	1	0
	1	0	1	0	1
j_1	0	1	1	0	1
	0	0	1	0	0
	1	0	1	0	1
j_2	0	1	0	1	0

Exercise: Even Matrices

Goal: Find the number of (i_1, i_2, j_1, j_2) , $i_1 \leq i_2, j_1 \leq j_2$

such that $\sum_{i'=i_1}^{i_2} \sum_{j'=j_1}^{j_2} x_{i',j'}$ is even.

- Calculate $S_{i,j}$ - $\mathcal{O}(n^2)$
- Total running time $\mathcal{O}(n^4)$ - 70 points

Exercise: Even Matrices

- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

	1	0	1	1	0
i_1	1	0	1	0	1
	0	1	1	0	1
	0	0	1	0	0
i_2	1	0	1	0	1
	0	1	0	1	0

Exercise: Even Matrices

- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

	1	0	1	1	0
i_1	1	0	1	0	1
	0	1	1	0	1
	0	0	1	0	0
i_2	1	0	1	0	1
	0	1	0	1	0

Exercise: Even Matrices

- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

	1	0	1	1	0
i_1	1	0	1	0	1
	0	1	1	0	1
	0	0	1	0	0
i_2	1	0	1	0	1
	0	1	0	1	0

Exercise: Even Matrices

- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

	1	0	1	1	0
i_1	1	0	1	0	1
	0	1	1	0	1
	0	0	1	0	0
i_2	1	0	1	0	1
	0	1	0	1	0

Exercise: Even Matrices

- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

	1	0	1	1	0
i_1	1	0	1	0	1
	0	1	1	0	1
	0	0	1	0	0
i_2	1	0	1	0	1
	0	1	0	1	0

Exercise: Even Matrices

- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

$$
\begin{array}{c}
\\ i_1 \\ \\ \\ i_2 \\ \\
\end{array}
\begin{bmatrix}
1 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 0
\end{bmatrix}
$$

Exercise: Even Matrices

- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

	1	0	1	1	0
i_1	1	0	1	0	1
	0	1	1	0	1
	0	0	1	0	0
i_2	1	0	1	0	1
	0	1	0	1	0

Exercise: Even Matrices

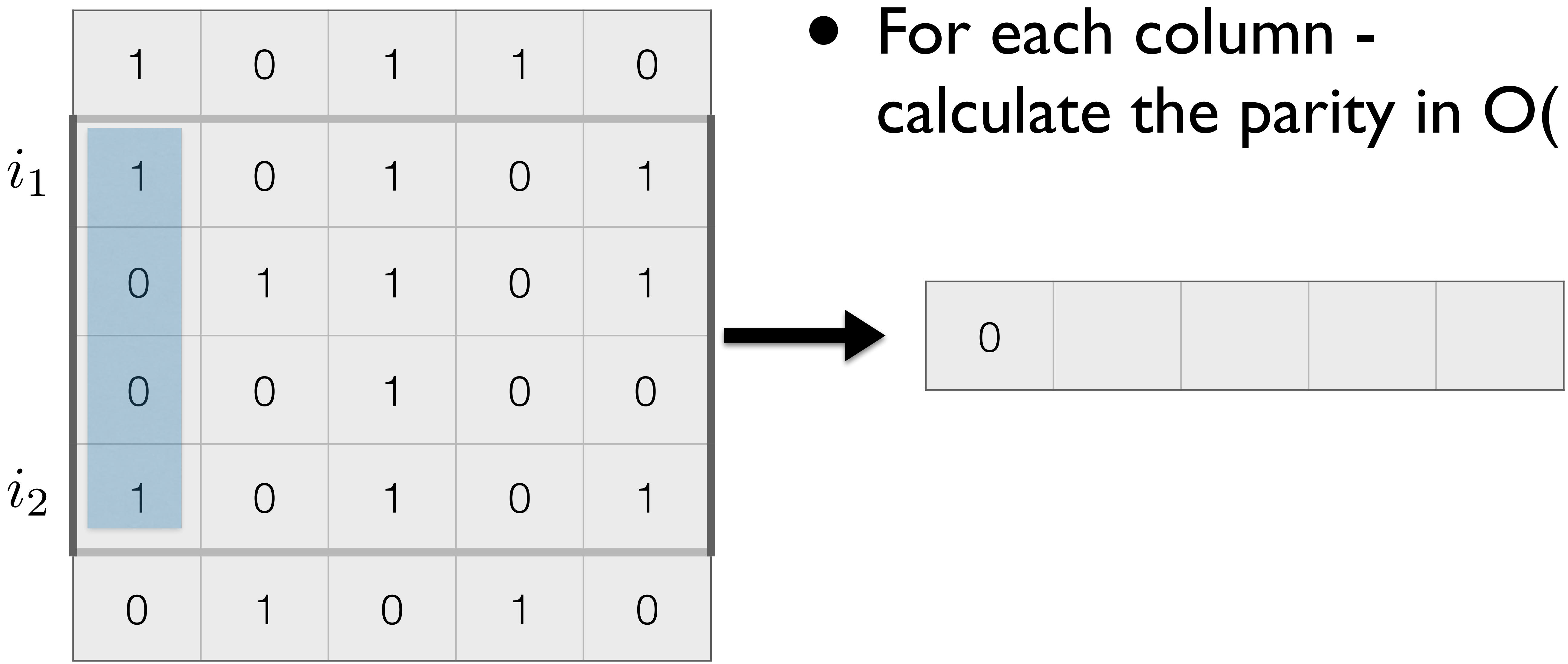
- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

	1	0	1	1	0
i_1	1	0	1	0	1
	0	1	1	0	1
	0	0	1	0	0
i_2	1	0	1	0	1
	0	1	0	1	0

Exercise: Even Matrices

- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

- For each column - calculate the parity in O(1)



Exercise: Even Matrices

- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

$$
\begin{array}{c|ccccc|}
 & 1 & 0 & 1 & 1 & 0 \\
i_1 & 1 & 0 & 1 & 0 & 1 \\
 & 0 & 1 & 1 & 0 & 1 \\
 & 0 & 0 & 1 & 0 & 0 \\
i_2 & 1 & 0 & 1 & 0 & 1 \\
 & 0 & 1 & 0 & 1 & 0
\end{array}
\longrightarrow
\begin{array}{|c|c|c|c|c|}
\hline
0 & 1 & & & \\
\hline
\end{array}
$$

- For each column - calculate the parity in O(1)

Exercise: Even Matrices

- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

	1	0	1	1	0
i_1	1	0	1	0	1
	0	1	1	0	1
	0	0	1	0	0
i_2	1	0	1	0	1
	0	1	0	1	0

- For each column - calculate the parity in O(1)

0	1	0		

Exercise: Even Matrices

- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

	1	0	1	1	0
i_1	1	0	1	0	1
	0	1	1	0	1
	0	0	1	0	0
i_2	1	0	1	0	1
	0	1	0	1	0

- For each column - calculate the parity in O(1)

0	1	0	0	

Exercise: Even Matrices

- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

	1	0	1	1	0
i_1	1	0	1	0	1
	0	1	1	0	1
	0	0	1	0	0
i_2	1	0	1	0	1
	0	1	0	1	0

- For each column - calculate the parity in O(1)

$\rightarrow$

0	1	0	0	1

Exercise: Even Matrices

- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

	1	0	1	1	0
i_1	1	0	1	0	1
	0	1	1	0	1
	0	0	1	0	0
i_2	1	0	1	0	1
	0	1	0	1	0

- For each column - calculate the parity in O(1)

0	1	0	0	1

Exercise: Even Matrices

1	0	1	0	1
0	1	1	0	1
0	0	1	0	0
1	0	1	0	1

0	1	0	0	1

Exercise: Even Matrices

1	0	1	0	1
0	1	1	0	1
0	0	1	0	0
1	0	1	0	1

0	1	0	0	1

Exercise: Even Matrices

1	0	1	0	1
0	1	1	0	1
0	0	1	0	0
1	0	1	0	1

0	1	0	0	1

same parity

Exercise: Even Matrices

1	0	1	0	1
0	1	1	0	1
0	0	1	0	0
1	0	1	0	1

0	1	0	0	1

same parity

Exercise: Even Matrices

	1	0	1	0	1
i_1	0	1	1	0	1
	0	0	1	0	0
i_2	1	0	1	0	1

0	1	0	0	1

same parity

Exercise: Even Matrices

	1	0	1	0	1
i_1	0	1	1	0	1
	0	0	1	0	0
i_2	1	0	1	0	1

0	1	0	0	1

same parity

Exercise: Even Matrices

1	0	1	0	1
0	1	1	0	1
0	0	1	0	0
1	0	1	0	1

0	1	0	0	1

Conclusion:

Number of even rectangles (left)

=

Number of even subsequences (right)

Exercise: Even Matrices

- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

	1	0	1	1	0
i_1	1	0	1	0	1
	0	1	1	0	1
	0	0	1	0	0
i_2	1	0	1	0	1
	0	1	0	1	0

- We have reduced the subproblem to 1-dim case (even pairs prob)

Exercise: Even Matrices

- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

	1	0	1	1	0
i_1	1	0	1	0	1
	0	1	1	0	1
	0	0	1	0	0
i_2	1	0	1	0	1
	0	1	0	1	0

- We have reduced the subproblem to 1-dim case (even pairs prob)

- 1-dim case can be solve in linear time

Exercise: Even Matrices

- Fix i_1, i_2 and consider only rectangles which contain rows from i_1 to i_2

	1	0	1	1	0
i_1	1	0	1	0	1
	0	1	1	0	1
	0	0	1	0	0
i_2	1	0	1	0	1
	0	1	0	1	0

- We have reduced the subproblem to 1-dim case (even pairs prob)
- 1-dim case can be solve in linear time
- How many subproblems?

Exercise: Even Matrices

- There are $\mathcal{O}(n^2)$ subproblems
- If implemented correctly, solution in $\mathcal{O}(n^3)$

Graph Traversals

- A graph is an ordered pair $G = (V, E)$
- V - set of vertices
- E - set of edges

Graph Traversals

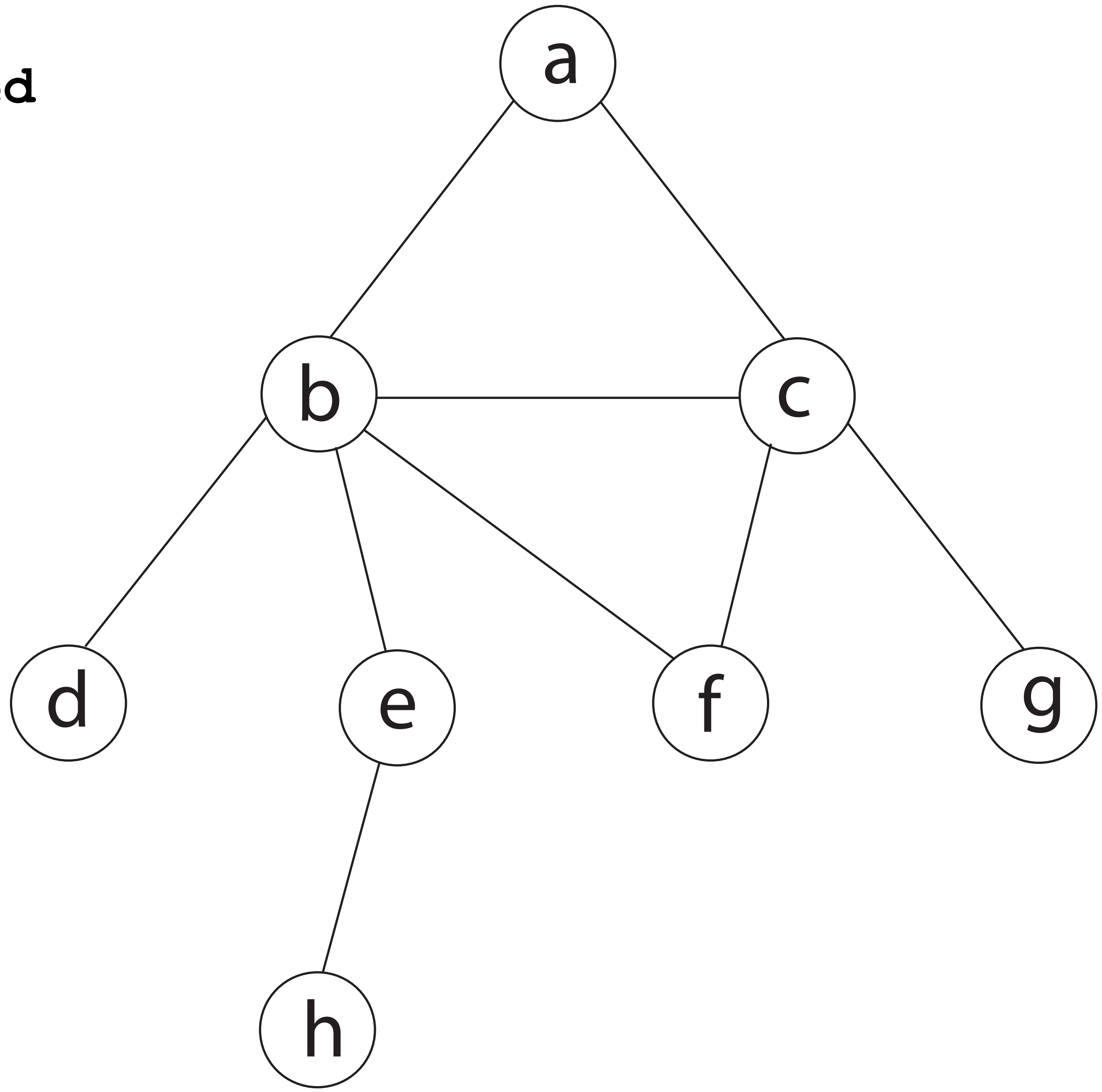
- Graphs are used to model relations between elements of a set
- Very general concept, hence often applicable.

Graph Traversals

- DFS and BFS are basic building blocks of most of the graph algorithms
- Both have time complexity of $O(|E|)$

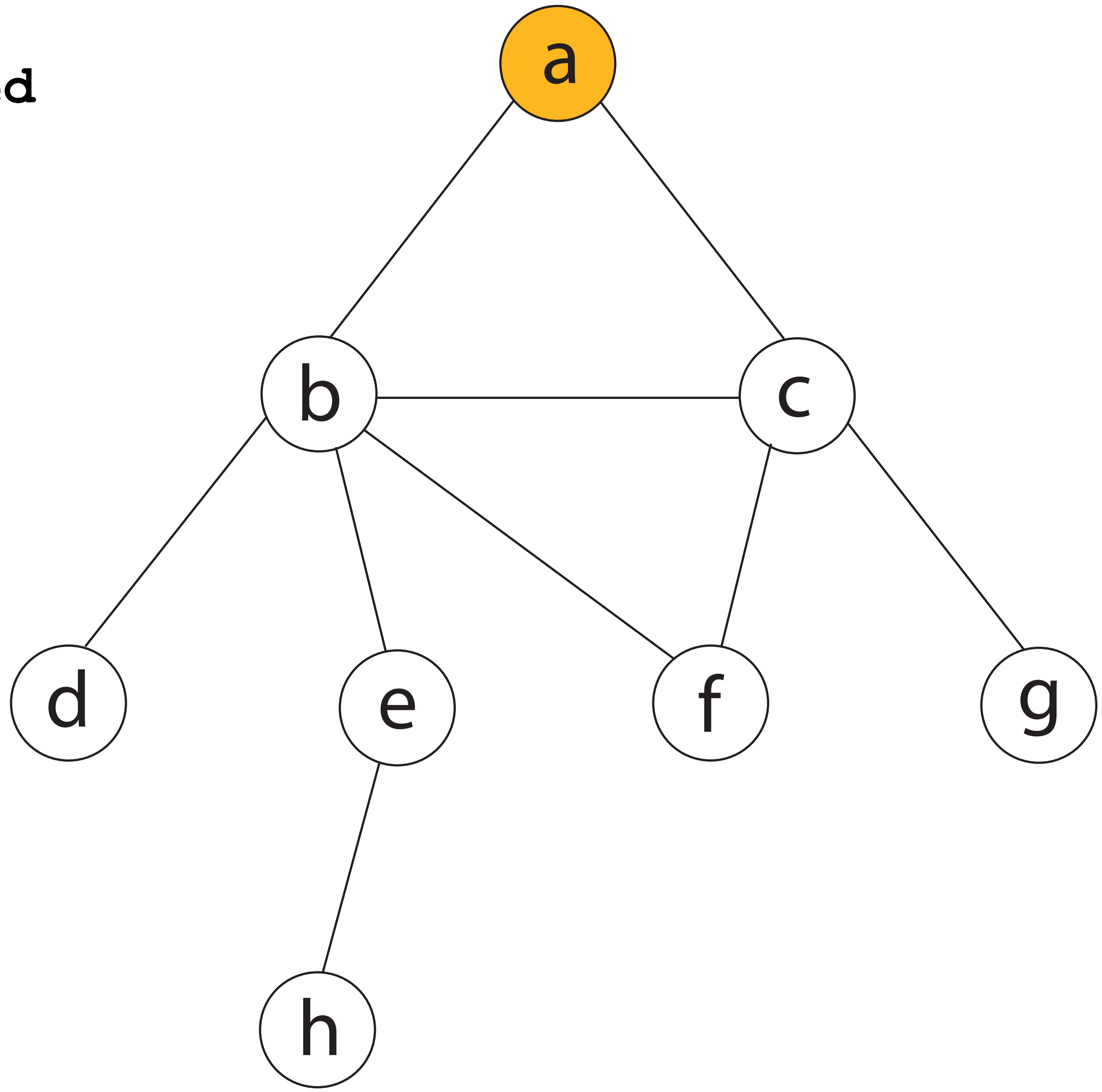
BFS - Breadth First Search

```
initialize queue q
mark all vertices as not visited
push starting vertex 'a' to q
mark 'a' as visited

while q not empty {
   v = front of q
   delete front of q
    for all neighbors u of v {
      if u is not visited {
        mark u as visited
        push u to q
      }
    }
}
```



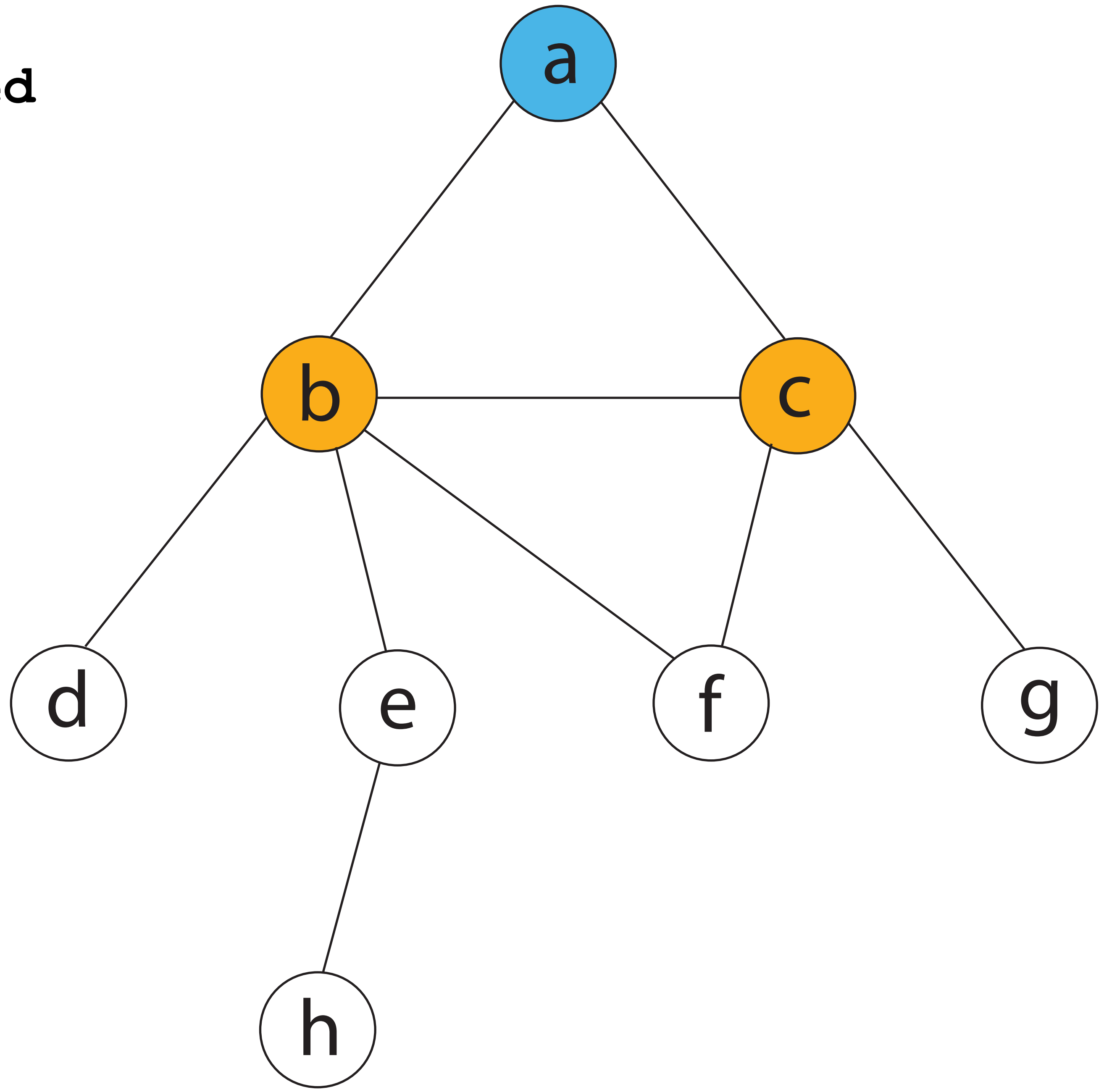
BFS - Breadth First Search

```
initialize queue q
mark all vertices as not visited
push starting vertex 'a' to q
mark 'a' as visited

while q not empty {
   v = front of q
   delete front of q
    for all neighbors u of v {
      if u is not visited {
        mark u as visited
        push u to q
      }
    }
}
```



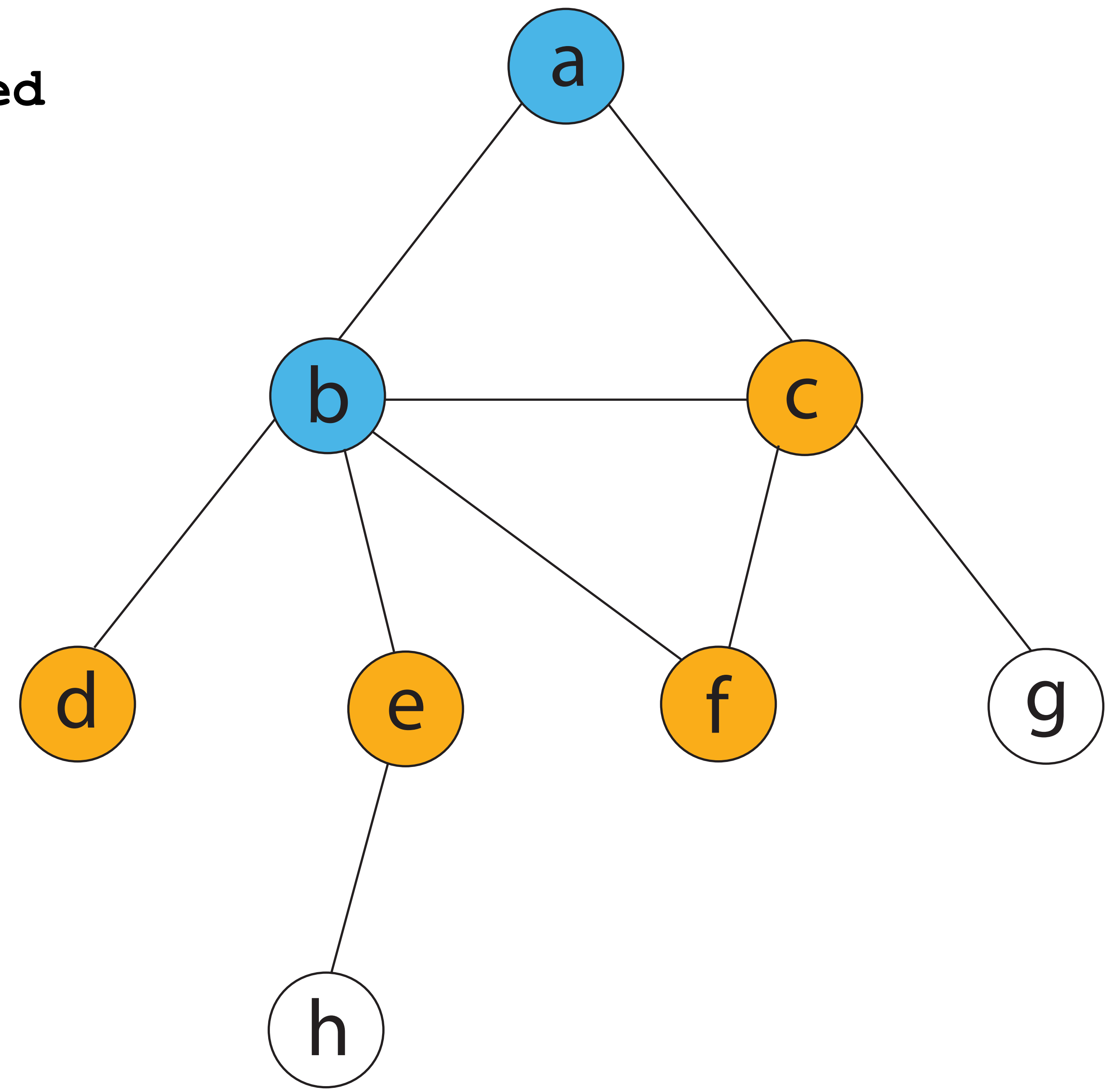
BFS - Breadth First Search

```
initialize queue q
mark all vertices as not visited
push starting vertex ‘a’ to q
mark ‘a’ as visited

while q not empty {
   v = front of q
   delete front of q
    for all neighbors u of v {
      if u is not visited {
        mark u as visited
        push u to q
      }
    }
}
```



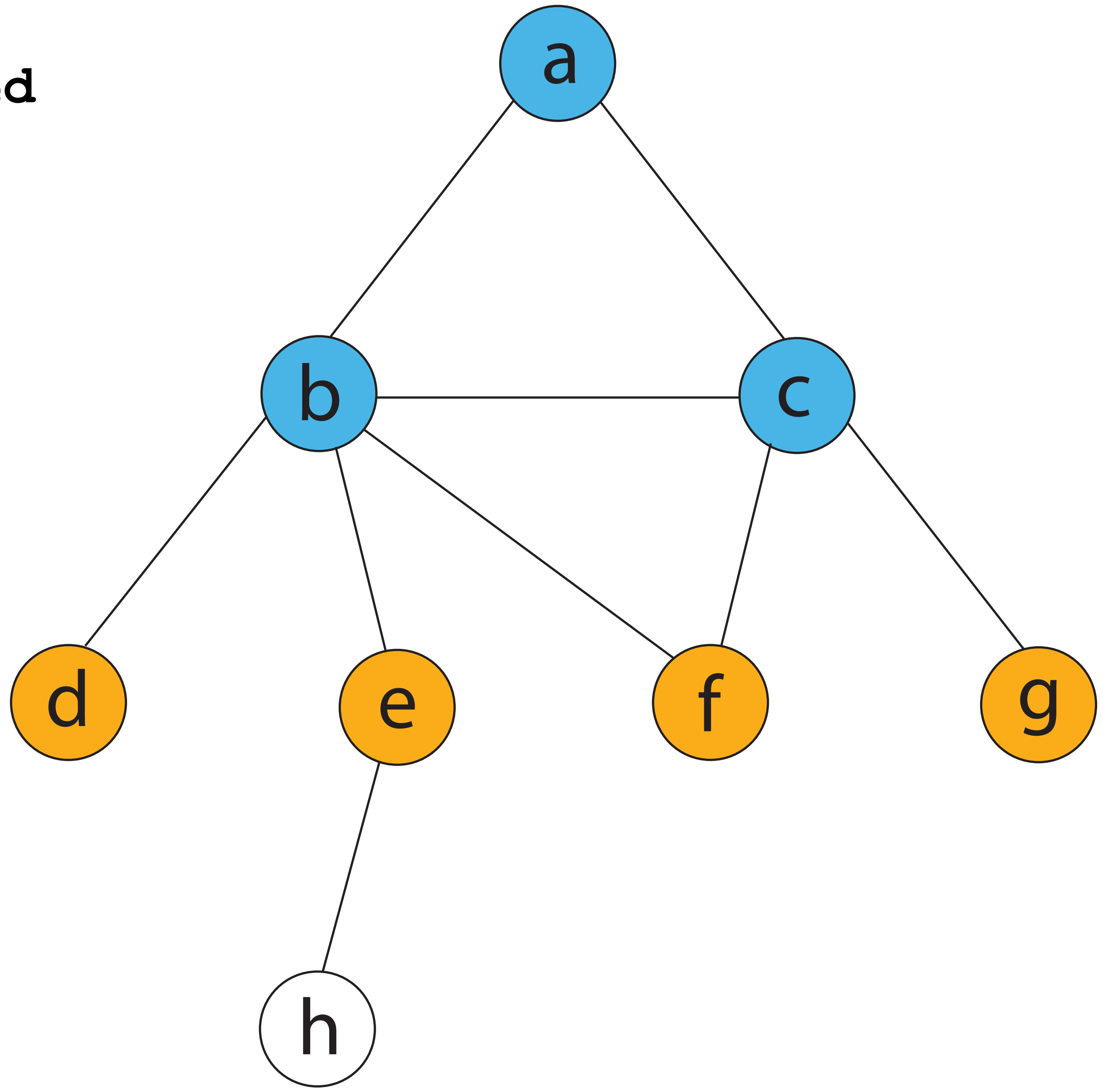
BFS - Breadth First Search

```
initialize queue q
mark all vertices as not visited
push starting vertex 'a' to q
mark 'a' as visited

while q not empty {
   v = front of q
   delete front of q
    for all neighbors u of v {
      if u is not visited {
        mark u as visited
        push u to q
      }
    }
}
```



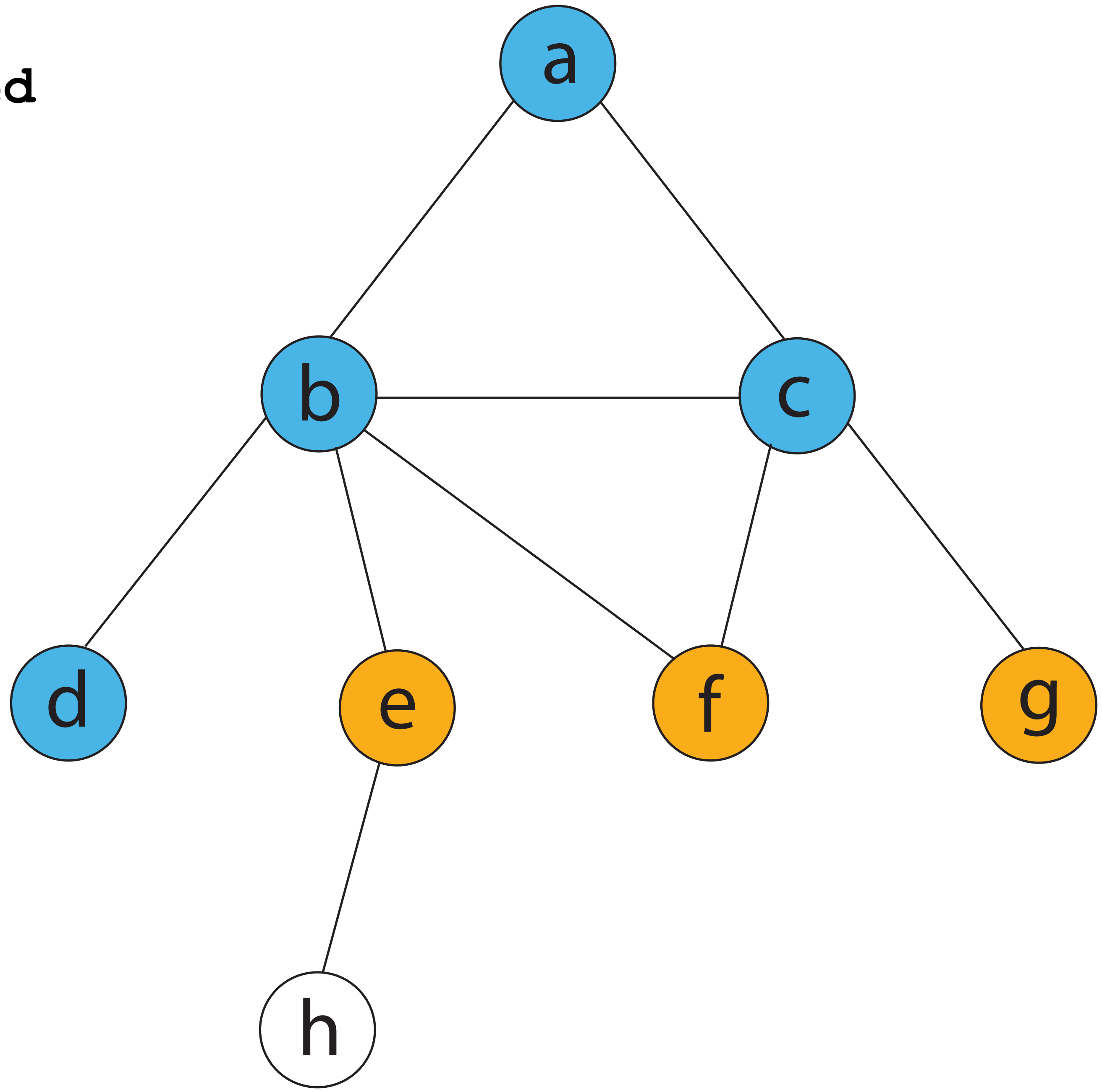
BFS - Breadth First Search

```
initialize queue q
mark all vertices as not visited
push starting vertex 'a' to q
mark 'a' as visited

while q not empty {
   v = front of q
   delete front of q
    for all neighbors u of v {
      if u is not visited {
        mark u as visited
        push u to q
      }
    }
}
```



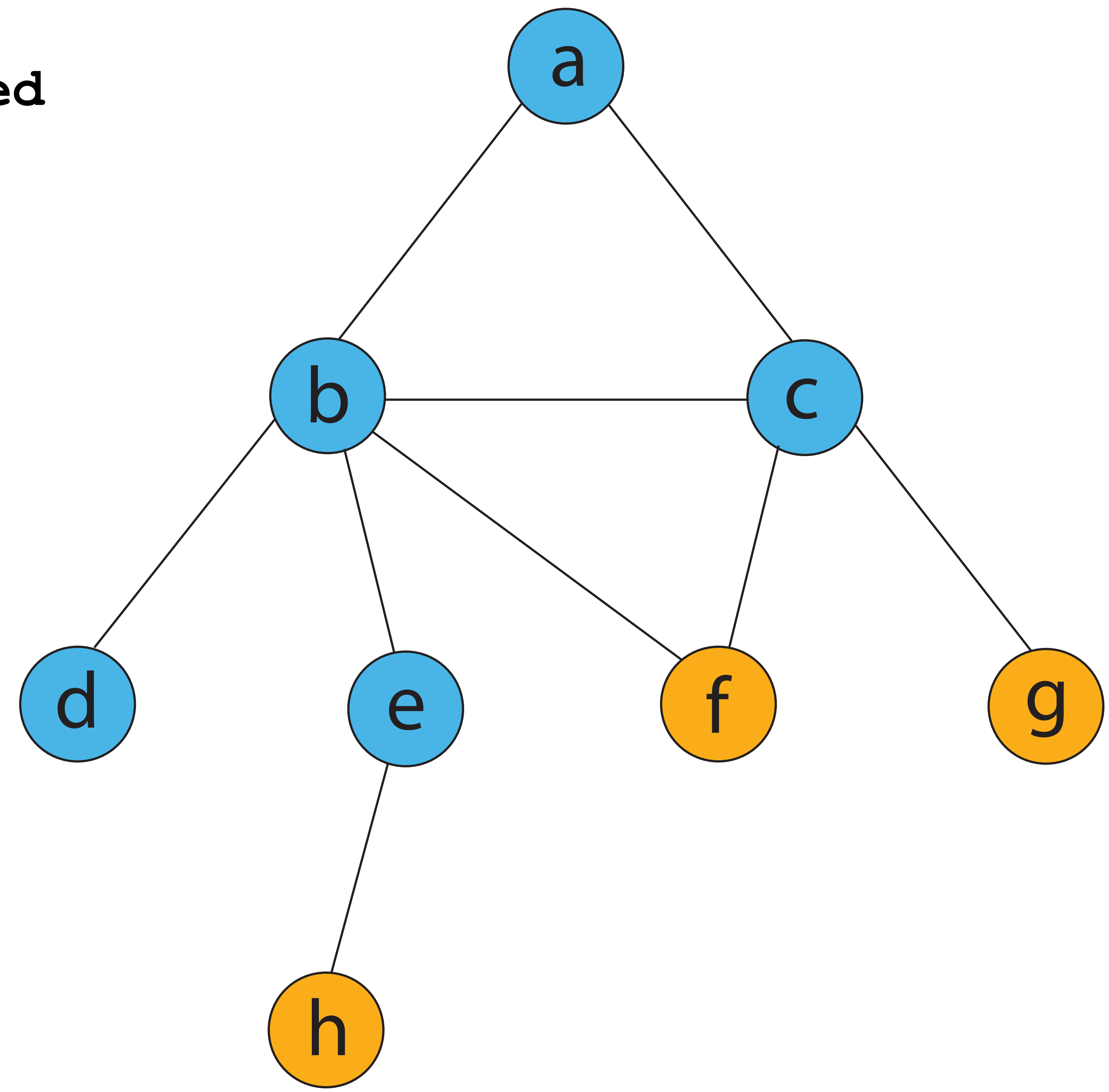
BFS - Breadth First Search

```
initialize queue q
mark all vertices as not visited
push starting vertex 'a' to q
mark 'a' as visited

while q not empty {
   v = front of q
   delete front of q
    for all neighbors u of v {
      if u is not visited {
        mark u as visited
        push u to q
      }
    }
}
```



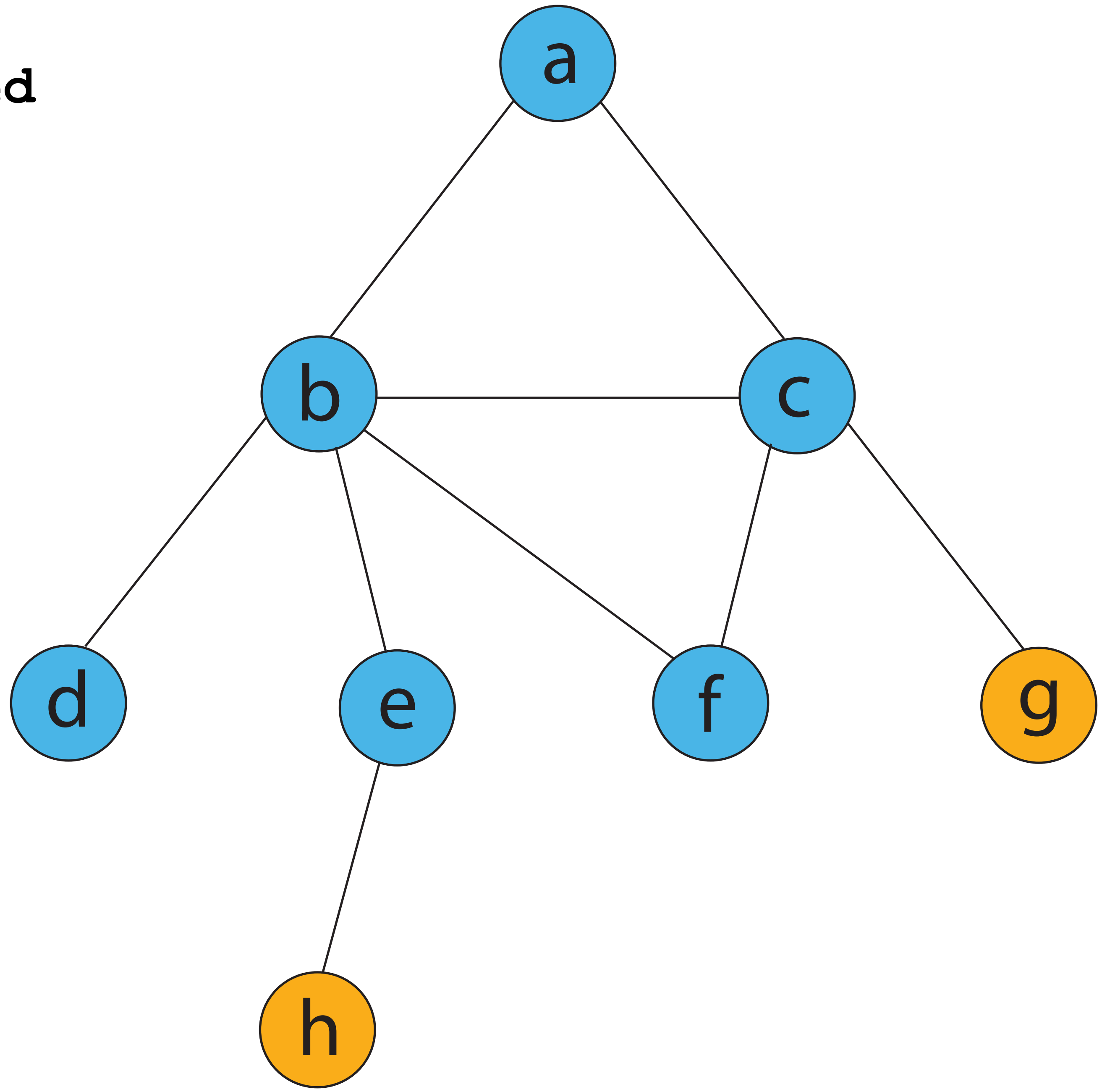
BFS - Breadth First Search

```
initialize queue q
mark all vertices as not visited
push starting vertex 'a' to q
mark 'a' as visited

while q not empty {
   v = front of q
   delete front of q
    for all neighbors u of v {
      if u is not visited {
        mark u as visited
        push u to q
      }
    }
}
```



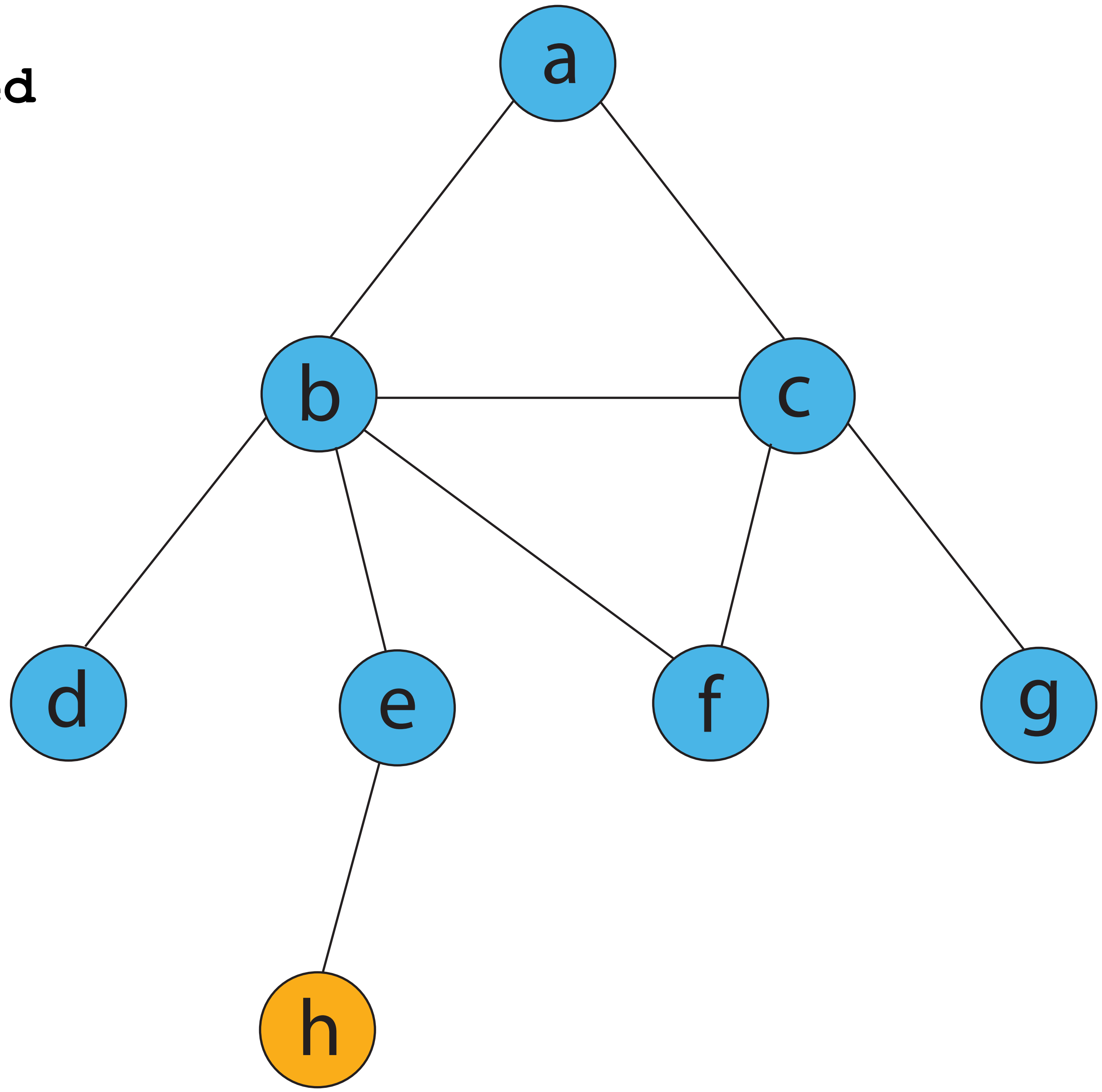
BFS - Breadth First Search

```
initialize queue q
mark all vertices as not visited
push starting vertex 'a' to q
mark 'a' as visited

while q not empty {
   v = front of q
   delete front of q
    for all neighbors u of v {
      if u is not visited {
        mark u as visited
        push u to q
      }
    }
}
```



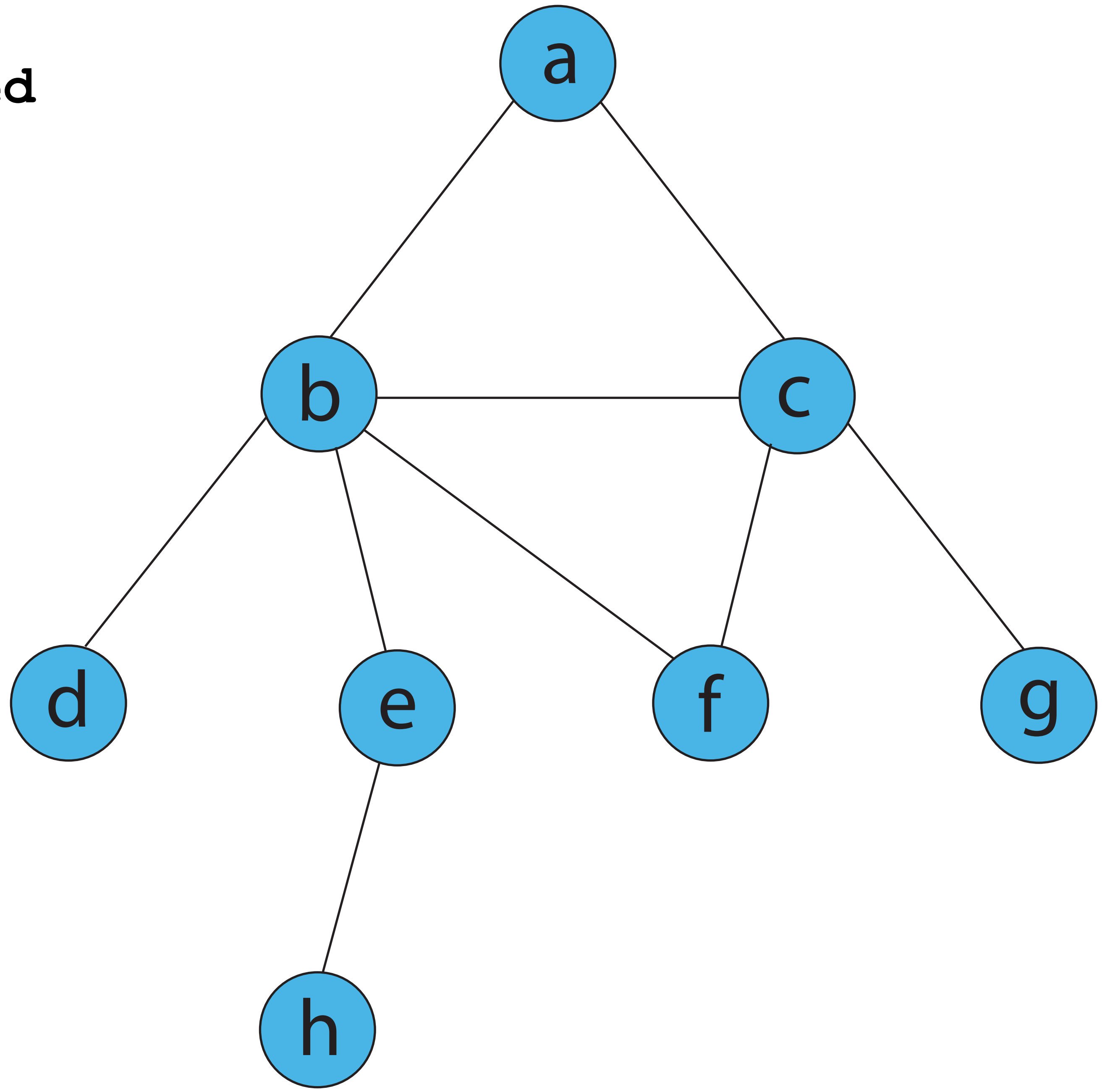
BFS - Breadth First Search

```
initialize queue q
mark all vertices as not visited
push starting vertex 'a' to q
mark 'a' as visited

while q not empty {
   v = front of q
   delete front of q
    for all neighbors u of v {
      if u is not visited {
        mark u as visited
        push u to q
      }
    }
}
```



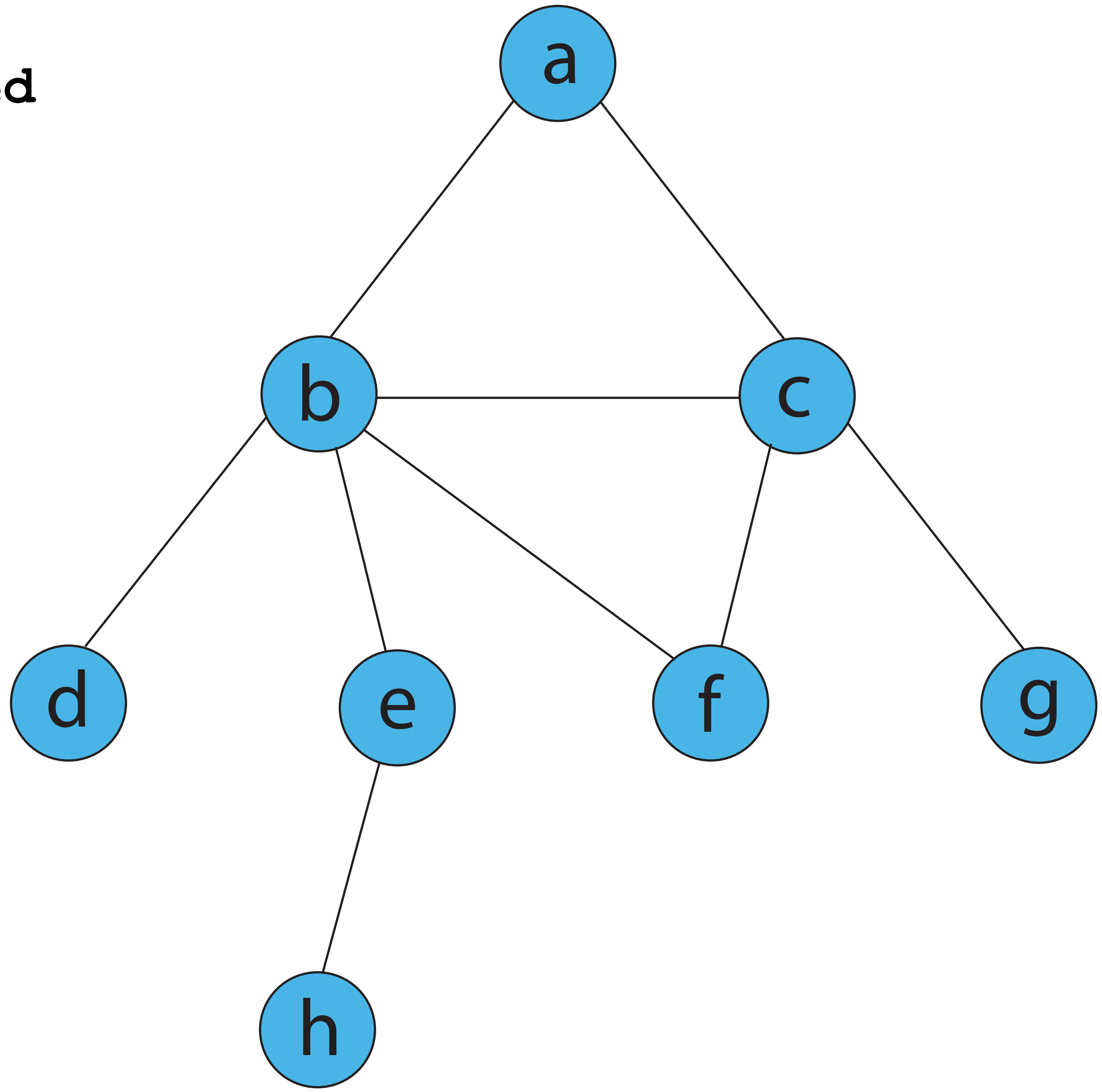
BFS - Breadth First Search

```
initialize queue q
mark all vertices as not visited
push starting vertex 'a' to q
mark 'a' as visited

while q not empty {
   v = front of q
   delete front of q
    for all neighbors u of v {
      if u is not visited {
        mark u as visited
        push u to q
      }
    }
}
```



BFS - Breadth First Search

```
initialize queue q
mark all vertices as not visited
push starting vertex 'a' to q
mark 'a' as visited

while q not empty {
   v = front of q
   delete front of q
    for all neighbors u of v {
      if u is not visited {
        mark u as visited
        push u to q
      }
    }
}
```



Order in which the vertices are visited is : a, b, c, d, e, f, g, h

BFS - Breadth First Search

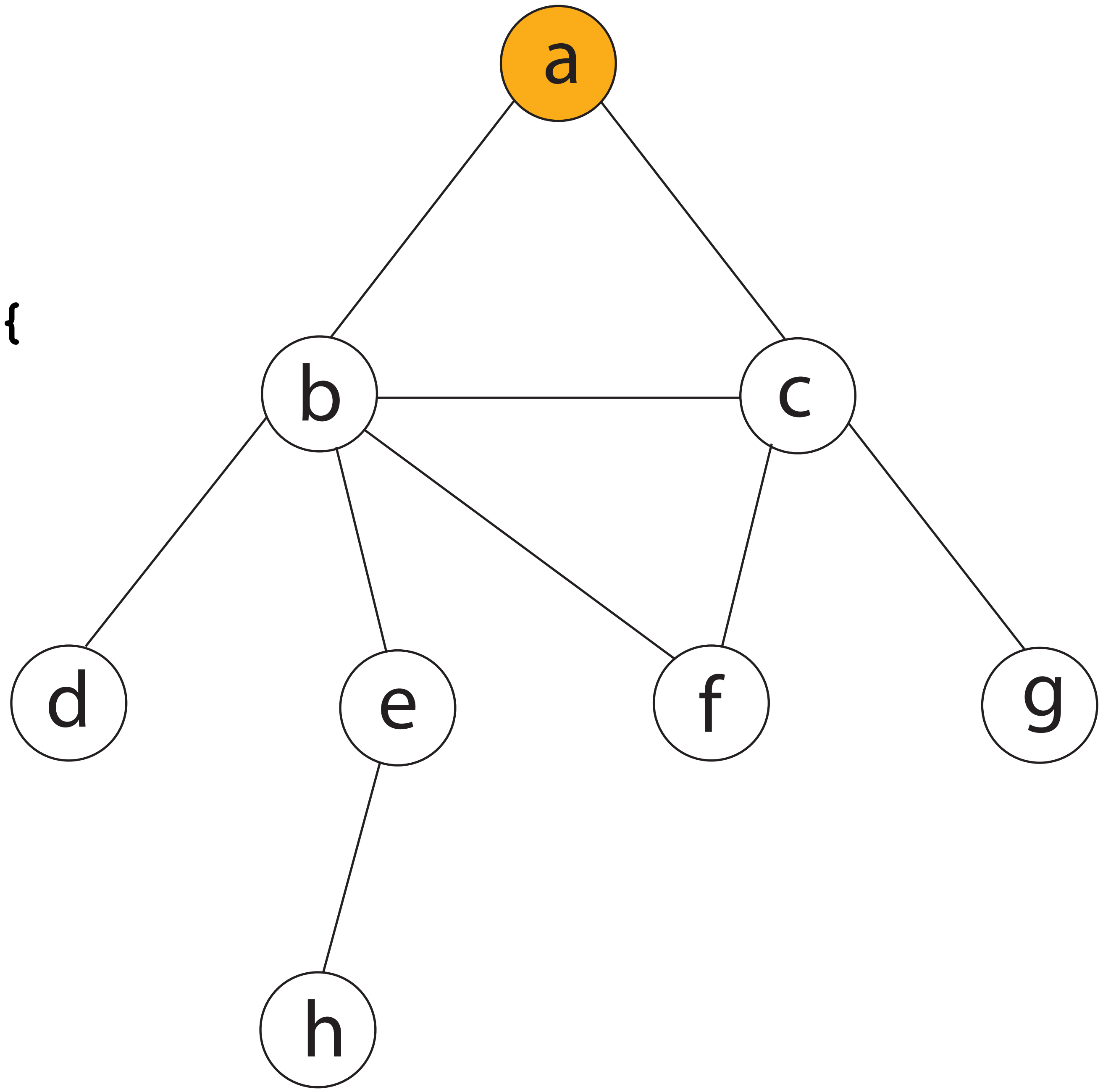
- BFS implicitly finds a shortest path from a starting vertex to any other. (Note: this is true only if the edges have the same weight.)
- You need to modify the code in order to construct the shortest path(s)

DFS - Depth First Search

- If you use stack instead of queue in the implementation of BFS, you get DFS
- Also, natural implementation of DFS with recursion

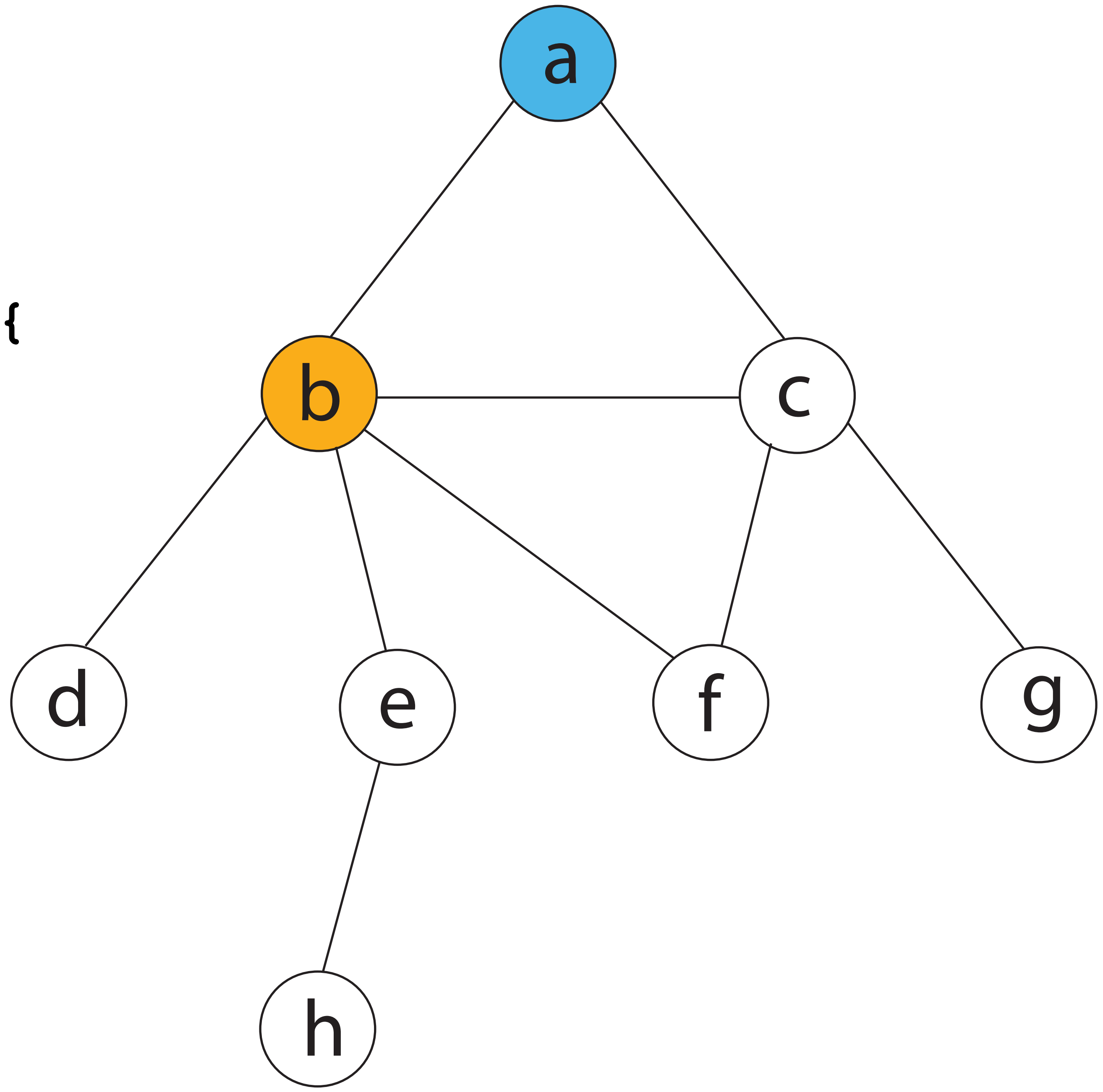
DFS - Depth First Search

```
proc DFS(vertex v){
  mark v as visited
  for all u neighbors of v {
    if u is not visited {
      DFS(u)
    }
  }
}
```



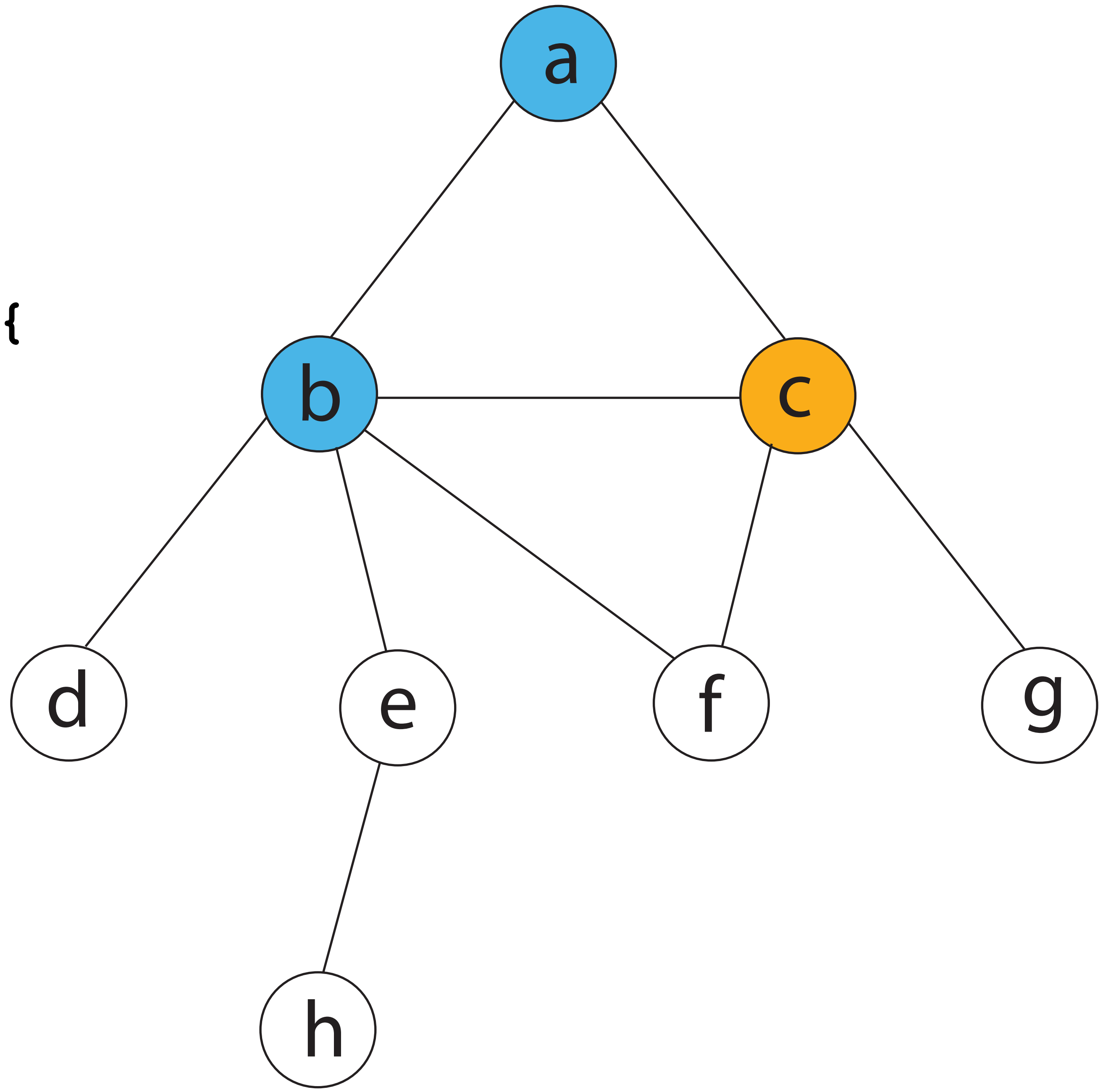
DFS - Depth First Search

```
proc DFS(vertex v){
  mark v as visited
  for all u neighbors of v {
    if u is not visited {
      DFS(u)
    }
  }
}
```



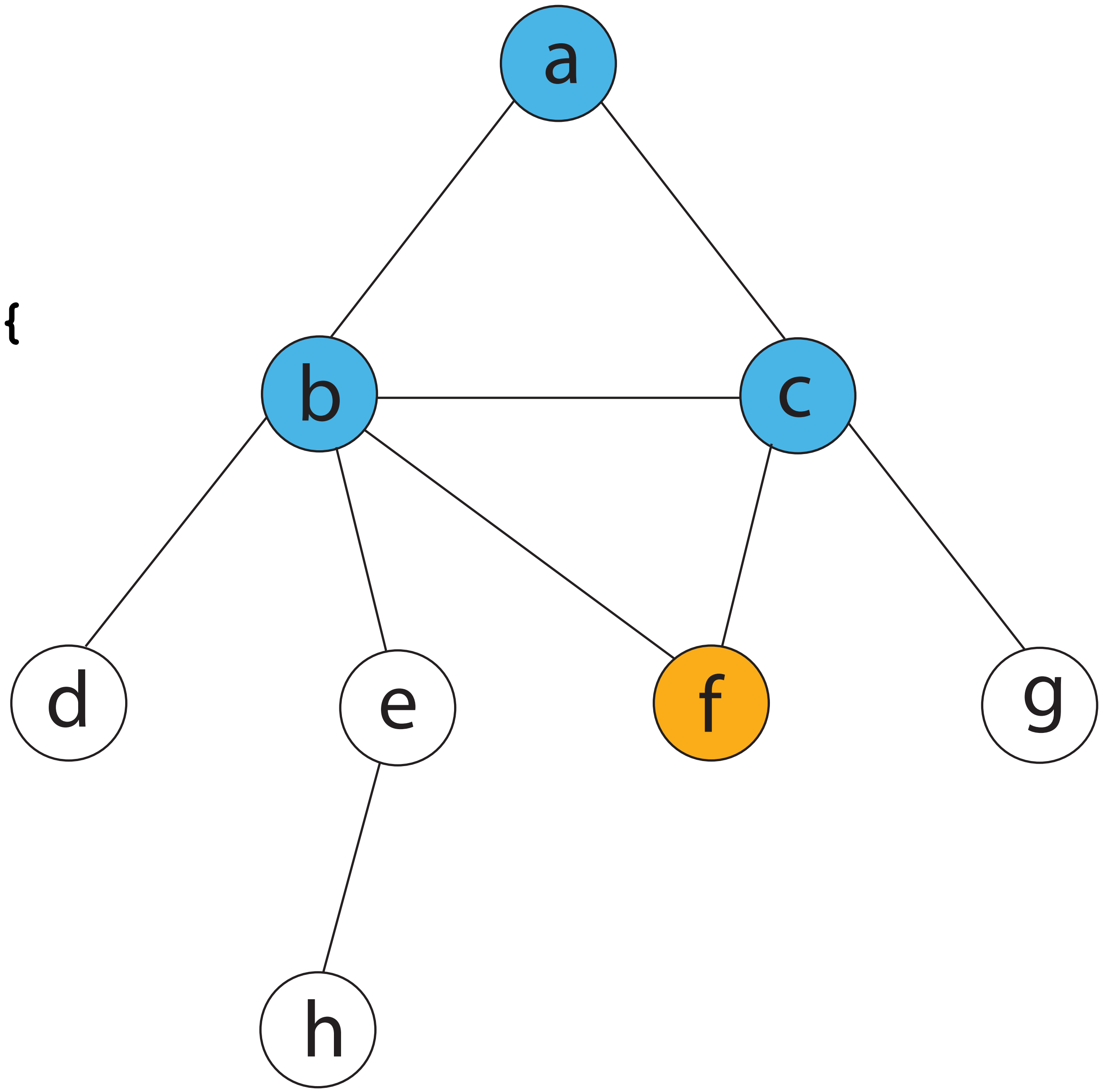
DFS - Depth First Search

```
proc DFS(vertex v){
  mark v as visited
  for all u neighbors of v {
    if u is not visited {
      DFS(u)
    }
  }
}
```



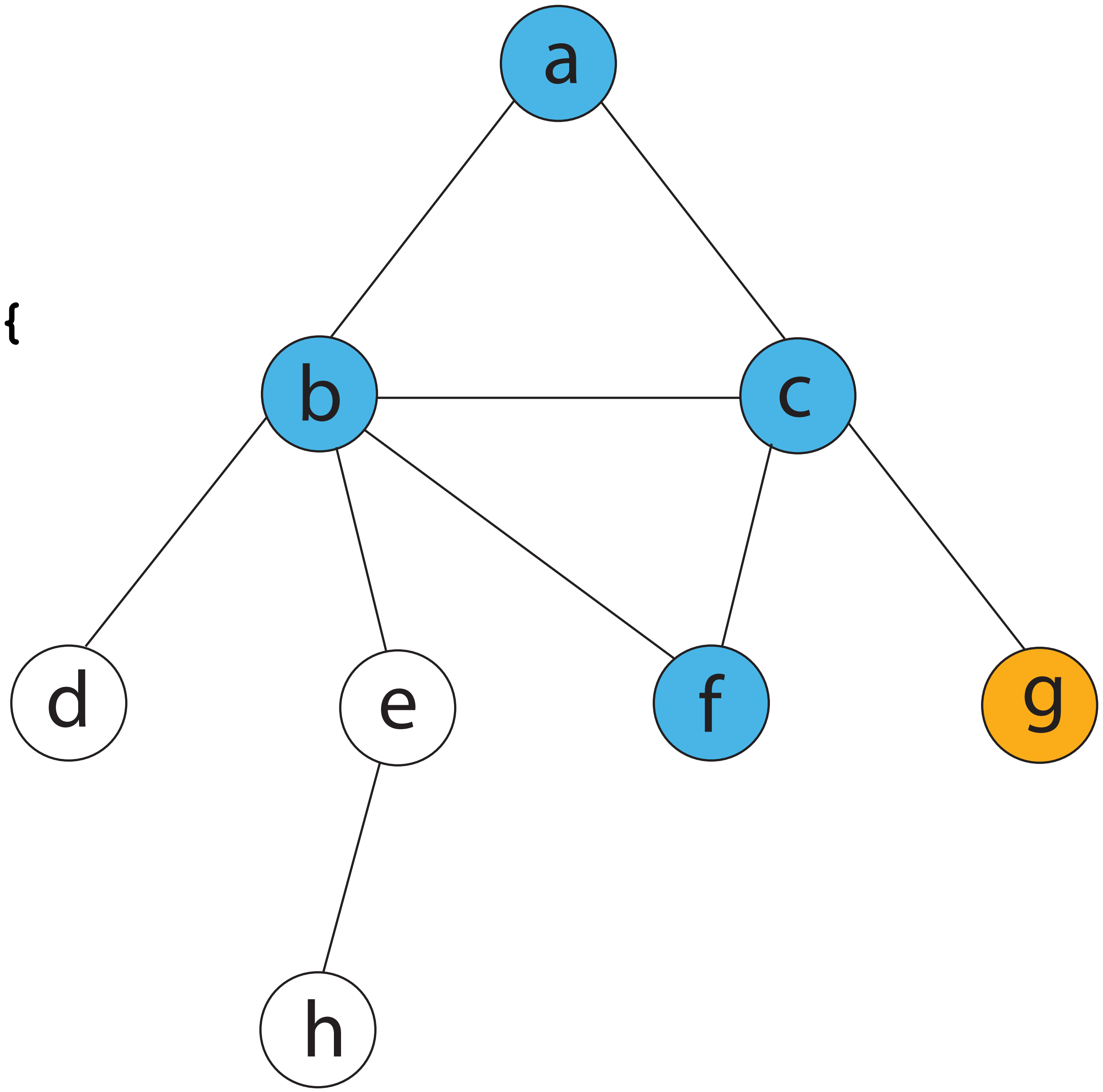
DFS - Depth First Search

```
proc DFS(vertex v){
  mark v as visited
  for all u neighbors of v {
    if u is not visited {
      DFS(u)
    }
  }
}
```



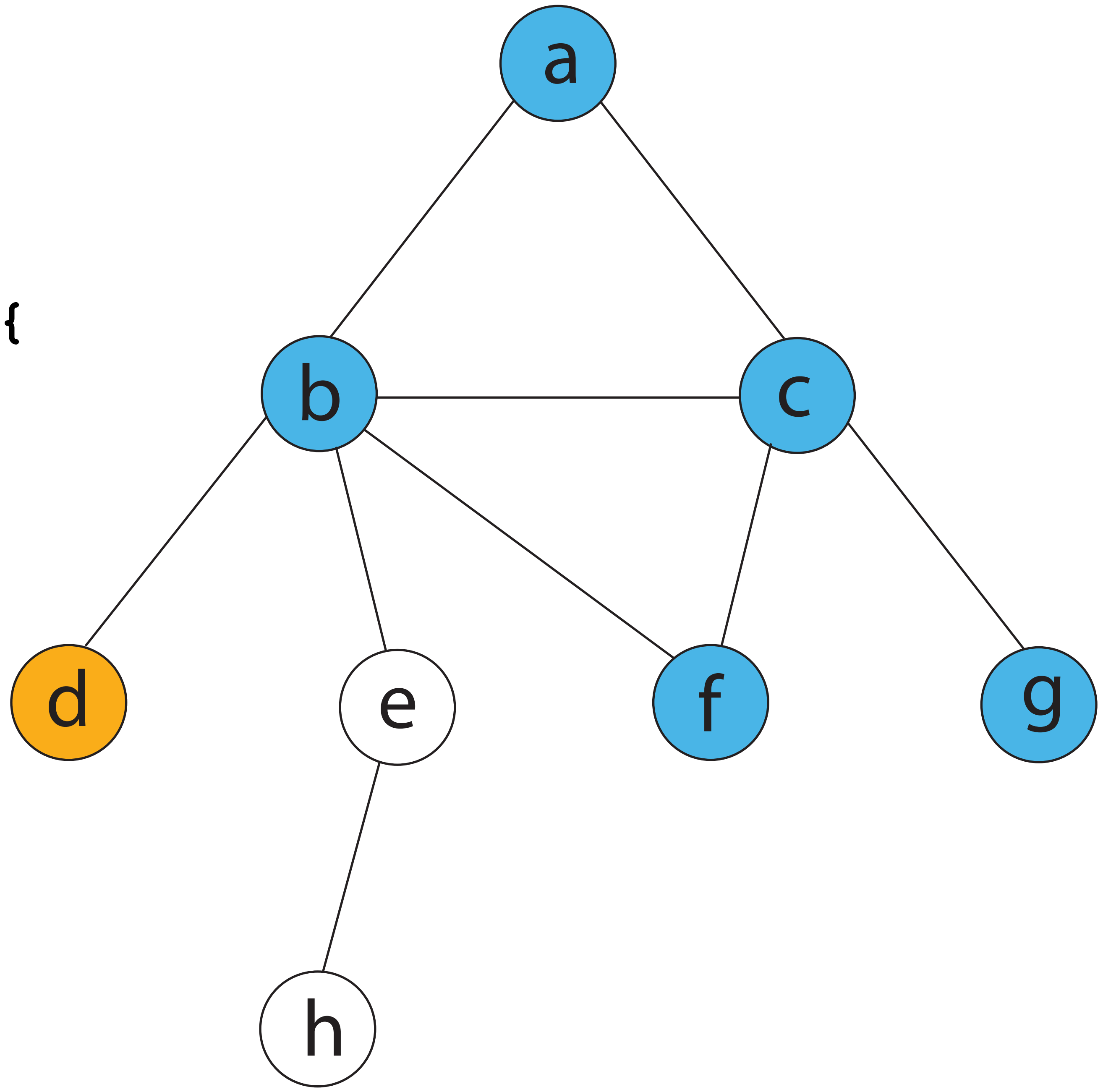
DFS - Depth First Search

```
proc DFS(vertex v){
  mark v as visited
  for all u neighbors of v {
    if u is not visited {
      DFS(u)
    }
  }
}
```



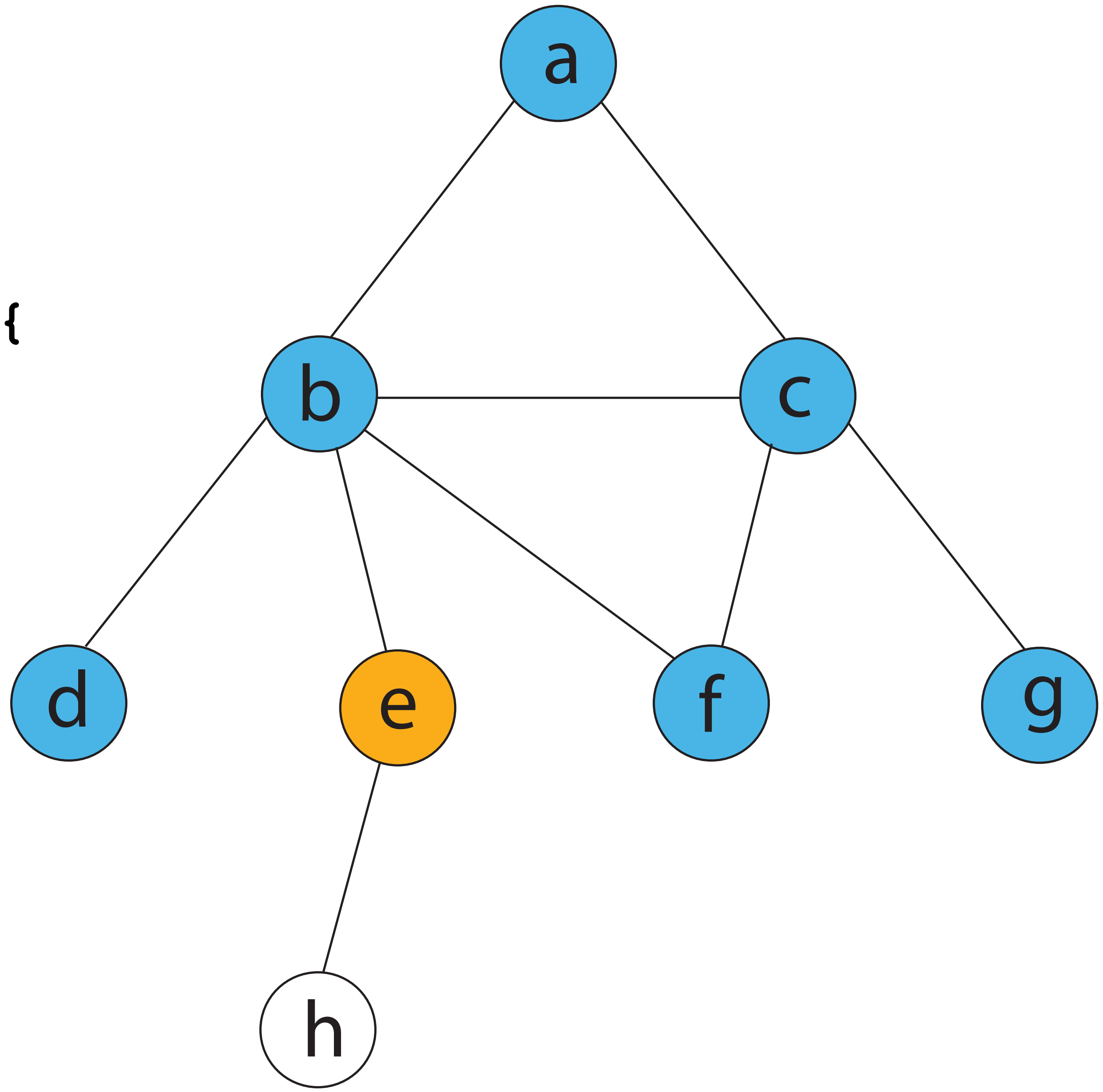
DFS - Depth First Search

```
proc DFS(vertex v){
  mark v as visited
  for all u neighbors of v {
    if u is not visited {
      DFS(u)
    }
  }
}
```



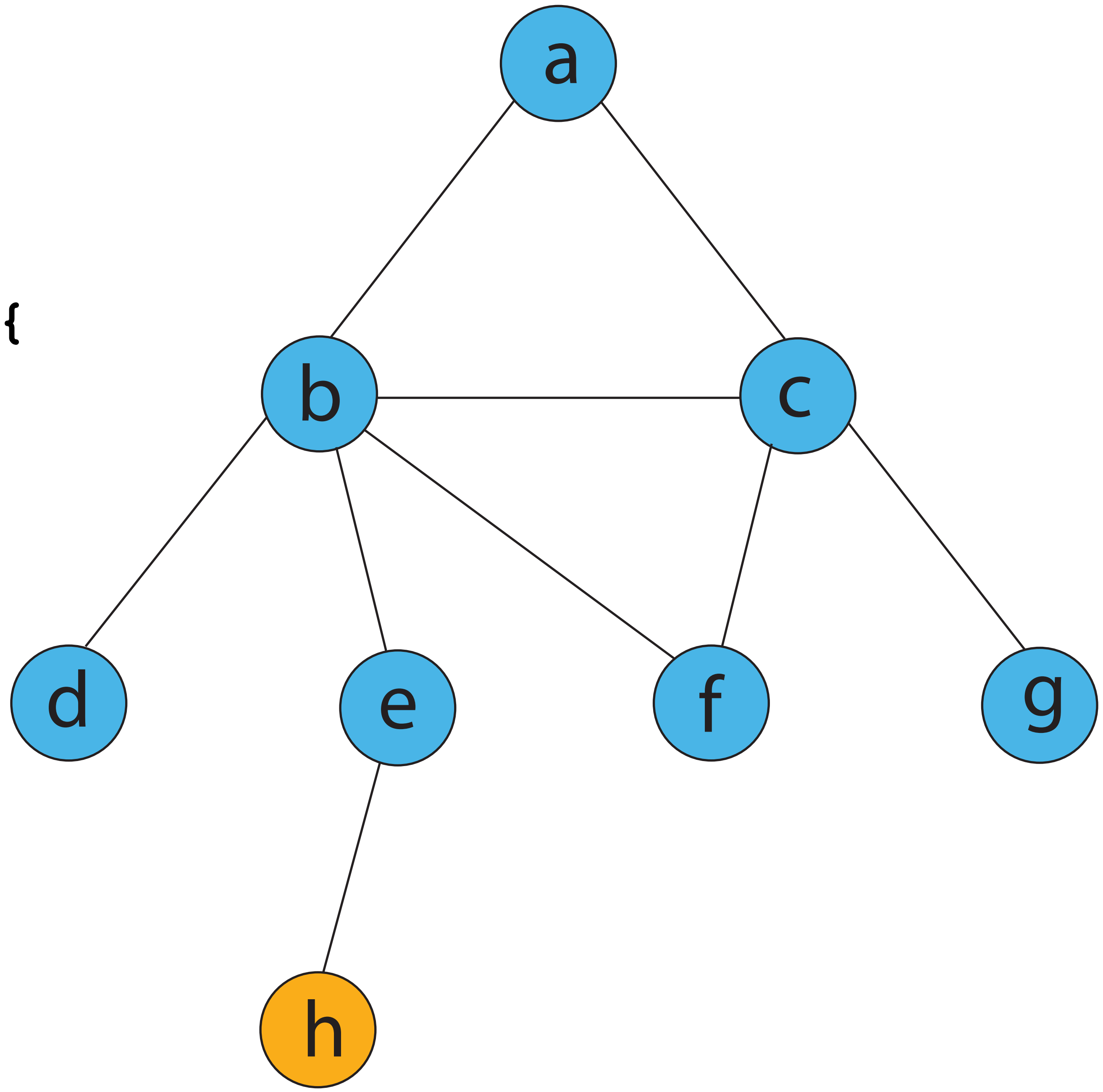
DFS - Depth First Search

```
proc DFS(vertex v){
  mark v as visited
  for all u neighbors of v {
    if u is not visited {
      DFS(u)
    }
  }
}
```



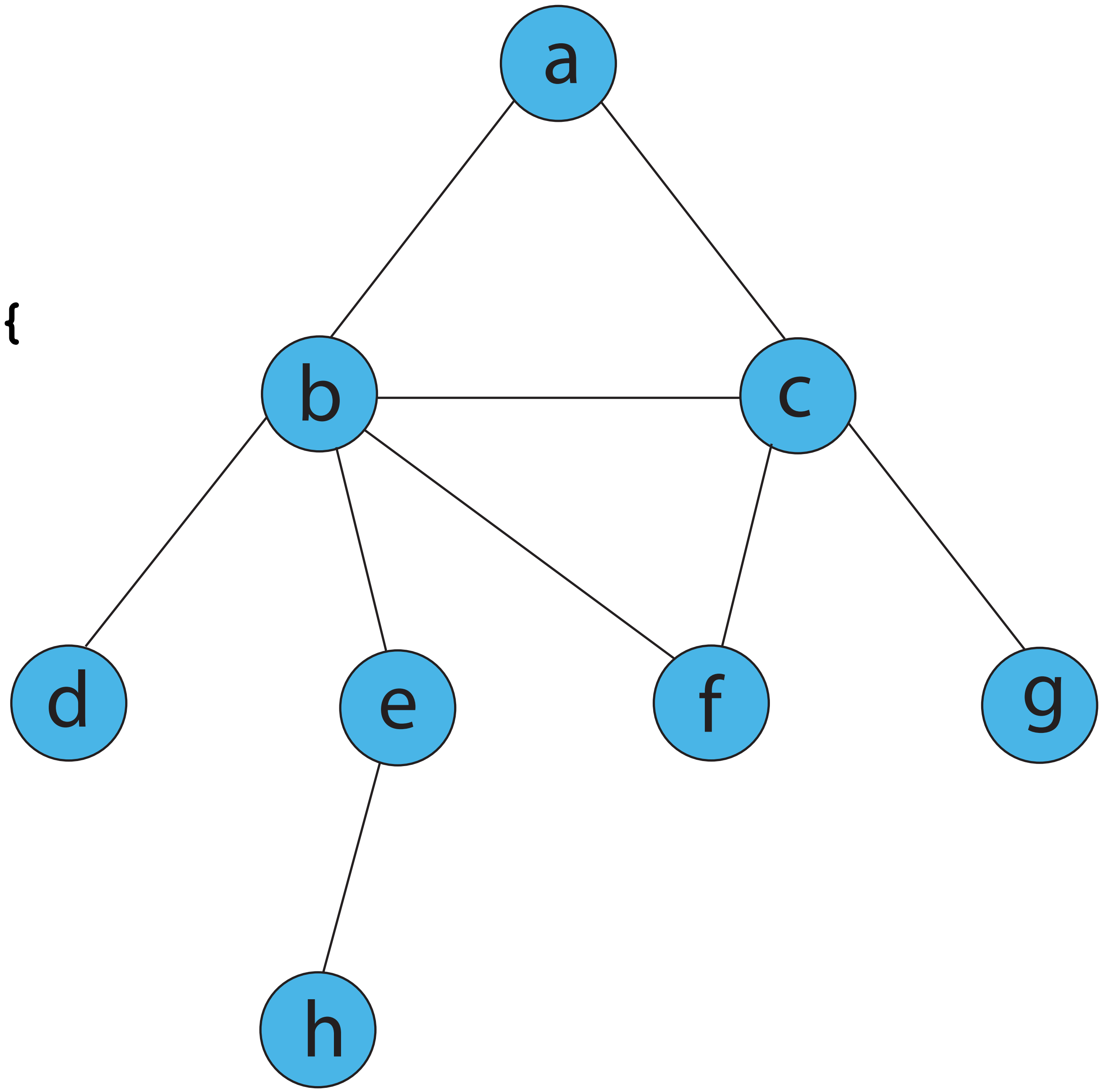
DFS - Depth First Search

```
proc DFS(vertex v){
  mark v as visited
  for all u neighbors of v {
    if u is not visited {
      DFS(u)
    }
  }
}
```



DFS - Depth First Search

```
proc DFS(vertex v){
  mark v as visited
  for all u neighbors of v {
    if u is not visited {
      DFS(u)
    }
  }
}
```



Order in which the vertices are visited is : a, b, c, f, g, d, e, h

Exercise: Race Tracks

Exercise: Race Tracks

- You can model the problem as a graph and the solution corresponds to a shortest path between two fixed vertices
- You can use BFS to find the shortest path

Exercise: Race Tracks

Naive model

- Vertices of the graph are points of the grid.
- And what are the edges? Two vertices are connected with an edge if we can hop from one grid point to the other

Exercise: Race Tracks

Naive model

- However, ability to hop from one grid point A to grid point B depends on the velocity (not fixed for a grid point)
- Hence, sometimes A and B could be connected and sometimes not.

Exercise: Race Tracks

Better model

- Vertices - a pair (grid point, velocity)
- Edges - now, two vertices $(p_1, v_1), (p_2, v_2)$ are connected if reaching the point p_1 with velocity v_1 enables us to hop to point p_2 with velocity v_2
- Number of vertices is 30 * 30 * 7 * 7 ~ 45000
- Degree of each vertex at most 9
- Hence, number of edges at most 405000

Exercise: Race Tracks

Better model

- Vertices - a pair (grid point, velocity)
- Edges - now, two vertices $(p_1, v_1), (p_2, v_2)$ are connected if reaching the point p_1 with velocity v_1 enables us to hop to point p_2 with velocity v_2
- Edges can be deduced from the "description" of a vertex, so no need to store the whole graph explicitly

Graph Traversals

What did we learn?

- Vertices of a graph can be represented with more complex objects than just numbers from 1 to n
- No need to always store the graph explicitly in order to perform a BFS or similar algorithm

Greedy Algorithms

- Often choices that seem best at particular moment turn out not to be optimal in the long run (E.g. Chess, Life, etc..)
- However, sometimes locally optimal choices are also globally optimal! This is when we can apply Greedy Algorithms.

Exercise: Interval Scheduling

- Your CPU needs to execute n jobs, described by time intervals $[s_1, f_1], \ldots, [s_n, f_n]$
- Job i starts at time s_i and finishes at time f_i
- Two jobs are incompatible if their intervals overlap
- What is the maximum number of mutually compatible jobs?

Exercise: Interval Scheduling

Approach to solving

- Come up with a property by which you will pick jobs one by one.
- This property should give you a measure of the locally optimal job.

Exercise: Interval Scheduling

Approach to solving

Natural candidates:

- Earliest start time - Consider jobs with ascending s_i
- Earliest finish time - Consider jobs with ascending f_i
- Shortest length - Consider jobs with ascending f_i - s_i
- Fewest conflicts - For each job i, count the number of conflicts with other jobs c_i . Consider jobs with ascending c_i

Exercise: Interval Scheduling

Approach to solving

Earliest start time
Earliest finish time
Shortest length
Fewest conflicts

Which one do you think will work?

Exercise: Interval Scheduling

Approach to solving

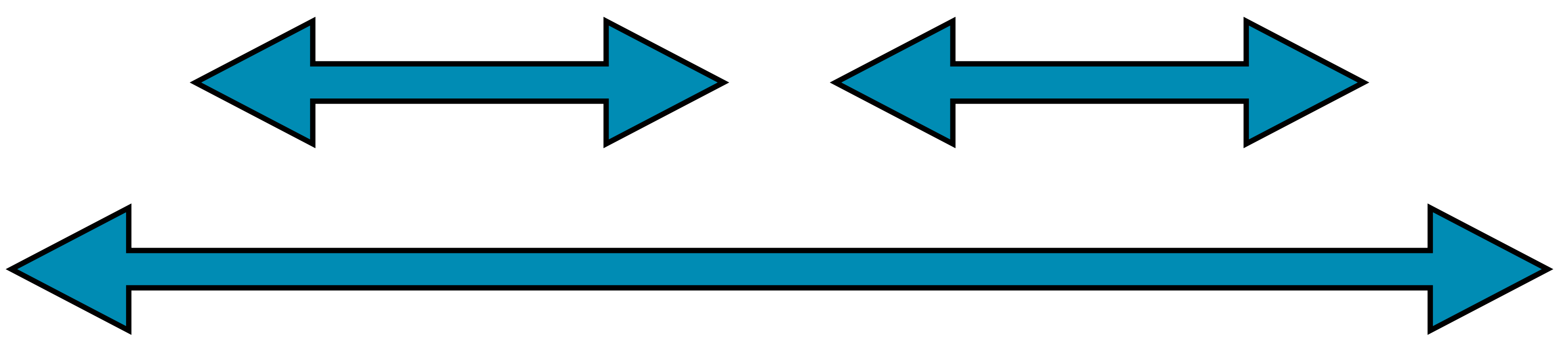
How to figure out if your greedy approach works (or doesn't work)?

- Find a counter example (and prove it doesn't work)
- Exchange argument: Assume you have an optimal solution. Modify the solution gradually until it is the same as the greedy one and prove that at each step you have the same number of jobs as before.

Exercise: Interval Scheduling

Approach to solving

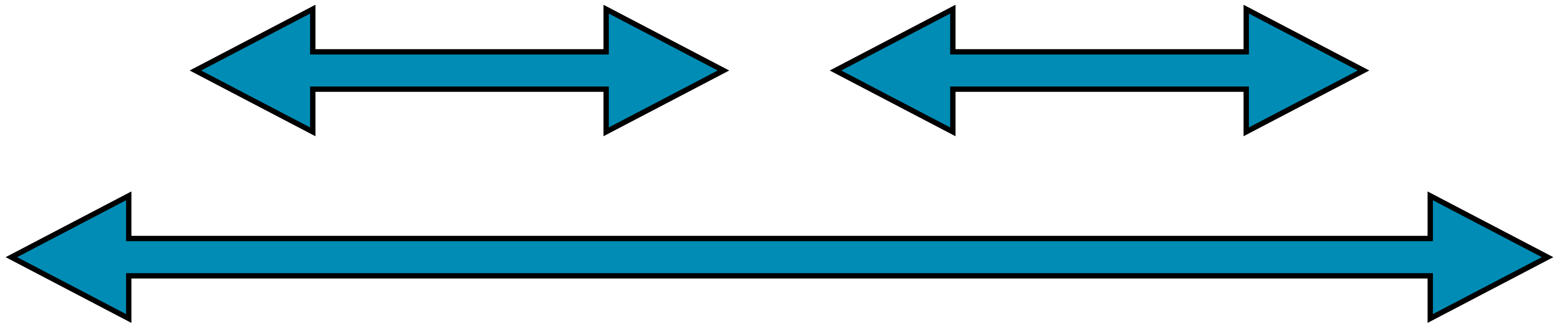
Earliest start time property.



Exercise: Interval Scheduling

Approach to solving

Earliest start time property.

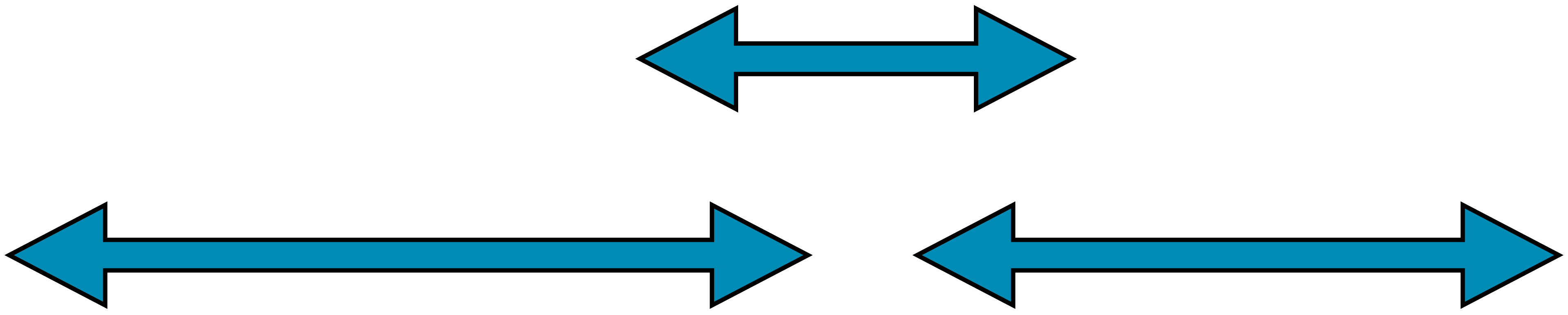


WRONG!

Exercise: Interval Scheduling

Approach to solving

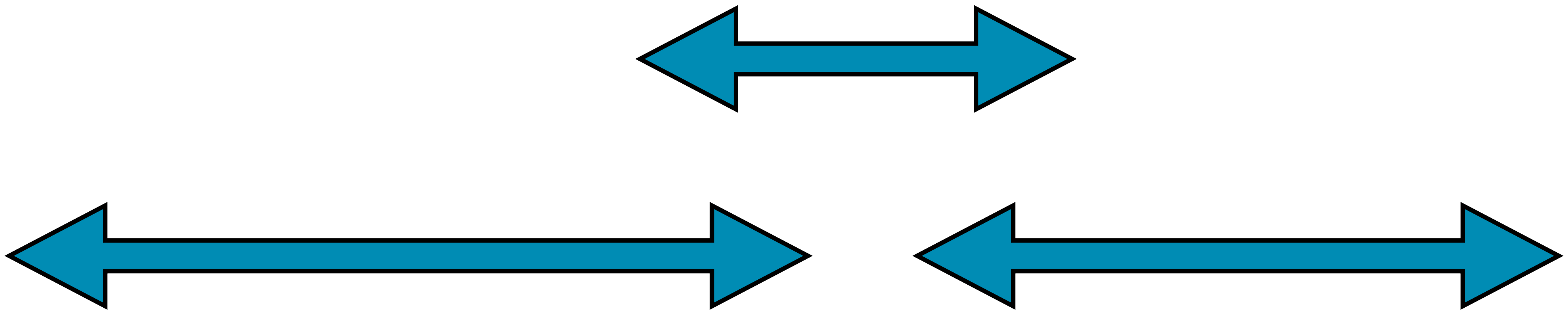
Shortest length



Exercise: Interval Scheduling

Approach to solving

Shortest length

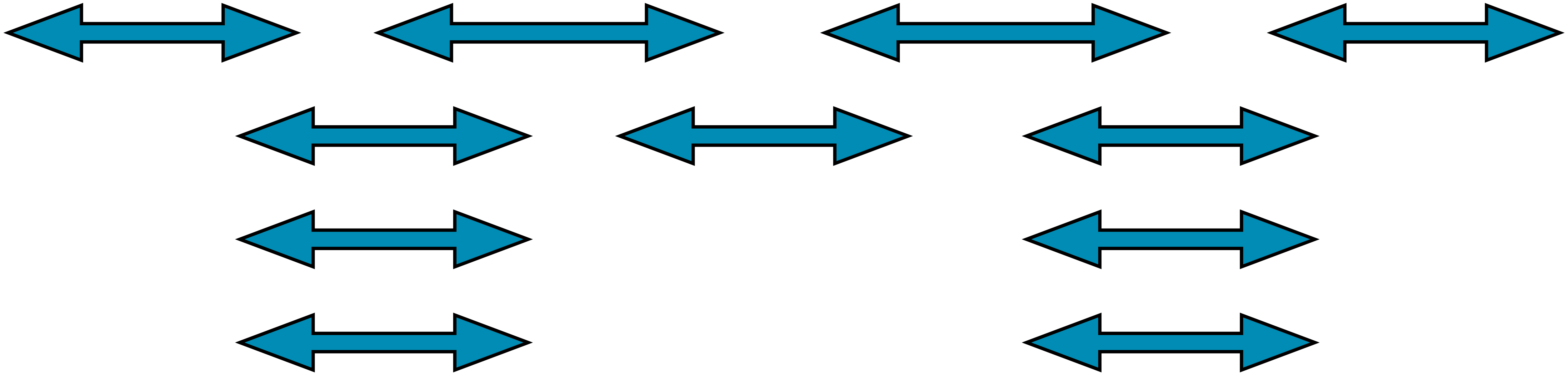


WRONG!

Exercise: Interval Scheduling

Approach to solving

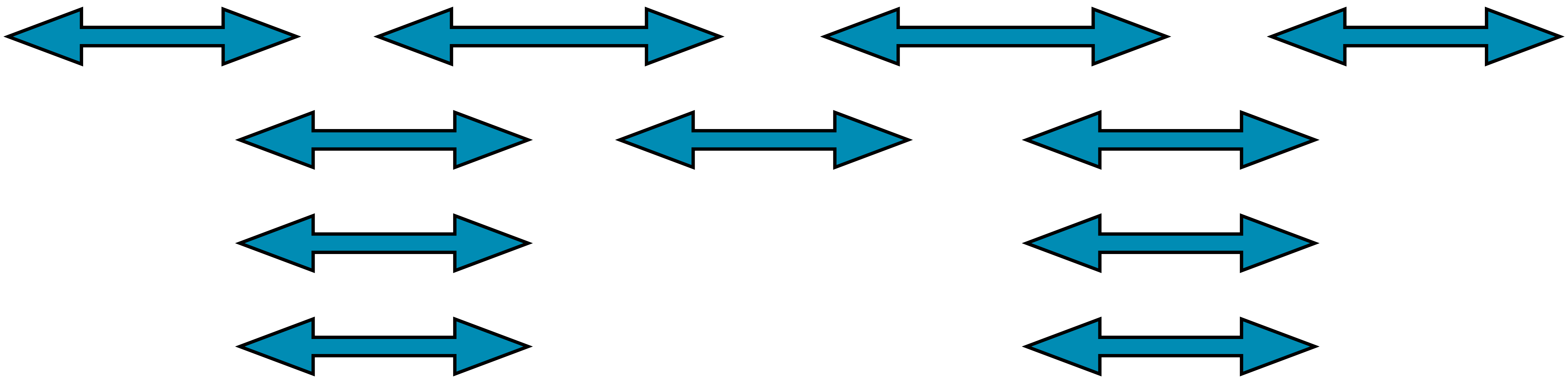
Fewest conflicts



Exercise: Interval Scheduling

Approach to solving

Fewest conflicts



WRONG!

Exercise: Interval Scheduling

Approach to solving

Earliest finish time - sketch of the proof

- Assume S is a set of intervals in the optimal solution
- Let $g_1, \ldots, g_k$ be all k jobs greedy algorithm would select, ordered by the earliest finish time
- If g_1 in S, then we are good
- If g_1 is not in S, then there are some jobs in S in conflict with it. However, there can be only one job in S in conflict with job g_1, denoted by c_1. Why?

Exercise: Interval Scheduling

Approach to solving

Earliest finish time

- Let S_1 be a set we get by removing job c_1 from S and inserting job g_1, i.e. $S_1 = S \setminus \{c_1\} \cup \{g_1\}$
- Note that $|S_1| = |S|$

- Now, consider g_2. If g_2 is not in S_1, then there is only one job c_2 in conflict with g_2.
- Let $S_2 = S_1 \setminus \{c_2\} \cup \{g_2\}$

- Repeat this until you inserted all the jobs from the greedy solution $g_1, \ldots, g_k \in S_k, |S_k| = |S|$

Greedy Algorithms

What did we learn?

- Some, but not all, problems can be solved with greedy approach.
- Finding a property by which we should greedily select can be non-obvious.
- We can prove that our greedy idea works with exchange argument, or disprove it with counterexample.