COMPSCI 220

Lectures 33-34: Course Review (additional slides only)

Algorithm analysis   Data sorting   Data searching

(Di)graphs   Graph algorithms

Lecturer: Georgy Gimel'farb

Contents of the Review Lectures

- Running time: Examples 1.5, 1.6, 1.2.1 from Textbook.
- Solving recurrences: Examples 1.29 – 1.32 from Textbook.
- Sorting: inversions; insertion, merge-, quick-, heap sort; heaps.
- Searching: BST, self-balanced search trees.
- Digraphs: representations; sub(di)graphs, classes of traversal arcs.
- DFS / BFS / PFS: examples; determining ancestors of a tree.
- Cycle detection; girth; topological sorting – examples.
- Graph connectivity; strong connected components.
- Maximum matchings; augmented paths – examples.
- Weighed (di)graphs: representations; diameter; radius; excentricity.
- SSSP: Dijkstra's and Bellman-Ford examples.
- APSP: Floyd's examples.
- MST: Prim's and Kruskal's examples.

Running Time of a Pseudocode Fragment

The running time for this fragment is $\Theta(f(n))$. What is $f(n)$?

$j \leftarrow 1$
for $i \leftarrow 1$ **step** $i \leftarrow i+1$ **while** $i \leq n^2$ **do**
  if $i = j$ **then**
    $j \leftarrow j \cdot n$
    for $k \leftarrow 1$ **step** $k \leftarrow k+1$ **while** $k \leq n$ **do**
      $//$ ...constant number C of elementary operations
    end for
  else
    for $k \leftarrow 1$ **step** $k \leftarrow k+n$ **while** $k \leq n^3+1$ **do**
      $//$ ...constant number C of elementary operations
    end for
  end if
end for

A. n^4; B. $n^3 \log n$; C. n^3; D. $n^2 \log n$; E. n^2

Running Time of a Pseudocode Fragment

The running time for this fragment is $\Theta(f(n))$. What is $f(n)$?

```
j ← 1
for i ← 1 step i ← i + 1 while i ≤ n^2 do        ← n^2 steps
    if i = j then
        j ← j · n
        for k ← 1 step k ← k + 1 while k ≤ n do        ← n steps
            // ...constant number C of elementary operations
        end for
    else
        for k ← 1 step k ← k + n while k ≤ n^3 + 1 do        ← n^2 steps
            // ...constant number C of elementary operations
        end for
    end if
end for
```

A. n^4; B. $n^3 \log n$; C. n^3; D. $n^2 \log n$; E. n^2

Running Time of a Pseudocode Fragment

The running time for this fragment is $\Theta(f(n))$. What is $f(n)$?

```
j ← 1
for i ← 1 step i ← i + 1 while i ≤ n^2 do          ← n^2 steps
    if i = j then        }  ← i = j only when j = 1, then n, then n^2
        j ← j · n        }
        for k ← 1 step k ← k + 1 while k ≤ n do    ← n steps
            // ...constant number C of elementary operations
        end for
    else
        for k ← 1 step k ← k + n while k ≤ n^3 + 1 do   ← n^2 steps
            // ...constant number C of elementary operations
        end for
    end if
end for
```

A. n^4; B. $n^3 \log n$; C. n^3; D. $n^2 \log n$; E. n^2

Running Time of a Pseudocode Fragment

The running time for this fragment is $\Theta(f(n))$. What is $f(n)$?

$j \leftarrow 1$
for $i \leftarrow 1$ **step** $i \leftarrow i + 1$ **while** $i \leq n^2$ **do** ← n^2 steps
  if $i = j$ **then**
    $j \leftarrow j \cdot n$ } ← $i = j$ only when $j = 1$, then n, then n^2
    for $k \leftarrow 1$ **step** $k \leftarrow k + 1$ **while** $k \leq n$ **do** ← n steps
      // ...constant number C of elementary operations
    end for
  else
    for $k \leftarrow 1$ **step** $k \leftarrow k + n$ **while** $k \leq n^3 + 1$ **do** ← n^2 steps
      // ...constant number C of elementary operations
    end for
  end if
end for

1. For $i = 1, n, n^2 \rightarrow Cn$ (the inner upper **for**-loop).
2. $n^2 - 3$ steps of $i \rightarrow Cn^2$ (the inner bottom **for**-loop).
3. $3Cn + (n^2 - 3) \cdot Cn^2 = C(3n - 3n^2 + n^4) \rightarrow f(n) = n^4$

A. n^4; B. $n^3 \log n$; C. n^3; D. $n^2 \log n$; E. n^2

Running Time of a Pseudocode Fragment

The running time for this fragment is $\Theta(f(n))$. What is $f(n)$?

```
j ← 1
for i ← 1 step i ← i + 1 while i ≤ n^2 do        ← n^2 steps
    if i = j then      }
        j ← j · n      }  ← i = j only when j = 1, then n, then n^2
        for k ← 1 step k ← k + 1 while k ≤ n do        ← n steps
            // ...constant number C of elementary operations
        end for
    else
        for k ← 1 step k ← k + n while k ≤ n^3 + 1 do        ← n^2 steps
            // ...constant number C of elementary operations
        end for
    end if
end for
```

1. For $i = 1, n, n^2 \rightarrow Cn$ (the inner upper **for**-loop).
2. $n^2 - 3$ steps of $i \rightarrow Cn^2$ (the inner bottom **for**-loop).
3. $3Cn + (n^2 - 3) \cdot Cn^2 = C(3n - 3n^2 + n^4) \rightarrow f(n) = n^4$

A. n^4; B. $n^3 \log n$; C. n^3; D. $n^2 \log n$; E. n^2

Big-Oh / Omega / Theta Definitions

- Let $f(n)$ and $g(n)$ be non-negative-valued functions, defined on non-negative integers, n.
- Let c and n_0 be a positive real constant and a positive integer, respectively.

If and only if there exist c and n_0 such that

$g(n) \leq cf(n)$ for all $n > n_0$ **then** $g(n)$ is $\mathrm{O}(f(n))$ ($g(n)$ is Big Oh of $f(n)$)
$g(n) \geq cf(n)$ for all $n > n_0$ **then** $g(n)$ is $\Omega(f(n))$ ($g(n)$ is Big Omega of $f(n)$

- Let c_1, c_2, and n_0 be two positive real constants and a positive integer, respectively.

If and only if there exist c_1, c_2 and n_0 such that

$c_1 f(n) \leq g(n) \leq c_2 f(n)$ for all $n > n_0$ **then** $g(n)$ is $\Theta(f(n))$

($g(n)$ is Big Theta of $f(n)$).

Big-Oh / Omega / Theta Properties

- **Scaling** (for $\mathrm{X} = \mathrm{O}, \Omega, \Theta$):

  $cf(n)$ is $\mathrm{X}\big(f(n)\big)$ for all constant factors $c > 0$.

- **Transitivity** (for $\mathrm{X} = \mathrm{O}, \Omega, \Theta$):

  If h is $\mathrm{X}(g)$ and g is $\mathrm{X}(f)$, then h is $\mathrm{X}(f)$.

- **Rule of sums** (for $\mathrm{X} = \mathrm{O}, \Omega, \Theta$):

  If $g_1 \in \mathrm{X}(f_1)$ and $g_2 \in \mathrm{X}(f_2)$, then $g_1 + g_2 \in \mathrm{X}(\max\{f_1, f_2\})$.

- **Rule of products** (for $\mathrm{X} = \mathrm{O}, \Omega, \Theta$):

  If $g_1 \in \mathrm{X}(f_1)$ and $g_2 \in \mathrm{X}(f_2)$, then $g_1 g_2 \in \mathrm{X}(f_1 f_2)$.

- **Limit rule:**

  Suppose the ratio's limit $\lim_{n \to \infty} \frac{f(n)}{g(n)} = L$ exists (may be infinite, ∞).

$$\text{Then} \begin{cases} \text{if} \quad L = 0 & \text{then} \quad f \in \mathrm{O}(g) \\ \text{if} \quad 0 < L < \infty & \text{then} \quad f \in \Theta(g) \\ \text{if} \quad L = \infty & \text{then} \quad f \in \Omega(g) \end{cases}$$

Solving a Recurrence

If the solution of the recurrence $T(n) = T(n-1) + \log_2 n$; $T(1) = 0$, is in $\Theta(f(n))$, what is $f(n)$?

Hint: The factorial $n! \approx n^n \mathrm{e}^{-n}\sqrt{2\pi n}$ where $\mathrm{e} = 2.718\ldots$ and $\pi = 3.1415\ldots$ are constants.

A. 2^n; B. $\log n$; C. n; D. $n \log n$; E. n^2

Telescoping:

$$\left.\begin{array}{lcl} T(n) & = & T(n-1) + \log_2 n \\ T(n-1) & = & T(n-2) + \log_2 (n-1) \\ \ldots & \ldots\ldots & \\ T(3) & = & T(2) + \log_2 3 \\ T(2) & = & T(1) + \log_2 2 \end{array}\right\} \rightarrow \left\{\begin{array}{lclcl} T(n) & - & T(n-1) & = & \log_2 n \\ T(n-1) & - & T(n-2) & = & \log_2 (n-1) \\ \ldots & \ldots\ldots & & \ldots\ldots & \\ T(3) & - & T(2) & = & \log_2 3 \\ T(2) & - & T(1) & = & \log_2 2 \end{array}\right.$$

Solving a Recurrence

If the solution of the recurrence $T(n) = T(n-1) + \log_2 n$; $T(1) = 0$, is in $\Theta(f(n))$, what is $f(n)$?

Hint: The factorial $n! \approx n^n e^{-n}\sqrt{2\pi n}$ where $e = 2.718\ldots$ and $\pi = 3.1415\ldots$ are constants.

A. 2^n; B. $\log n$; C. n; D. $n \log n$; E. n^2

Telescoping:

$$\left.\begin{array}{lll} T(n) & = & T(n-1) + \log_2 n \\ T(n-1) & = & T(n-2) + \log_2(n-1) \\ \ldots & \ldots\ldots & \\ T(3) & = & T(2) + \log_2 3 \\ T(2) & = & T(1) + \log_2 2 \end{array}\right\} \rightarrow \left\{\begin{array}{lllll} T(n) & - & \cancel{T(n-1)} & = & \log_2 n \\ \cancel{T(n-1)} & - & \cancel{T(n-2)} & = & \log_2(n-1) \\ \ldots & & \ldots\ldots & & \ldots\ldots \\ \cancel{T(3)} & - & \cancel{T(2)} & = & \log_2 3 \\ \cancel{T(2)} & - & T(1) & = & \log_2 2 \end{array}\right.$$

Solving a Recurrence

If the solution of the recurrence $T(n) = T(n-1) + \log_2 n$; $T(1) = 0$, is in $\Theta(f(n))$, what is $f(n)$?

Hint: The factorial $n! \approx n^n e^{-n}\sqrt{2\pi n}$ where $e = 2.718\ldots$ and $\pi = 3.1415\ldots$ are constants.

A. 2^n; B. $\log n$; C. n; D. $n \log n$; E. n^2

Telescoping:

$$\left.\begin{array}{lcl} T(n) & = & T(n-1) + \log_2 n \\ T(n-1) & = & T(n-2) + \log_2(n-1) \\ \ldots & \ldots\ldots & \\ T(3) & = & T(2) + \log_2 3 \\ T(2) & = & T(1) + \log_2 2 \end{array}\right\} \rightarrow \left\{\begin{array}{lclcl} T(n) & - & \cancel{T(n-1)} & = & \log_2 n \\ \cancel{T(n-1)} & - & \cancel{T(n-2)} & = & \log_2(n-1) \\ \ldots & & \ldots\ldots & & \ldots\ldots \\ \cancel{T(3)} & - & \cancel{T(2)} & = & \log_2 3 \\ \cancel{T(2)} & - & T(1) & = & \log_2 2 \end{array}\right.$$

Summing left and right columns: $T(n) - T(1) = \log_2 n + \ldots + \log_2 2$

Solving a Recurrence

If the solution of the recurrence $T(n) = T(n-1) + \log_2 n$; $T(1) = 0$, is in $\Theta(f(n))$, what is $f(n)$?

Hint: The factorial $n! \approx n^n e^{-n}\sqrt{2\pi n}$ where $e = 2.718\ldots$ and $\pi = 3.1415\ldots$ are constants.

A. 2^n; B. $\log n$; C. n; D. $n \log n$; E. n^2

Telescoping:

$$\left.\begin{array}{lcl} T(n) & = & T(n-1) + \log_2 n \\ T(n-1) & = & T(n-2) + \log_2(n-1) \\ \ldots & & \ldots\ldots \\ T(3) & = & T(2) + \log_2 3 \\ T(2) & = & T(1) + \log_2 2 \end{array}\right\} \rightarrow \left\{\begin{array}{lclcl} T(n) & - & \cancel{T(n-1)} & = & \log_2 n \\ \cancel{T(n-1)} & - & \cancel{T(n-2)} & = & \log_2(n-1) \\ \ldots & & \ldots\ldots & & \ldots\ldots \\ \cancel{T(3)} & - & \cancel{T(2)} & = & \log_2 3 \\ \cancel{T(2)} & - & T(1) & = & \log_2 2 \end{array}\right.$$

Summing left and right columns: $T(n) - T(1) = \log_2 n + \ldots + \log_2 2$

$$\begin{array}{lcl} T(n) & = & 0 + \log_2 2 + \log_2 3 + \ldots + \log_2(n-1) + \log_2 n = \log_2(n!) \\ & = & n \log_2 n - n \log_2 e + \frac{1}{2}(\log_2 n + \log_2 \pi + 1), \text{ i.e.,} \end{array}$$

$$T(n) \in \Theta(n \log n)$$

Data Structures and Algorithms

Static ADT: 1D and multidimensional arrays.

Dynamic ADT:

Linked lists	Stacks, queues	Priority queues, heaps
Tables (associative lists,dictionaries)		Hash tables
Trees	Binary search trees (BST): AVL, red-black, AA	
	Multiway search trees: B-trees	
Digraphs / graphs		Disjoint sets

Algorithms:

- Sort/select: insertion-, merge-, quick-, heap sort; quickselect
- Search: sequential, binary (dynamic – binary search tree)
- Hash function: division, folding, truncation, middle-squaring
- Hashing: separate chaining (SC), open addressing (OALP, OADH)
- Graph: DFS/BFS/PFS, connected components, MST (Kruskal, Prim), matching, SSSP (Dijkstra, Bellman-Ford), APSP (Floyd)

Sorting Algorithms

Algorithm	Complexity for n items		Comments
	Worst case	Average case	
Data sorting – comparison-based algorithms			
Insertion sort	$O(n^2)$	$O(n^2)$	Selection, Bubble sort
Mergesort	$O(n \log n)$	$O(n \log n)$	Extra space $O(n)$
Quicksort	$O(n^2)$	$O(n \log n)$	Randomised pivots: the worst case $O(n \log n)$
Heapsort	$O(n \log n)$	$O(n \log n)$	Priority queue (heap)
Data sorting – non-comparison-based algorithms			
Counting sort	$O(n)$	$O(n)$	Constrained range of integer search keys
Data selection – comparison-based algorithms			
Quickselect	$O(n^2)$	$O(n)$	Randomised pivots: the worst case $O(n)$

Search Algorithms

Algorithm	Complexity for n items		Comments
	Worst case	Average case	
Data search – comparison-based algorithms			
Seq search	$\mathrm{O}(n)$	$\mathrm{O}(n)$	Unsorted data list
Binary search	$\mathrm{O}(\log n)$	$\mathrm{O}(\log n)$	Sorted static list
BST	$\mathrm{O}(n)$	$\mathrm{O}(\log n)$	Balancing: $\mathrm{O}(\log n)$
B-trees	Tree height	Ave height	Opt height: $\approx \log_m n$

Algorithm	Time $T_{\dots}(\lambda)$ of search for m items		Comments
	Unsuccessful	Successful	
Data search – hash tables of size n with load factor $\lambda = \frac{m}{n}$			
SC	$1 + \lambda$	$1 + \frac{\lambda}{2}$	$\lambda \geq 1$
OALP	$\frac{1}{2}\left(1 + \left(\frac{1}{1-\lambda}\right)^2\right)$	$\frac{1}{2}\left(1 + \frac{1}{1-\lambda}\right)$	$\lambda \leq 0.75$
OADH	$\frac{1}{1-\lambda}$	$\frac{1}{\lambda}\ln\left(\frac{1}{1-\lambda}\right)$	$\lambda \leq 0.75$

Digraphs: Computer Representations

$$G = \Big(\quad V = \{0,1,2,3,4\}, \quad E = \big\{(0,2),(1,0),(1,2),(1,3),(3,1),(4,2),(3,4)\big\} \quad \Big)$$

Adjacency lists representing the set E of arcs:

$$\big\{\{2\},\{0,2,3\},\underbrace{\{.\}}_{\emptyset},\{1,4\},\{2\}\big\}$$ or

2		
0	2	3
1	4	
2		

Adjacency matrix representing the set E of arcs:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Sub(di)graphs

$$G = \big(\quad V = \{0,1,2,3,4\}, \\ E = \{(0,2),(1,0),(1,2),(1,3),(3,1),(4,2),(3,4)\} \quad \big)$$

Sub(di)graph $G' = (V', E')$; $V' \subseteq V$; if $(u,v) \in E' \subseteq E$, then $u, v \in V'$:

$$G' = \big(V' = \{1,2,3\}, E' = \{(1,2),(3,1)\}\big)$$
$$G' = \big(V' = \{0,1,2\}, E' = \{(1,2)\}\big)$$

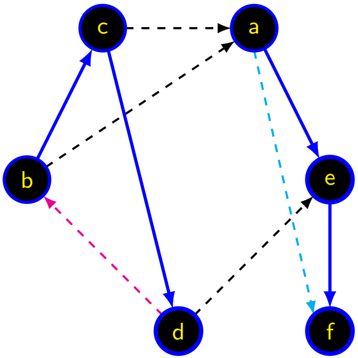
Induced sub(di)graph $G' = (V', E')$; $E' = \{(u,v) \in E : \; u, v \in V'\}$:

$$G' = \big(V' = \{1,2,3\}, E' = \{(1,2),(1,3),(3,1)\}\big)$$
$$G' = \big(V' = \{0,1,2\}, E' = \{(0,2),(1,0),(1,2)\}\big)$$

Spanning sub(di)graph: $G' = (V', E')$; $V' = V$; $E' \subseteq E$

$$G' = \big(V' = \{0,1,2,3,4\}, E' = \{(0,2),(1,2),(3,4)\}\big)$$
$$G' = \big(V' = \{0,1,2,3,4\}, E' = \{(1,0),(1,2),(1,3),(3,4)\}\big)$$

Classes of Traversal Arcs



v	a	b	c	d	e	f
$seen[v]$	0	6	7	8	1	2
$done[v]$	5	11	10	9	4	3

Search forest F: a set of disjoint trees spanning a digraph G after its traversal.

An arc $(u, v) \in E(G)$, i.e., (c, d), is a **tree arc** if it belongs to one of the trees of F:
$seen(\mathsf{c}) < seen(\mathsf{d}) < done(\mathsf{d}) < done(\mathsf{c})$

The arc (u, v), being not a tree arc, is

- **forward** if u is an ancestor of v in F:
  $seen(\mathsf{a}) < seen(\mathsf{f}) < done(\mathsf{f}) < done(\mathsf{a})$
- **back** if u is a descendant of v in F:
  $seen(\mathsf{b}) < seen(\mathsf{d}) < done(\mathsf{d}) < done(\mathsf{b})$, and
- **cross arc** if neither u nor v is an ancestor of the other in F:
  $seen(\mathsf{a}) < done(\mathsf{a}) < seen(\mathsf{b}) < done(\mathsf{b})$

DFS / BFS / PFS in Graph Algorithms

DFS / BFS complexity:

- $\Theta(n+m)$ – adjacency lists
- $\Theta(n^2)$ – an adjacency matrix

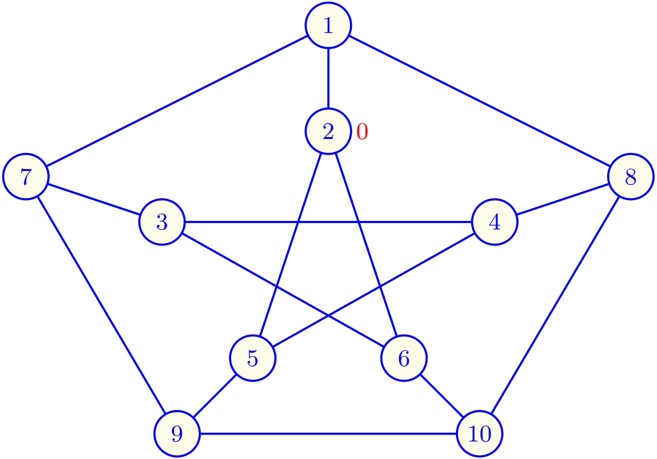
PFS compexity:

- $\Omega(n^2)$ – an array of keys
- $\Omega(n \log n)$ – a binary heap

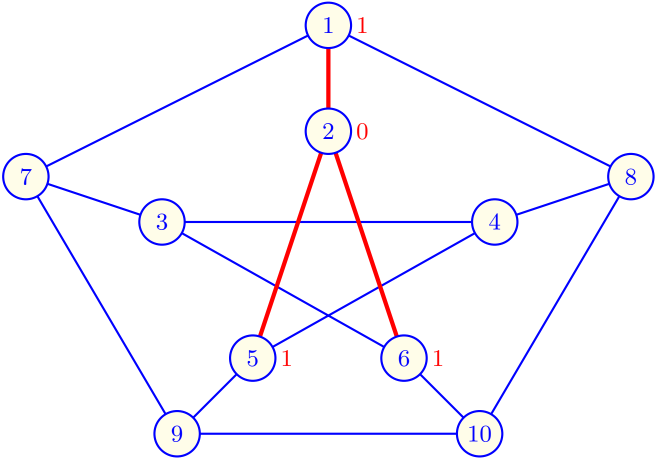
Graph algorithms:

- **Cycle detection**: by running the BFS.
- **Girth computation**: by running the BFS or DFS.
- **Topological ordering**: zero-indegree sorting or the DFS.
- **Strongly connected components**: two runs of the DFS.
- **Maximum matching**: $\mathrm{O}(n^2 m)$
  - Finding an augmenting path: $\mathrm{O}(m)$ with adjacency lists.
  - At most $O(n)$ augmenting paths to be found.
  - An augmenting path for each of $\mathrm{O}(n)$ non-matched vertices.
  - Repeating the process for each modified matching.

Girth Example: Petersen Graph

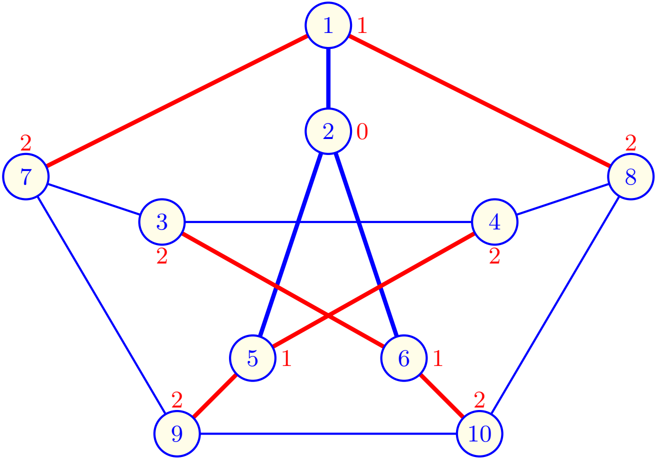


BFS starting at the vertex 2: $\left\{\begin{array}{c|cccccccccc} v \in V & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \hline d[v] & \cdot & 0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{array}\right.$



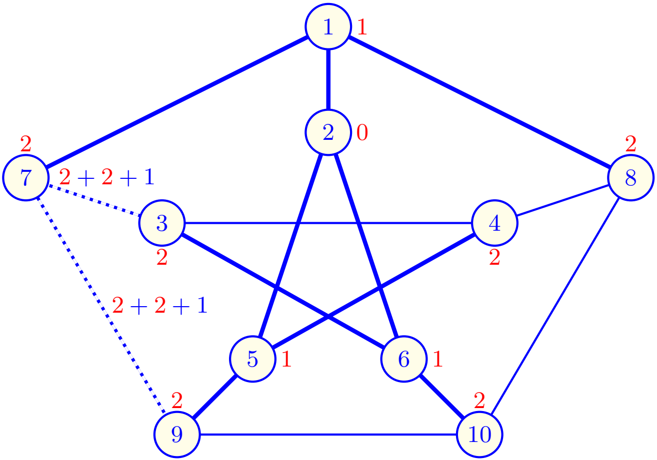
BFS starting at the vertex 2: $\left\{\begin{array}{c|cccccccccc} v \in V & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \hline d[v] & 1 & 0 & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot \end{array}\right\}$

Girth Example: Petersen Graph — BFS step 2

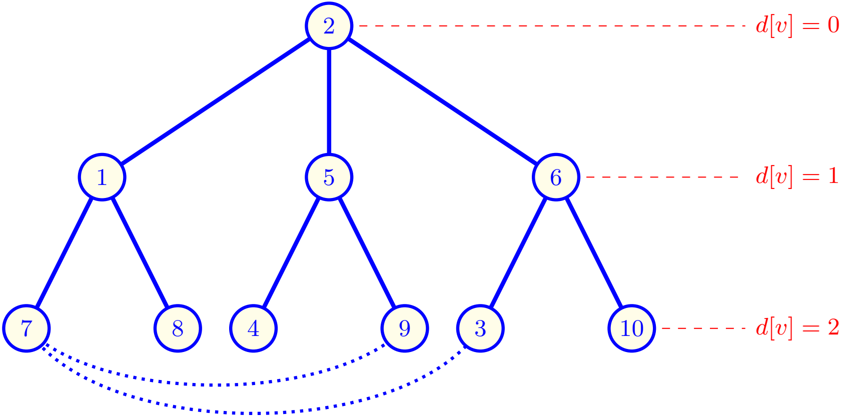


BFS starting at the vertex 2:

$v \in V$	1	2	3	4	5	6	7	8	9	10
$d[v]$	1	0	2	2	1	1	2	2	2	2



As is easily checked, the Petersen graph has girth of 5.



2
1
5
6
7
8
4
9
3
10
$d[v] = 0$
$d[v] = 1$
$d[v] = 2$

Graph Algorithms: Weighted (Di)graphs

Single-source shortest path (SSSP):

- **Dijkstra's algorithm**:
  - $\Theta(n^2)$ – scanning an array for the minimum distance.
  - $\mathrm{O}\big((n+m)\log n\big)$ – a priority queue (a binary heap).
  - $\mathrm{O}\big(m + n\log n\big)$ – with a Fibonacci heap.
- **Bellman–Ford algorithm**:
  - $\Theta(n^3)$ – an adjacency matrix.
  - $\Theta(n, m)$ – adjacency lists

All-pairs shortest paths (APSP):

- **Floyd's algorithm**: $\Theta(n^3)$.

Minimal spanning tree (MST):

- **Prim's algorithm**: $\mathrm{O}(m + n\log n)$ (like Dijkstra's).
- **Kruskal's algorithm**: $\mathrm{O}(m\log n)$.