SIT221: Data Structures and Algorithms

Lecture 12: Revisit

Week 11 recording for prac

	Recording Name	Session Name	Date	Duration
Week 11	Week 11 - prac - recording 1	Week 11 - prac	24/09/2017 1:19 pm	00:20:54
Week 10	Week 10 - prac - recording 1	Week 10 - prac	16/09/2017 4:19 pm	01:06:30
Week 9	Week 9 - prac - recording 1	Week 9 - prac	9/09/2017 4:23 pm	00:33:14
Week 8	Week 8 - prac - recording 1	Week 8 - prac	2/09/2017 4:08 pm	01:09:28
Week 7	Week 7 - prac - recording 1	Week 7 - prac	26/08/2017 4:11 pm	00:39:42
Week 4	SIT221 - Data Structures And Algorithms - recording 1	SIT221 - Data Structures And Algorithms	3/08/2017 11:22 am	00:42:34
Week 3	SIT221 - Data Structures And Algorithms - recording 1	SIT221 - Data Structures And Algorithms	27/07/2017 10:55 am	00:40:01
Week 2	Week 2 - prac - recording 1	Week 2 - prac	20/07/2017 11:01 am	00:31:17

Reminder

- Assignment 2 is due 11:59pm 1 Oct, 2017.
- No further extension will be given

What have we studied so far?

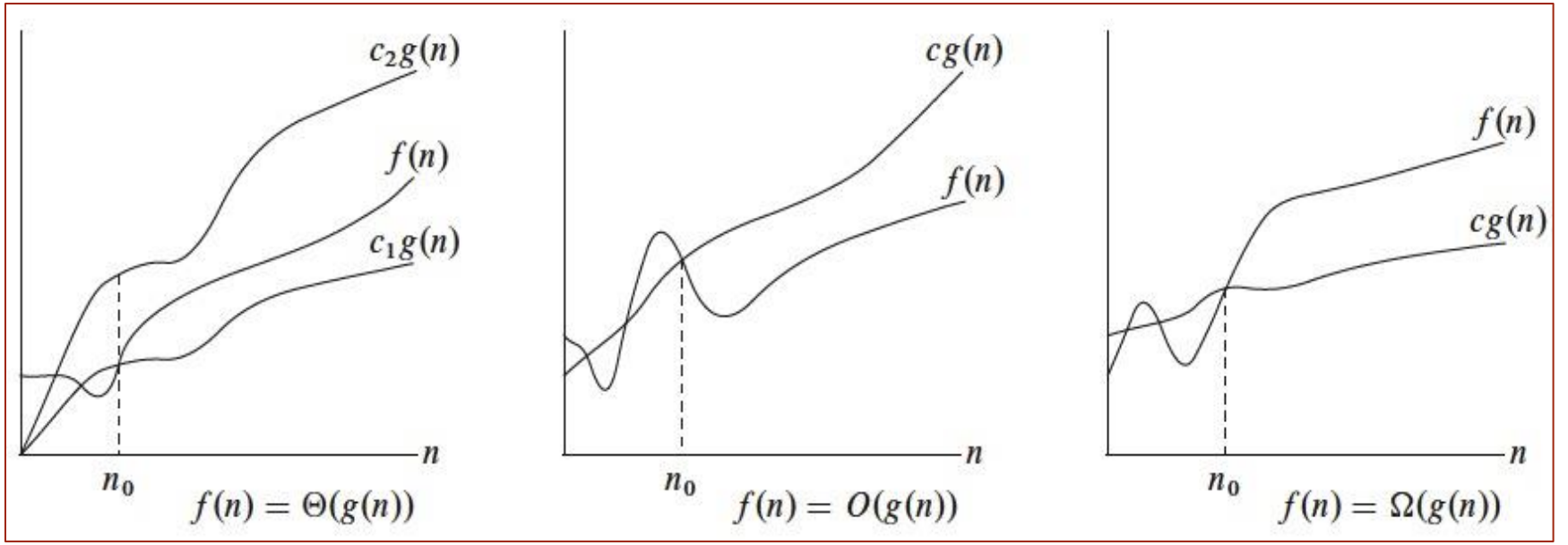
- Algorithm analysis
- Sorting and searching
- Linked lists
- Stacks and queues
- Hash tables
- Trees
- Graphs
- Advanced algorithms on trees and graphs
- Dynamic programming & greedy algorithms
- LINQ, Unit testing & debugging
- Sample exam

Algorithm analysis

- Machines do not really dominate the efficiency
  - Example
    - Sort an array of 1 million elements.
    - Two algorithms
      - A => complexity = 2 * (number of elements)^2,
      - B => complexity = 50* number of elements * log(number of elements)
    - Two machines
      - C1 (1 billion instructions per second), runs algorithm A
      - C2 (10 millions instructions per second), runs algorithm B
    - Time to sort 1 million elements on C1 = 2 * (10^6)^2 / 10^9) = 2000 seconds
    - Time to sort 1 million elements on C2 = 50 * 10^6 * log (10^6) / 10^7 = 100 seconds !!!
- How efficient is my algorithm? How to improve?
- Given two algorithms A, B; which one is more efficient?

Algorithm analysis (cont)

- Computational complexity analysis of algorithms
  - Running time T(n) - number of operations to reach a solution
  - Big-O, Big-Ω, Big-Θ



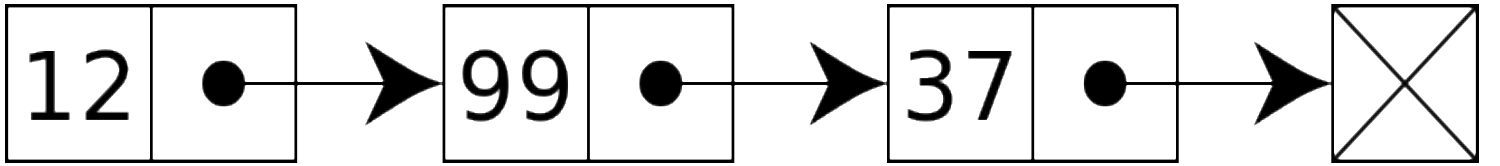
Sorting & Searching

- Sorting
  - Bubble sort
  - Insertion sort
  - Selection sort
  - Merge sort
  - Quick sort
- Searching
  - Linear Search
  - Binary Search

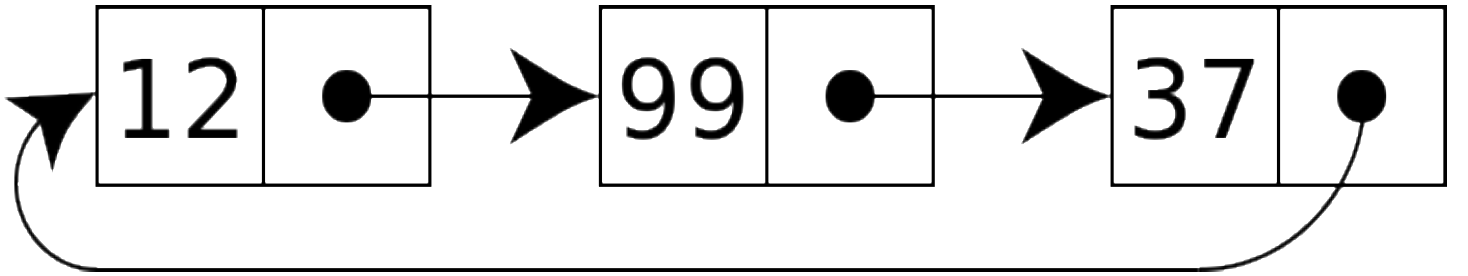
Linked Lists – A recursive data structure

Basic operations
- Traversal
- N-th Node
- Add
- Insert
- RemoveAt

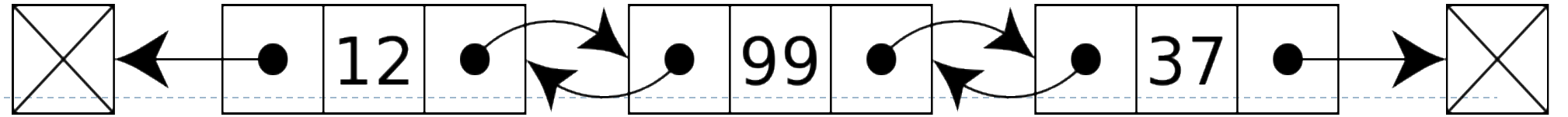
Singly linked lists



Circular linked lists

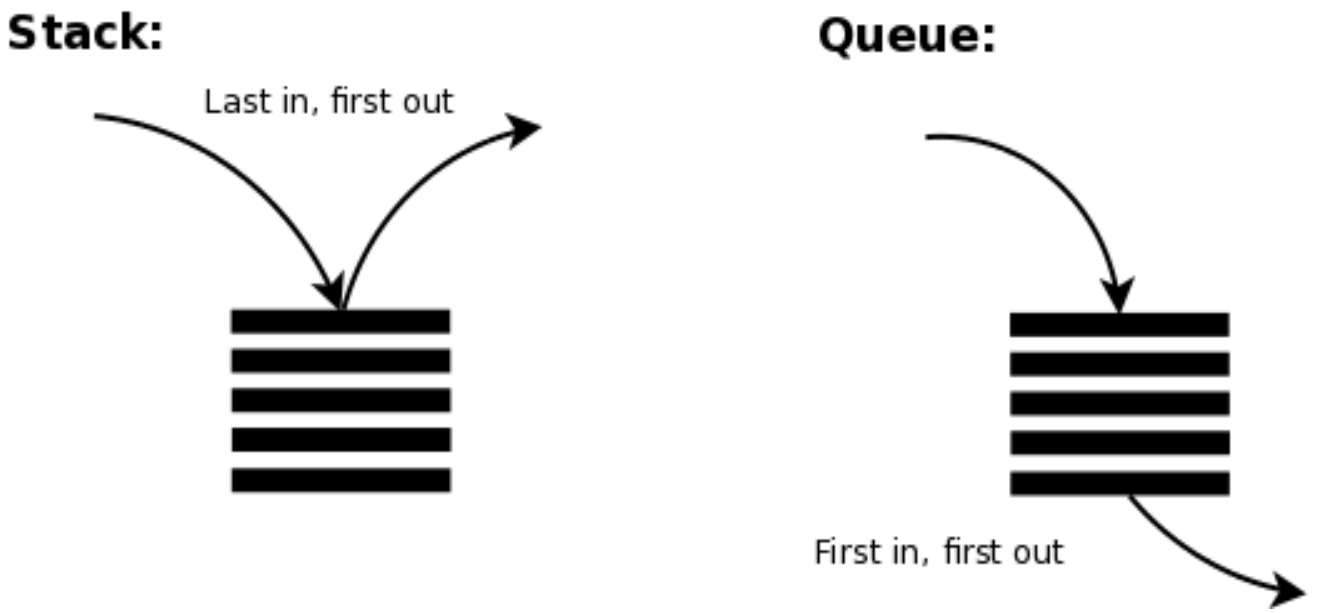


Doubly linked lists



Stacks & Queues

- Last-in First-out (LIFO) = Stack = Reverse = Backtrack
- First-in First-out (FIFO) = Queue

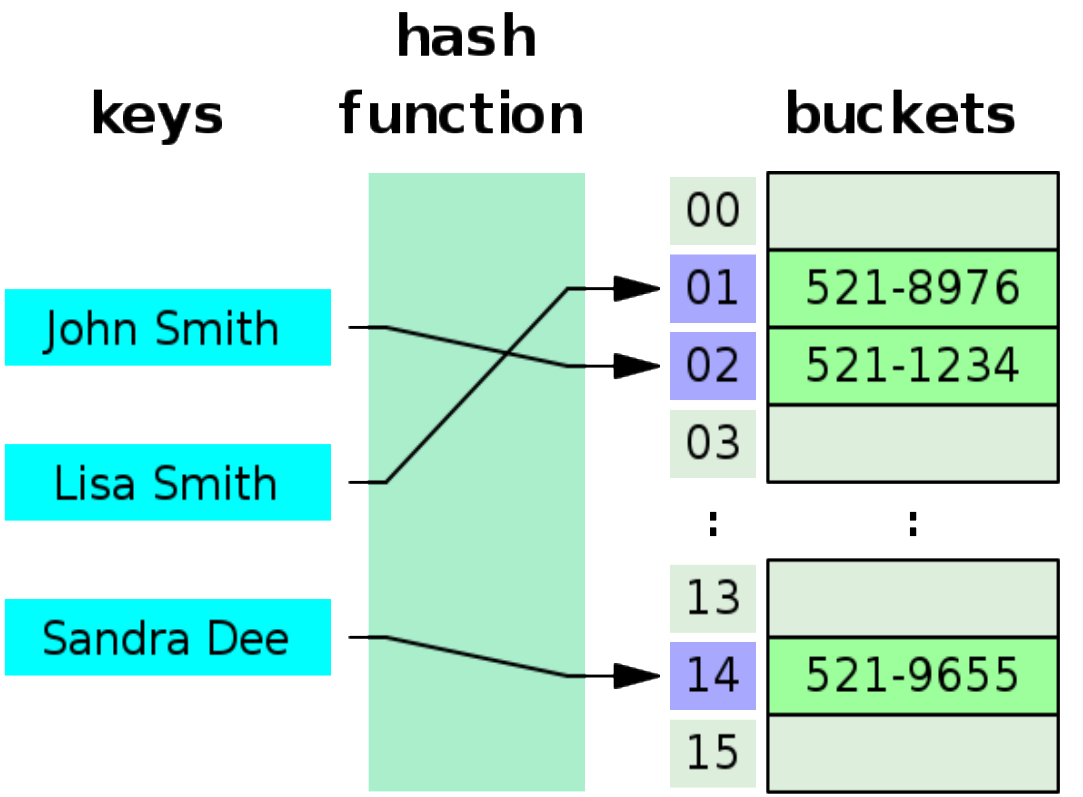


Hash table or Associative Arrays

- What's a dictionary? A collection of KeyValuePair<KEY, VALUE>
- No duplicate keys allowed, same key always has same hash code
- Ideally different keys will generate different hash codes, but…not always

Basic operations

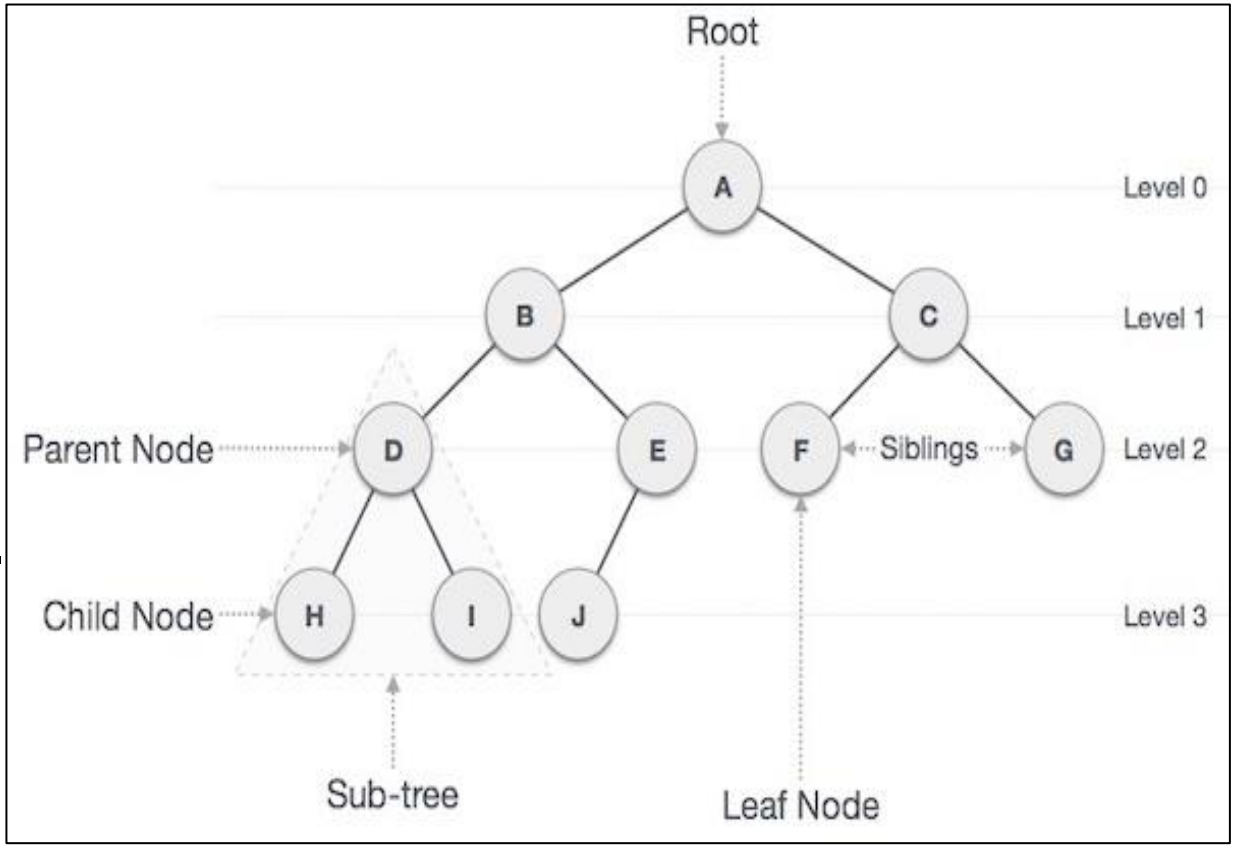
- Insert(key, value) in constant time
- Delete(key) in constant time
- Contains(key) in constant time
- ValueAt(key) in constant time



Phone book using hashtable

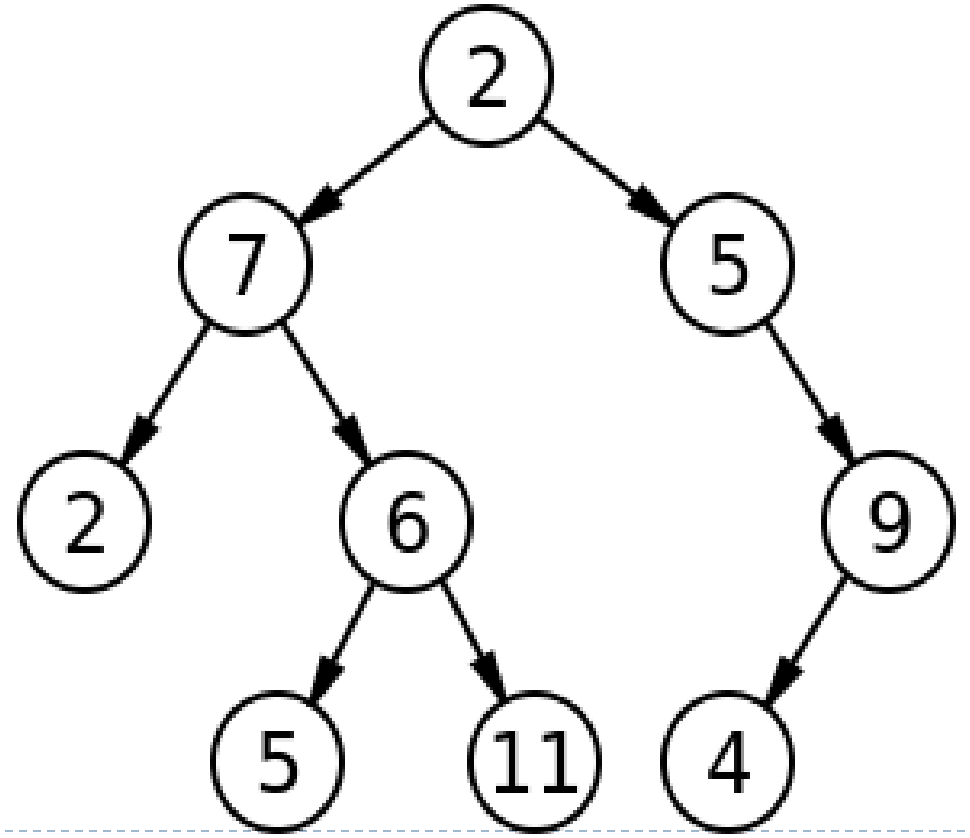
Tree (hierarchical data structure)

- One **Root** node.
- **Siblings** – same parent
- **Leaf** node – no children
- **Node depth** – number of edges to/from root node.
- **Height** – number of edges from node to leaf node.
- No **cycles/loop**



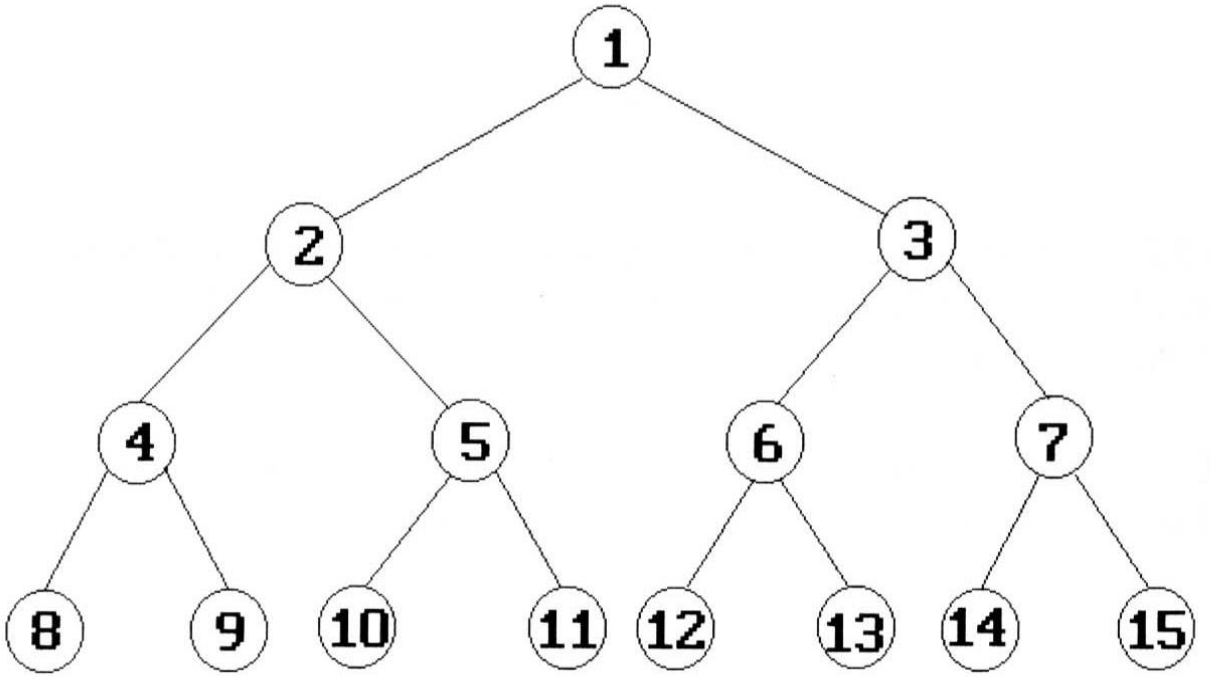
Special types of trees (1. Binary Tree)

- Each node can have at most 2 children – left child & right child



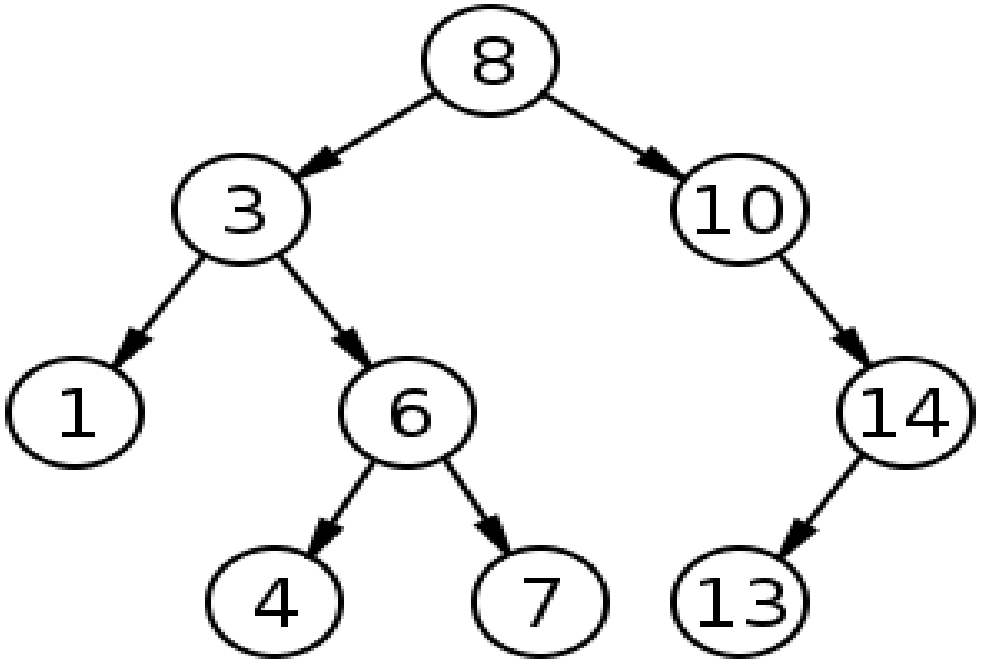
Special types of trees (2. Complete/full Binary Tree)

- A binary tree
- Every level, *except leaves*, is completely filled



Special types of trees (3. Binary Search Tree - BST)

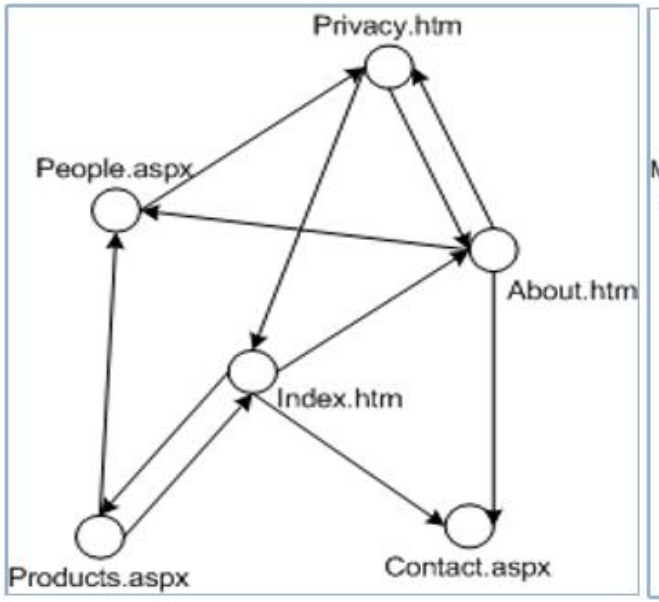
- A binary tree.
- For each node, nodes on the left are less than nodes on the right.
- Basic operations
  - Add
  - Search (binary search)
  - Min/Max
  - Traverse
    - Pre-order (root, left, right)
    - In-order (left, root, right)
    - Post-order (left, right, root)

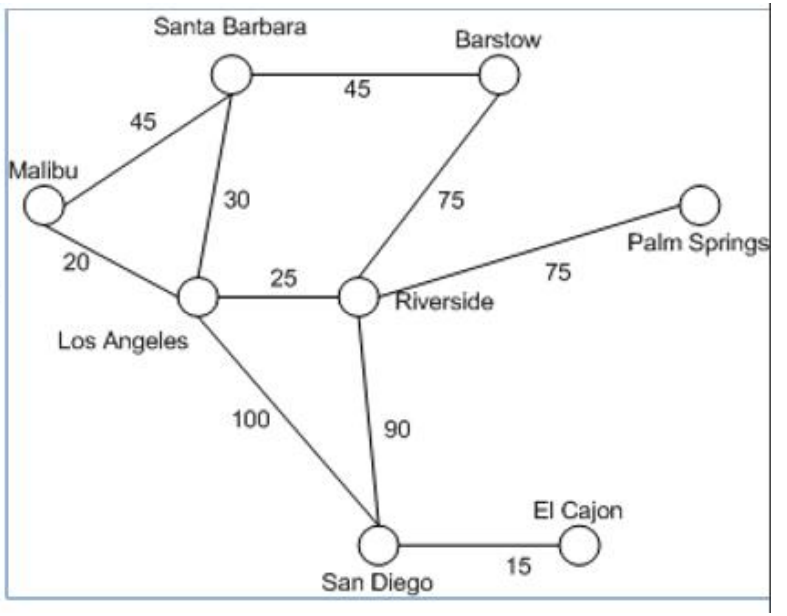


Graph

- What are the key elements in this diagram?
  - Nodes/vertices, e.g. pages, individuals, etc.
  - Links/edges, e.g. to represent navigations from one page to another, route from a city to another, etc.
- Any graph can be described as: Vertices & edges **G = (V, E)**
- Prac: Quickgraph

  http://quickgraph.codeplex.com/



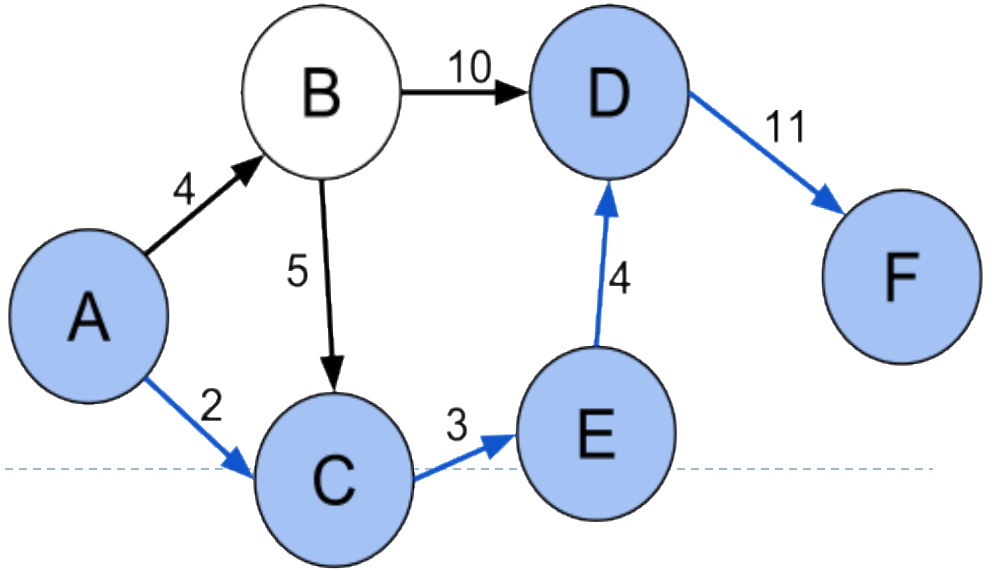


Advanced algorithms on trees and graphs

- Traversing trees
  - Breadth first search (BFS): Traverse level by level (Queues can be used in implementation)
  - Depth first search (DFS): Traverse in depth (Stacks can be used in implementation)

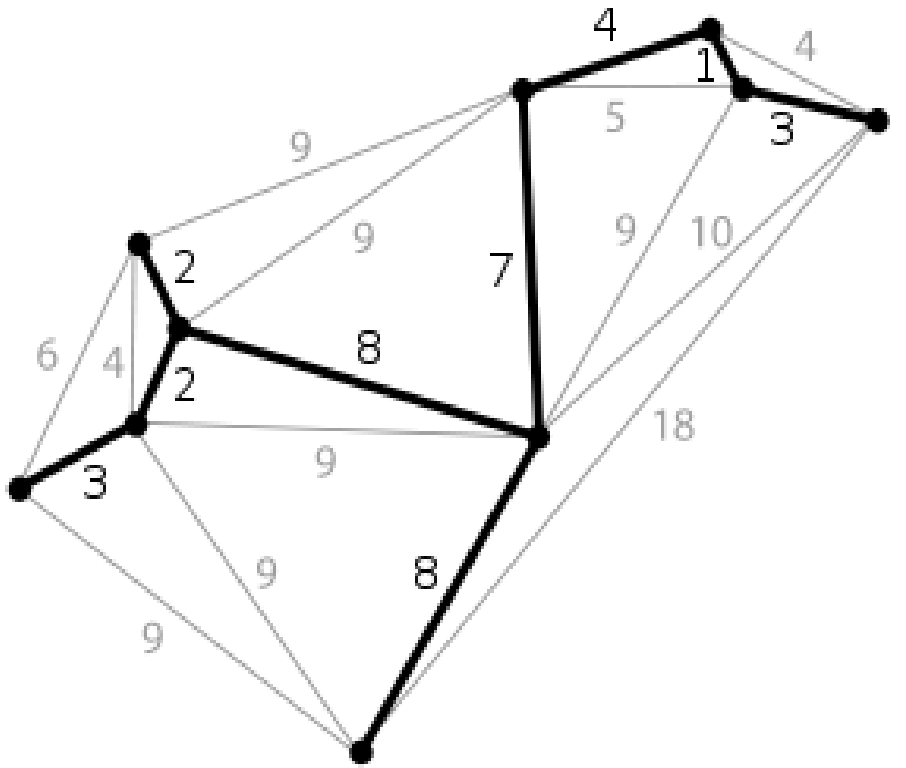
Advanced algorithms on trees and graphs (cont)

- Shortest Path: Using Dijkstra's algorithm
- What are the possible paths between A & F? which one is the shortest?
  - Path1: A → B → D → F [cost = 4 + 10 + 11]
  - Path2: A → B → C → E → D → F [cost = 4+5+3+4 + 11]
  - Path3: A→ C → E → D → F [cost = 2 + 3 + 4 + 11]



Advanced algorithms on trees and graphs (cont)

- Minimum Spanning Tree (MST)
  - Finding a low-cost tree connecting a set of nodes.
  - Minimal total weighting for its edges.
  - A graph with *n* vertices will have a spanning tree with *n*-1 edges.
  - Prim's and Kruskal's algorithm

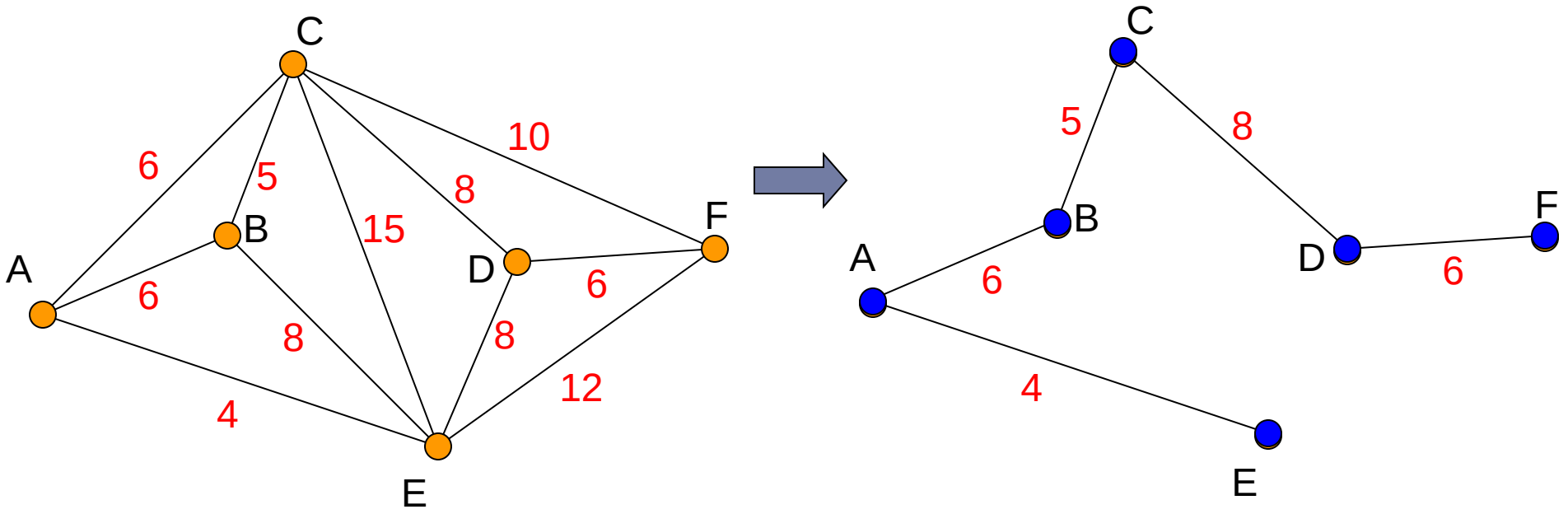


Prim's algorithm

1. $T = \emptyset$
2. Randomly select a vertex and add this vertex to T.
3. If every vertex of G is in T, then stop. Otherwise, go to step 4.
4. Find an edge which
   i) connects a vertex $\in$ T to a vertex $\notin$ T, and
   ii) has minimal weight.

   Add this edge to T and go back to step 2.

Prim's algorithm - Example



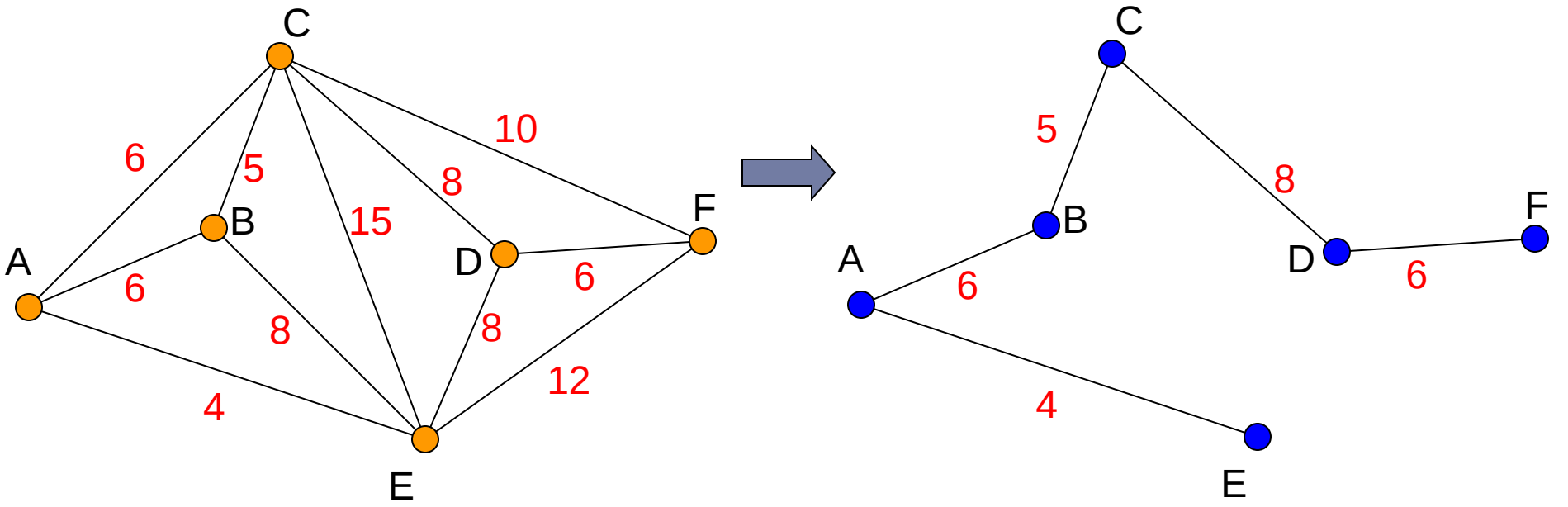
$$\sum weights = 29$$

Kruskal's algorithm

1. $T = (V, E_T)$ với $E_T = Ø$.
2. If T is connected*, then stop. Otherwise, go to Step 3.
3. Select an edge $\notin E_T$ with minimum weight such that this edge does not create any cycles in T when it is added into T. Go back to Step 2.

*A graph is connected if there always exists routes between any pair of nodes

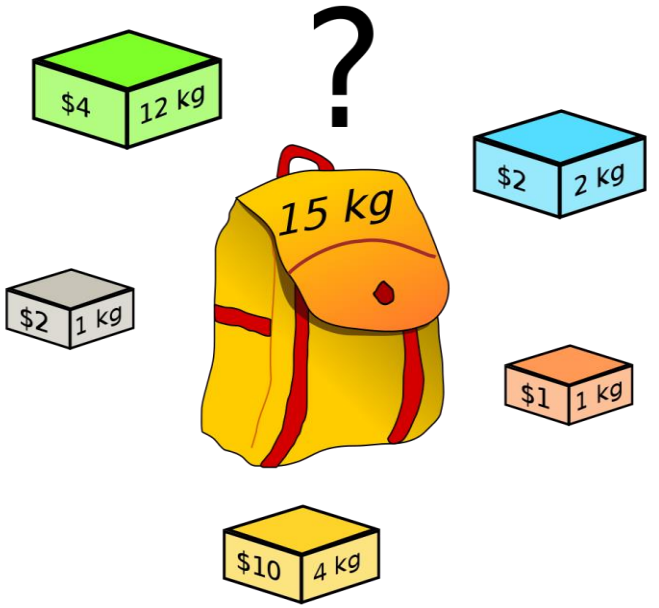
Kruskal's algorithm - example



$$\sum weights = 29$$

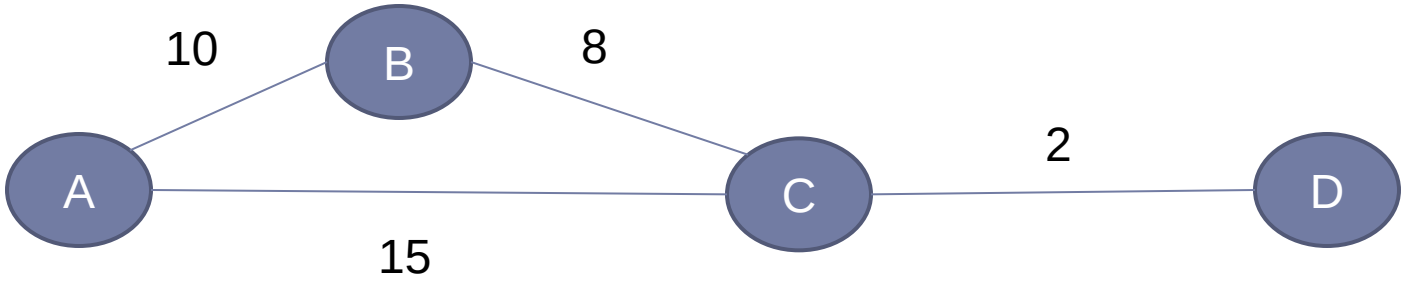
Dynamic programming

- Divide & Conquer + Memorize
- It moves bottom-up (instead of top-down) while storing solutions to sub-problems to use in solving the original problems.
- Example: 0/1 knapsack problem



Greedy Algorithms

- "take what you can get now" strategy.
- Work in phases, in each phase the currently best decision is made
- Not guaranteed optimal solution.



LINQ, Unit Testing & Debugging

- LINQ

```
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

var lowNums =
    from n in numbers
    where n < 5
    select n;
```

```
List<Product> products = GetProductList();

var expensiveInStockProducts =
    from p in products
    where p.UnitsInStock > 0 && p.UnitPrice > 3.00M
    select p;
```

Lambda expressions

▲ 1 of 2 ▼ (extension) IEnumerable<string> IEnumerable<string>.Where<string>(Func<string, bool> predicate)
Filters a sequence of values based on a predicate.
predicate: *A function to test each element for a condition.*

```
string[] digits = { "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine" };
var shortDigits = digits.Where((digit, index) => digit.Length<index);
Console.WriteLine("Short digits");
foreach (var d in shortDigits)
{
    Console.WriteLine("The word {0} is shorter than its value", d);
}
```

```
Short digits
The word five is shorter than its value
The word six is shorter than its value
The word seven is shorter than its value
The word eight is shorter than its value
The word nine is shorter than its value
```

Unit testing – using NUnit

- A unit testing framework for .NET applications
- You write your test cases using C#.
- You can re-run whenever you need to.
- It can be used in continuous delivery, you commit your code, build, run test cases, and publish.

What makes a good unit test

- AAA: Arrange, Act, Assert
- Let's take an example of a calculator that adds, subtracts,…two numbers

```
public void TestAddTwoPositives() {
    //Arrange
    int v1 = 10;
    int v2 = 20;
    Int expect = v1 + v2;

    //Act
    Int Actual = Calc.Add(v1, v2);

    //Assert
    Assert.AreEqual( expected , actual);
}
```

Debugging

- Break point
- Step into
- Step over
- Step out
- Locals / Watch
- Call Stack
- Console.WriteLine && Debug.Write

Sample Exam