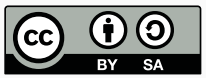
VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

Brute Force and Exhaustive Search

Jiří Dvorský, Ph.D.

Presentation status to date February 24, 2025

Department of Computer Science
VSB – Technical University of Ostrava



Lecture outline

Lecture outline (cont.)

Graph traversal

Depth-first graph traversal

Breadth-first graph traversal

Brute Force Strategy

Characteristics

Brute force is a straightforward approach to solving a problem, usually **directly based on the problem definition** and definitions of the concepts involved.

Brute Force Strategy – examples

Exponentiation problem

Compute an for a nonzero number a and a nonnegative integer n. By the definition of exponentiation

$$a^n = \underbrace{a \times a \times \cdots \times a}_{n \text{ times}}$$

Brute force solution – simply computing an by multiplying 1 by a n times.

More examples

- the consecutive integer checking algorithm for computing GCD, and
- the definition-based algorithm for matrix multiplication from previous presentation.

Brute Force Strategy

1. general strategy – it is difficult to find a problem where “doesn’t work”,
2. does not generally lead to efficient algorithms, but for some problems, e.g., matrix multiplication, pattern matching, algorithms based on this strategy are applicable to larger inputs,
3. is an acceptable strategy when it is not worthwhile to deal with more sophisticated algorithms – ad hoc problem solving,
4. it’s always a useful strategy for solving problems with small input sizes, and
5. it’s also important as a benchmark against which to compare more efficient algorithms solving the same problem.

Brute Force and Exhaustive Search

Sorting Algorithms

Sorting Algorithms

- We are given an array of n elements for which an ordering relation is defined (i.e. the relation “less than”), for example let’s take integers.
- The task is to rearrange the elements of the array into a non-decreasing sequence – the element of the array at the lower index must be less than or equal to the element at the higher index.
- The question is: is there a sorting algorithm that solves the problem in a brute force, completely straightforward way?

SelectSort – Initial consideration

Question

Which element in the array do we know exactly where it belongs?

Answer

The smallest! It belongs at the beginning of the array, at the lowest index! (Note: The same applies to the largest element.)

SelectSort – Algorithm Principle

1. Select the smallest element of the n array elements and replace it with the first element of the array.
2. Select the smallest element from the remaining $n - 1$ elements of the array and replace it with the second element of the array.
3. In general, in the i-th step, select the smallest element from the remaining $n - i$ elements and replace it with the i-th element.
4. After $n - 1$ steps, the array is sorted.

SelectSort – Example

89	45	68	90	29	34	17
17	45	68	90	29	34	89
17	29	68	90	45	34	89
17	29	34	90	45	68	89
17	29	34	45	90	68	89
17	29	34	45	68	90	89
17	29	34	45	68	89	90

SelectSort – pseudocode

Input : Array $A[0 \ldots n-1]$ with ordering defined on array elements
Output: Sorted array A

```
1 for i ← 0 to n − 2 do
2 |   min ← i;
3 |   for j ← i + 1 to n − 1 do
4 |   |   if A[j] < A[min] then
5 |   |   |   min ← j;
6 |   |   end
7 |   end
8 |   Swap (A[i], A[min]);
9 end
```

Exchanging values of two variables

```
1 Procedure Swap(x, y)
  |   Input  : Parameters x and y of the same data type
  |   Output: Interchanged values of x and y
2 |   aux ← x;
3 |   x ← y;
4 |   y ← aux;
5 end
```

SelectSort – Analysis

1. Input size – number of elements n.
2. Basic operations – element comparison (sometimes number of element swaps).
3. The number of basic operations depends only on n – worst, best and average cases merge.
4. Constructing equations

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1]$$

$$= \sum_{i=0}^{n-2} (n-1-i) = \frac{1}{2} n(n-1) \approx \frac{1}{2} n^2 = \Theta(n^2)$$

SelectSort – Analysis (cont.)

Remarks

- The detailed calculation of the sum can be found in the example “Uniqueness of elements” in the previous lesson.
- The complexity of the algorithm does not depend on the unorderedness of the input array. So the algorithm is **not natural**.
- But the algorithm is **stable** – the first of several identical elements is always taken as the minimum.
- The algorithm is **in situ** – only a constant amount of extra memory is needed.

SelectSort

Animation

An animation of the SelectSort algorithm is available in a separate file.

Brute Force and Exhaustive Search

Sequential search

Sequential search

- Typical example of a brute force strategy solution.
- Strong side of the algorithm – simplicity.
- Weak side of the algorithm – high complexity.

Input : Array $A[0 \dots n-1]$ and searched element x
Output: Index of the first occurrence of element x in array A, otherwise -1

```
1 for i ← 0 to n – 1 do
2 |   if A[i] = x then
3 |   |   return i;
4 |   end
5 end
6 return -1;
```

The algorithm is also referred to as **linear search**.

Linear search – complexity

	Number of comparisons
Worst-case scenario	n
Best-case scenario	1
Average-case scenario	$\frac{1}{2}p(n+1) + n(1-p)$

where p is the probability of successful search

Linear search – using a sentinel

Input : Array $A[0 \ldots n]$ and searched element x
Output: Index of the first occurrence of element x in array A, otherwise -1

```
1 A[n] ← x;
2 i ← 0;
3 while A[i] ≠ x do
4 |   i ← i + 1;
5 end
6 if i < n then return i;
7 return -1;
```

Brute Force and Exhaustive Search

Brute force string matching

Brute force string matching

Task

Find the pattern p in the text t.

Brute Force Solution

1. Attach the pattern to the beginning of the text.
2. Start comparing characters in the pattern and the text.
3. If all characters of the pattern match the text – found.
4. If we find a mismatch, move the pattern one position forward and continue with point 2

$$\begin{array}{ccccccccc} t_0 & \cdots & t_i & \cdots & t_{i+j} & \cdots & t_{i+m-1} & \cdots & t_{n-1} \\ & & \updownarrow & & \updownarrow & & \updownarrow & & \\ & & p_0 & \cdots & p_j & \cdots & p_{m-1} & & \end{array}$$

Brute force search – pseudocode

Input : Pattern p, text t and starting position s
Output: Position of the first occurrence of p in text t or -1

```
1 for i ← s to |t| - |p| do
2 |   j ← 0;
3 |   while j < |p| do
4 |   |   if p[j] ≠ t[i + j] then break;
5 |   |   j ← j + 1;
6 |   end
7 |   if j = |p| then return i;
8 end
9 return -1;
```

Brute force search – example

First attempt

Shift by 1 character

Second attempt

Shift by 1 character

Brute force search – example (cont.)

Third attempt

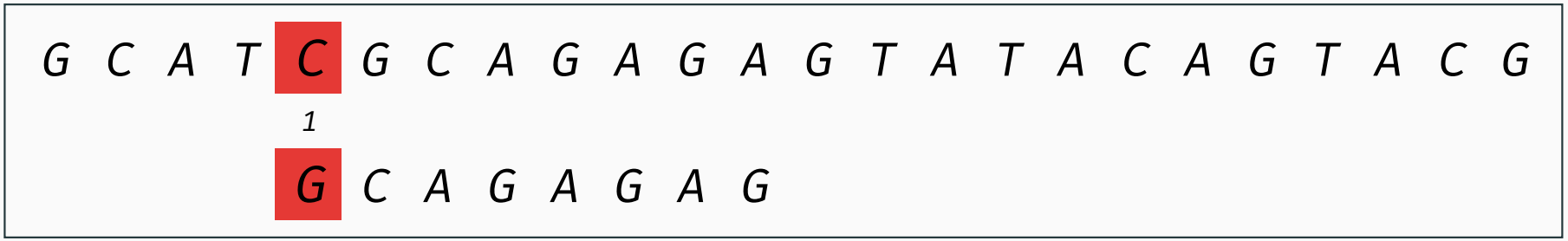
Shift by 1 character

Fourth attempt

Shift by 1 character

Brute force search – example (cont.)

Fifth attempt



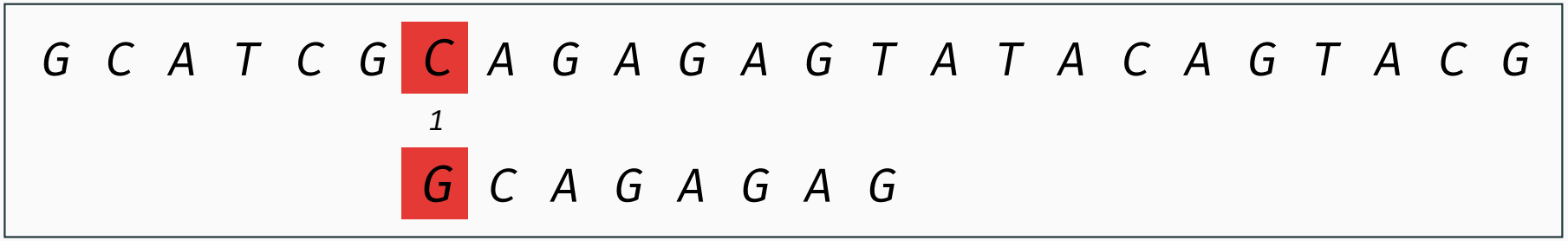
Shift by 1 character

Sixth attempt

Shift by 1 character

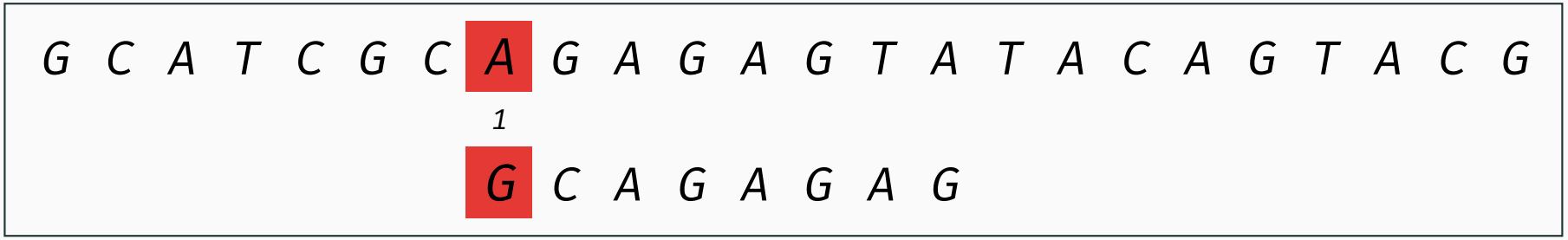
Brute force search – example (cont.)

Seventh attempt



Shift by 1 character

Eighth attempt



Shift by 1 character

Brute force search – example (cont.)

Ninth attempt

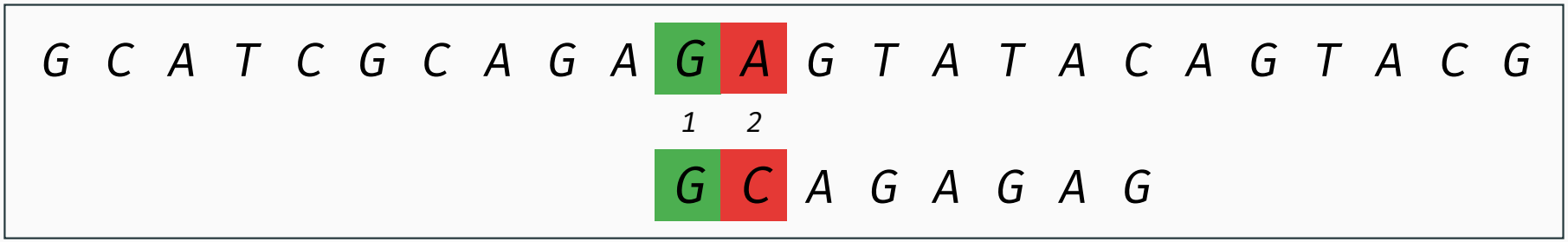
Shift by 1 character

Tenth attempt

Shift by 1 character

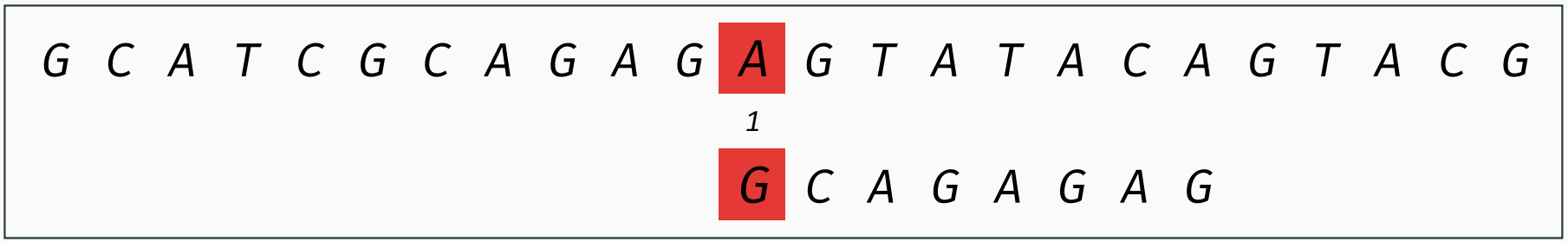
Brute force search – example (cont.)

Eleventh attempt



Shift by 1 character

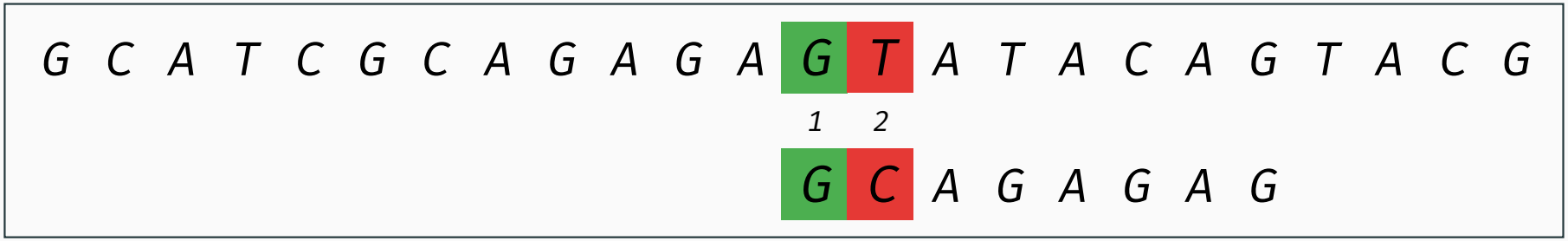
Twelfth attempt



Shift by 1 character

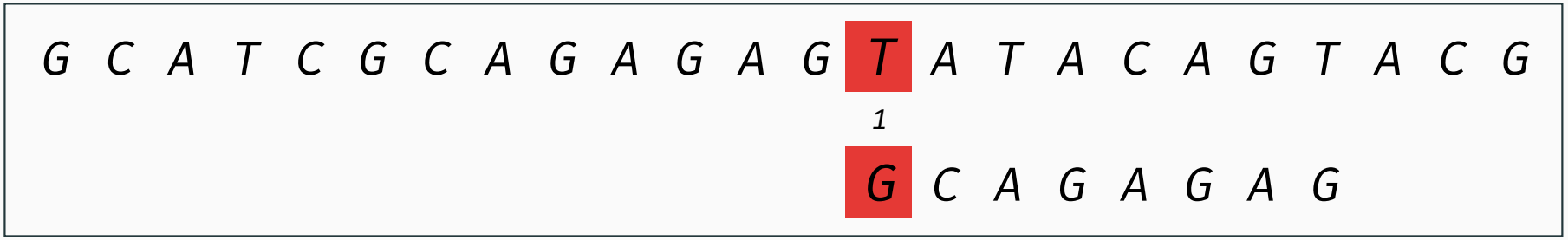
Brute force search – example (cont.)

Thirteenth attempt



Shift by 1 character

Fourteenth attempt



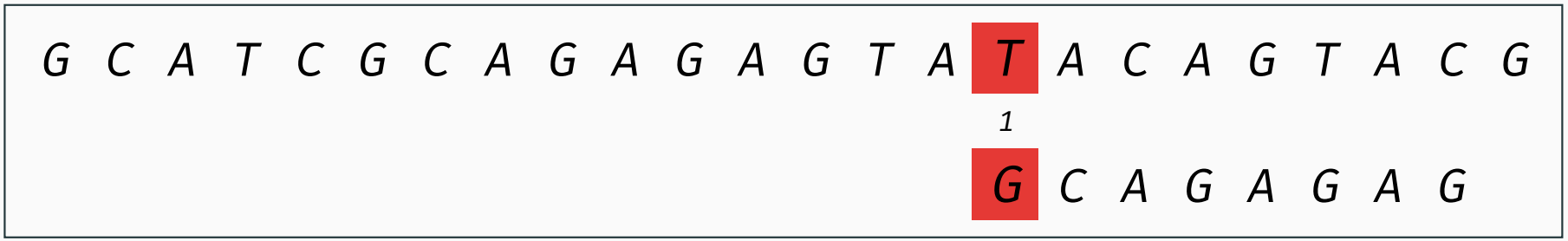
Shift by 1 character

Brute force search – example (cont.)

Fifteenth attempt

Shift by 1 character

Sixteenth attempt



Shift by 1 character

Brute force search – example (cont.)

Seventeenth attempt

Shift by 1 character

The algorithm performed a total of 30 character comparisons.

Brute force string matching – algorithm complexity

- Input size – length of the text n and length of the sample m.
- Basic operation – comparison of the sample character and the text character.
- Dependency only on input size – no, it also depends on when the first mismatch is found.
- Worst-case scenario – text $a^{n-1}b$, sample $a^{m-1}b$
  - in each attempt we perform all m comparisons of the sample with the text
  - at the same time, we make all $n - m + 1$ attempts.
  - In total, we perform $m(n - m + 1)$ comparisons, the algorithm falls into $O(mn)$.
- Best-case scenario – sample is found at the beginning of the text, complexity $O(m)$.
- Natural languages – shift occurs after several (k_L) comparisons, worst-case complexity $O(k, n) = O(n)$.

Brute Force and Exhaustive Search

Closest pair problem

Closest pair problem

Problem definition

Find two mutually closest points from a set of n points.

- This is one of the problems of **computational geometry**.
- Points can lie in a plane or generally in some multidimensional space.
- Points can represent real-world objects or records in a database, texts...
- Examples of applications:
  - Air traffic control – we are looking for the two closest aircraft in airspace.
  - Clustering – hierarchical clustering algorithms gradually merge the closest clusters into one larger cluster.

Closest pair problem – assumptions

Let's assume a set of n points $\{P_1, \ldots, P_n\}$, to each point P_i corresponds a vector $\vec{p}_i$ with components

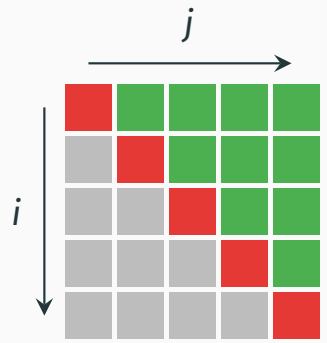
$$\vec{p}_i = (x_i, y_i)$$

in the usual Cartesian coordinates.

The distance of points $\vec{p}_i$ and $\vec{p}_j$ will be calculated using the Euclidean distance

$$d(\vec{p}_i, \vec{p}_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Closest pair problem – solution by brute force



- Calculate the distance of all pairs of points $\vec{p}_i$ and $\vec{p}_j$ and find the minimum.
- It is sufficient to calculate only pairs of points for $j = i + 1, \dots n$.

Input : Set of points $\{\vec{p}_1, \vec{p}_2, \dots, \vec{p}_n\}$
Output: Distance of the two closest points

```
1 Minimum distance ← ∞;
2 for i ← 1 to n − 1 do
3 |   for j ← i + 1 to n do
4 |   |   Minimum distance ←
  |   |     min(Minimum distance, sqrt((x_i − x_j)^2 + (y_i − y_j)^2));
5 |   end
6 end
7 return Minimum distance;
```

Closest pair problem – solution by brute force, complexity

1. Input size – number of points n
2. Basic operation
   - Calculation of square root – not a trivial matter[1].
   - Calculation of square root can be avoided – it is an increasing function, we can look for the minimum of "squares" of distances.
   - We will take the exponentiation of differences of coordinates as the basic operation.
3. Number of basic operations depends only on n – worst, best and average case are the same.

Closest pair problem – solution by brute force, complexity (cont.)

4. Establishment of relationships

$$\begin{aligned} C(n) &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2 = 2 \sum_{i=1}^{n-1} (n - i) \\ &= 2[(n-1) + (n-2) + \cdots + 1] = (n-1)n \in \Theta(n^2) \end{aligned}$$

5. Removal of square root – reduction of complexity by a constant factor, no improvement in asymptotic complexity, still $\Theta(n^2)$ algorithm.
6. Later we will show an algorithm with linear logarithmic complexity.

[1] *https://en.wikipedia.org/wiki/Methods_of_computing_square_roots*

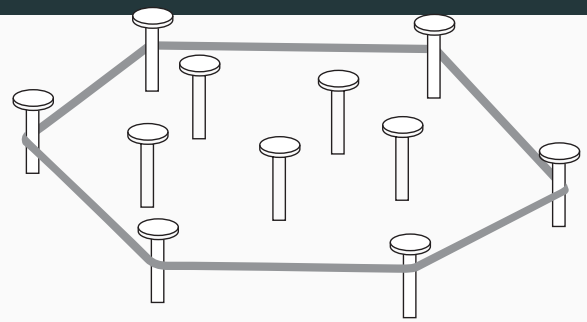
Brute Force and Exhaustive Search

Convex hull of a set

Convex Hull

Problem Statement

The task is to find the convex hull of a set of points in space.



- This is one of the problems of **computational geometry**.
- Points can lie in a plane or generally in some multidimensional space.
- Examples of applications:
  - Collision detection – computer graphics, autonomous vehicles,
  - GIS – point sensors, creating an area from this data,
  - Optimization tasks – the vertices of the convex hull are extreme in some way; a convex polygon is created as the intersection of a finite number of half-spaces; a half-space is defined by an inequality...

Convex set of points

Definition

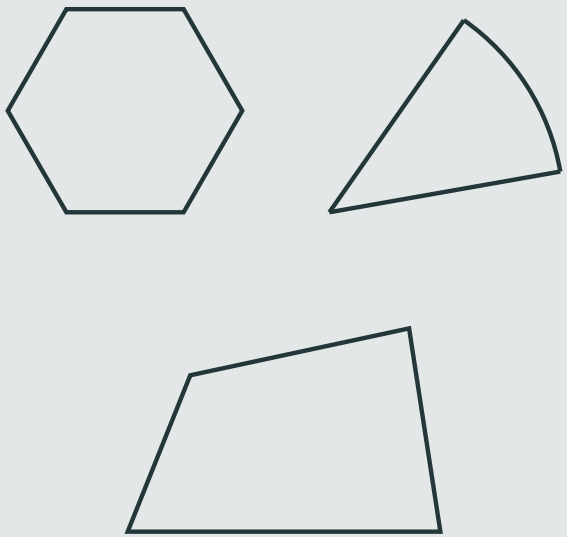
A set of points M in a plane is called **convex**, if for any pair of points $p, q \in M$ the line segment connecting points p and q belongs to the set M.

A set that is not convex is called **non-convex**.

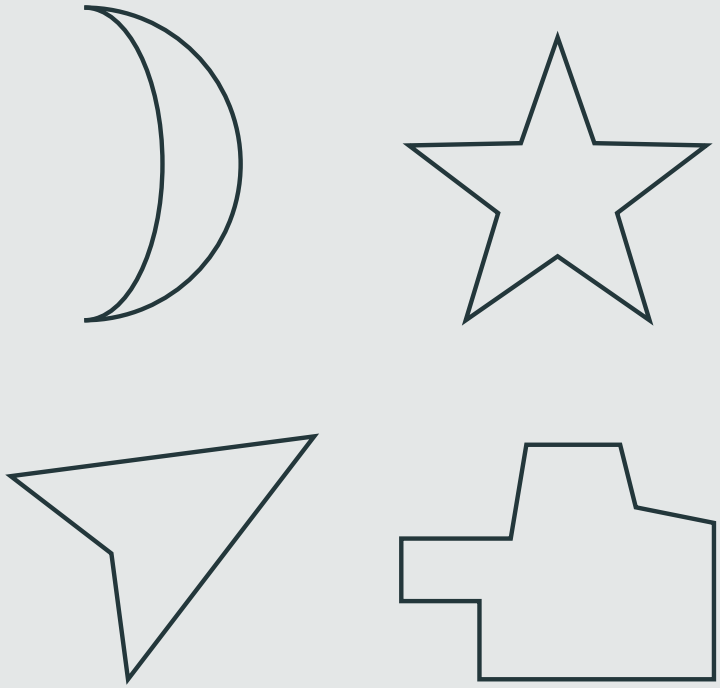
If we imagine the boundary of the set as opaque, the convexity of the set means intuitively that from each of its points every point is visible.

Convex set of points – examples

Convex shapes



Nonconvex shapes



Convex hull

Definition

The convex hull of a set of points M is called the smallest convex set that contains M.

The expression ”smallest” means that the convex hull of the set M must be a subset of any other convex subset containing the set M.

Convex hull

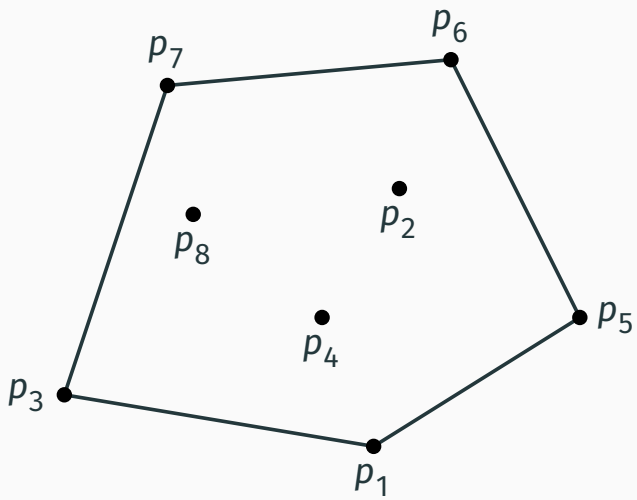
- two-element set – line segment connecting both points.

Convex hull (cont.)

- three-element set – triangle, if the points do not lie on a line, otherwise it is a line segment connecting the most distant points.

Theorem

The convex hull of a set of points M with more than two points that do not lie on one line is a convex polygon, whose vertices belong to M.



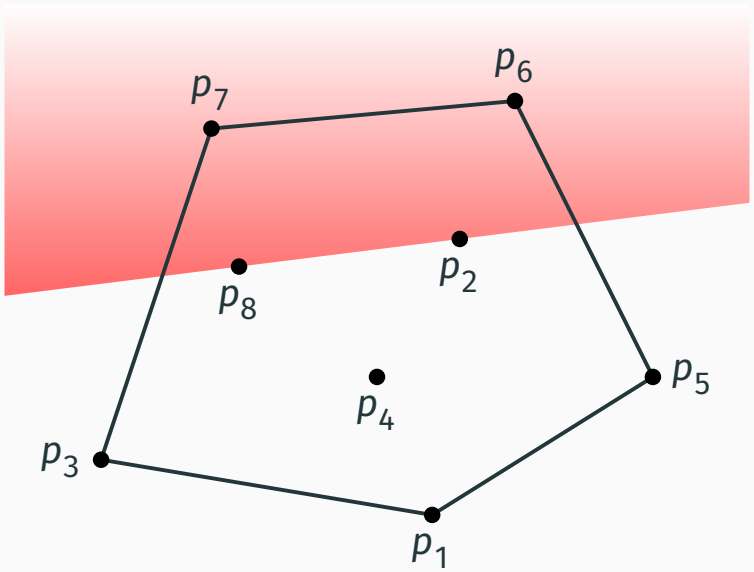
The points of the set M that specify the convex hull M are called **extreme points**.

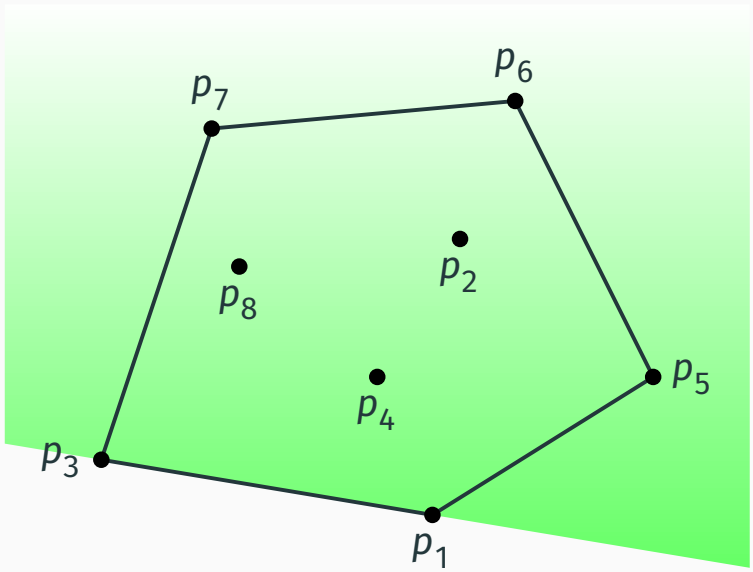
Convex hull (cont.)

The problem of finding the convex hull of a set M is reduced to finding the extreme points.

Convex hull – brute force solution

The line segment $\vec{p}_i\vec{p}_j$ belongs to the convex hull of the set M if and only if all other points from M lie in one of the half-planes defined by the line $\vec{p}_i\vec{p}_j$.





Convex hull – brute force solution (cont.)

The general equation of the line passing through points $\vec{p}_i$ and $\vec{p}_j$ can be written as

$$ax + by + c = 0,$$

where

$$\begin{aligned} a &= y_j - y_i \\ b &= x_i - x_j \\ c &= y_i x_j - x_i y_j \end{aligned}$$

Convex hull – brute force solution (cont.)

The line defines two half-planes:

$$ax + by + c < 0 \quad (1)$$

$$ax + by + c > 0 \quad (2)$$

It is therefore sufficient to verify that for the remaining $n - 2$ points, either inequality (1) or (2) holds.

Convex hull – complexity analysis

- We must check all $\frac{1}{2}n(n-1)$ pairs of points and simultaneously
- for each line defined by one pair of points we must verify the validity of inequalities (??) and (??) for the remaining $n-2$ points.
- Overall, therefore $\left[\frac{1}{2}n(n-1)\right](n-2) \in O(n^3)$.

Brute Force and Exhaustive Search

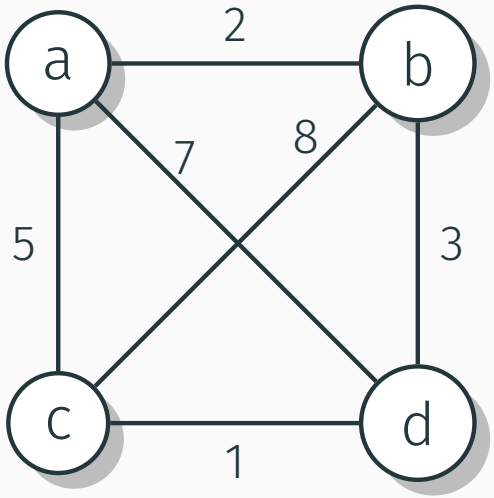
Exhaustive search

Exhaustive search

- Part of solving many problems – finding **one element**, with some **specific property**, from a set of elements, so-called domain, which **exponentially** or faster grows with the problem size.
- The searched element is typically **of combinatorial nature** – permutation, combination, subset.
- Typically it is about **optimization tasks** – typically we search for maximum, minimum. For example, we minimize path length, maximize profit.

Exhaustive search is a problem-solving strategy based on brute force, consisting of testing all elements of the considered domain.

Traveling Salesman Problem – example



Route	Route length l
$a \to b \to c \to d \to a$	$2 + 8 + 1 + 7 = 18$
$a \to b \to d \to c \to a$	$2 + 3 + 1 + 5 = 11$
$a \to c \to b \to d \to a$	$5 + 8 + 3 + 7 = 23$
$a \to c \to d \to b \to a$	$5 + 1 + 3 + 2 = 11$
$a \to d \to b \to c \to a$	$7 + 3 + 8 + 5 = 23$
$a \to d \to c \to b \to a$	$7 + 1 + 8 + 2 = 18$

Exhaustive search

Traveling Salesman Problem

Knapsack problem

Brute Force and Exhaustive Search

Graph traversal

Depth first and breadth first graph traversal

Algorithm for depth-first graph traversal

```
   Input  : Graph G(V, E), initial vertex s ∈ V
   Output: DF-tree
 1 Init(V, s);
 2 while stack ≠ ∅ do
 3 |   u ← Top(stack);
 4 |   switch state [u] do
 5 |   |   case discovered do
 6 |   |   |   ProcessDiscoveredVertex(u);
 7 |   |   end
 8 |   |   case current do
 9 |   |   |   ProcessCurrentVertex(u);
10 |   |   end
11 |   end
12 end
```

Algorithm for depth-first graph traversal (cont.)

```
1  Procedure Init(V, s)
       Input  : Vertices set V, initial vertex s ∈ V
2      foreach v ∈ V do
3          state [v] ← unknown;
4          d [v] ← undefined;
5          f [v] ← undefined;
6          π [v] ← undefined;
7      end
8      stack ← ∅;
9      state [s] ← discovered;
10     Push(stack, s);
11     time ← 0;
12 end
```

Algorithm for depth-first graph traversal (cont.)

```
1  Procedure ProcessDiscoveredVertex(u)
   |  Input  : Vertex u ∈ V
2  |  state [u] ← current;
3  |  d [u] ← time ← time + 1;
4  |  foreach v ∈ Adj(G, u) do
5  |  |  if state [v] = unknown then
6  |  |  |  state [v] ← discovered;
7  |  |  |  π [v] ← u;
8  |  |  |  Push(stack, v);
9  |  |  end
10 |  end
11 end
```

Algorithm for depth-first graph traversal (cont.)

```
1 Procedure ProcessCurrentVertex(u)
  |   Input  : Vertex u ∈ V
2 |   state[u] ← finished;
3 |   f[u] ← time ← time + 1;
4 |   Pop(stack);
5 end
```

Animation of depth-first graph traversal – legend

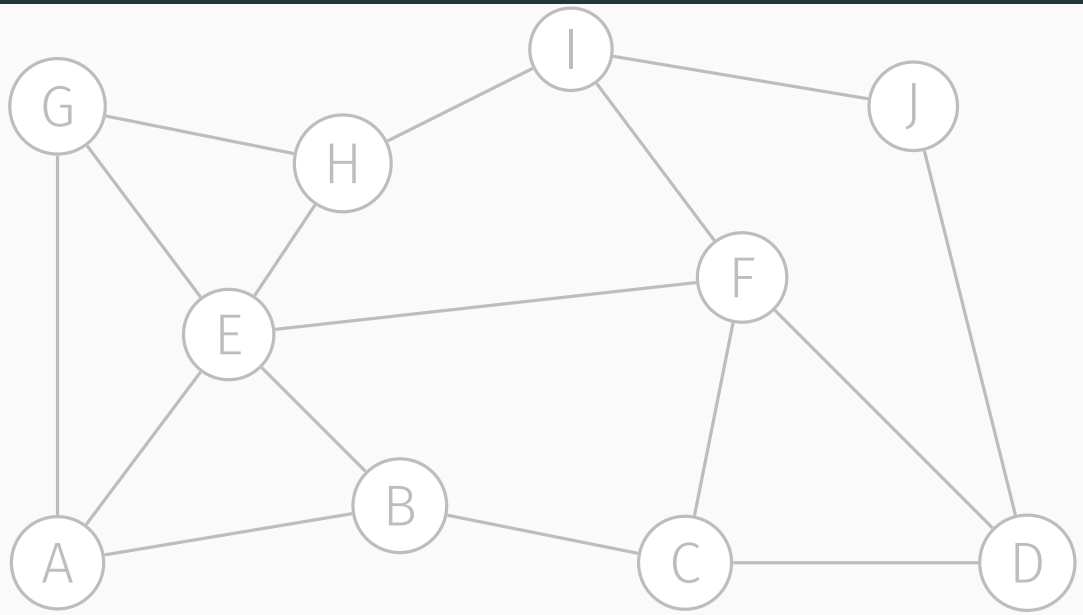
Graph vertices

gray	vertex in unknown state
yellow	vertex in discovered state
red	vertex in current state
blue	vertex in finished state

Graph edges

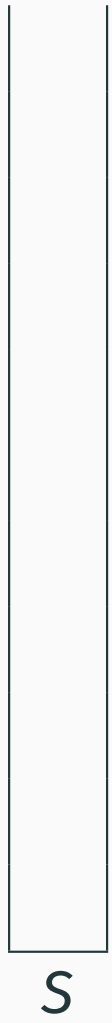
gray	edge between vertices in unknown state or edge not belonging to the DF-tree
yellow	edge incident with vertices in discovered state
red	edge incident with vertex in current state
blue	edge between vertices in finished state

Depth-first graph traversal, step 1

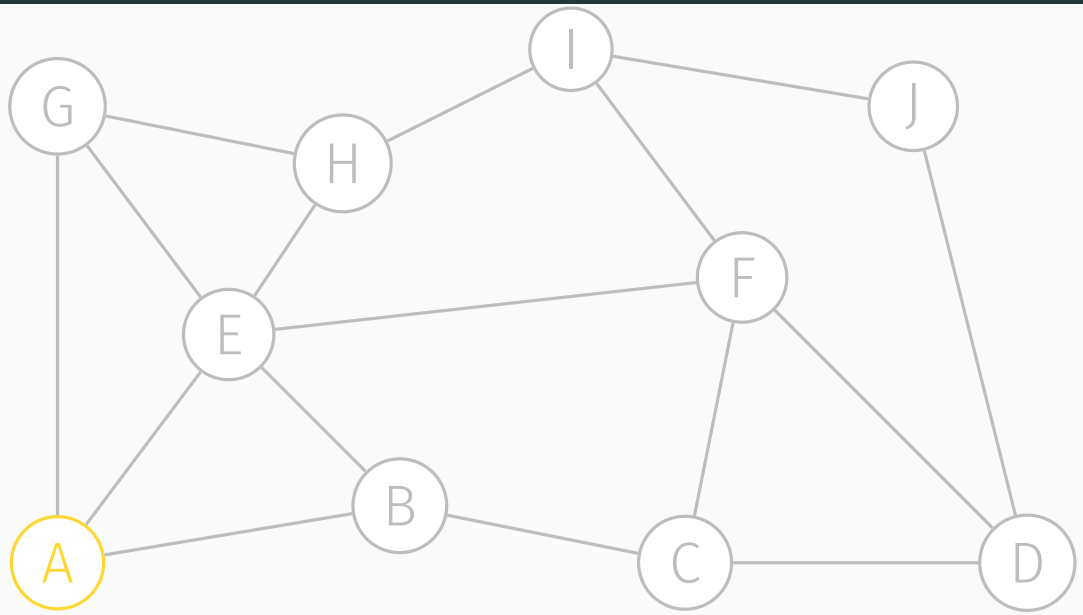


v	$d[v]$	$f[v]$	$\pi[v]$
A	∞	∞	/
B	∞	∞	/
C	∞	∞	/
D	∞	∞	/
E	∞	∞	/

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	/
G	∞	∞	/
H	∞	∞	/
I	∞	∞	/
J	∞	∞	/



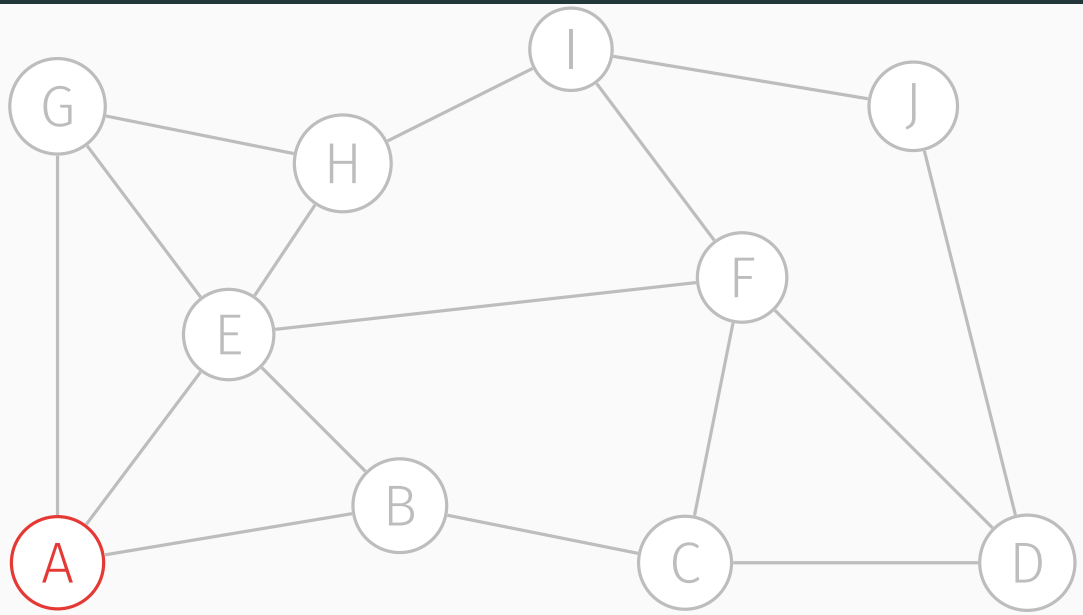
Depth-first graph traversal, step 2



v	$d[v]$	$f[v]$	$\pi[v]$
A	∞	∞	/
B	∞	∞	/
C	∞	∞	/
D	∞	∞	/
E	∞	∞	/

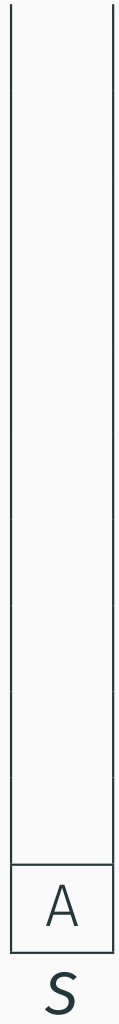
v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	/
G	∞	∞	/
H	∞	∞	/
I	∞	∞	/
J	∞	∞	/

Depth-first graph traversal, step 3

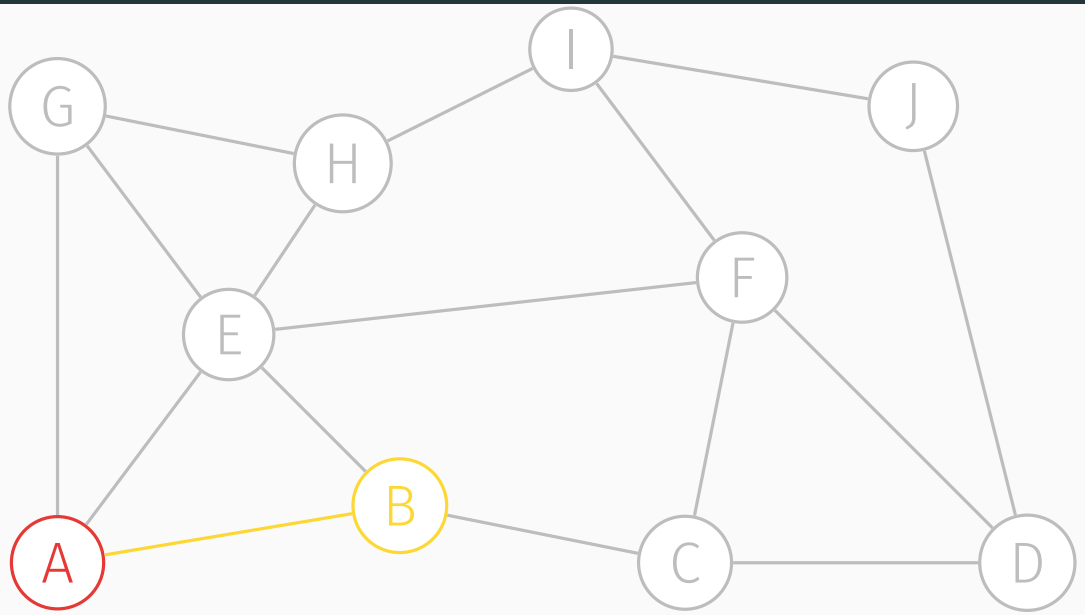


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	/
C	∞	∞	/
D	∞	∞	/
E	∞	∞	/

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	/
G	∞	∞	/
H	∞	∞	/
I	∞	∞	/
J	∞	∞	/

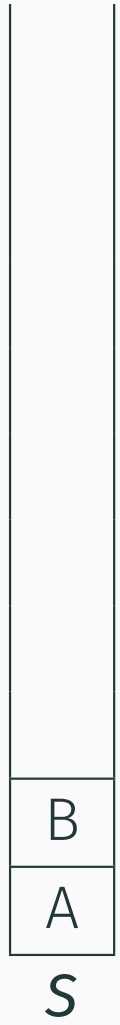


Depth-first graph traversal, step 4

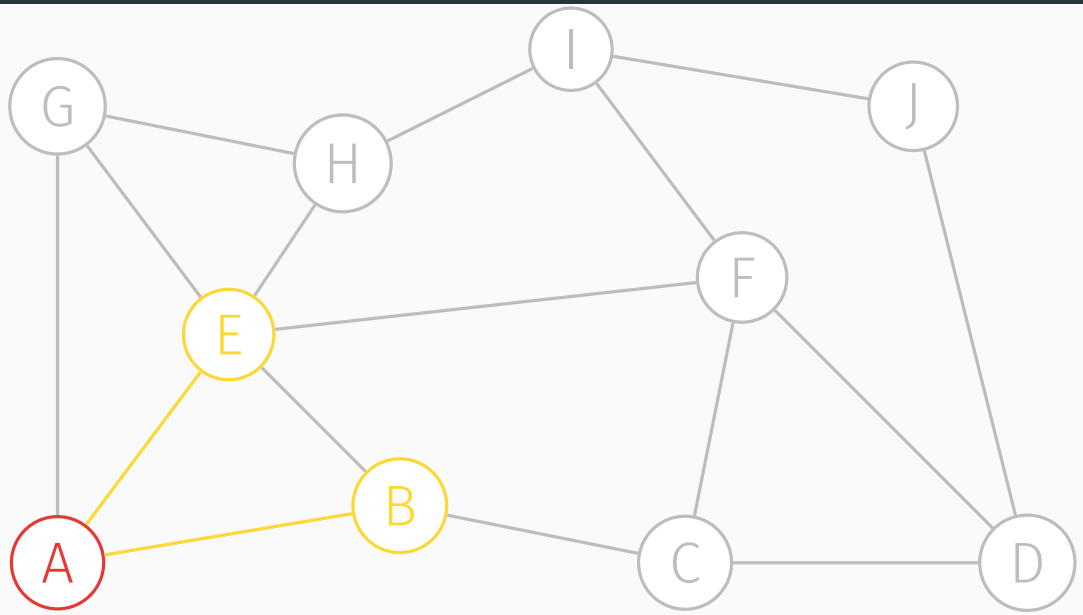


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	∞	∞	/
D	∞	∞	/
E	∞	∞	/

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	/
G	∞	∞	/
H	∞	∞	/
I	∞	∞	/
J	∞	∞	/

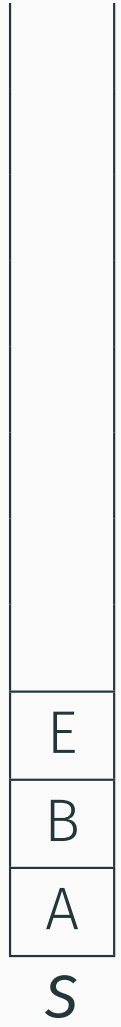


Depth-first graph traversal, step 5

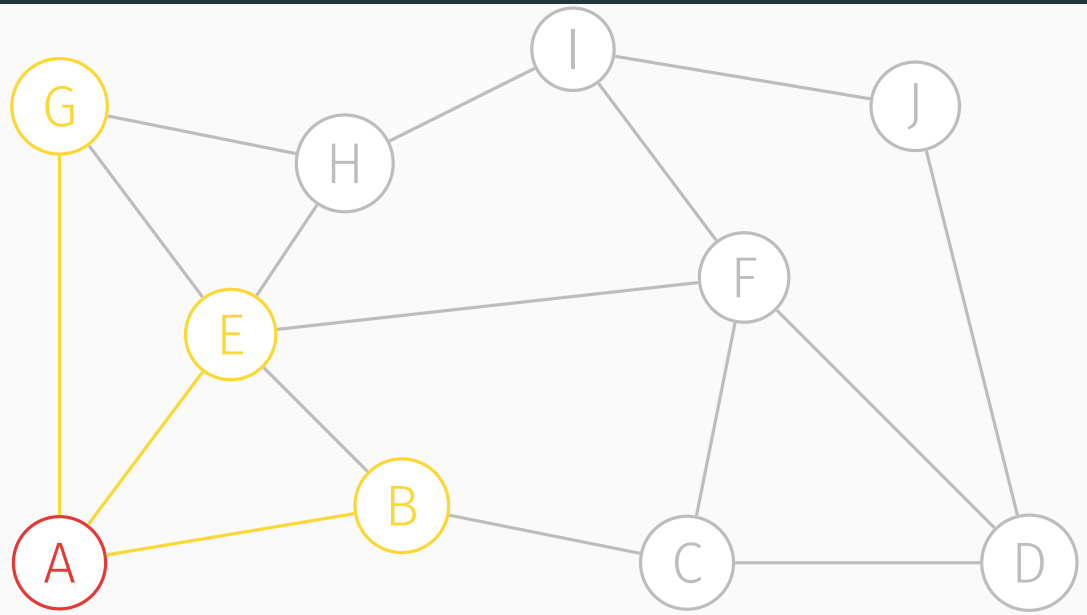


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	∞	∞	/
D	∞	∞	/
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	/
G	∞	∞	/
H	∞	∞	/
I	∞	∞	/
J	∞	∞	/



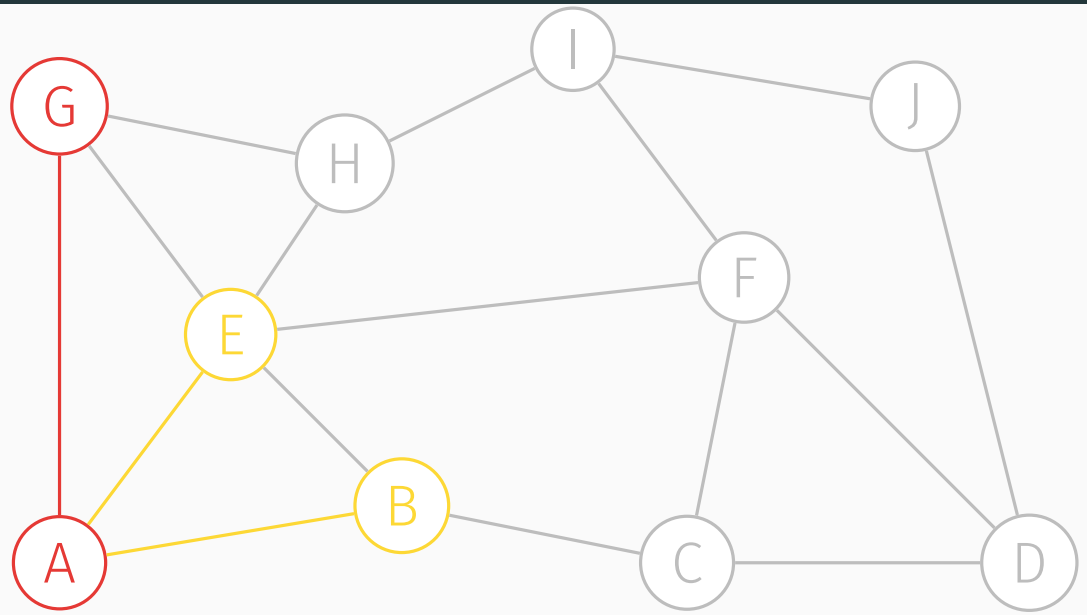
Depth-first graph traversal, step 6



v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	∞	∞	/
D	∞	∞	/
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	/
G	∞	∞	A
H	∞	∞	/
I	∞	∞	/
J	∞	∞	/

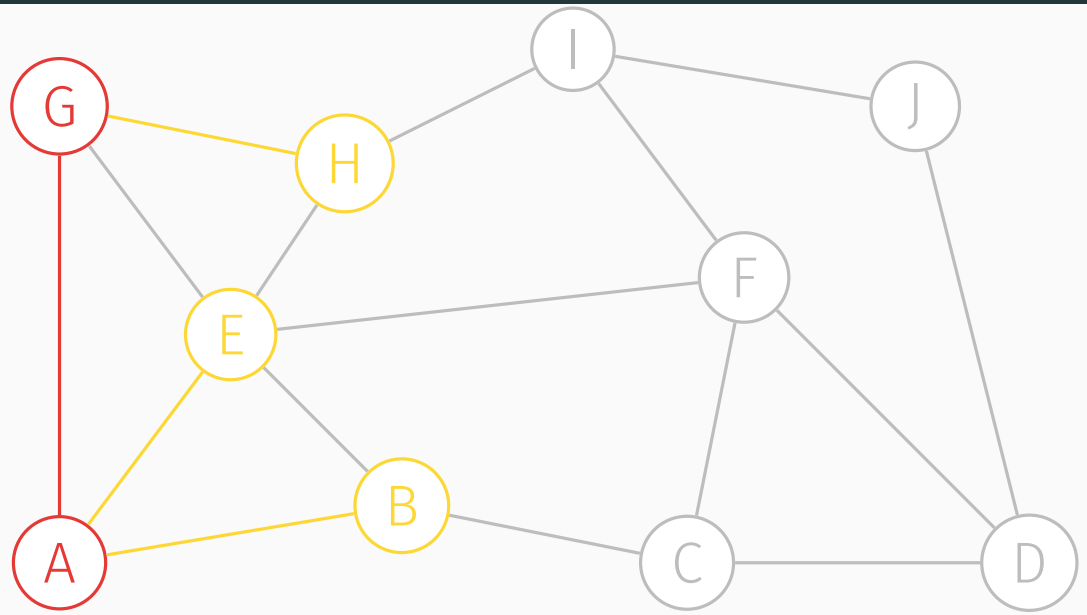
Depth-first graph traversal, step 7



v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	∞	∞	/
D	∞	∞	/
E	∞	∞	A

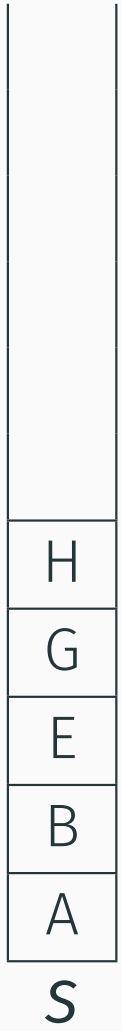
v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	/
G	2	∞	A
H	∞	∞	/
I	∞	∞	/
J	∞	∞	/

Depth-first graph traversal, step 8

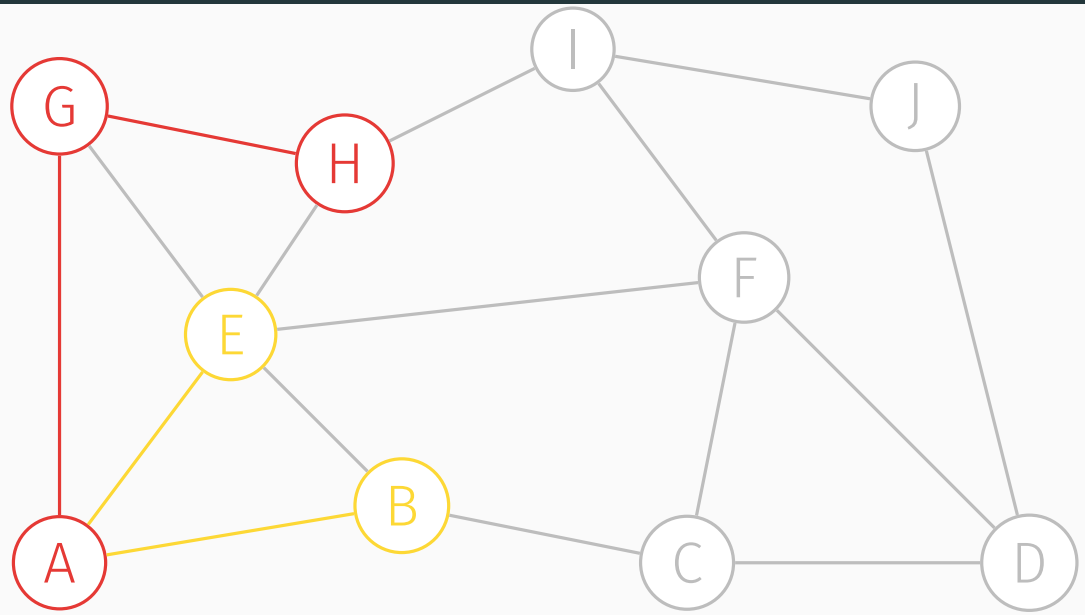


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	∞	∞	/
D	∞	∞	/
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	/
G	2	∞	A
H	∞	∞	G
I	∞	∞	/
J	∞	∞	/

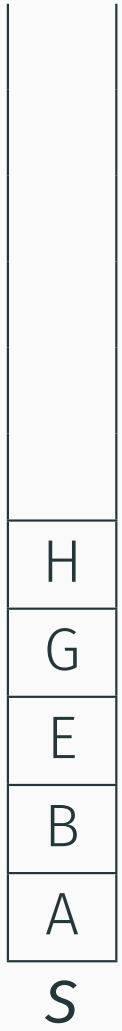


Depth-first graph traversal, step 9

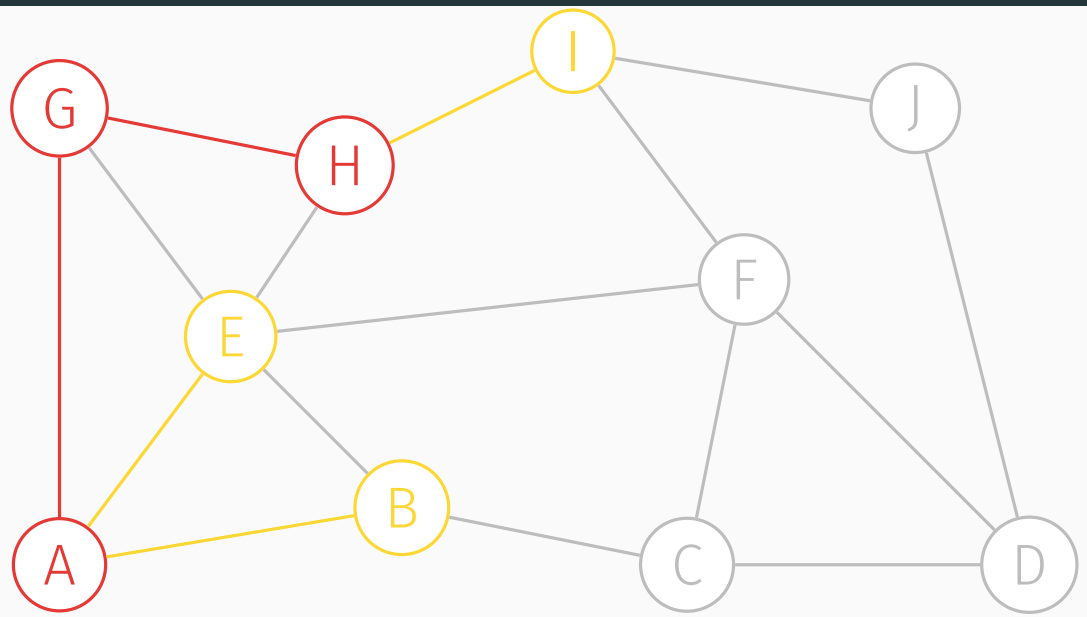


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	∞	∞	/
D	∞	∞	/
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	/
G	2	∞	A
H	3	∞	G
I	∞	∞	/
J	∞	∞	/

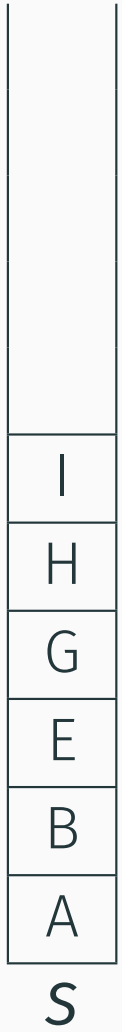


Depth-first graph traversal, step 10

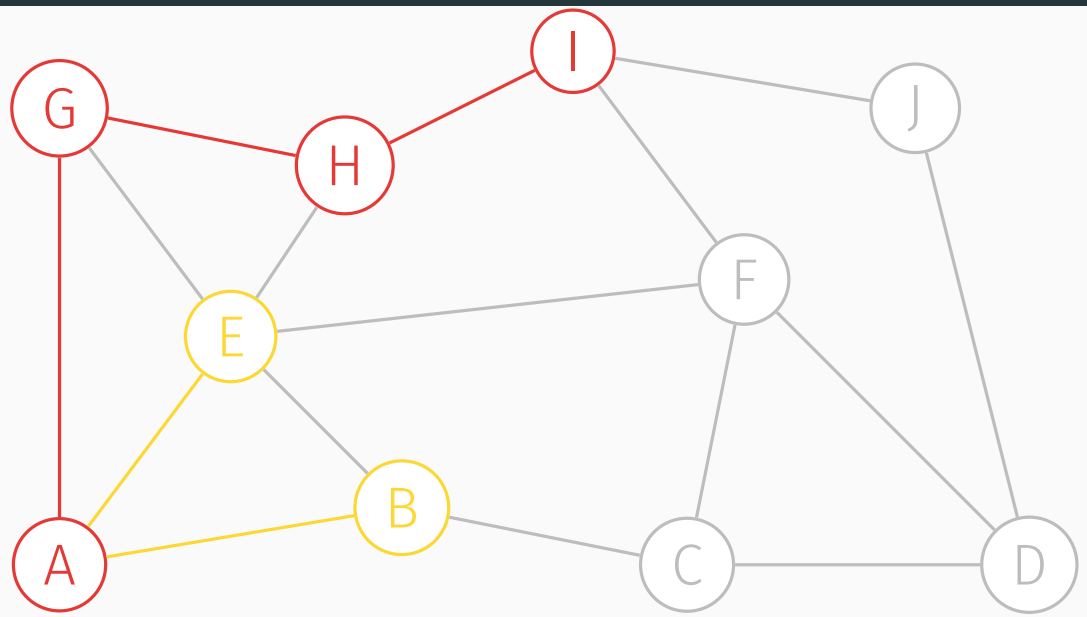


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	∞	∞	/
D	∞	∞	/
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	/
G	2	∞	A
H	3	∞	G
I	∞	∞	H
J	∞	∞	/

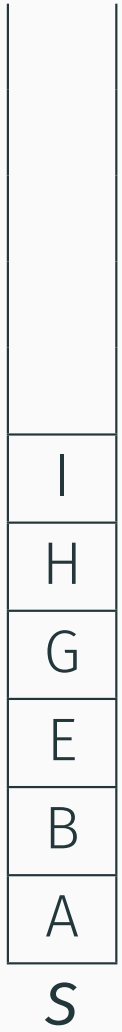


Depth-first graph traversal, step 11

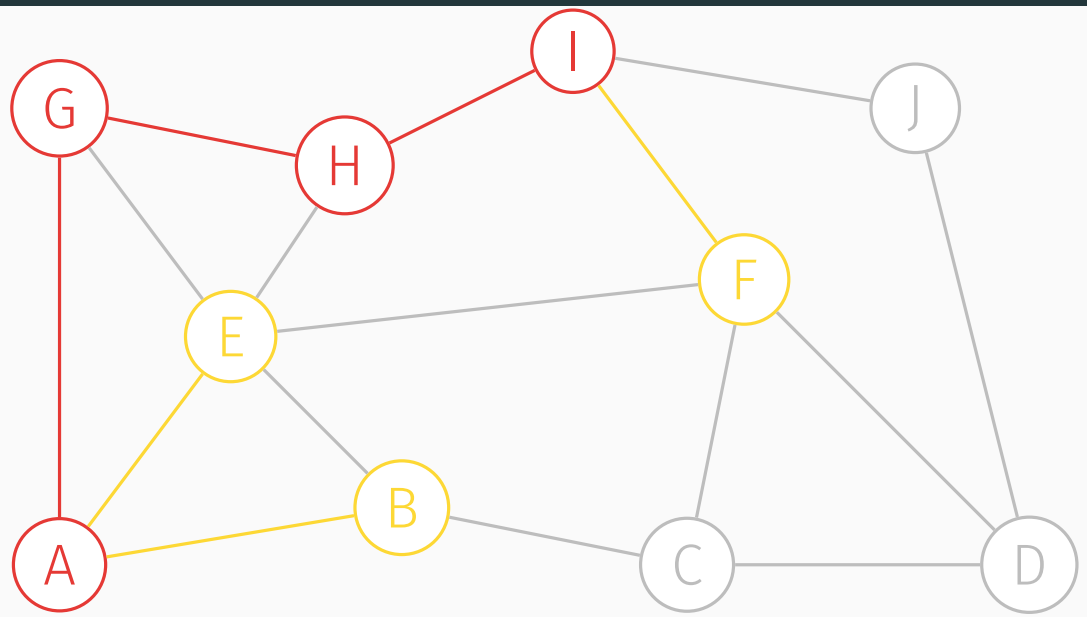


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	∞	∞	/
D	∞	∞	/
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	/
G	2	∞	A
H	3	∞	G
I	4	∞	H
J	∞	∞	/

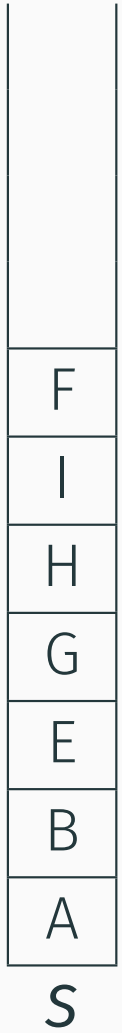


Depth-first graph traversal, step 12

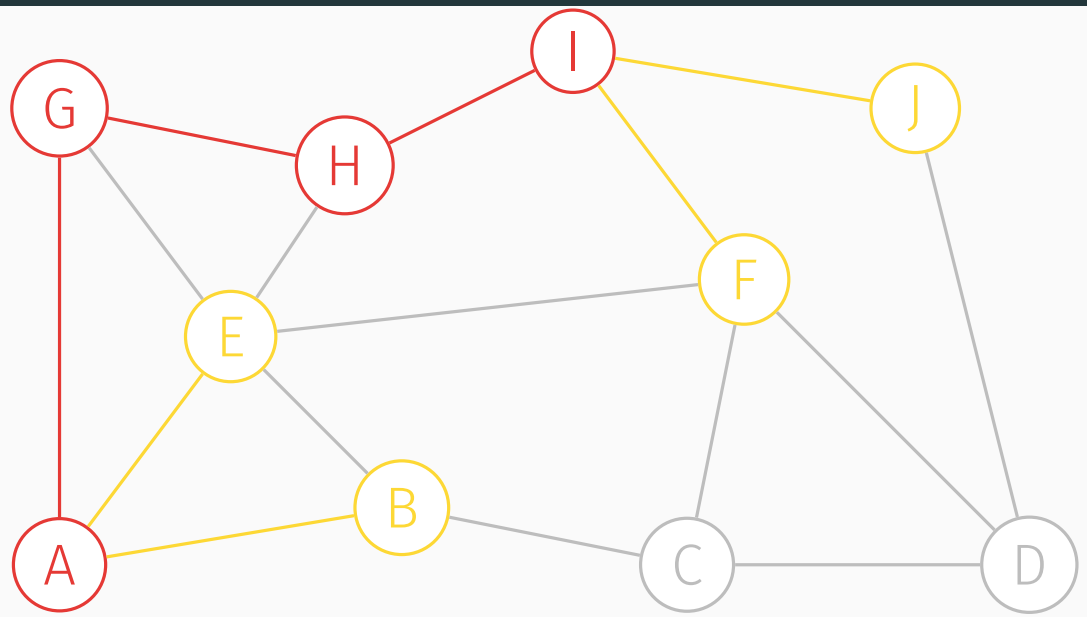


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	∞	∞	/
D	∞	∞	/
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	I
G	2	∞	A
H	3	∞	G
I	4	∞	H
J	∞	∞	/

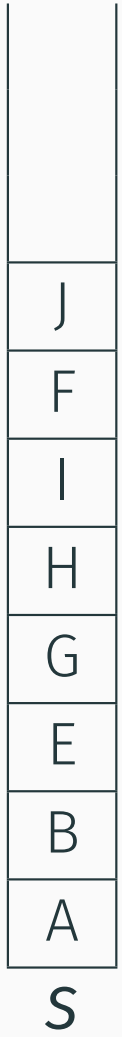


Depth-first graph traversal, step 13

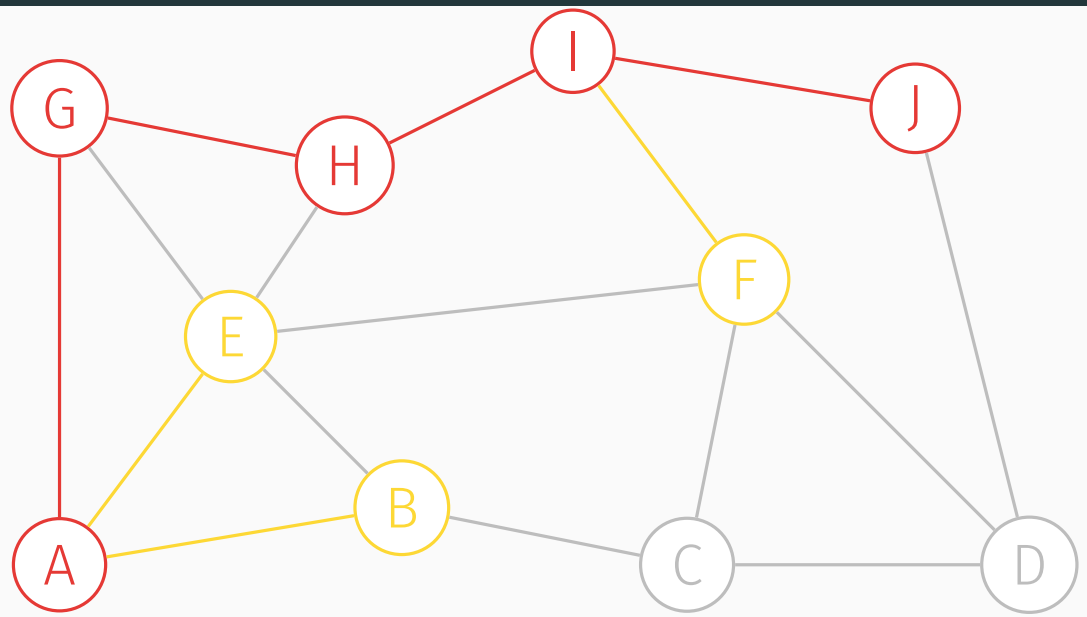


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	∞	∞	/
D	∞	∞	/
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	I
G	2	∞	A
H	3	∞	G
I	4	∞	H
J	∞	∞	I

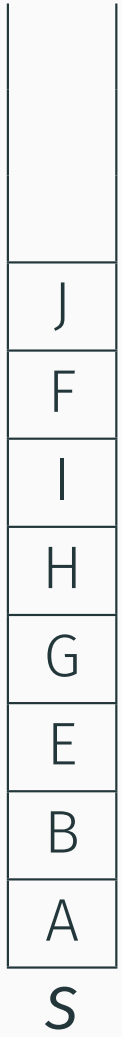


Depth-first graph traversal, step 14

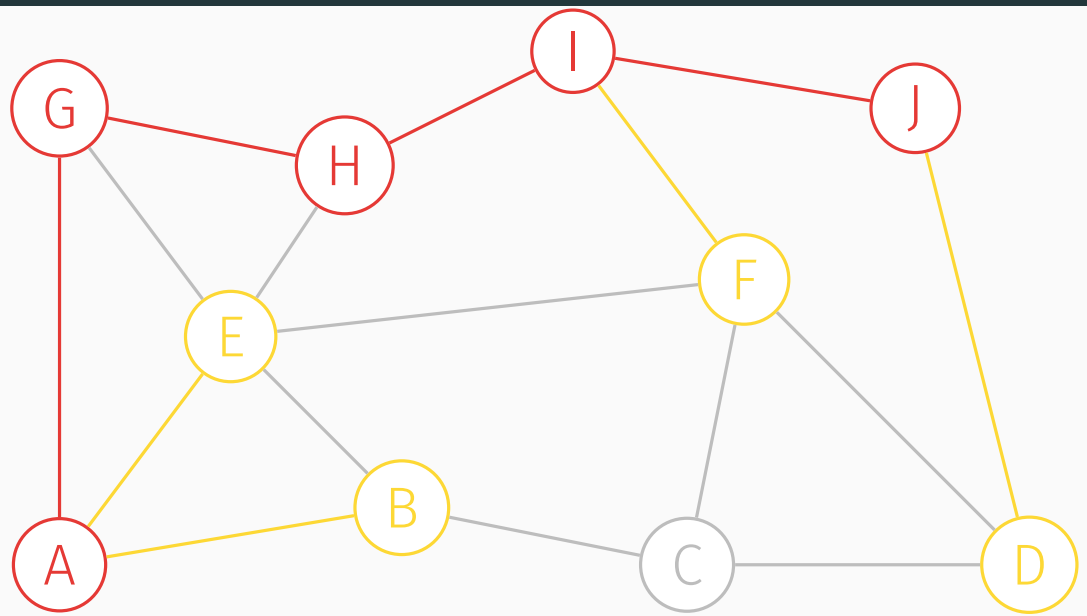


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	∞	∞	/
D	∞	∞	/
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	I
G	2	∞	A
H	3	∞	G
I	4	∞	H
J	5	∞	I

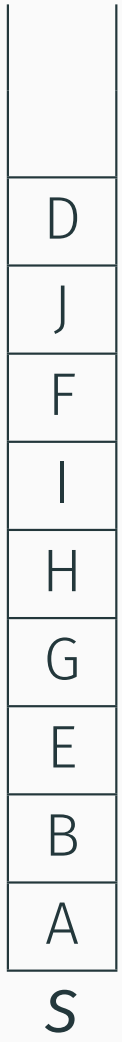


Depth-first graph traversal, step 15

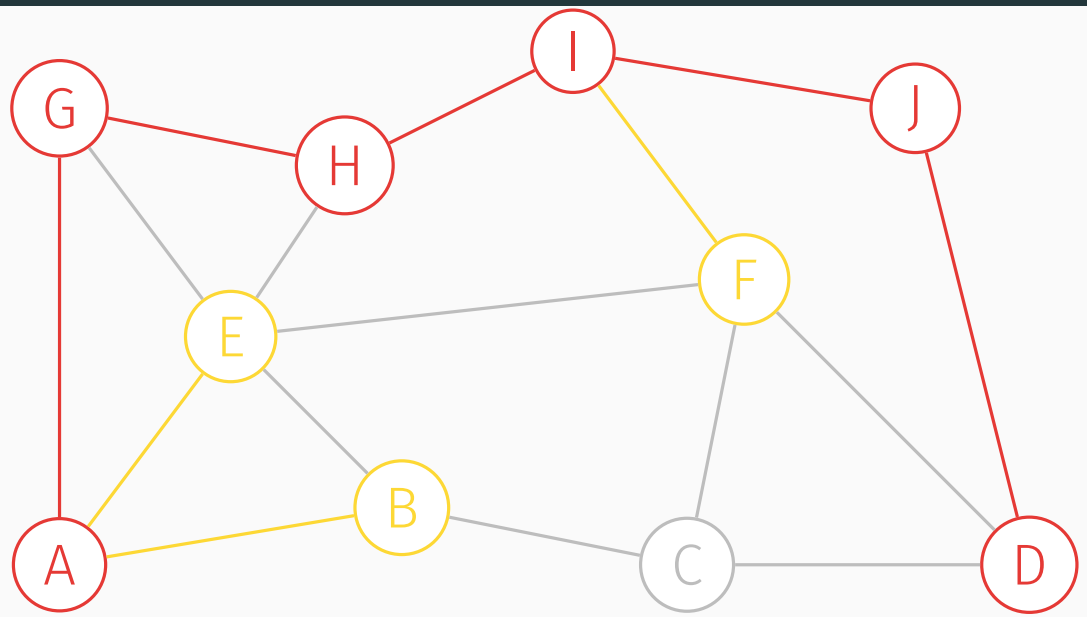


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	∞	∞	/
D	∞	∞	J
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	I
G	2	∞	A
H	3	∞	G
I	4	∞	H
J	5	∞	I

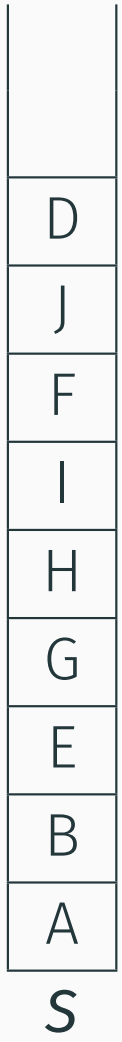


Depth-first graph traversal, step 16

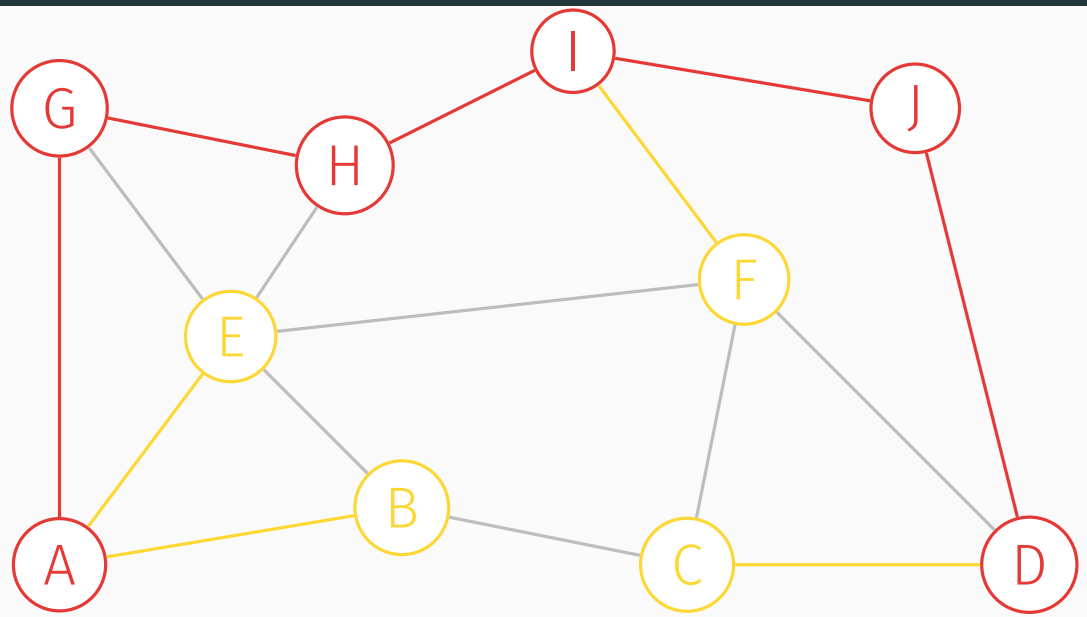


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	∞	∞	/
D	6	∞	J
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	I
G	2	∞	A
H	3	∞	G
I	4	∞	H
J	5	∞	I

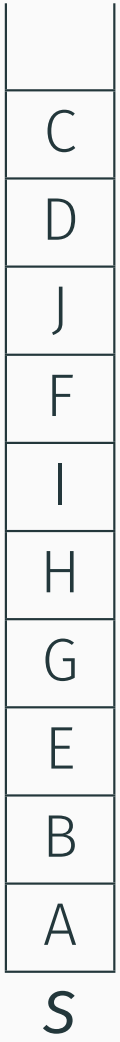


Depth-first graph traversal, step 17

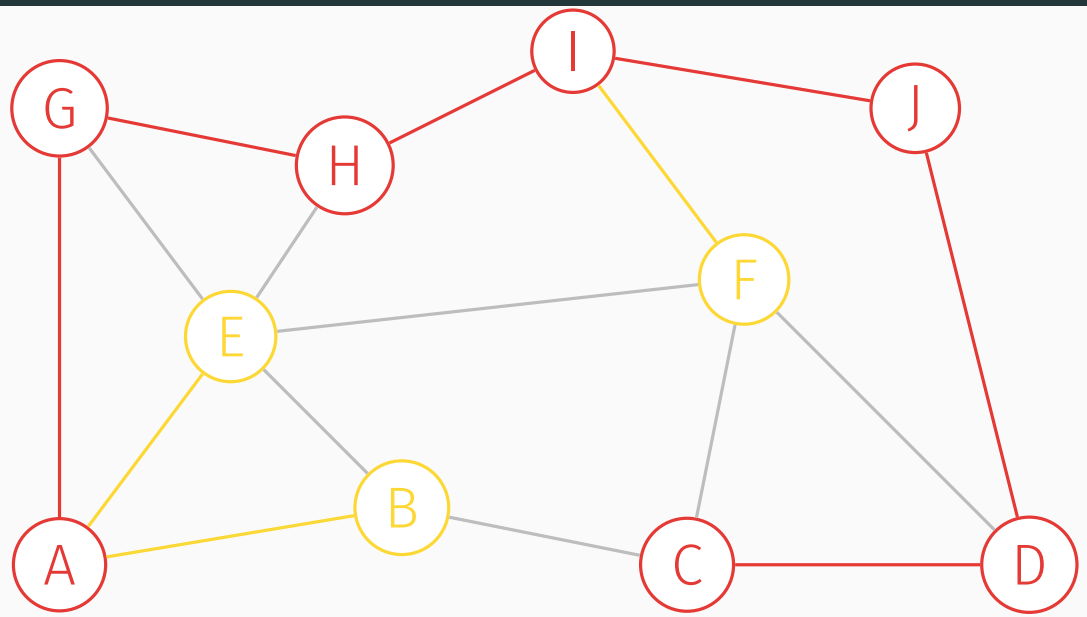


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	∞	∞	D
D	6	∞	J
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	I
G	2	∞	A
H	3	∞	G
I	4	∞	H
J	5	∞	I

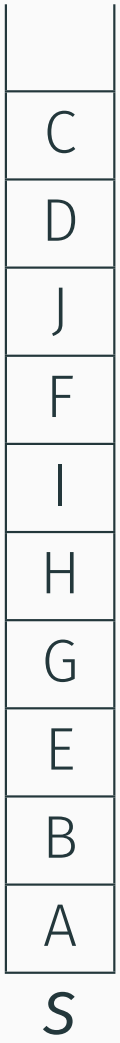


Depth-first graph traversal, step 18

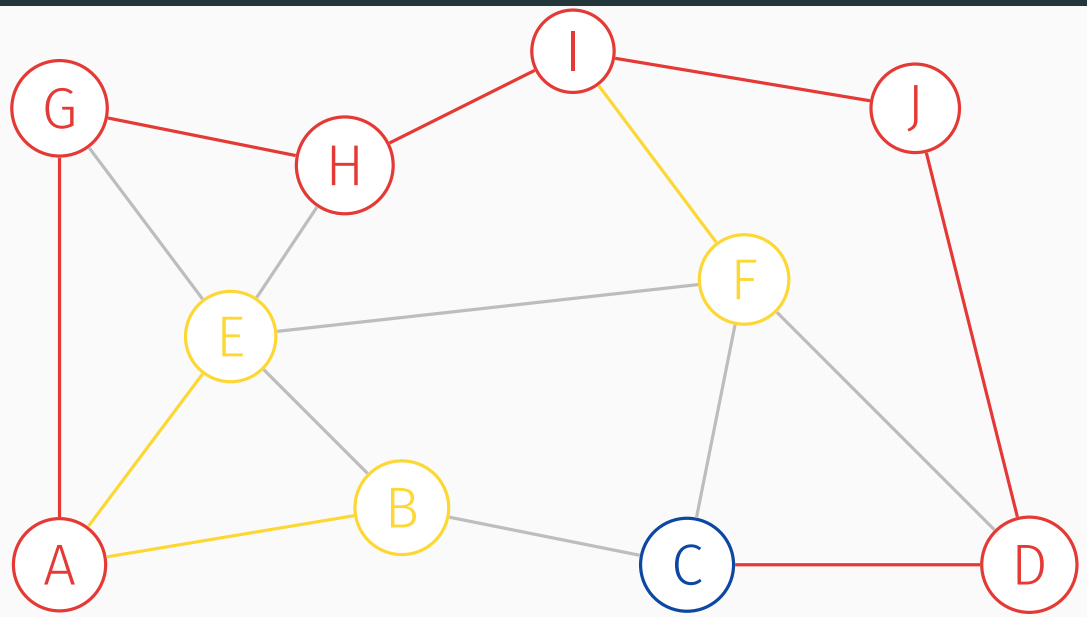


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	7	∞	D
D	6	∞	J
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	I
G	2	∞	A
H	3	∞	G
I	4	∞	H
J	5	∞	I

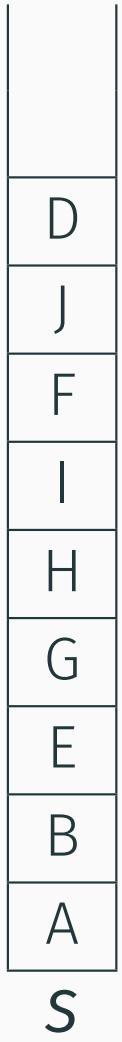


Depth-first graph traversal, step 19

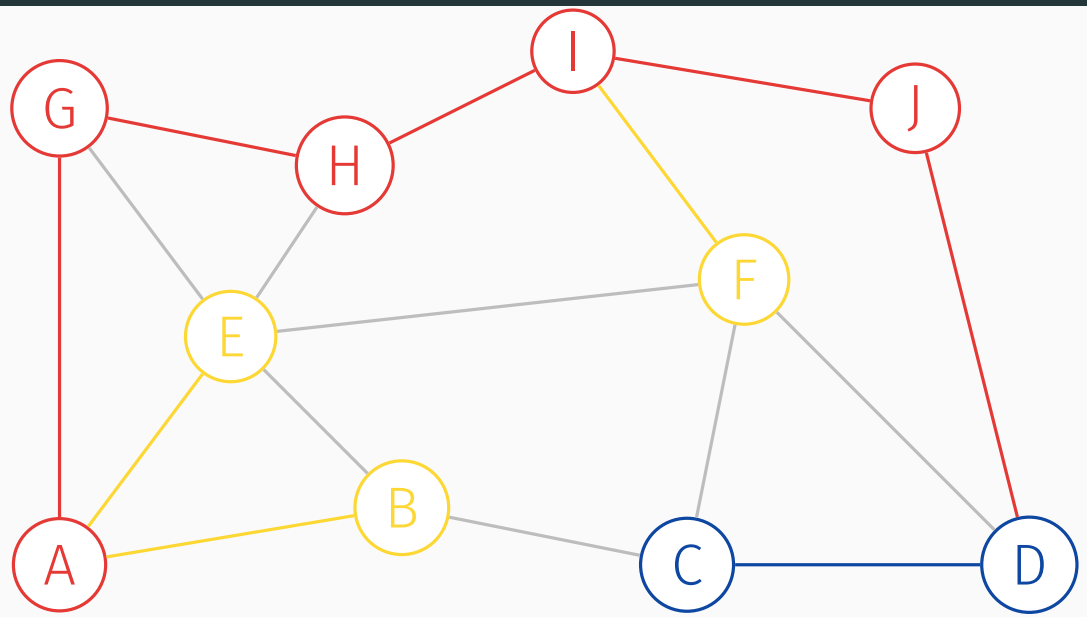


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	7	8	D
D	6	∞	J
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	I
G	2	∞	A
H	3	∞	G
I	4	∞	H
J	5	∞	I

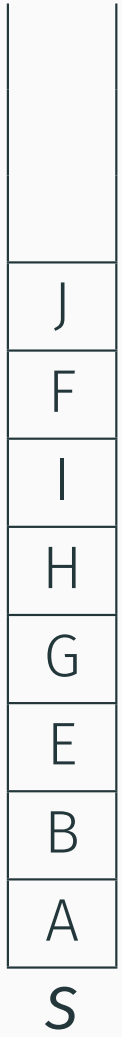


Depth-first graph traversal, step 20

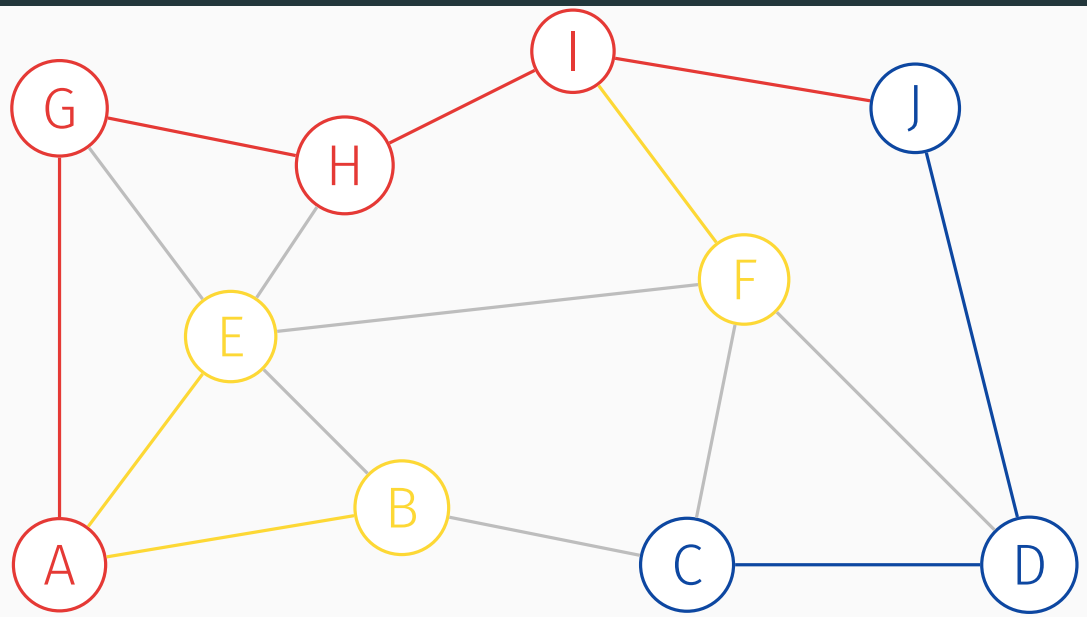


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	7	8	D
D	6	9	J
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	I
G	2	∞	A
H	3	∞	G
I	4	∞	H
J	5	∞	I

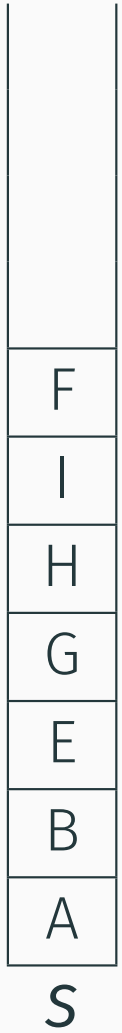


Depth-first graph traversal, step 21

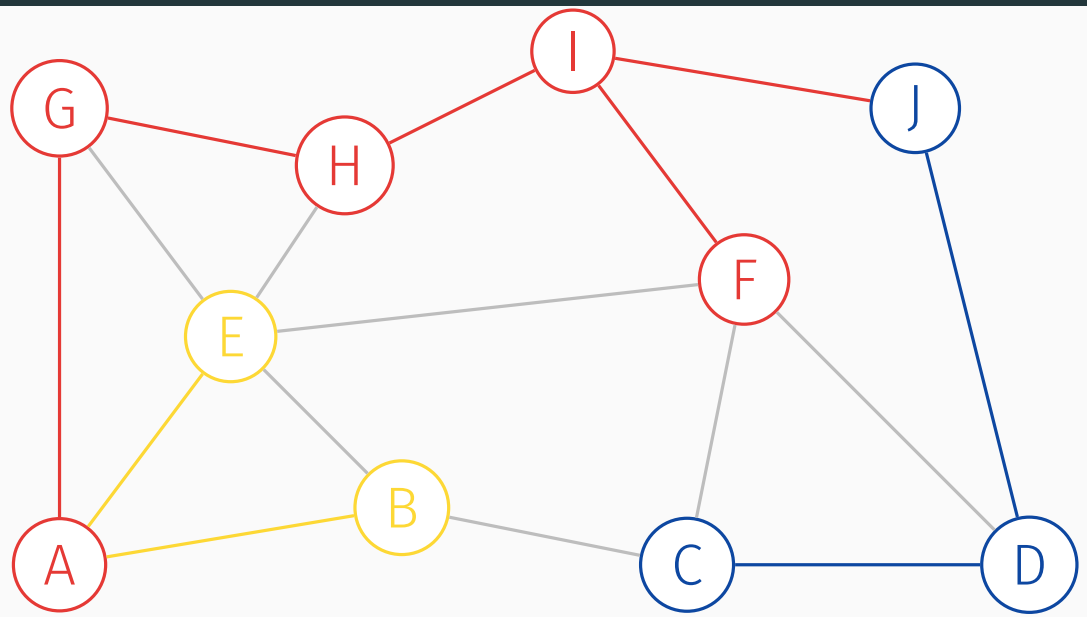


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	7	8	D
D	6	9	J
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	∞	∞	I
G	2	∞	A
H	3	∞	G
I	4	∞	H
J	5	10	I

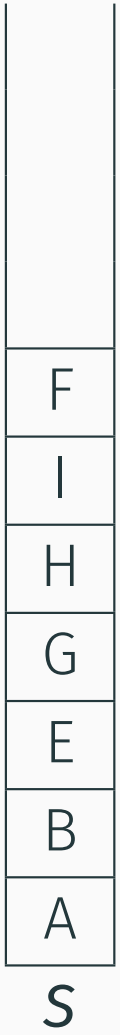


Depth-first graph traversal, step 22

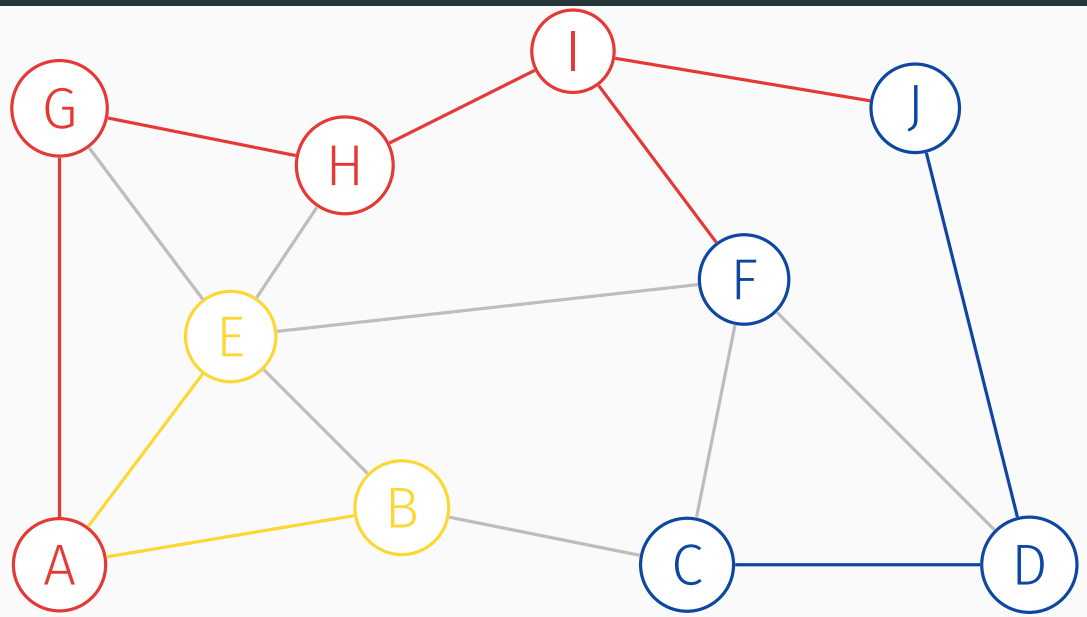


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	7	8	D
D	6	9	J
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	11	∞	I
G	2	∞	A
H	3	∞	G
I	4	∞	H
J	5	10	I

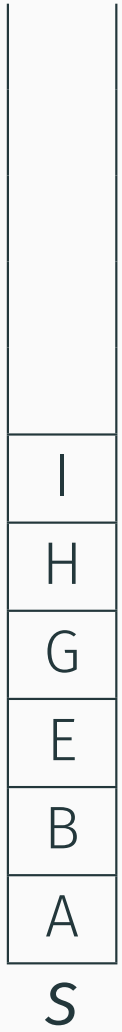


Depth-first graph traversal, step 23

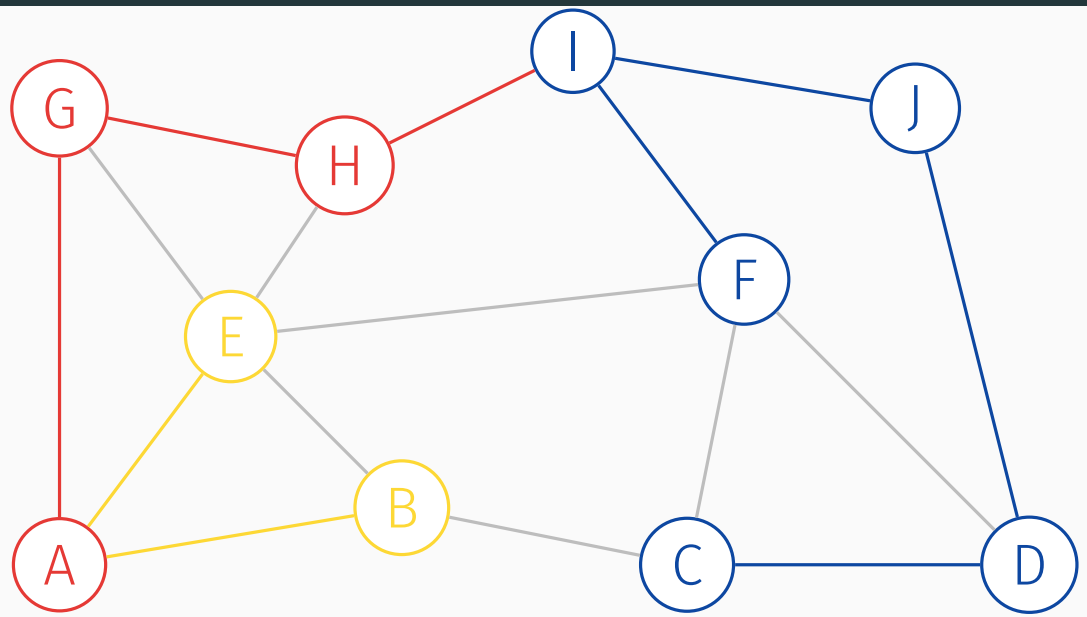


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	7	8	D
D	6	9	J
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	11	12	I
G	2	∞	A
H	3	∞	G
I	4	∞	H
J	5	10	I

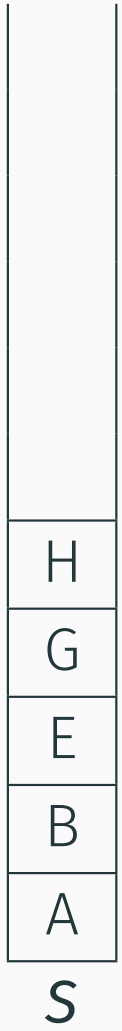


Depth-first graph traversal, step 24

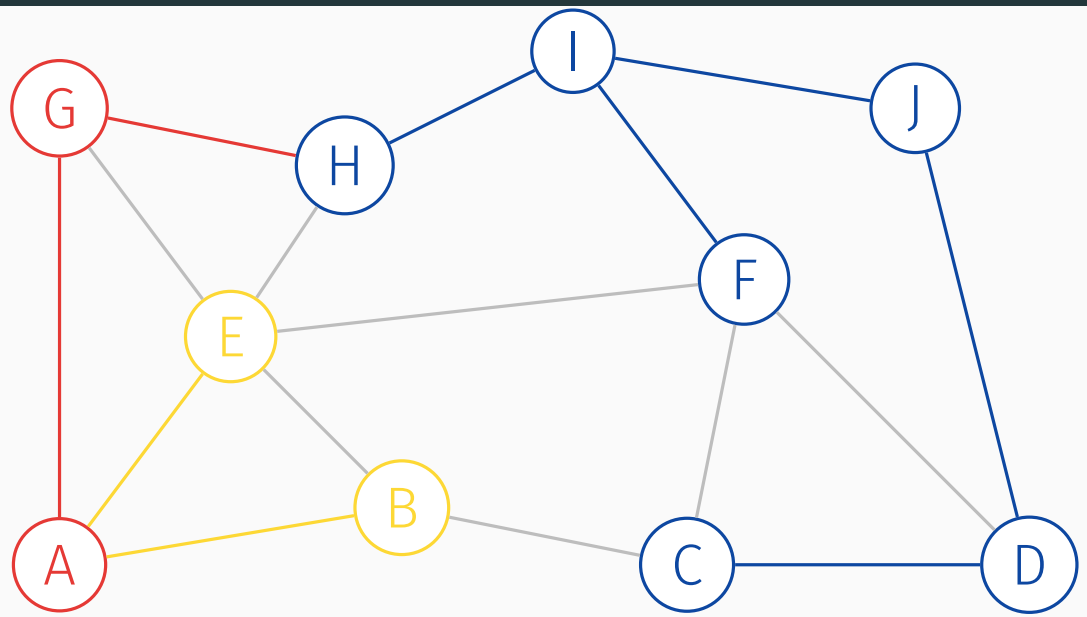


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	7	8	D
D	6	9	J
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	11	12	I
G	2	∞	A
H	3	∞	G
I	4	13	H
J	5	10	I



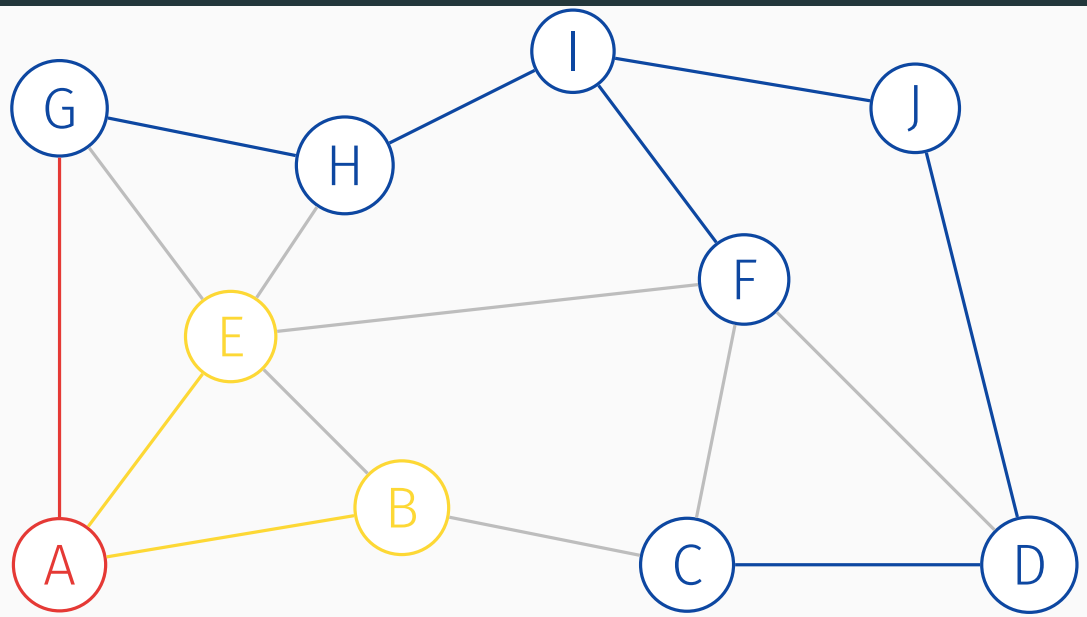
Depth-first graph traversal, step 25



v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	7	8	D
D	6	9	J
E	∞	∞	A

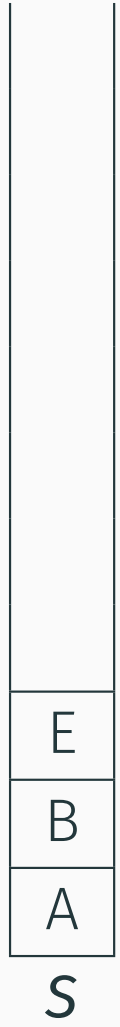
v	$d[v]$	$f[v]$	$\pi[v]$
F	11	12	I
G	2	∞	A
H	3	14	G
I	4	13	H
J	5	10	I

Depth-first graph traversal, step 26

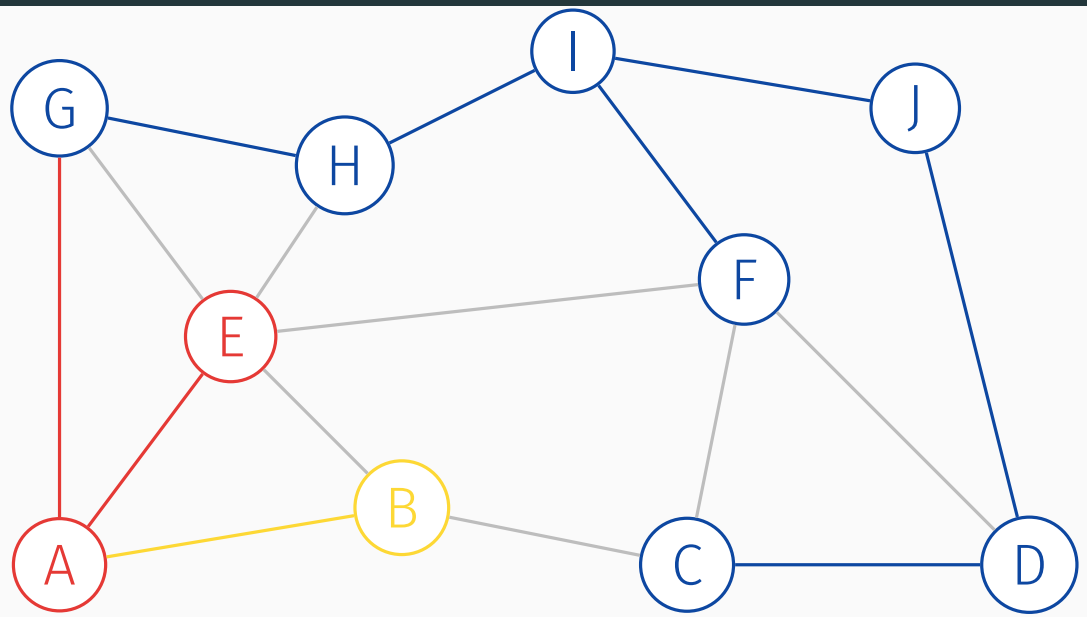


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	7	8	D
D	6	9	J
E	∞	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	11	12	I
G	2	15	A
H	3	14	G
I	4	13	H
J	5	10	I

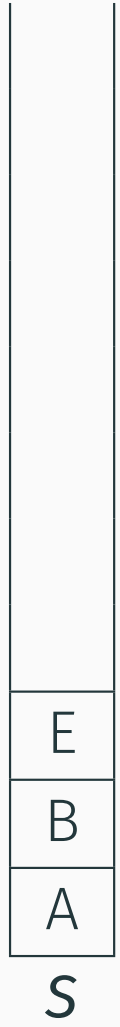


Depth-first graph traversal, step 27

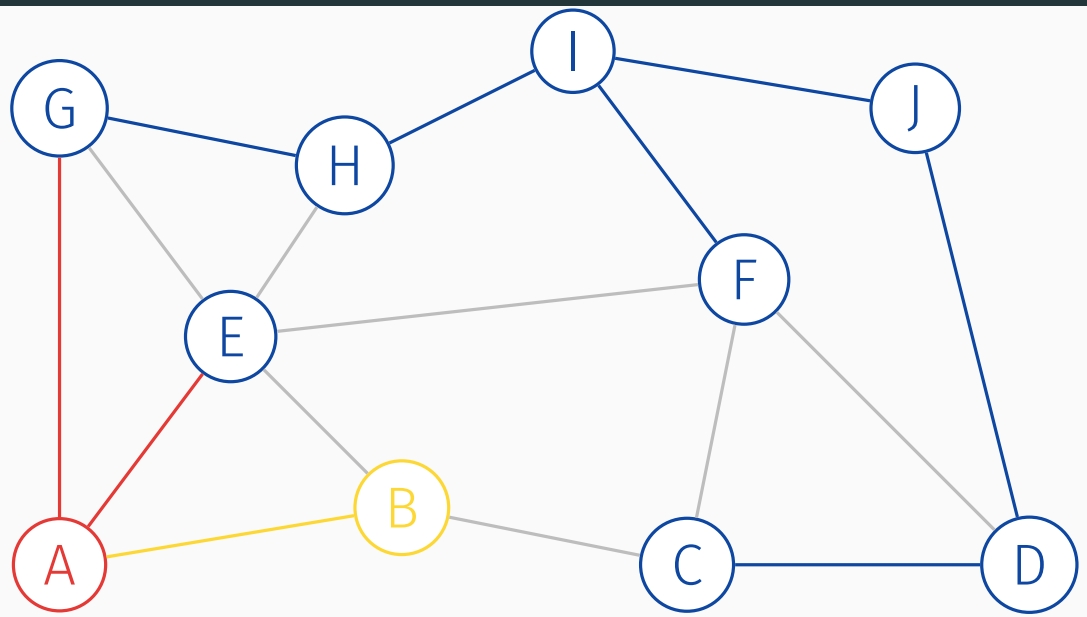


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	7	8	D
D	6	9	J
E	16	∞	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	11	12	I
G	2	15	A
H	3	14	G
I	4	13	H
J	5	10	I

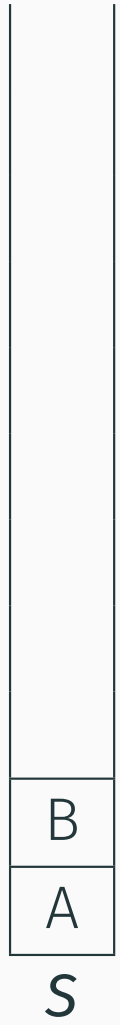


Depth-first graph traversal, step 28

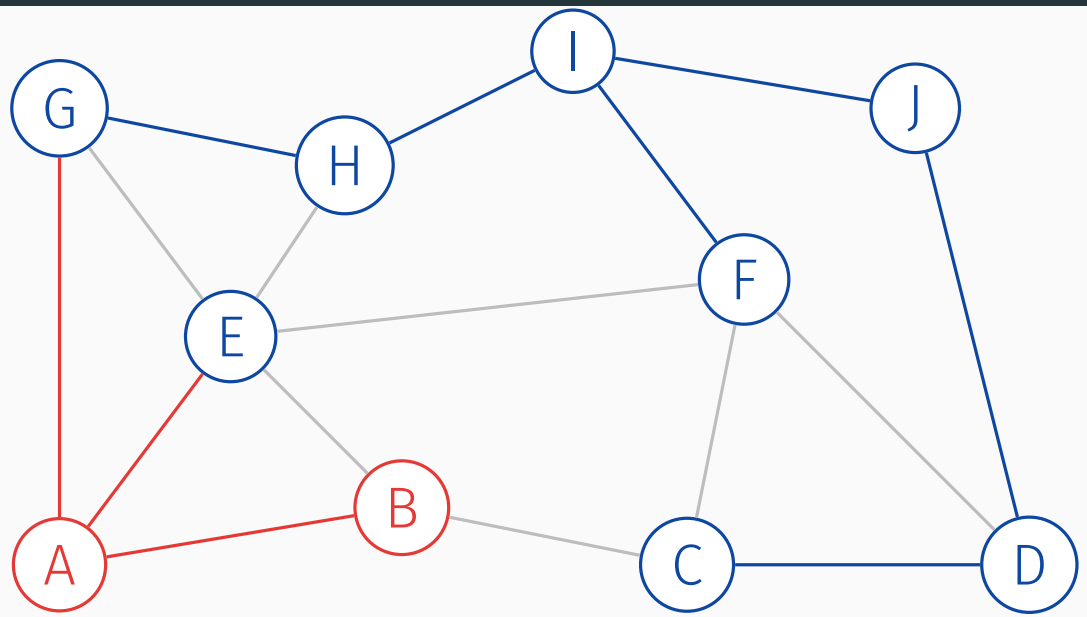


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	∞	∞	A
C	7	8	D
D	6	9	J
E	16	17	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	11	12	I
G	2	15	A
H	3	14	G
I	4	13	H
J	5	10	I

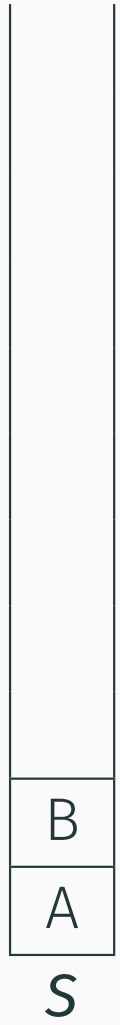


Depth-first graph traversal, step 29

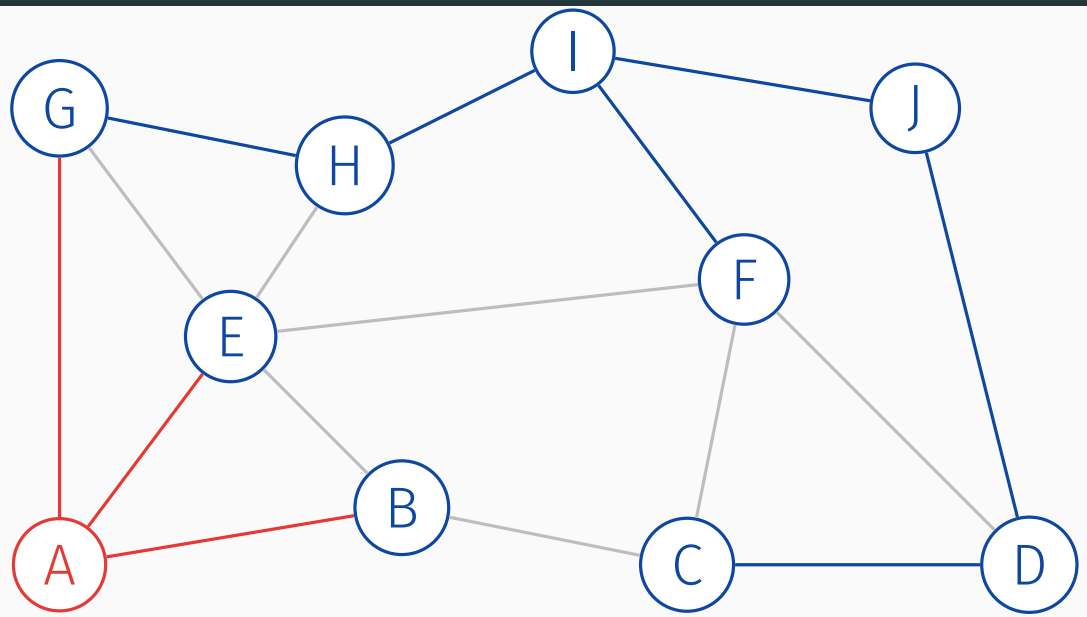


v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	18	∞	A
C	7	8	D
D	6	9	J
E	16	17	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	11	12	I
G	2	15	A
H	3	14	G
I	4	13	H
J	5	10	I



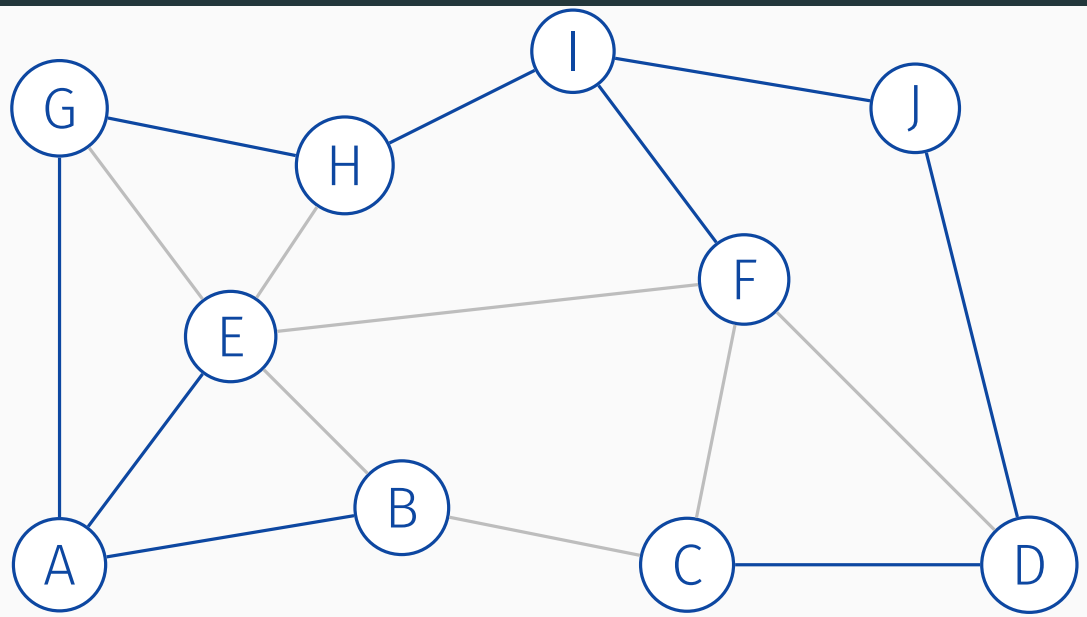
Depth-first graph traversal, step 30



v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/
B	18	19	A
C	7	8	D
D	6	9	J
E	16	17	A

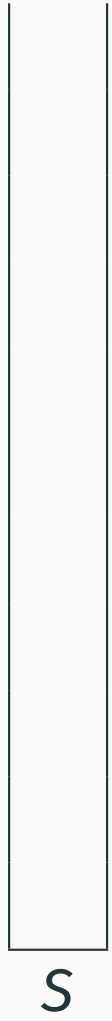
v	$d[v]$	$f[v]$	$\pi[v]$
F	11	12	I
G	2	15	A
H	3	14	G
I	4	13	H
J	5	10	I

Depth-first graph traversal, step 31



v	$d[v]$	$f[v]$	$\pi[v]$
A	1	20	/
B	18	19	A
C	7	8	D
D	6	9	J
E	16	17	A

v	$d[v]$	$f[v]$	$\pi[v]$
F	11	12	I
G	2	15	A
H	3	14	G
I	4	13	H
J	5	10	I



Algorithm for breadth-first graph traversal

Input : Graph $G(V, E)$, initial vertex $s \in V$
Output: BF tree

```
1  foreach u ∈ V/{s} do
2  |   state [u] ← unknown;
3  |   d [u] ← ∞;
4  |   π [u] ← nothing;
5  end
6  Q ← ∅;
7  state [s] ← discovered;
8  d [s] ← 0;
9  π [s] ← nothing;
10 Enqueue(Q, s);
```

Algorithm for breadth-first graph traversal (cont.)

```
11 while Q ≠ ∅ do
12 |   u ← Dequeue(Q);
13 |   foreach v ∈ Adj(G, u) do
14 |   |   if state [v] = unknown then
15 |   |   |   state [v] ← discovered;
16 |   |   |   d [v] ← d [u] + 1;
17 |   |   |   π [v] ← u;
18 |   |   |   Enqueue (Q,v);
19 |   |   end
20 |   end
21 |   state [u] ← finished;
22 end
```

Animation of breadth-first graph traversal – legend

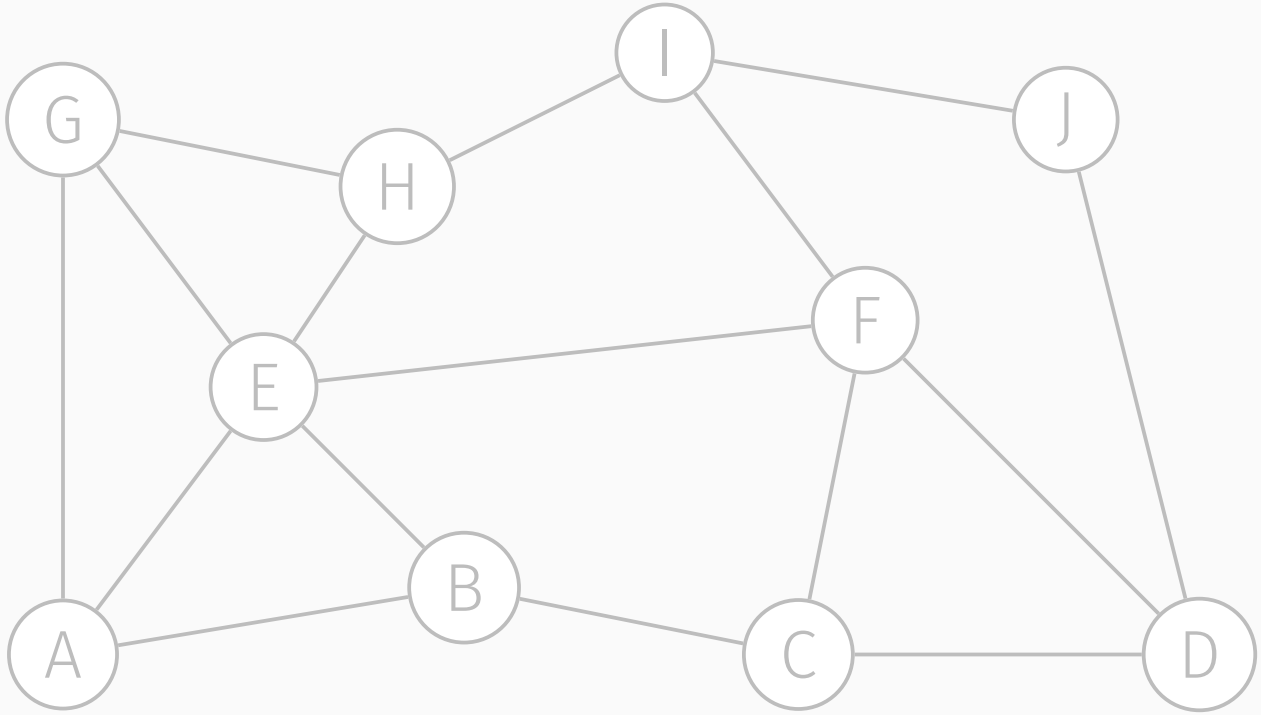
Graph vertices

grey	vertex in unknown state
yellow	vertex in discovered state
red	currently processed vertex
blue	vertex in finished state

Graph edges

grey	edge between vertices in unknown state or edge not belonging to the BF-tree
yellow	edge incident with vertices in discovered state
red	edge incident with currently processed vertex
blue	edge between vertices in finished state

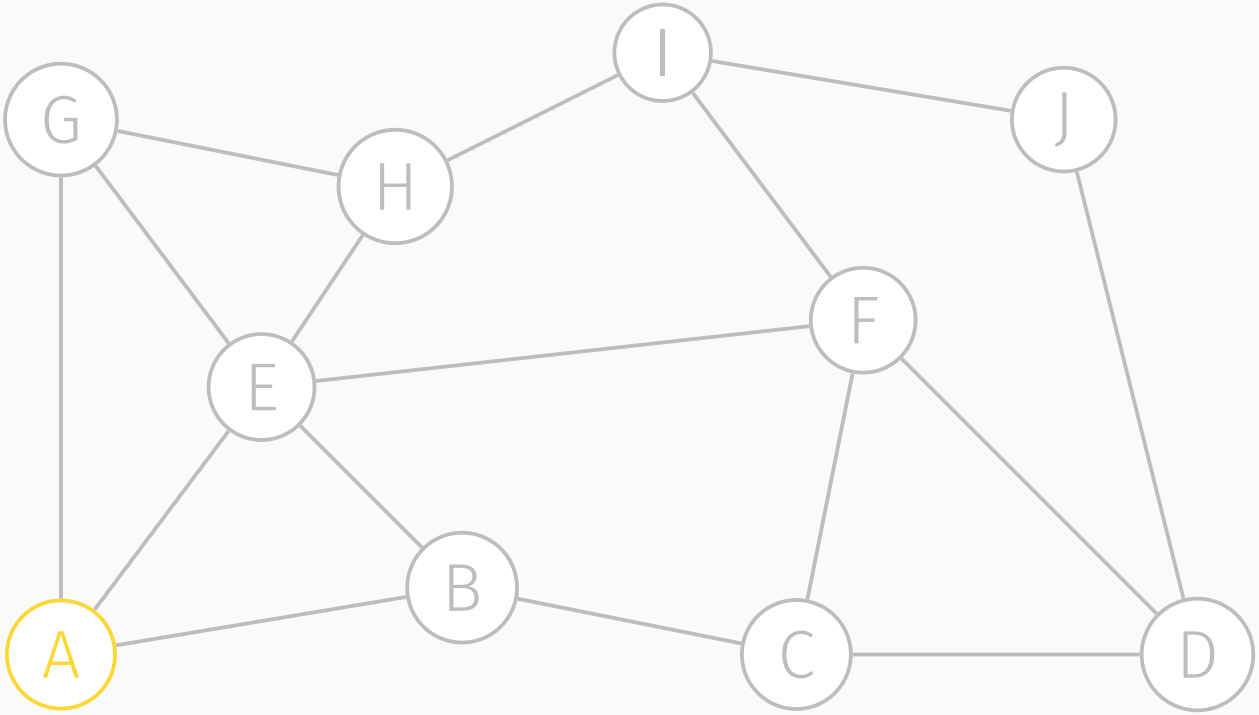
Breadth-first graph traversal, step 1



Q [ ]

v	$d[v]$	$\pi[v]$
A	∞	/
B	∞	/
C	∞	/
D	∞	/
E	∞	/
F	∞	/
G	∞	/
H	∞	/
I	∞	/
J	∞	/

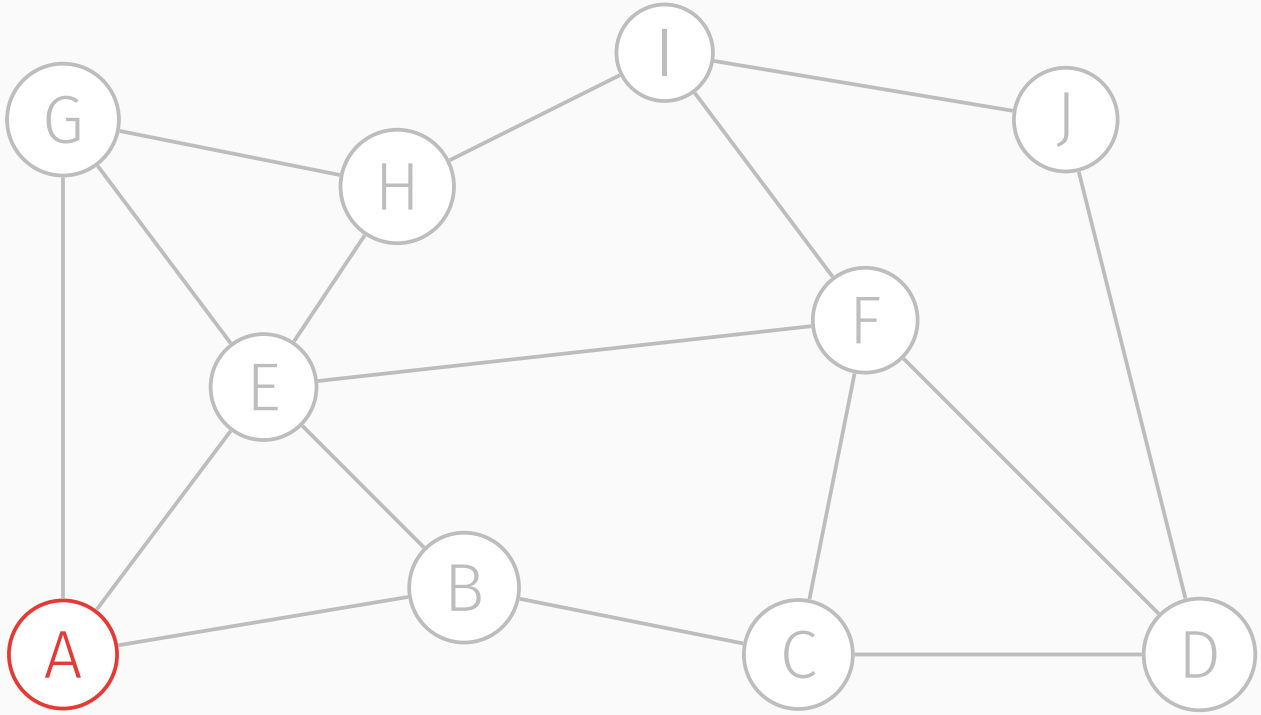
Breadth-first graph traversal, step 2



Q | A |

v	$d[v]$	$\pi[v]$
A	0	/
B	∞	/
C	∞	/
D	∞	/
E	∞	/
F	∞	/
G	∞	/
H	∞	/
I	∞	/
J	∞	/

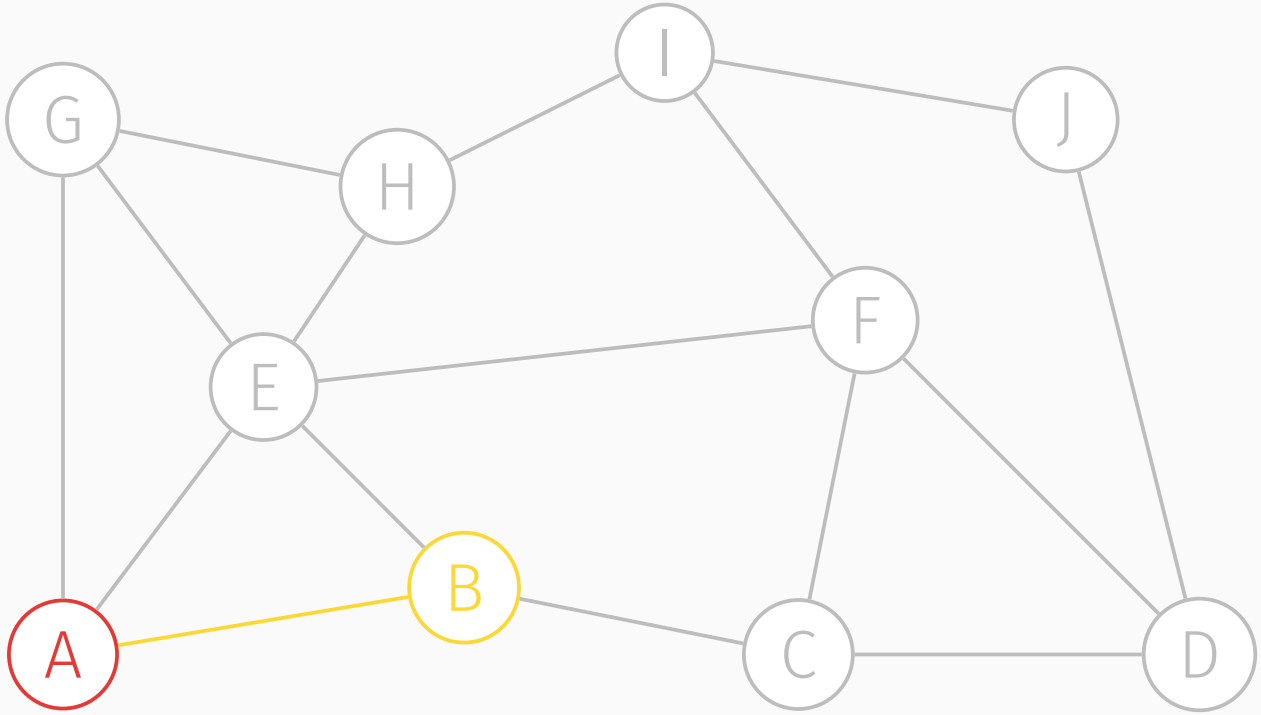
Breadth-first graph traversal, step 3



Q [ ]

v	$d[v]$	$\pi[v]$
A	0	/
B	∞	/
C	∞	/
D	∞	/
E	∞	/
F	∞	/
G	∞	/
H	∞	/
I	∞	/
J	∞	/

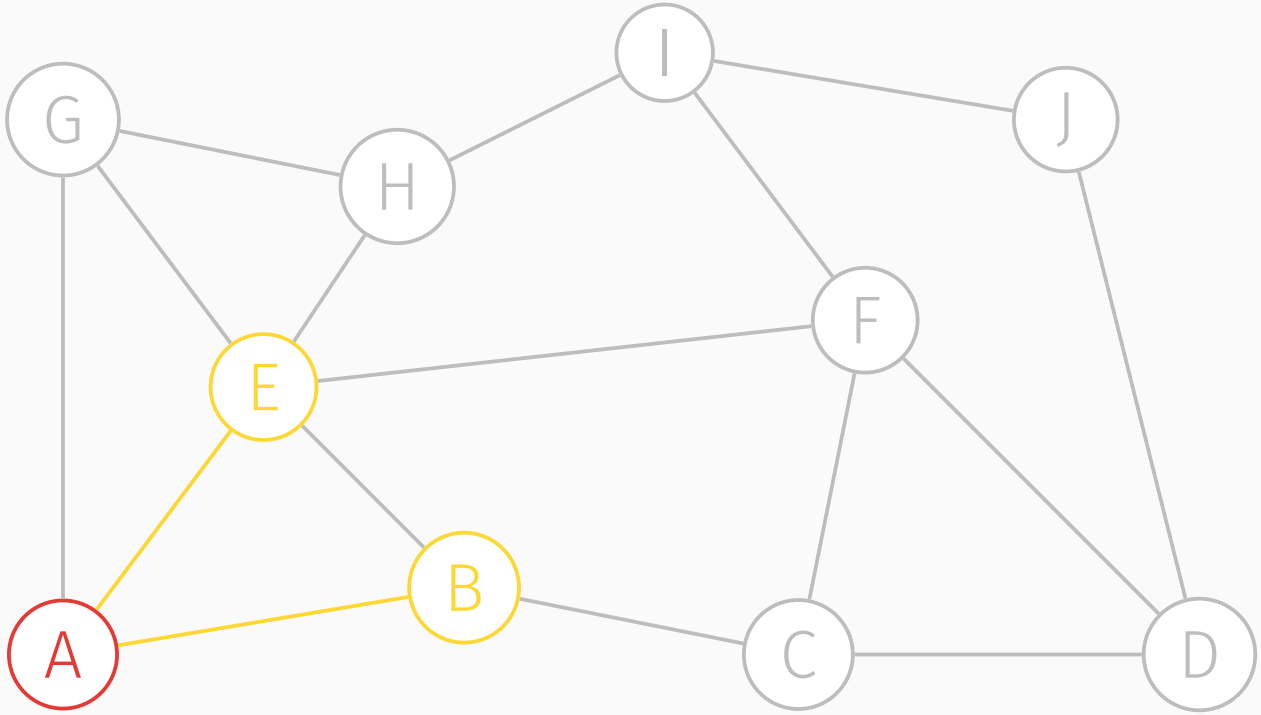
Breadth-first graph traversal, step 4



Q | B |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	∞	/
D	∞	/
E	∞	/
F	∞	/
G	∞	/
H	∞	/
I	∞	/
J	∞	/

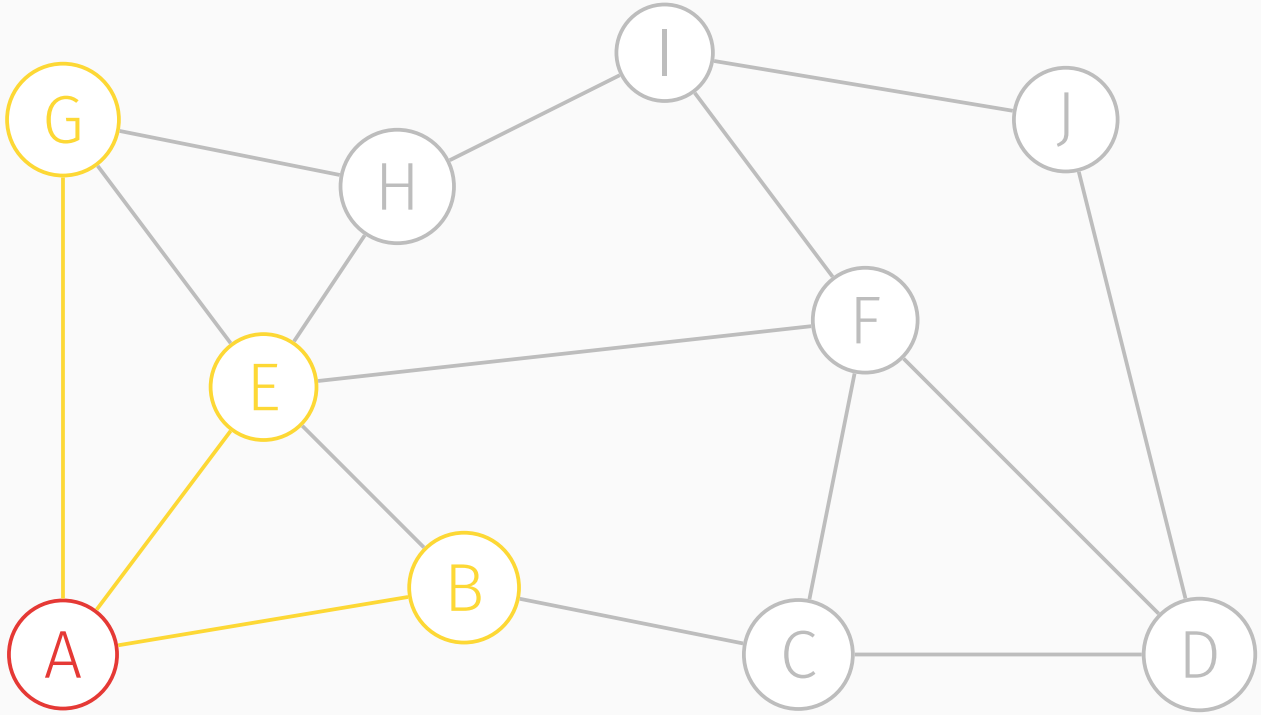
Breadth-first graph traversal, step 5



Q | B | E |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	∞	/
D	∞	/
E	1	A
F	∞	/
G	∞	/
H	∞	/
I	∞	/
J	∞	/

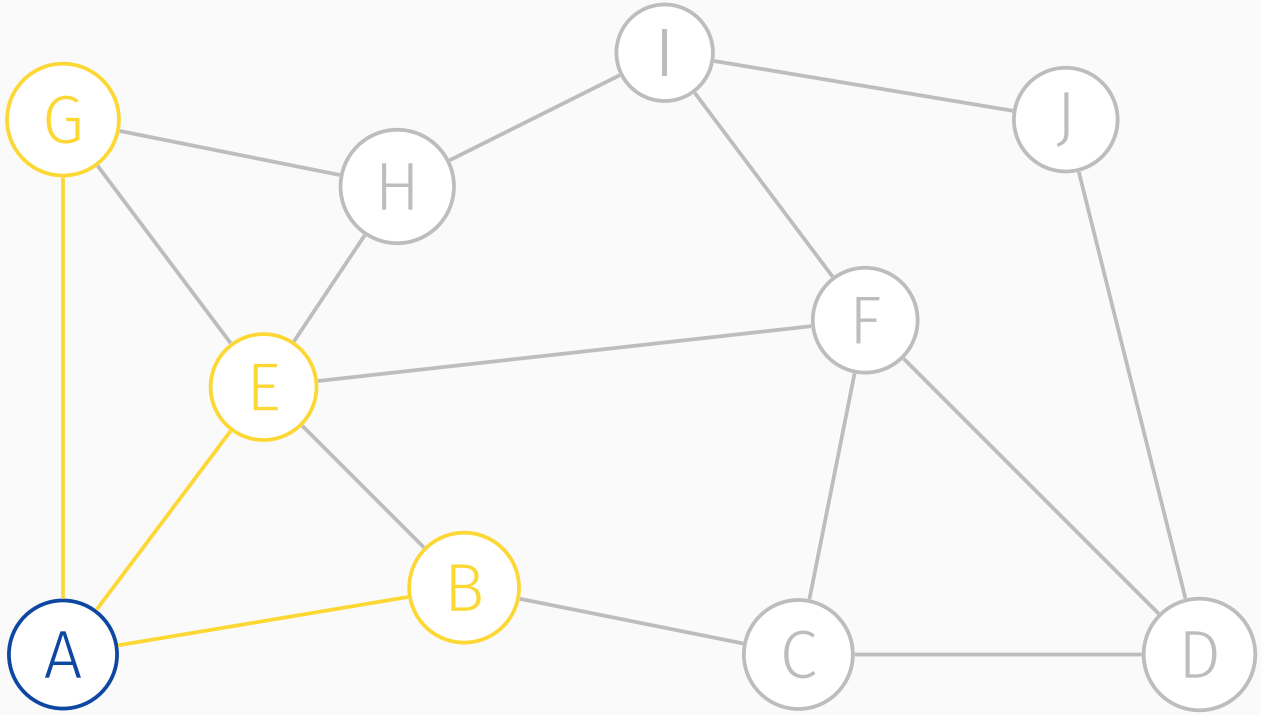
Breadth-first graph traversal, step 6



Q | B | E | G |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	∞	/
D	∞	/
E	1	A
F	∞	/
G	1	A
H	∞	/
I	∞	/
J	∞	/

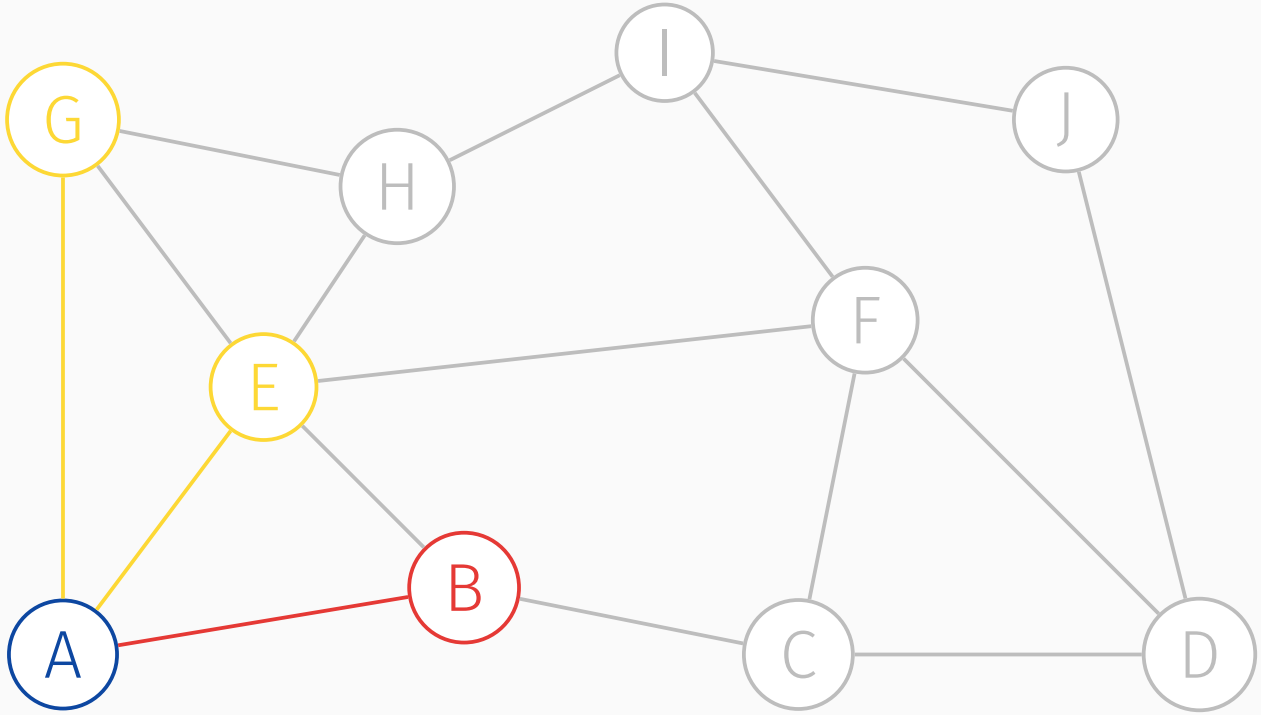
Breadth-first graph traversal, step 7



Q | B | E | G |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	∞	/
D	∞	/
E	1	A
F	∞	/
G	1	A
H	∞	/
I	∞	/
J	∞	/

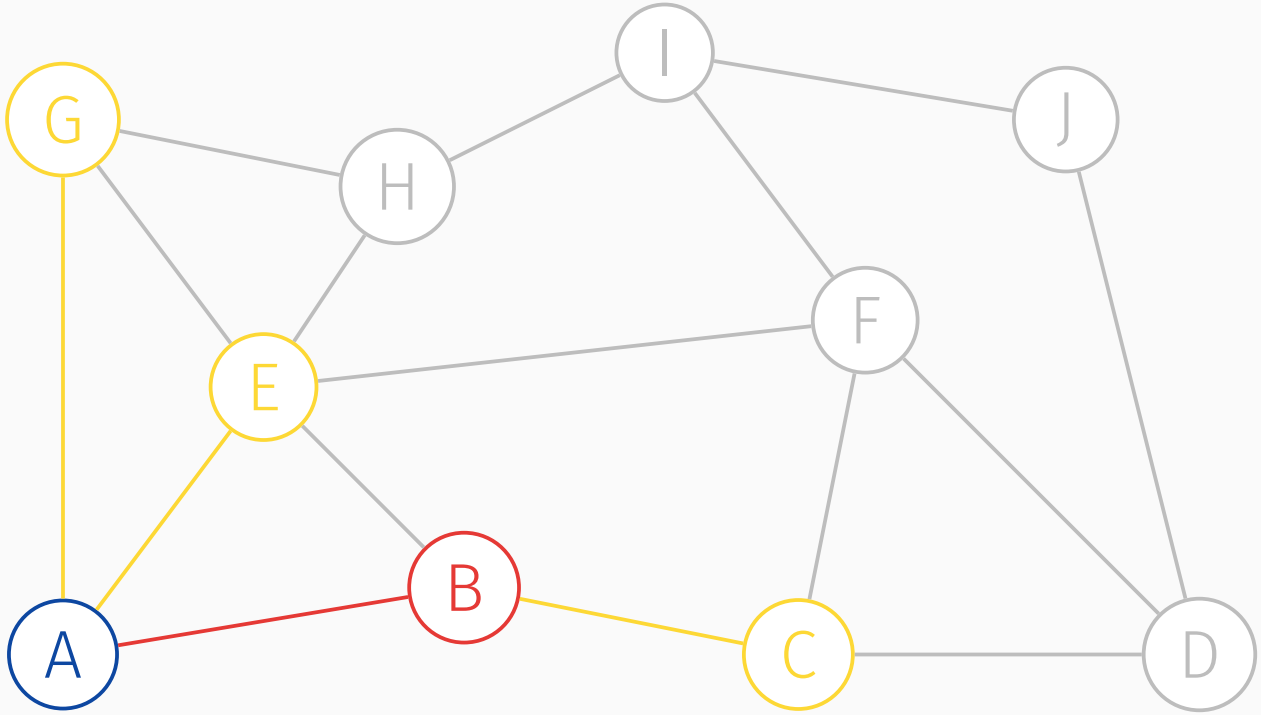
Breadth-first graph traversal, step 8



Q | E | G |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	∞	/
D	∞	/
E	1	A
F	∞	/
G	1	A
H	∞	/
I	∞	/
J	∞	/

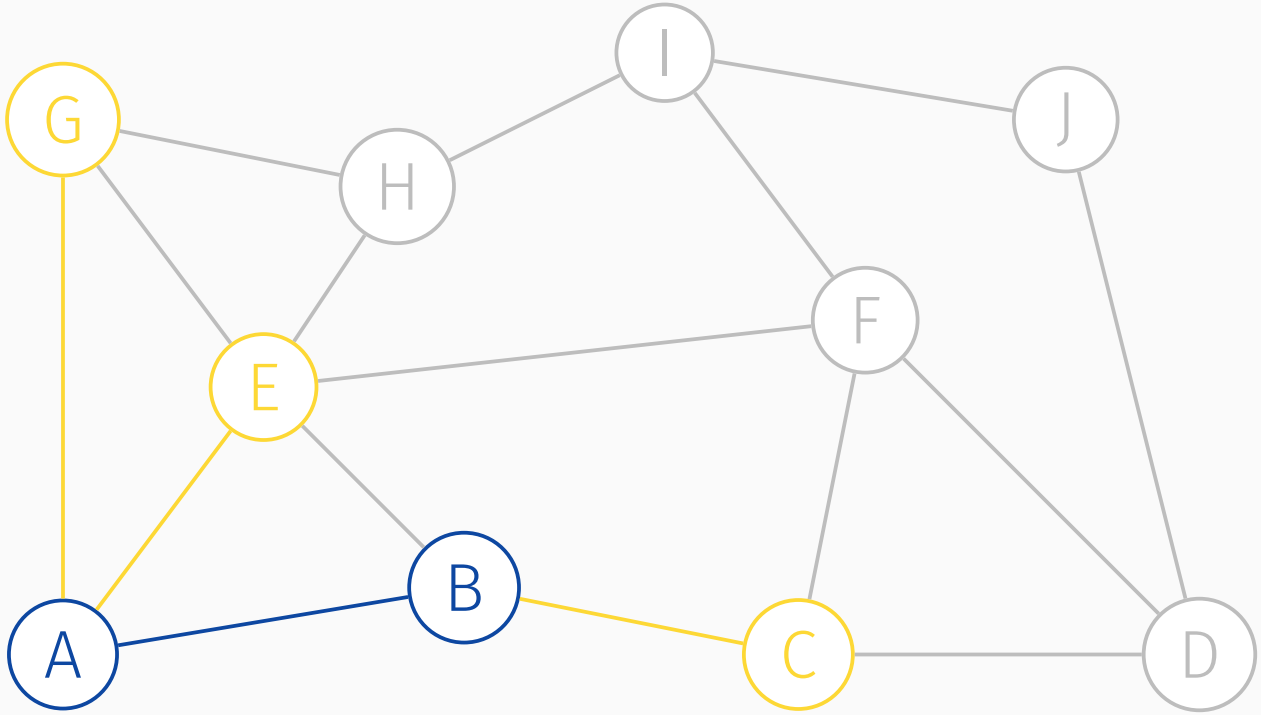
Breadth-first graph traversal, step 9



Q | E | G | C |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	∞	/
G	1	A
H	∞	/
I	∞	/
J	∞	/

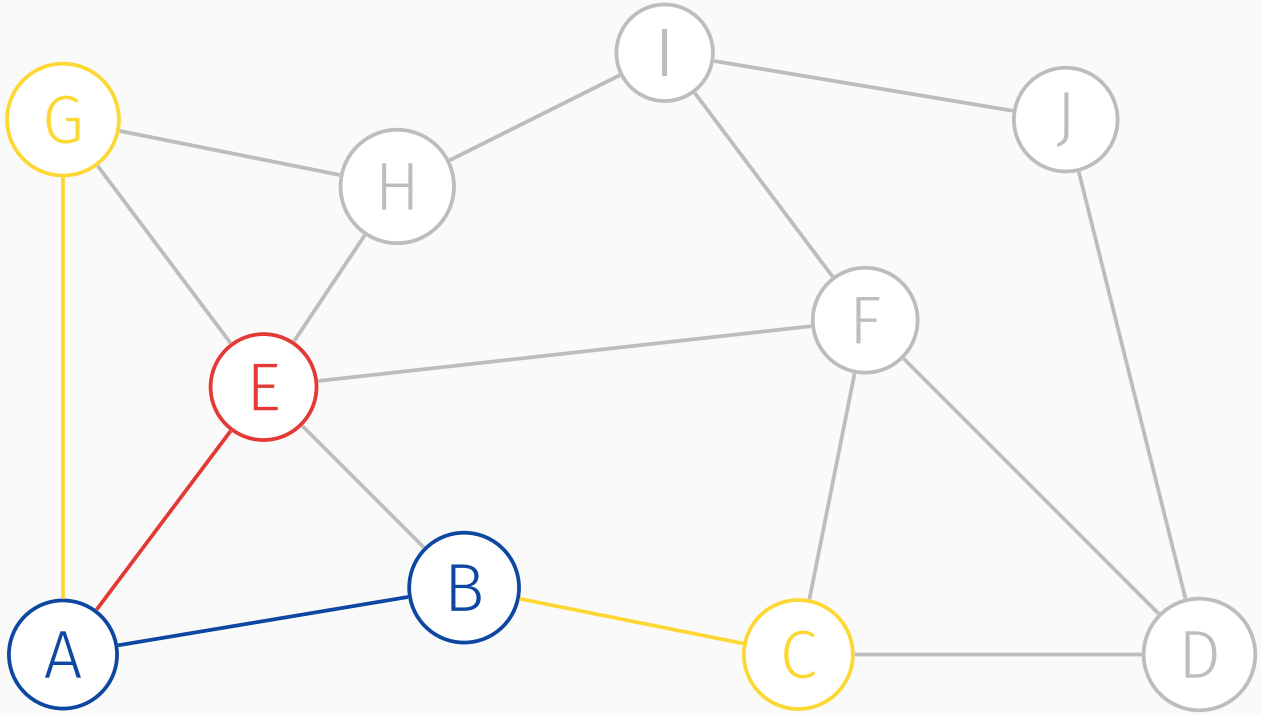
Breadth-first graph traversal, step 10



Q | E | G | C |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	∞	/
G	1	A
H	∞	/
I	∞	/
J	∞	/

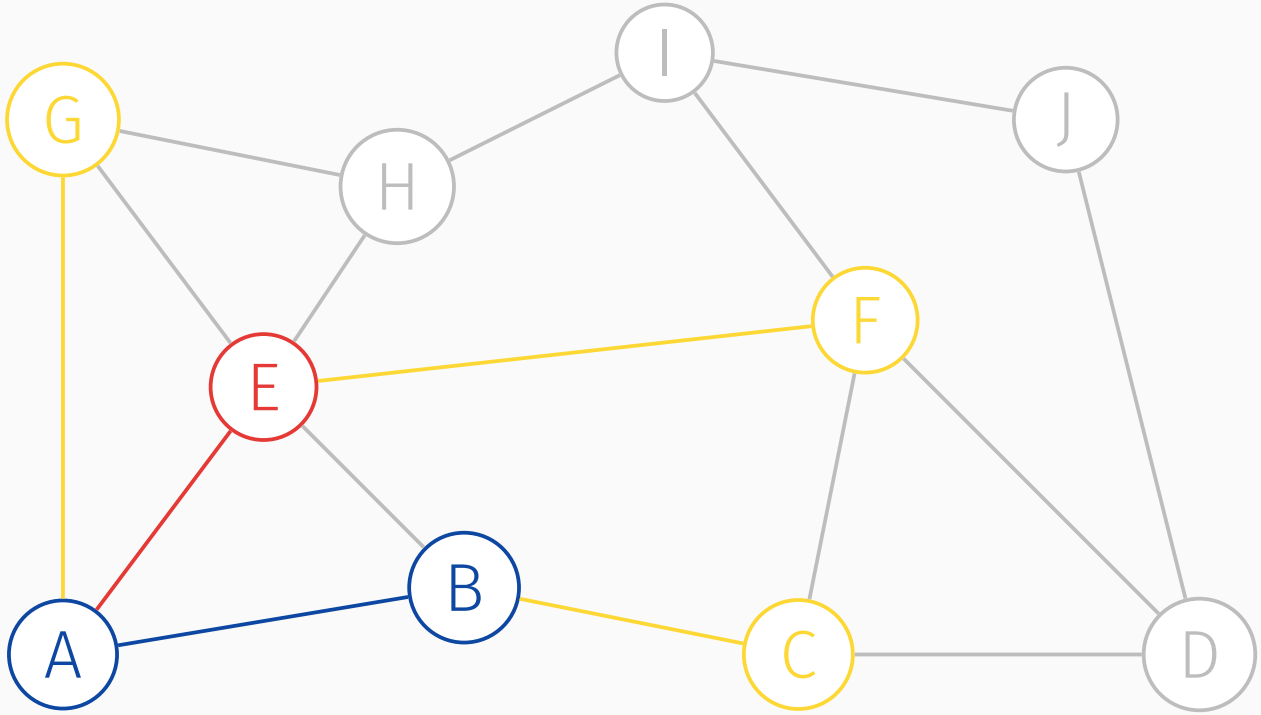
Breadth-first graph traversal, step 11



Q | G | C |

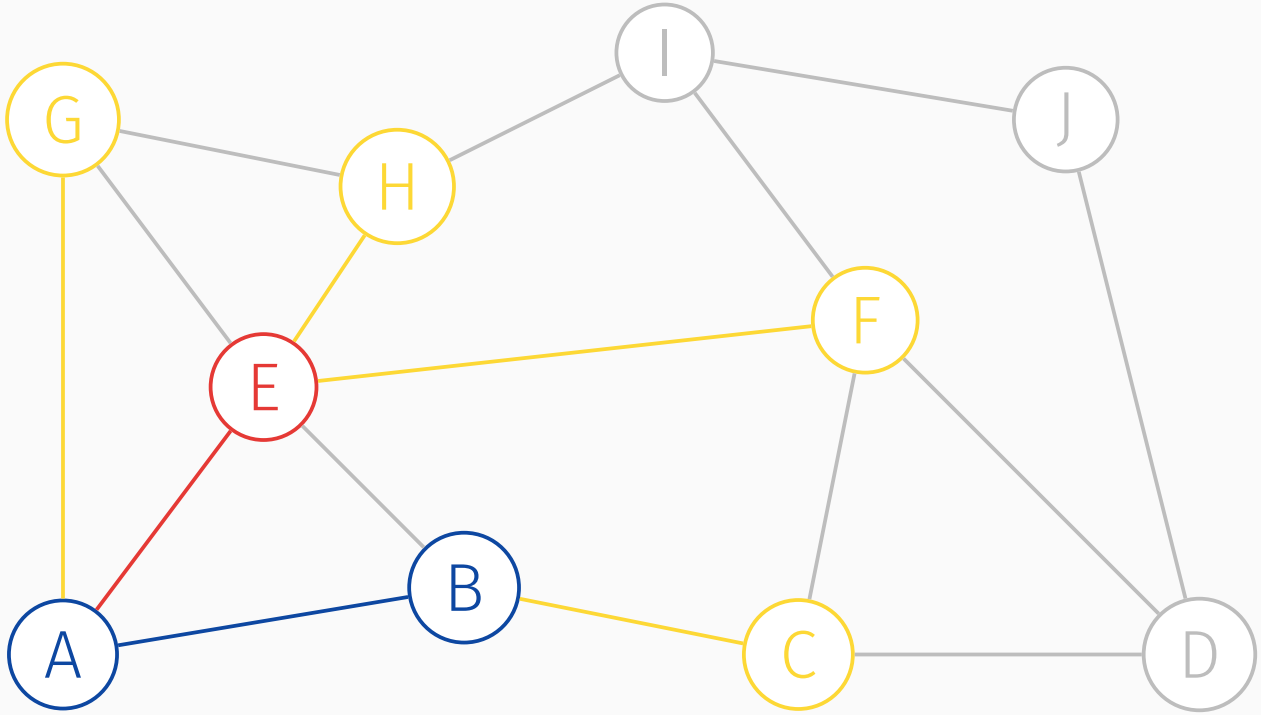
v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	∞	/
G	1	A
H	∞	/
I	∞	/
J	∞	/

Breadth-first graph traversal, step 12



v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	2	E
G	1	A
H	∞	/
I	∞	/
J	∞	/

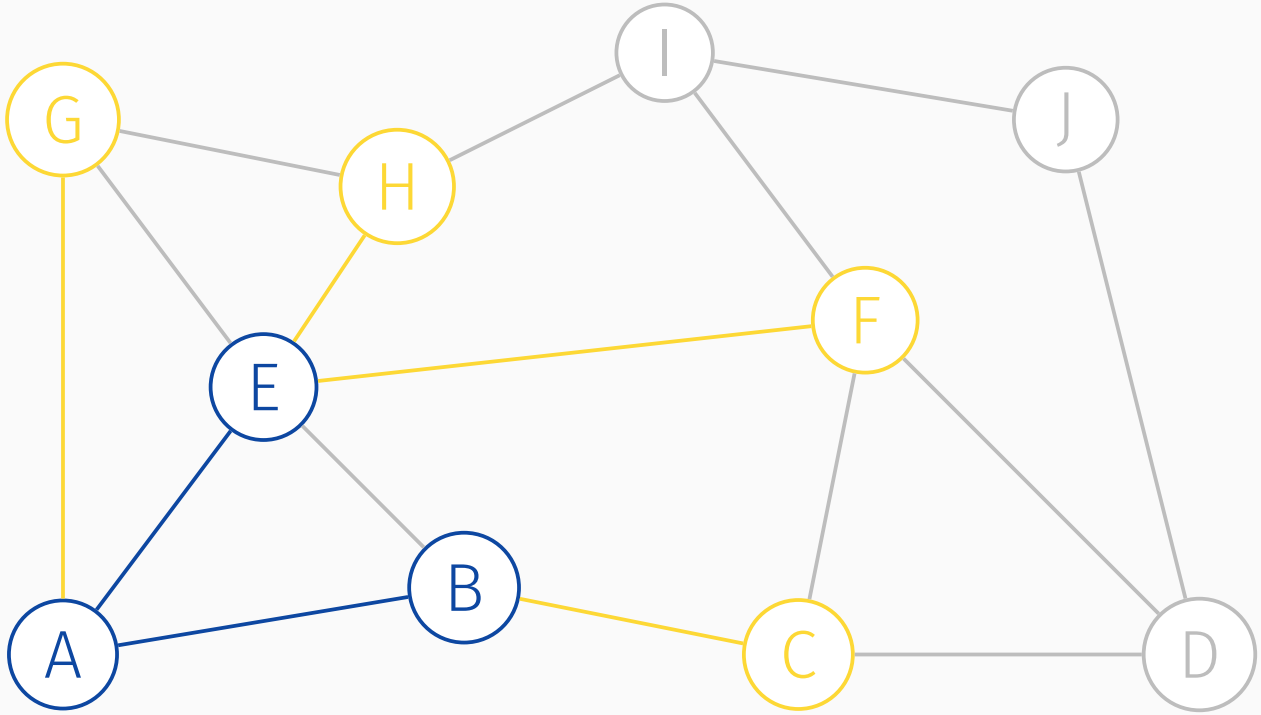
Breadth-first graph traversal, step 13



v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	2	E
G	1	A
H	2	E
I	∞	/
J	∞	/

Q | G | C | F | H |

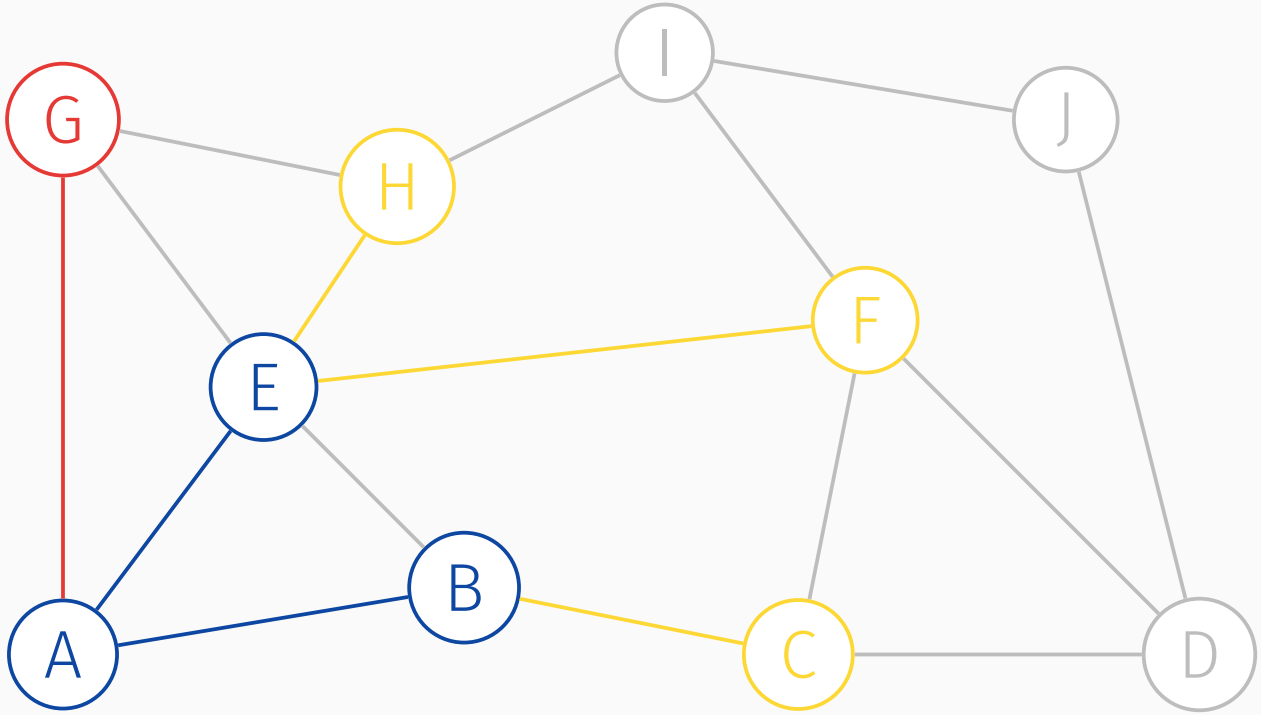
Breadth-first graph traversal, step 14



v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	2	E
G	1	A
H	2	E
I	∞	/
J	∞	/

Q | G | C | F | H |

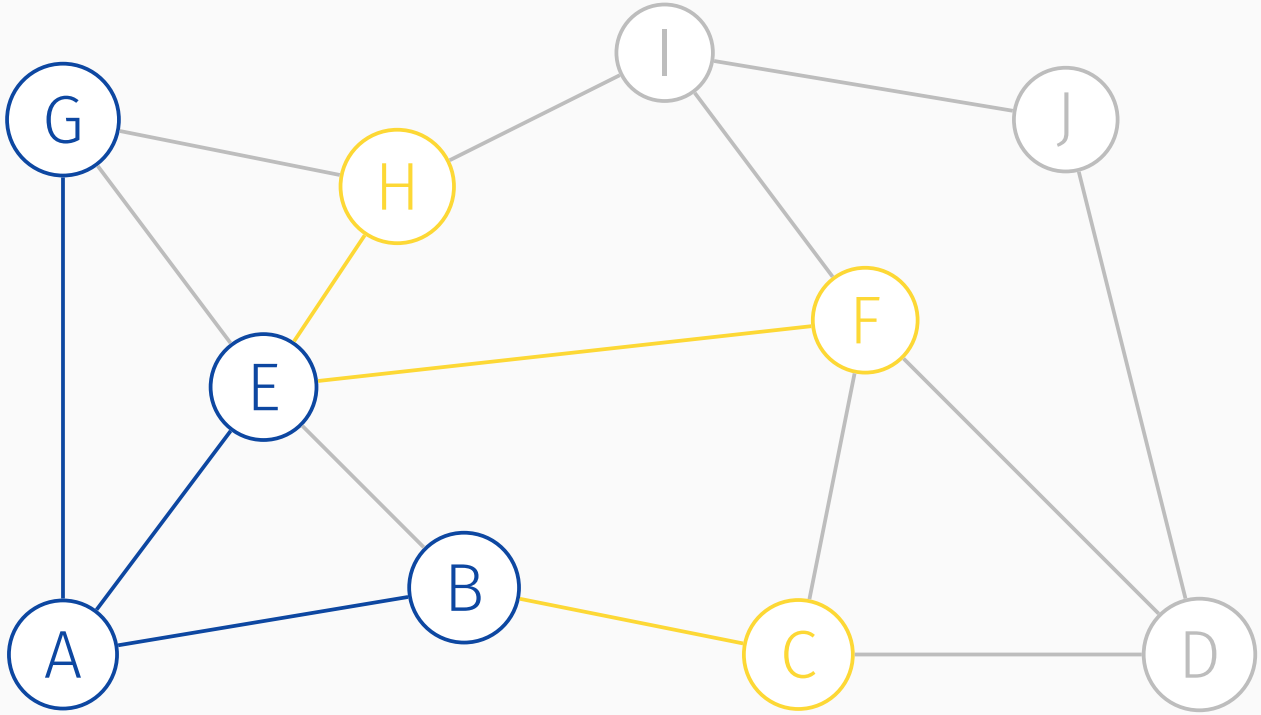
Breadth-first graph traversal, step 15



Q | C | F | H |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	2	E
G	1	A
H	2	E
I	∞	/
J	∞	/

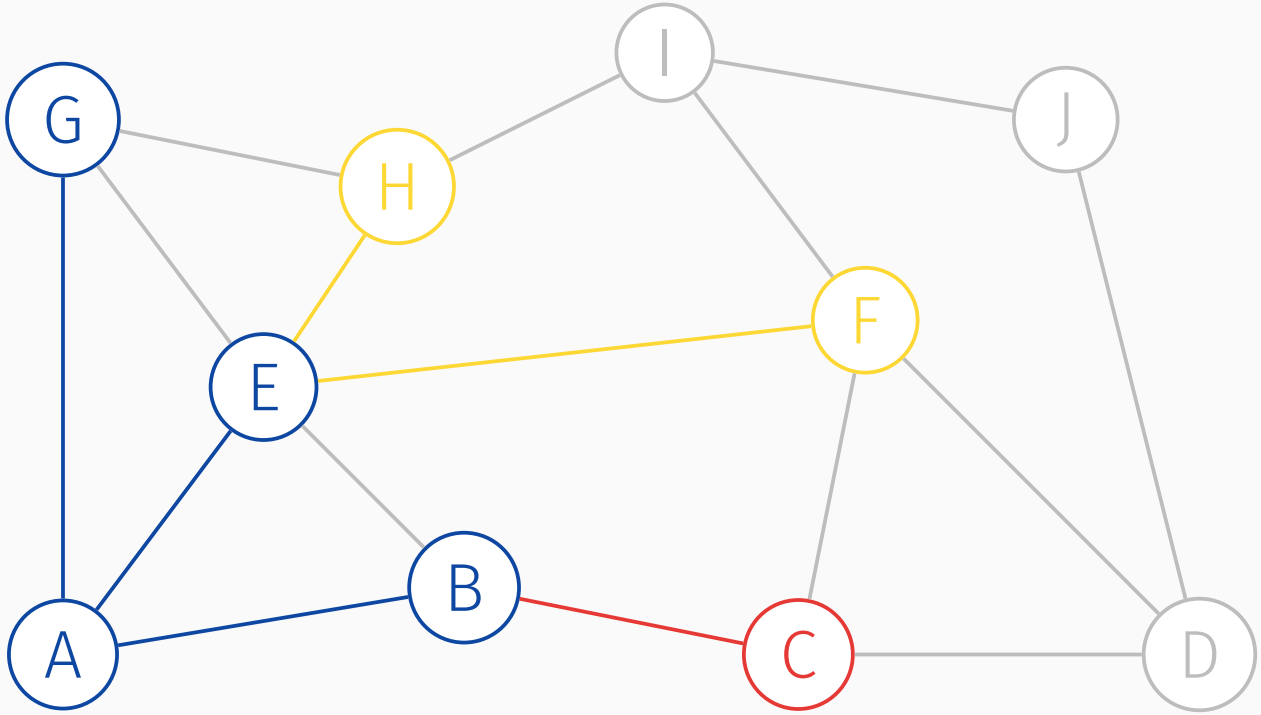
Breadth-first graph traversal, step 16



Q | C | F | H |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	2	E
G	1	A
H	2	E
I	∞	/
J	∞	/

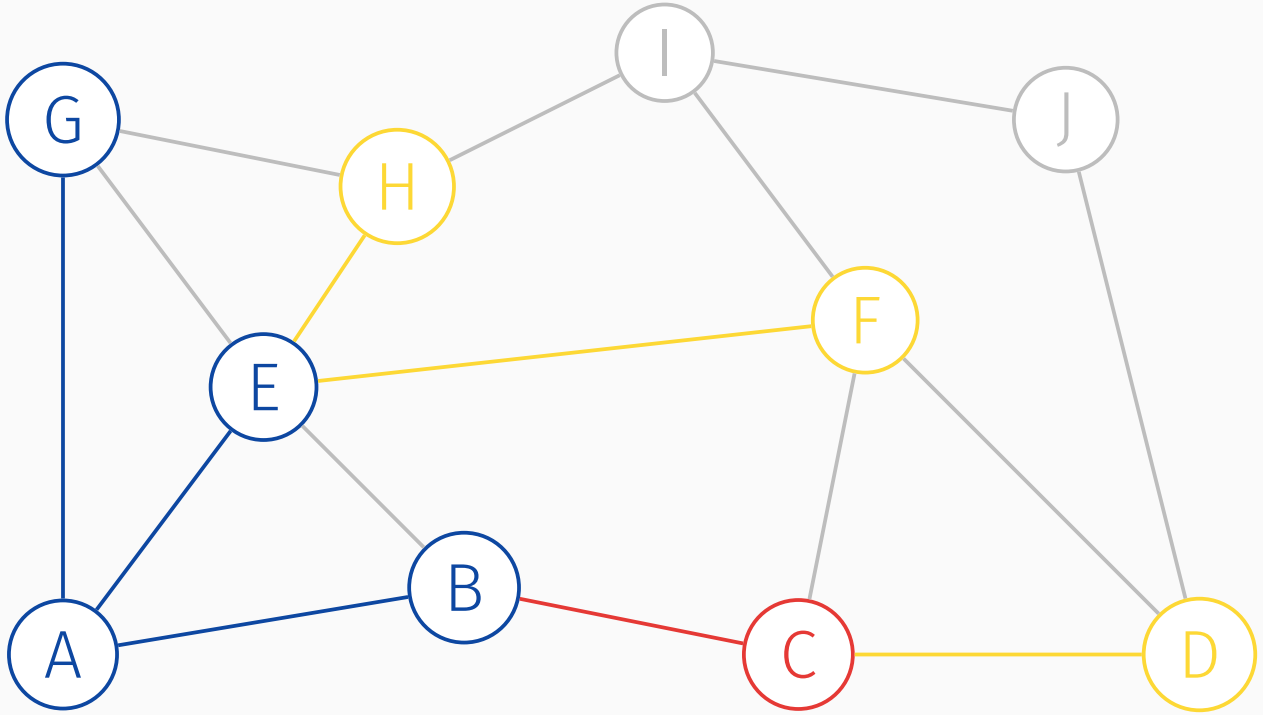
Breadth-first graph traversal, step 17



Q | F | H |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	2	E
G	1	A
H	2	E
I	∞	/
J	∞	/

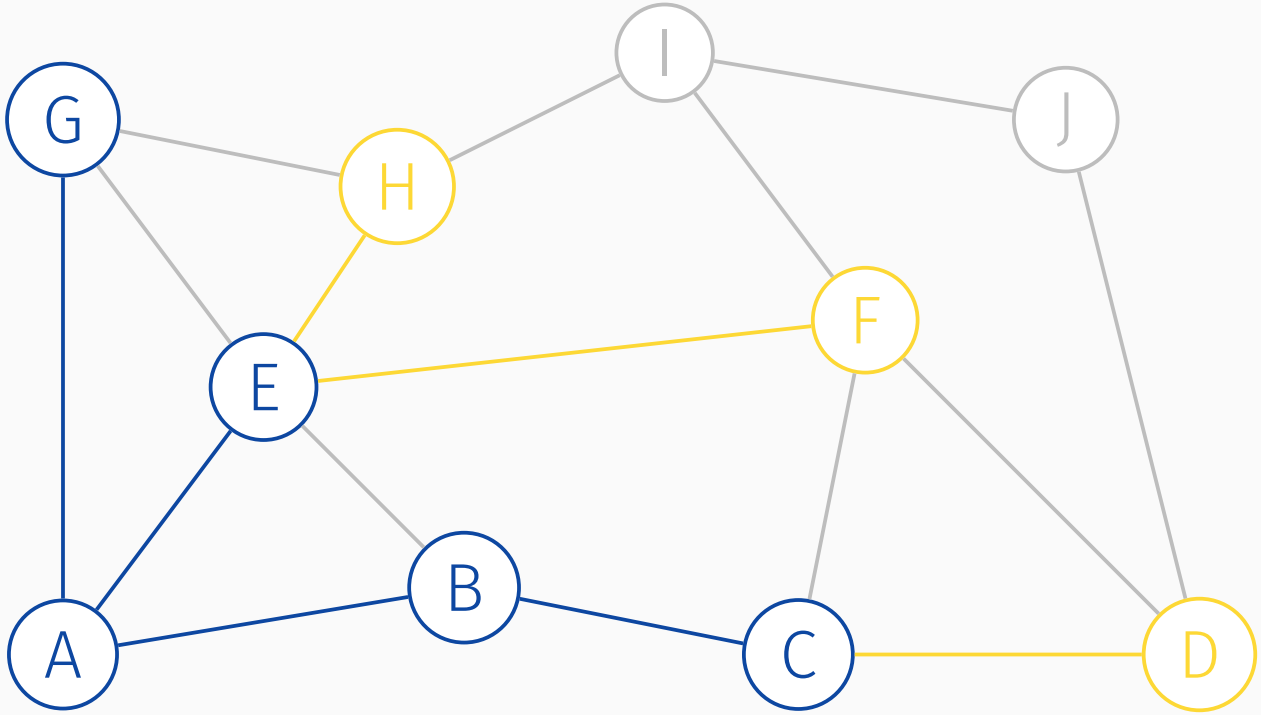
Breadth-first graph traversal, step 18



Q | F | H | D |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	∞	/
J	∞	/

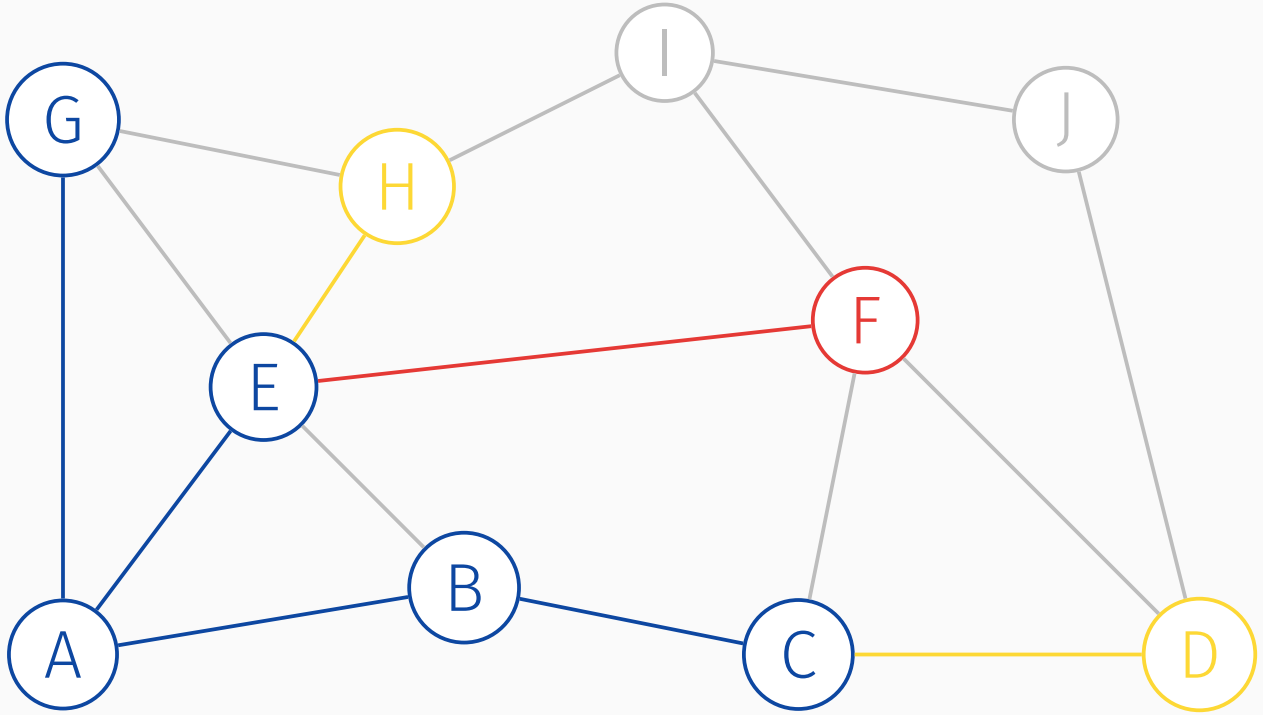
Breadth-first graph traversal, step 19



Q | F | H | D |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	∞	/
J	∞	/

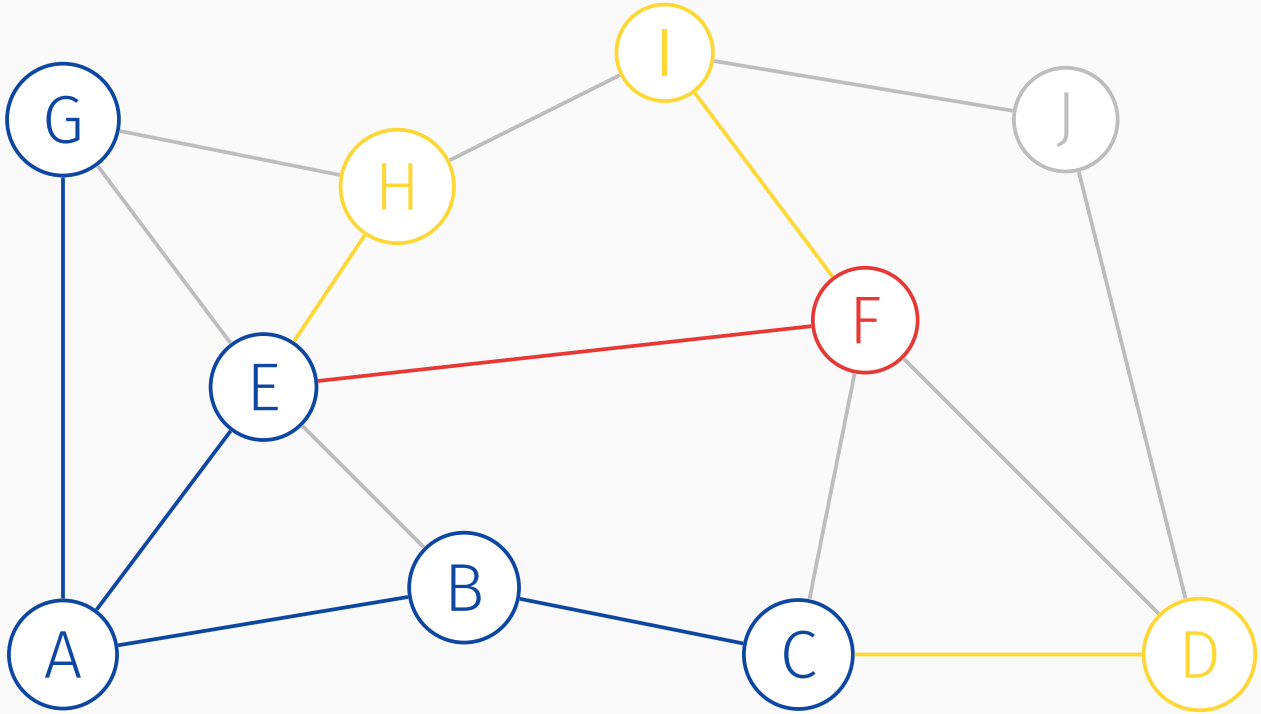
Breadth-first graph traversal, step 20



Q | H | D |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	∞	/
J	∞	/

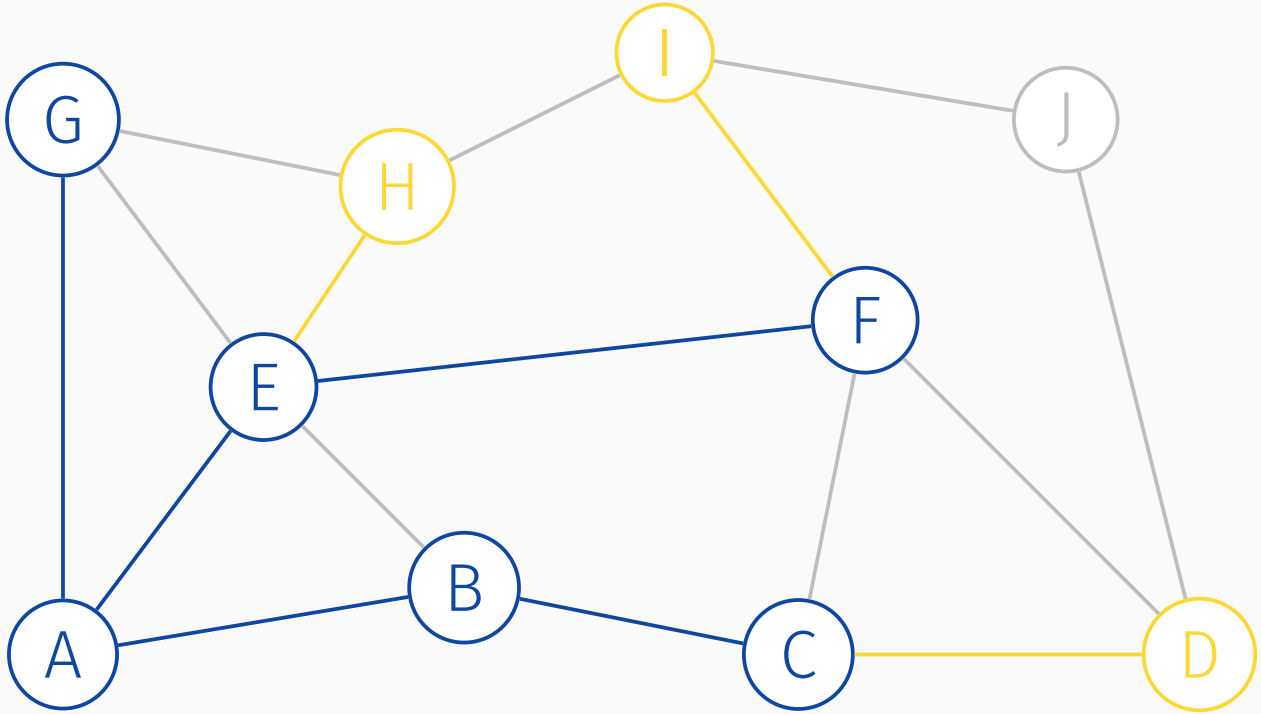
Breadth-first graph traversal, step 21



Q | H | D | I |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	∞	/

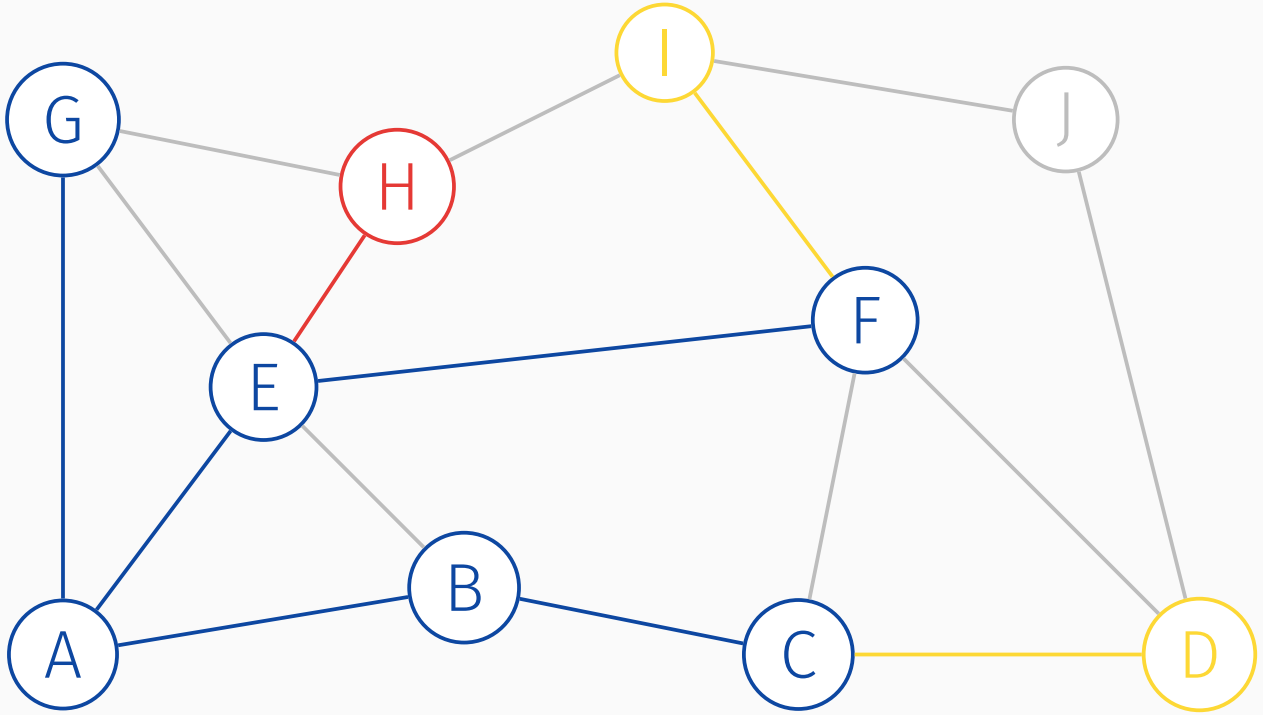
Breadth-first graph traversal, step 22



Q | H | D | I |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	∞	/

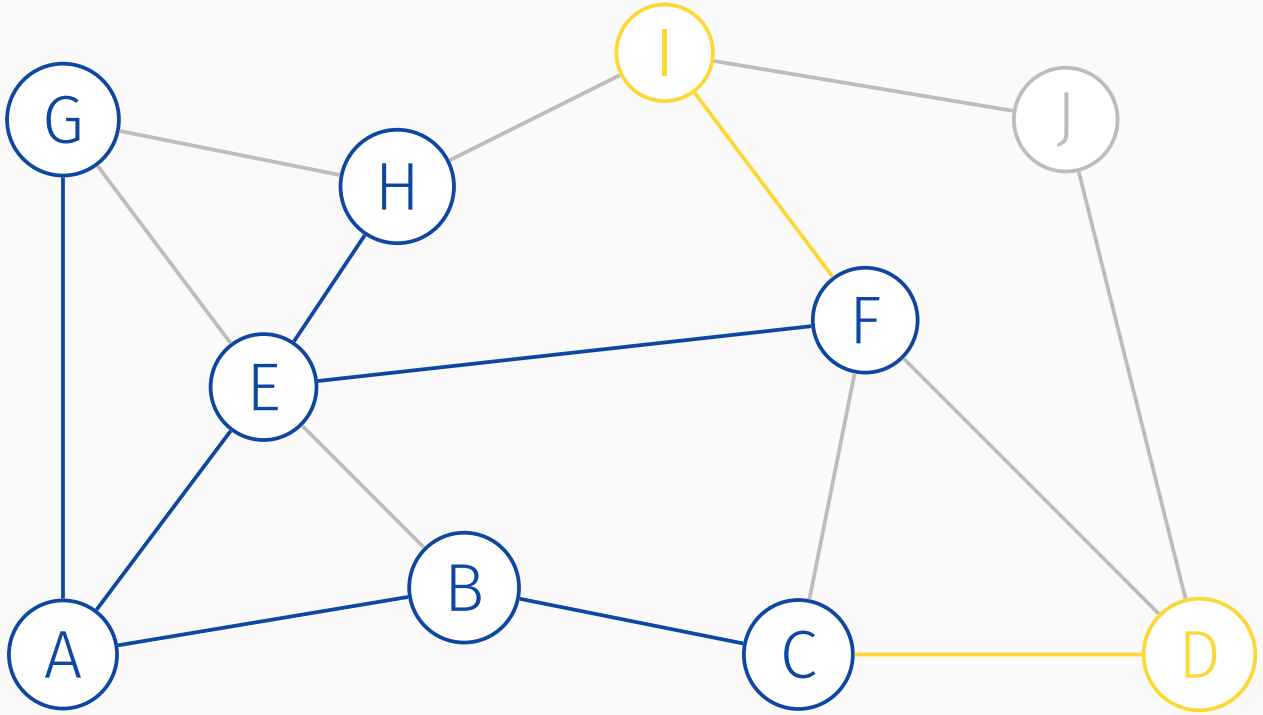
Breadth-first graph traversal, step 23



Q | D | I |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	∞	/

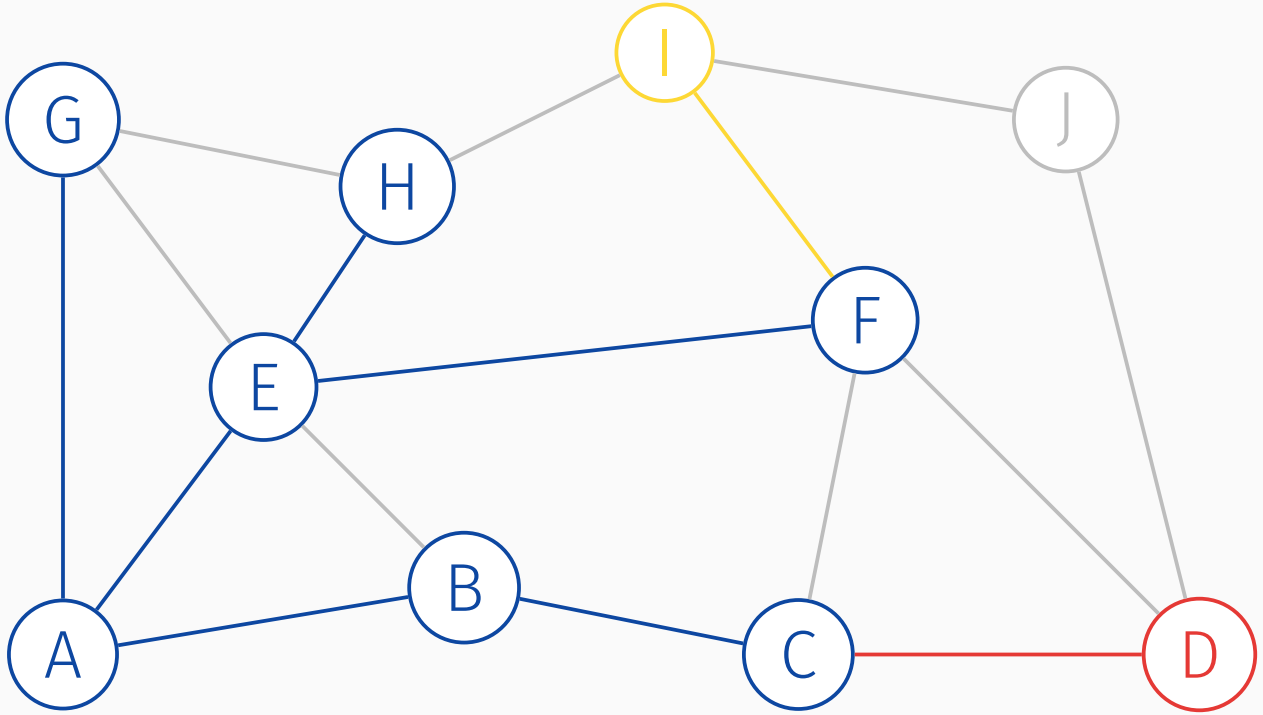
Breadth-first graph traversal, step 24



Q | D | I |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	∞	/

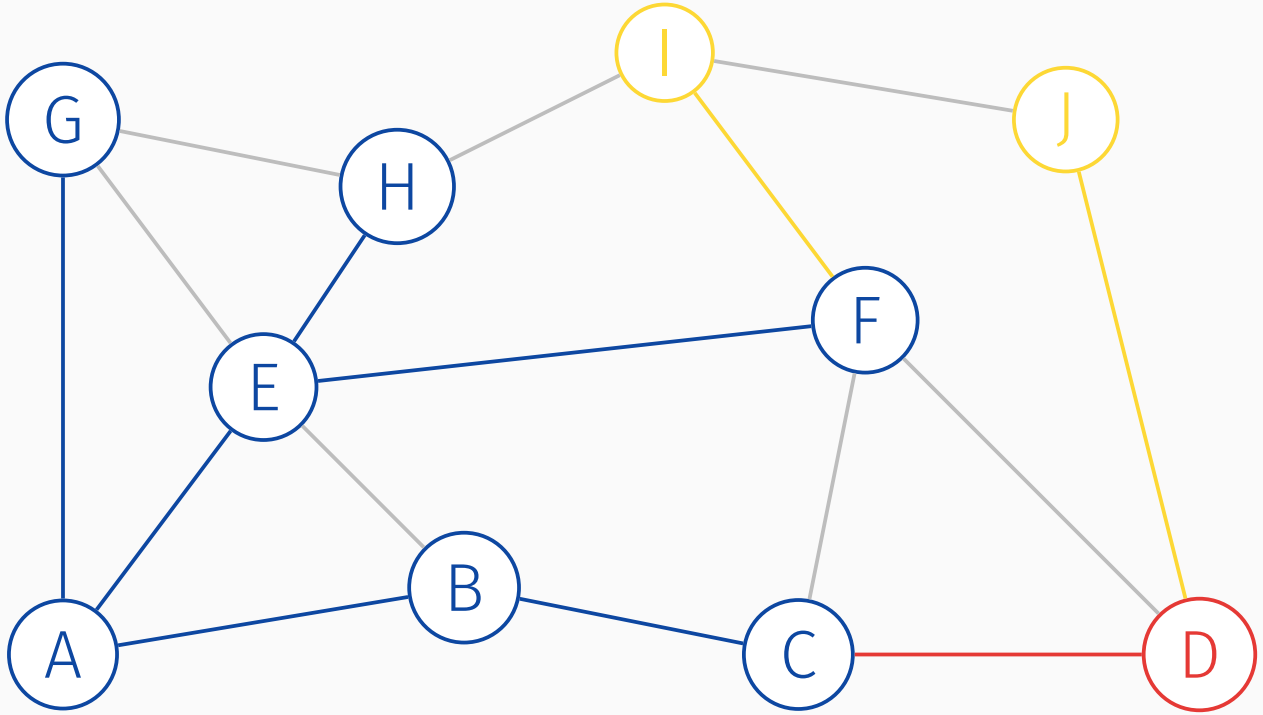
Breadth-first graph traversal, step 25



Q | I |

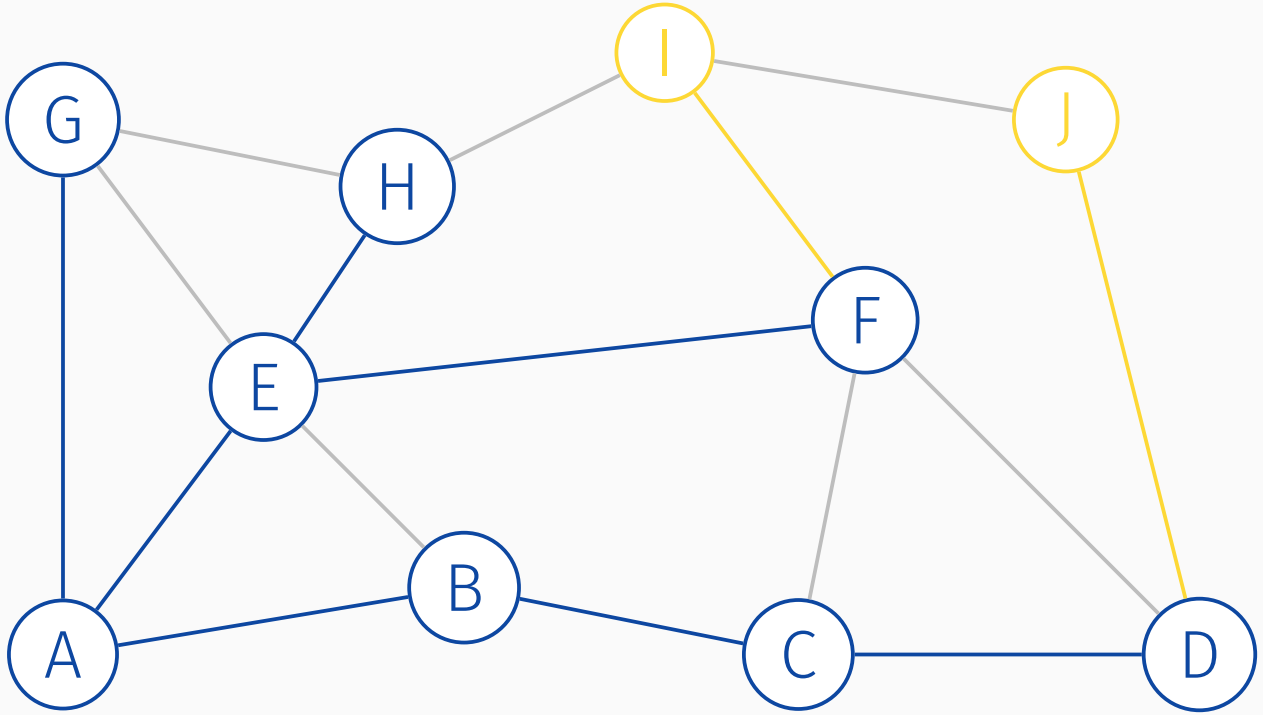
v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	∞	/

Breadth-first graph traversal, step 26



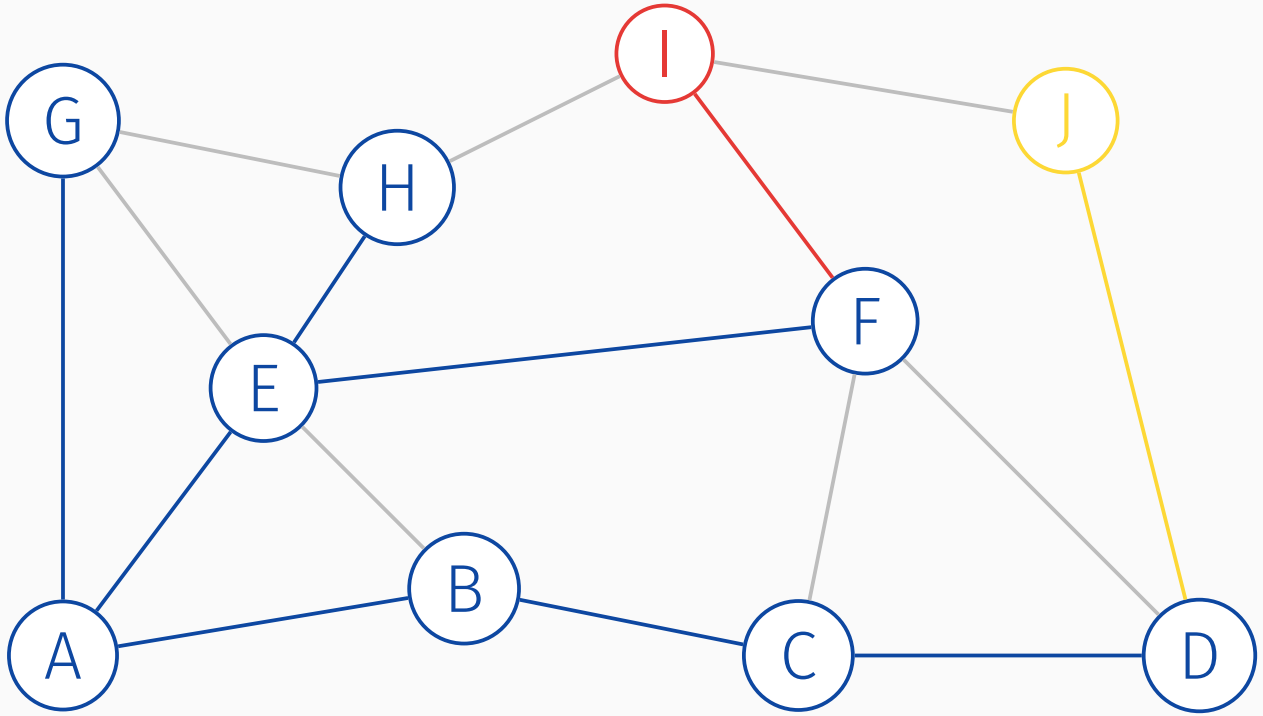
v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	4	D

Breadth-first graph traversal, step 27



v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	4	D

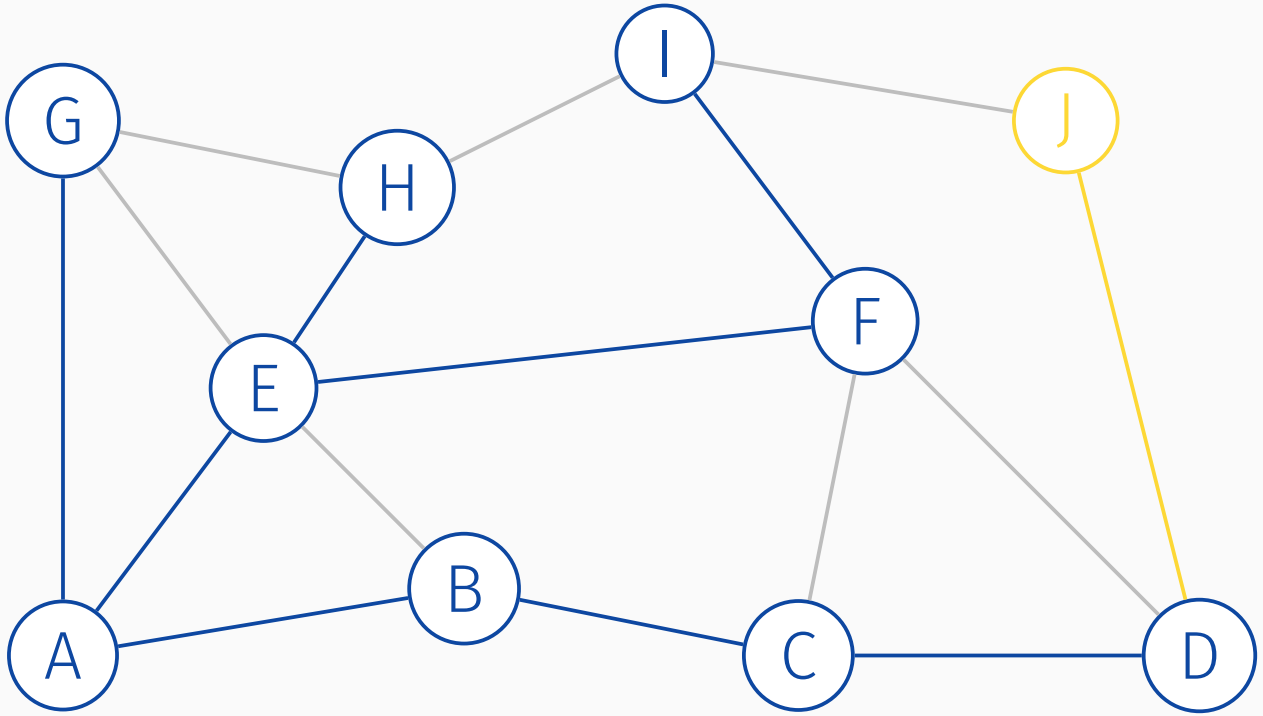
Breadth-first graph traversal, step 28



Q | J |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	4	D

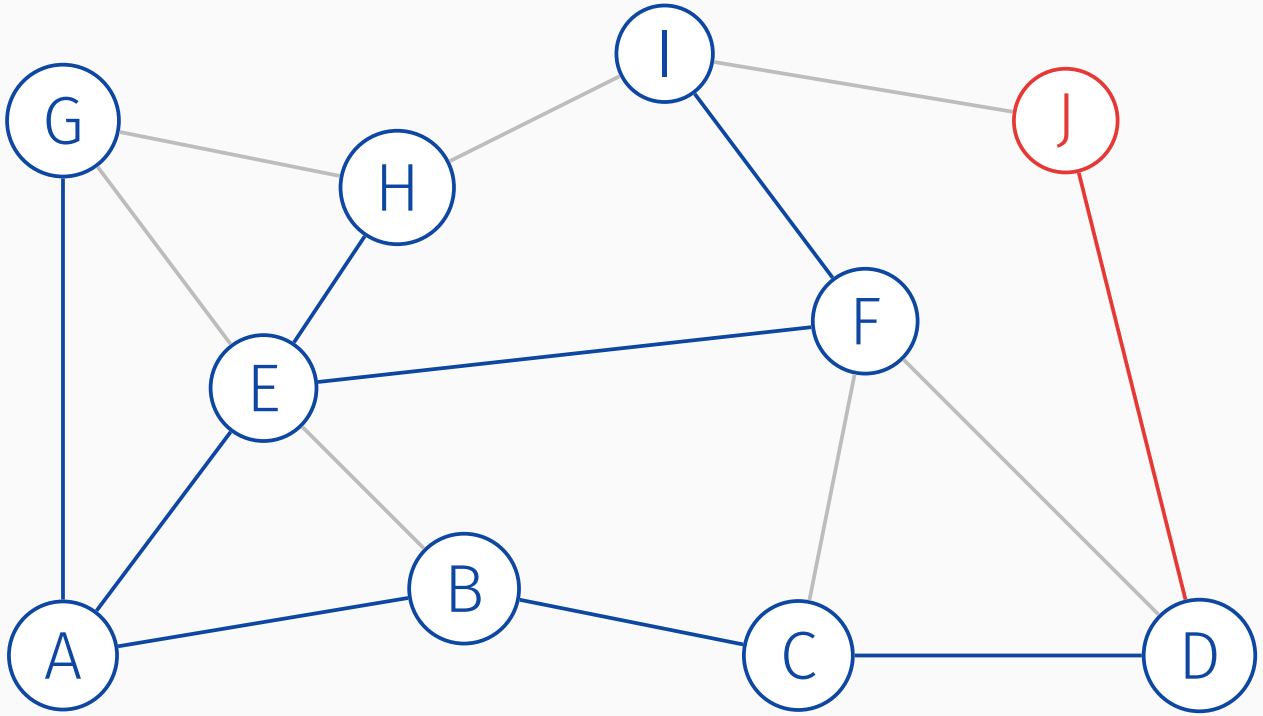
Breadth-first graph traversal, step 29



Q | J |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	4	D

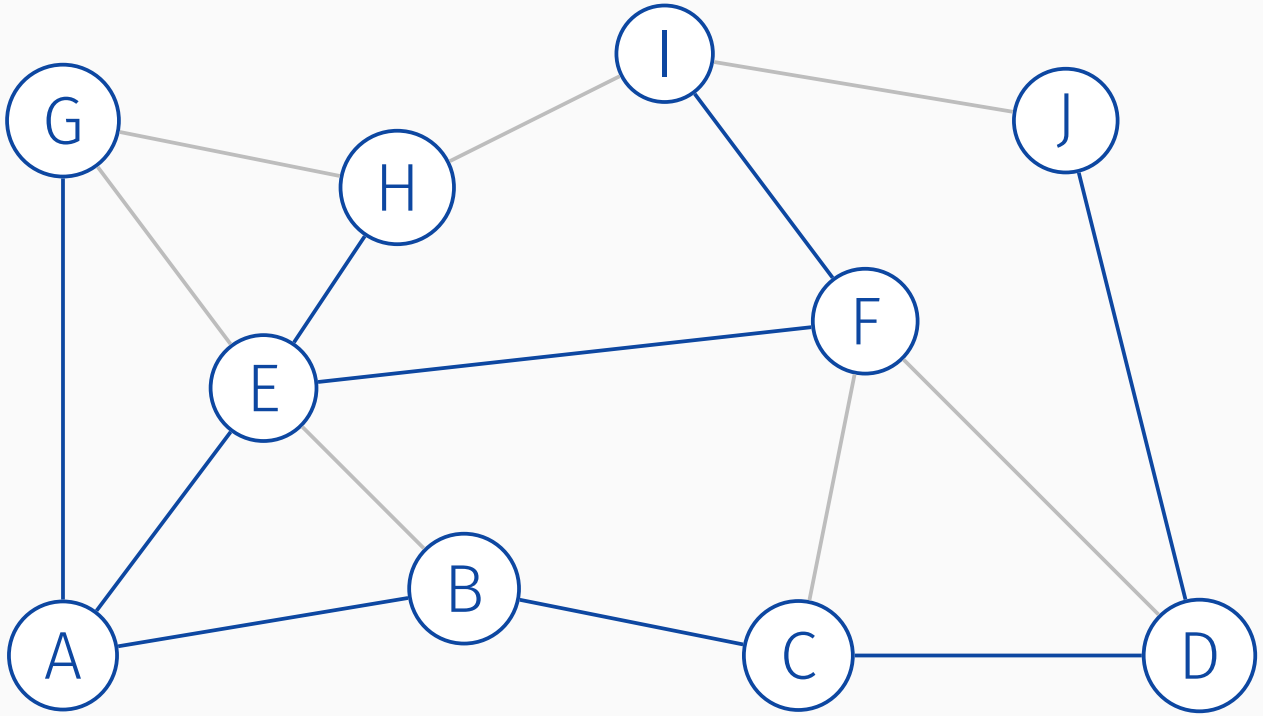
Breadth-first graph traversal, step 30



Q |

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	4	D

Breadth-first graph traversal, step 31



Q

v	$d[v]$	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	4	D

Depth-first and breadth-first graph traversal

Animation

For both graph traversal algorithms, two animations are available:

- path search in a maze and
- application in computer graphics – area filling.

The animations are available as separate file animations.

Sources for independent study

- Brute force problem solving strategies
  - book [1], chapter 3, pages 97 – 98
- Selection sort
  - book [1], chapter 3.1, pages 98 – 100
- Bubble sort
  - book [1], chapter 3.1, pages 100 – 101
- Shaker sort
  - book [2], chapter 4, pages 78 – 79
- Sequential search
  - book [1], chapter 3.2, pages 104 – 104
- Brute force string matching
  - book [1], chapter 3.2, pages 105 – 106

Sources for independent study (cont.)

- Closest pair problem
  - book [1], chapter 3.3, pages 108 – 109
- Convex hull problem
  - book [1], chapter 3.3, pages 109 – 113
- Exhaustive search
  - book [1], chapter 3.4, page 115
- Traveling salesman problem
  - book [1], chapter 3.4, page 116
- Knapsack problem
  - book [1], chapter 3.4, pages 116 – 117
- Depth-first graph traversal
  - book [1], chapter 3.5, pages 122 – 125
- Breadth-first graph traversal
  - book [1], chapter 3.5, pages 125 – 128

Thanks for your attention