CMP418: Algorithm and Complexity Analysis (3 units)

Lecture 3: Brute Force and Exhaustive Search

MR. M. YUSUF

Chapter 3 Outline

- Selection Sort and Bubble Sort
- Sequential Search and Brute-Force String Matching
- Closest-Pair and Convex-Hull Problems by Brute Force
- Exhaustive Search
- Depth-First Search and Breadth-First Search

What is Brute Force Approach (BFA)?

- The first algorithm design Approach
- A straightforward approach to solving problem,
- Usually based on problem statement and definitions of the concepts involved
- "**Force**" comes from using computer power not intellectual power
- In short, "**brute force**" means "Just do it!"
- It is the only general approach that always works
- Seldom gives efficient solution, but one can easily improve the brute force version.
- Serves as a yardstick to compare with more efficient solutions

Brute Force Case Study – Selection Sort

- We start selection sort by **<u>scanning the entire given list</u>** to find its smallest element and **<u>Exchange it with the first element</u>**,
  - putting the smallest element in its final position in the sorted list.
- Then we scan the list, **<u>starting with the second element</u>**, to find the smallest among the last n - 1 elements and **<u>exchange it with the second element</u>**,
  - putting the second smallest element in its final position.
- On the i-th pass (i goes from 0 to n-2) the algorithm searches for the smallest item among the **<u>last n-i elements and swaps it with A_i</u>**

$$\begin{tikzcd}[column sep=small] A_0 \le A_1 \le \ldots \le A_{i-1} \mid & A_i \arrow[rr, leftarrow, bend left] \arrow[rr, rightarrow, bend left] & , \ldots\ldots, & A_{min}, & \ldots\ldots, A_{n-1} \end{tikzcd}$$

already sorted   the last n-i elements

Brute Force Case Study – Selection Sort Algorithm

```
ALGORITHM SelectionSort(A[0,..n-1])
for i <- 0 to n-2 do
    min <- i
    for j <- i+1 to n-1 do
        if A[j] < A[min]
            min <- j
    swap A[i] and A[min]
```

| 89 45 68 90 29 34 **17**
17 | 45 68 90 **29** 34 89
17 29 | 68 90 45 **34** 89
17 29 34 45 | 90 **68** 89
17 29 34 45 68 | 90 **89**
17 29 34 45 68 89 | 90

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

C(n) ϵ Θ(n^2)
# of key swaps ϵ Θ(n)

Brute Force Case Study – Bubble Sort

- To **compare adjacent elements** of the list and exchange them **if they are out of order.**
- By doing it repeatedly, we end up "**bubbling up**" the largest element to the last position on the list.
- The next pass bubbles up the second largest element, and so on, until after n - 1 passes the list is sorted.
- Pass i ($0 \le i \le n - 2$) of bubble sort can be represented by the following diagram:

$$A_0, \ldots \ldots, A_j \overset{?}{\longleftrightarrow} A_{j+1}, \ldots \ldots, A_{n-i-1} \mid \underbrace{A_{n-i} \le \ldots \le A_{n-1}}_{\text{On their final positions}}$$

Brute Force Case Study – Bubble Sort…

```
**ALGORITHM** BubbleSort(A[0..n-1])
for i <- 0 to n-2 do
    for j <- 0 to n-2-i do
        if A[j+1] < A[j]
            swap A[j] and A[j+1]
```

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

$C(n) \in \Theta(n^2)$

What about 89, 45, 68, 90, 29, 34, 17 ?

Could you improve it?

Brute Force Case Study – Bubble Sort…

What about sorting 89, 45, 68, 90, 29, 34, 17 ?

Pass 1

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

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

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

45, 68, 89, 90, 29, 34, 17 No Swap

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

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

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

n-2-i , i = 0

Pass 2

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

45, 68, 89, 29, 34, 17, 90 No Swap

45, 68, 89, 29, 34, 17, 90 No Swap

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

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

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

n-2-I, i=1

Brute Force Case Studies – Bubble Sort…

What about sorting 89, 45, 68, 90, 29, 34, 17 ?

Pass 3

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

45, 68, 29, 34, 17, 89, 90 No Swap

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

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

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

n-2-I, i=2

Pass 4

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

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

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

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

n-2-I, i=3

Brute Force Case Studies – Bubble Sort…

What about sorting 89, 45, 68, 90, 29, 34, 17 ?

Pass 5

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

29, 34, 17, 45, 68, 89, 90 No Swap

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

n-2-I, i=4

Pass 6

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

17, 29, 45, 68, 68, 89, 90 **Sorted!**

n-2-I, i=5

What is the difference between Selection and Bubble sort?...

What about sorting 89, 45, 68, 90, 29, 34, 17 ?

```
ALGORITHM SelectionSort(A[0,..n-1])
for i <- 0 to n-2 do
    min <- i
    for j <- i+1 to n-1 do
        if A[j] < A[min]
            min <- j
    swap A[i] and A[min]
```

```
ALGORITHM BubbleSort(A[0..n-1])
for i <- 0 to n-2 do
    for j <- 0 to n-2-i do
        if A[j+1] < A[j]
            swap A[j] and A[j+1]
```

Chapter 3 Outline

- Selection Sort and Bubble Sort
- Sequential Search and Brute-Force String Matching
- Closest-Pair and Convex-Hull Problems by Brute Force
- Exhaustive Search
- Depth-First Search and Breadth-First Search

Brute Force Case Study – Sequential Search

- The algorithm simply compares successive elements of a given list with a given search key until either a match is encountered (successful search)
- Or the list is exhausted without finding a match (unsuccessful search).
- A simple extra trick is often employed in implementing sequential search: if we append the search key to the end of

Brute Force Case Studies – Sequential Search…

```
ALGORITHM SequentialSearch(A[0..n-1], K)
//Output: index of the first element in A, whose
//value is equal to K or -1 if no such element is found
i <- 0
while i < n and A[i] ≠ K do
    i <- i+1
    if i == n
        return i
    else
        return -1
```

$C_{worst}(n) = n$

Brute Force Case Studies – String Matching

- Given a string of ***n*** characters called the ***text*** and a string of ***m*** characters ($m \leq n$) called the ***pattern,***
- We find a substring of the text that matches the pattern.
- To put it more precisely, we want to find i-the index of the leftmost character of the first matching substring in the text

$$\begin{tikzcd}
t_0 & \cdots & t_i \arrow[d, leftrightarrow] & \cdots & t_{i+j} \arrow[d, leftrightarrow] & \cdots & t_{i+m-1} \arrow[d, leftrightarrow] & \cdots & t_{n-1} & \textbf{Text } \boldsymbol{T} \\
 & & p_0 & \cdots & p_j & \cdots & p_{m-1} & & & \textbf{Pattern } \boldsymbol{P}
\end{tikzcd}$$

p_0 should be tested with up to $t_?$

Brute Force Case Study – String Matching…

```
ALGORITHM BruteForceStringMatching(T[0..n-1], P[0..m-1])
for i <- 0 to n-m do
    j <- 0
    while j < m and P[j] = T[i+j] do
        j <- j+1
    if j = m
        return i
return -1
```

N O B O D Y _ N O T I C E D _ H I M

$C_{worst}(n, m) = m(n-m+1) \in O(nm)$

$C_{avg}(n, m) \in \Theta(n)$

Brute Force Case Study – String Matching…

- Length of text = n
- Length of pattern = m
- Maximum number of comparison = m(n – m + 1)

Exercise: How many comparisons (both successful and unsuccessful) will be made by the brute force algorithm in searching for each of the following patters in the binary tree of **one thousand zeros**

Pattern 1 = 00001

Pattern 2 = 01010

Chapter 3 Outline

- Selection Sort and Bubble Sort
- Sequential Search and Brute-Force String Matching
- Closest-Pair Problem by Brute Force
- Exhaustive Search
- Depth-First Search and Breadth-First Search

Brute Force Case Study – Closest-Pair

- Find the two closest points in a set of n points
- Points can be airplanes (most probable collision candidates), database records, DNA sequences, etc.
- Cluster analysis: pick two points, if they are close enough they are in the same cluster, pick another point,
- Euclidean distance, $d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$
- Brute-force: compute distance between each pair of disjoint points and find a pair with the smallest distance $d(p_i, p_j) = d(p_j, p_i)$, so we consider only $d(p_i, p_j)$ for $i < j$

Brute Force Case Study – Closest-Pair…

```
ALGORITHM BruteForceClosestPair(P)
//Input: A list P of n (n≥2) points p1(x1,y1),
//p2(x2,y2), …, pn(xn,yn)
//Output: distance between closest pair
d <- ∞
for i <- 1 to n-1 do
    for j <- i+1 to n do
        d <- min( d, sqrt($(x_i - x_j)^2 + (y_i - y_j)^2$ ) )
return d
```

C(n) =

$$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} 2$$

$$= 2\sum_{i=1}^{n-1}(n-i)$$

$= 2[(n-1)+(n-2)+\dots+1]$

$= (n-1)n \in \Theta(n^2)$

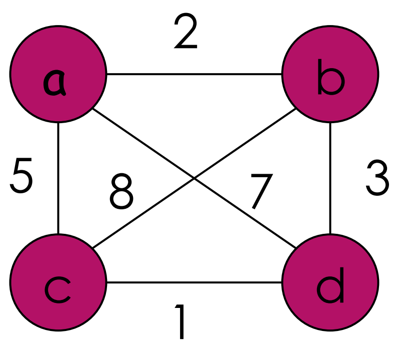
Chapter 3 Outline

- Selection Sort and Bubble Sort
- Sequential Search and Brute-Force String Matching
- Closest-Pair and Convex-Hull Problems by Brute Force
- Exhaustive Search
- Depth-First Search and Breadth-First Search

Brute Force Case Study – Exhaustive Search

- Traveling Salesman Problem (TSP)
  - Find the shortest tour through a given set of n cities that visits each city exactly once before returning to the city where it started
  - Can be conveniently modeled by a weighted graph; vertices are cities and edge weights are distances
  - Same as finding "Hamiltonian Circuit": find a cycle that passes through all vertices exactly once

Brute Force Case Study
Exhaustive Search: Travelling Sales Man (TSM)



Consider only when b precedes c

$\frac{1}{2}$(n-1)! permutations

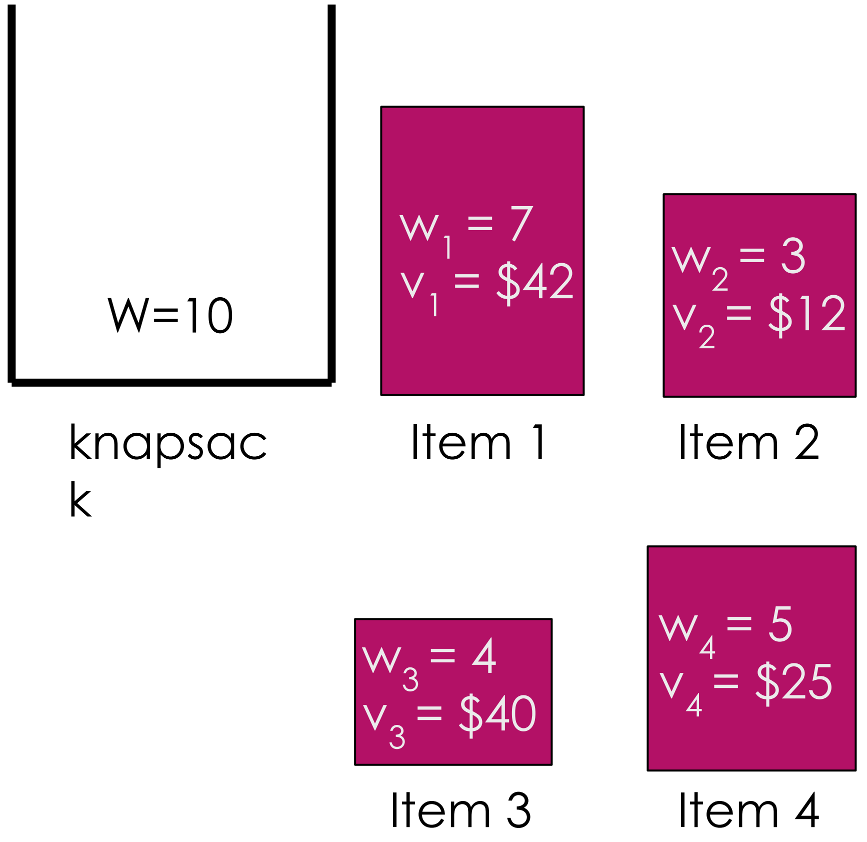
a □ b □ c □ d □ a	2+8+1+7 = 18
a □ b □ d □ c □ a	2+3+1+5 = 11 ⟵ optimal
a □ c □ b □ d □ a	5+8+3+7 = 23
a □ c □ d □ b □ a	5+1+3+2 = 11 ⟵ optimal
a □ d □ b □ c □ a	7+3+8+5 = 23
a □ d □ c □ b □ a	7+1+8+2 = 18

Brute Force Case Studies
Exhaustive Search: Knapsack Problem (KP)

- Given n items of weights $w_1, w_2, \ldots, w_n$ and values $v_1, v_2, \ldots, v_n$ and a knapsack of capacity W, find the most valuable subset of the items that fit into the knapsack
- A transport plane has to deliver the most valuable set of items to a remote location without exceeding its capacity
- How do you solve this?
- Brute force: Generate all possible subsets of the n items, compute total weight of each subset to identify feasible subsets, and find the subset of the largest value

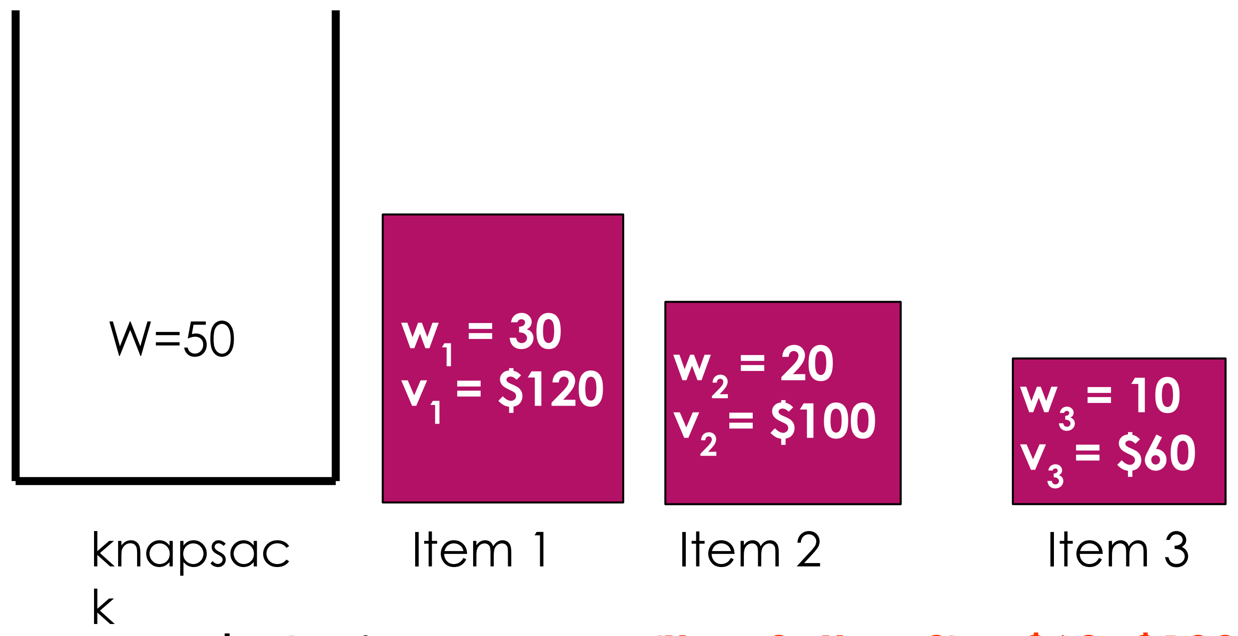
Brute Force Case Study Exhaustive Search: KP1

subset	weight	value
∅	0	$0
{1}	7	$42
{2}	3	$12
{3}	4	$40
{4}	5	$25
{1,2}	10	$54
{1,3}	11	!feasible
{1,4}	12	!feasible
{2,3}	7	$52
{2,4}	8	$37
{3,4}	9	$65
{1,2,3}	14	!feasible
{1,2,4}	15	!feasible
{1,3,4}	16	!feasible
{2,3,4}	12	!feasible
{1,2,3,4}	19	!feasible



Works for this example! ☺

Brute Force Case Study Exhaustive Search: KP2



Item 1: \$4/unit
Item 2: \$5/unit
Item3: \$6/unit

{Item3, Item2} = \$60+\$100 = \$160

{Item3, Item1} = \$60+\$120 = \$180

{Item2, Item1} = \$100+\$120 = \$220

Brute Force Case Study - Exhaustive Search

- A brute force approach to combinatorial problems (which require generation of permutations, or subsets)
- Generate every element of problem domain
- Select feasible ones (the ones that satisfy constraints)
- Find the desired one (the one that optimizes some objective function)
- For both TSM and KP, exhaustive search gives exponential time complexity.
- These are NP-hard problems, no known polynomial-time algorithm

Brute Force Case Study
Exhaustive Search: Assignment Problem (AP)

- There are n people who need to be assigned to execute n jobs, one person per job.
- C[i, j] : cost that would accrue if i-th person is assigned to j-th job.
- Find an assignment with the minimum total cost.
- **Generate all permutations of <1, 2, 3, 4>, and compute total cost, find the smallest cost**

Cost matrix

	Job 1	Job 2	Job 3	Job 4
Person 1	9	2	7	8
Person 2	6	4	3	7
Person 3	5	8	1	8
Person 4	7	6	9	4

Brute Force Case Study Exhaustive Search: AP

$$C = \begin{pmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{pmatrix}$$

1) < 1, 2, 3, 4 > cost = 9+4+1+4 = 18
2) < 1, 2, 4, 3 > cost = 9+4+8+9 = 30
3) < 1, 3, 2, 4 > cost = 9+3+8+4 = 24
4) < 1, 3, 4, 2 > cost = 9+3+8+6 = 26
5) < 1, 4, 2, 3 > cost = 9+7+8+9 = 33
6) < 1, 4, 3, 2 > cost = 9+7+1+6 = 23

7) < 2, 1, 3, 4 > cost = 2+6+1+4 = 13
8) < 2, 1, 4, 3 > cost = 2+6+8+9 = 25
9) < 2, 3, 1, 4 > cost = 2+3+5+4 = 14
10) < 2, 3, 4, 1 > cost = 2+3+8+7 = 20
11) < 2, 4, 1, 3 > cost = 2+7+5+9 = 23
12) < 2, 4, 3, 1 > cost = 2+7+1+7 = 17

13) < 3, 1, 2, 4 > cost = 7+6+8+4 = 25
13) < 3, 1, 4, 2 > cost = 7+6+8+6 = 27
15) < 3, 2, 1, 4 > cost = 7+4+1+4 = 16
16) < 3, 2, 4, 1 > cost = 7+4+8+7 = 26
17) < 3, 4, 1, 2 > cost = 7+7+5+6 = 25
18) < 3, 4, 2, 1 > cost = 7+7+8+7 = 29

19) < 4, 1, 2, 3 > cost = 8+6+8+9 = 31
20) < 4, 1, 3, 2 > cost = 8+6+1+6 = 21
21) < 4, 2, 1, 3 > cost = 8+4+5+9 = 26
22) < 4, 2, 3, 1 > cost = 8+4+1+7 = 20
23) < 4, 3, 1, 2 > cost = 8+3+1+6 = 18
24) < 4, 3, 2, 1 > cost = 8+3+8+7 = 26

Complexity is Ω(n!)

Efficient algorithm exists Called the "Hungarian method".

Worst Assignment = 5

Best Assignment = 7

Complexity is Ω(4!) = 24

Chapter 3 Outline

- Selection Sort and Bubble Sort
- Sequential Search and Brute-Force String Matching
- Closest-Pair and Convex-Hull Problems by Brute Force
- Exhaustive Search
- Depth-First Search and Breadth-First Search

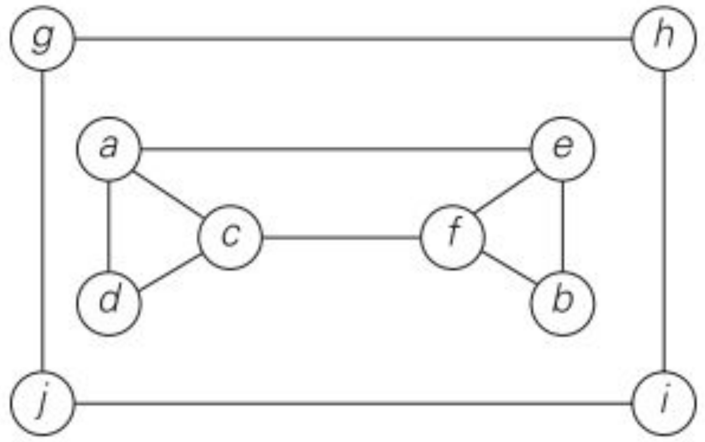
Graph Traversals

- Problem domain can be a graph
- Exhaustive search needs to visit each vertex and do something at it
- Two important graph-traversal algorithms: depth-first search, breadth first search

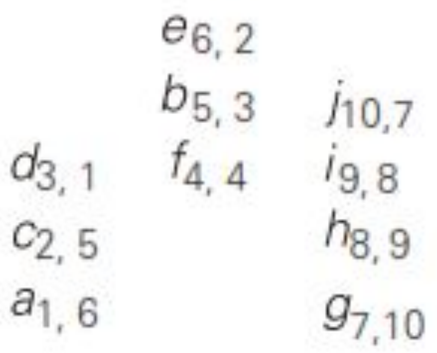
Graph Traversals: Depth-First Search (DFS)

- Start at a vertex, mark it as visited
- Go to one of unvisited neighbors, mark it visited, and so on.
- At dead end, backs up one edge to the vertex it came from and tries to visit unvisited vertices from there.
- Eventually halts after backing up to the starting vertex, by then one connected component has been traversed.
- If unvisited vertices remain, dfs starts at one of them.

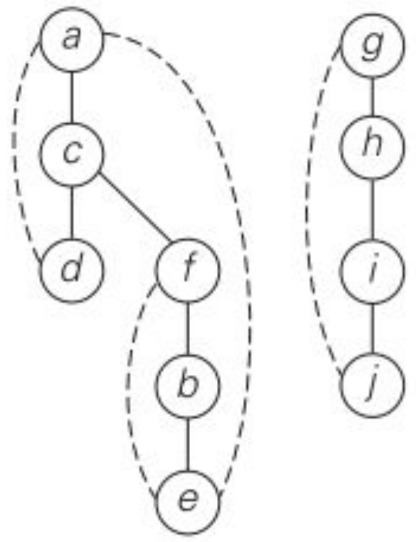
Graph Traversals: DFS



Graph



Traversal's stack: Traversal's stack (the first subscript number indicates the order in which a vertex is visited, i.e., pushed onto the stack; the second one indicates the order in which it becomes a dead-end, i.e., popped off the stack).



DFS **forest** with the tree and back edges shown with solid and dashed lines, respectively.

Graph Traversals: DFS

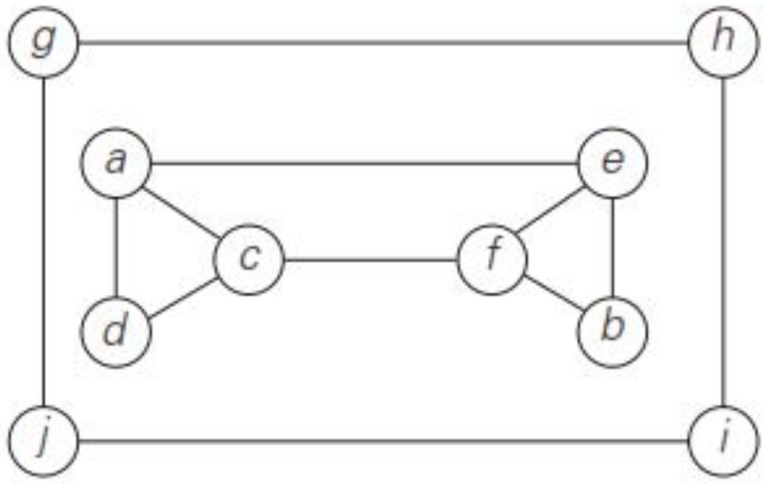
```
ALGORITHM: DFS(G)
// Input: Graph=<V, E>
mark each vertex in V with 0 as a mark of being "unvisited"
count <- 0
for each vertex v in V do
    if v is marked with 0
        dfs(v)

dfs(v)
count <- count+1; mark v with count
for each vertex w in V adjacent to v do
    if w is marked with 0
        dfs(w)
```

Graph Traversals: BFS

- Start at a vertex, mark it visited
- Go to each of its neighbors and put them in a list
- Delete the starting vertex.
- Start at one of the remaining in the list, mark it visited, and so on...

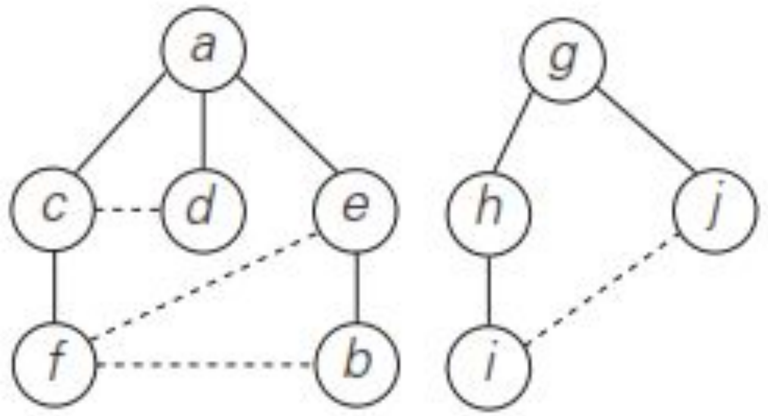
Graph Traversals: BFS



Graph

$a_1\ c_2\ d_3\ e_4\ f_5\ b_6$
$g_7\ h_8\ j_9\ i_{10}$

Traversal queue, with the numbers indicating the order in which the vertices are visited, i.e., added to (and removed from) the queue.



BFS forest with the tree and cross edges shown with solid and dotted lines, respectively.

Graph Traversals: BFS

```
ALGORITHM BFS(G)
mark each vertex in V with 0 as a mark of being “unvisited”
count <- 0
**for** each vertex v in V **do**
    **if** v is marked with 0
        bfs(v)
bfs(v)
Count <- count+1; mark v with count and initialize a queue with v
**while** the queue is not empty **do**
    **for** each vertex w in V adjacent to the front vertex **do**
        **if** w is marked with 0
            count <- count+1; mark w with count
            add w to queue
    remove the front vertex from the queue
```

Summary of DFS and BFS

	DFS	**BFS**
Data structure	Stack	Queue
Edge types	Tree and back edges	Tree and cross edges
Applications	Connectivity, Acyclicity, Articulation points	Connectivity, Acyclicity, Minimum-edge paths
Efficiency for adjacency matrix	$\Theta(\|V\|^2)$	$\Theta(\|V\|^2)$
Efficiency for adjacency list	$\Theta(\|V\| + \|E\|)$	$\Theta(\|V\| + \|E\|)$

Summary

- *BFA is a straightforward approach to solving a problem, usually directly* based on the **problem statement** and **definitions** of the concepts involved.
- The principal strengths of the BFA are **wide applicability** and **simplicity**; its principal weakness is the **subpar efficiency** of most brute-force algorithms.
- *Exhaustive search is a brute-force approach to combinatorial problems.*
- The *TSM, the KP, and the AP are typical examples of problems that can be solved, at least* theoretically, by exhaustive-search algorithms (ESA).
- *DFS and BFS are two principal* graph-traversal algorithms.

Thank You