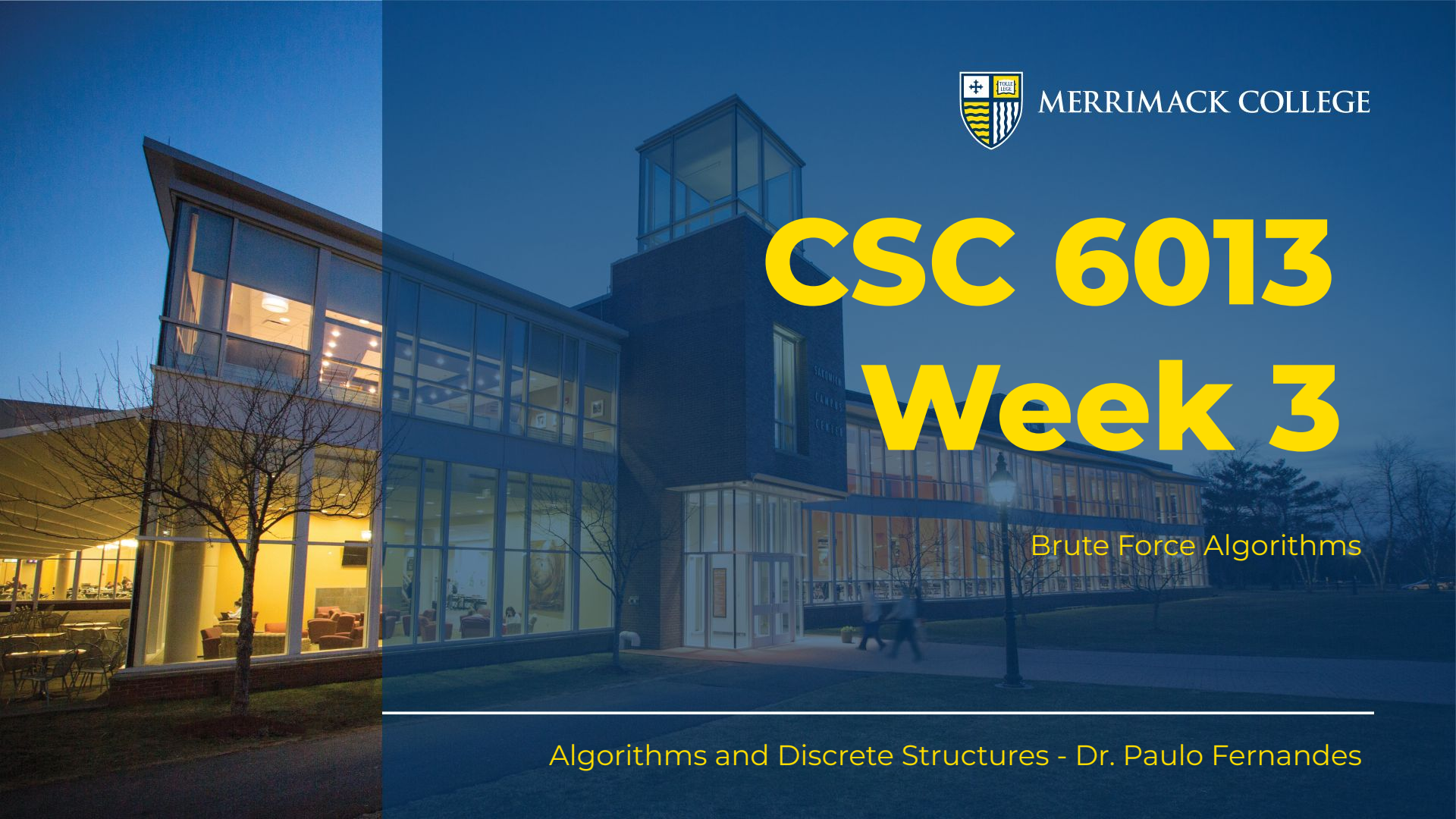


MERRIMACK COLLEGE

CSC 6013 Week 3

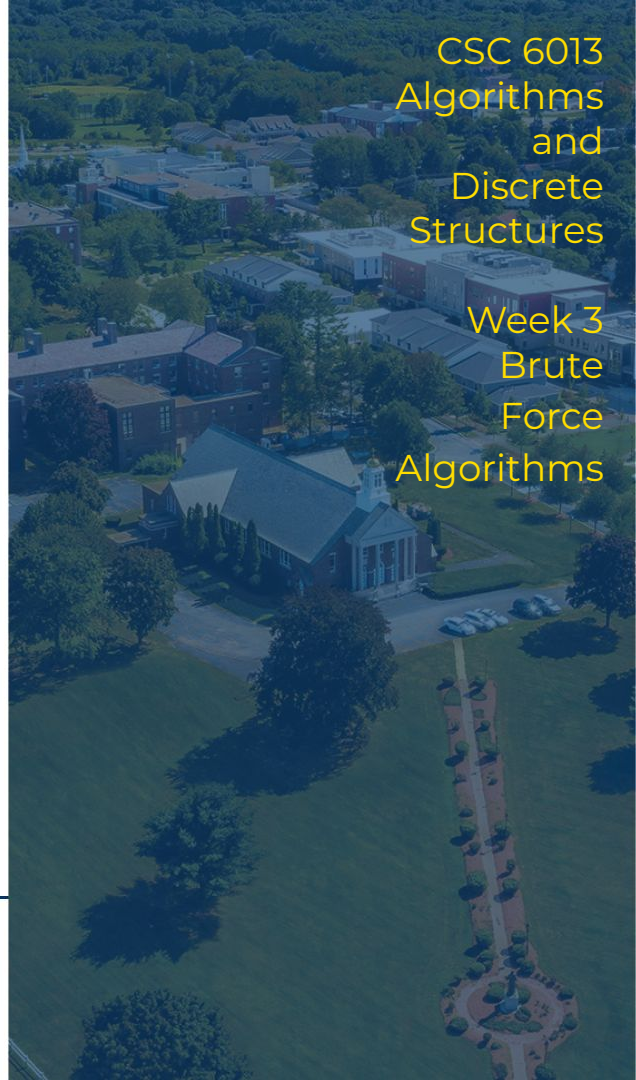
Brute Force Algorithms

Algorithms and Discrete Structures - Dr. Paulo Fernandes

Presentation Agenda

Week 3

- Why brute force?
  a. Enumeration versus Ingenuity
  b. Concepts - Combinatorics and Summations
- Brute Force Examples
  a. selection sort
  b. bubble sort
  c. string matching
  d. closest pair
  e. breadth first search in graphs (BFS)
- This Week's tasks



Why brute force?

Find the cases ...
Enumerate them ...
Then just do it!

We call brute force algorithms the class of algorithms that we do not search for tricks and clever shortcuts, but instead we just try to enumerate the cases to analyze, and we just go one by one, without much wit, just sheer computations, and this relates to brute force solutions.

Brute force, in this sense, is not a lack of sophistication, just a solution relying on sheer processing power.

- **Enumeration versus Ingenuity**
- Basics Concepts
  - Combinatorics and
  - Summations (again)

Enumeration versus Ingenuity

Brute Force is not a bad thing...

The term
Brute Force Search
is frequently employed.

- Brute Force is a **straightforward approach** to solve a problem, usually directly based on the problem statement and definitions of the concepts involved.
- Brute Force algorithms are based on solving the problem directly by doing the work to be done just doing the tasks directly;
- The **Counting positive elements in an array** algorithm seem last week is a Brute Force one, as we pass by all elements of the array and we check if they are positive, counting plus one.

```python
# Input: An array A with n Real numbers
# Output: Return the number of positive values in the array

def countPositiveElements(A):
    count = 0
    for x in A:
        if x>0:
            count += 1
    return count

from random import random
A = []
for _ in range(1000):
    A.append((random()-0.5)*100)
print("Positive elements:", countPositiveElements(A))
```

O(n)

Wikipedia: Brute-force search.

Enumeration versus Ingenuity

Brute Force is not clever...
but it is not necessarily a bad choice.

- There is no other way to count the number of positive number in an array, so the brute force solution is the best one can do.
- In Brute Force we just enumerates what needs to be done:
  - For counting positive elements, we just go over the ***n*** elements of the array.
- However, some problems may benefit from a clever approach.

```python
# Input: An array A with n Real numbers
# Output: Return the number of positive values in the array

def countPositiveElements(A):
    count = 0
    for x in A:
        if x>0:
            count += 1
    return count

from random import random
A = []
for _ in range(1000):
    A.append((random()-0.5)*100)
print("Positive elements:", countPositiveElements(A))
```

O(n)

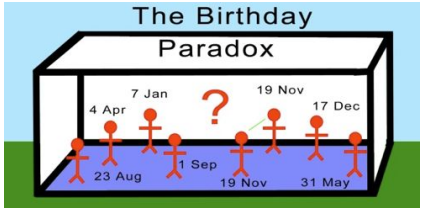
Doing a assignment, sometimes you need to think, sometimes you just have to do it!

Text: Brute Force Approach and its pros and cons.

Enumeration versus Ingenuity

Let's see a different problem...

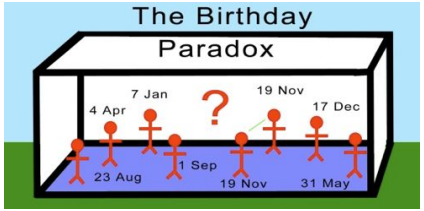
- Let's say you have a list of ***n*** people with names and date of births (dob) and you want to create an algorithm that returns ***True*** if two people have the same dob (month and day only).
  - The brute force approach would be verify to all possible pairs of people if they have the same dob:
    - The algorithm return ***True*** if one pair has the same dob, or ***False*** otherwise
  - Such algorithm will have to go through ***n*** choose ***2*** pairs, which is $\frac{n(n-1)}{2}$ pairs.
  - ... it is not that too bad, but there is a clever way...



Text: Birthday Paradox.

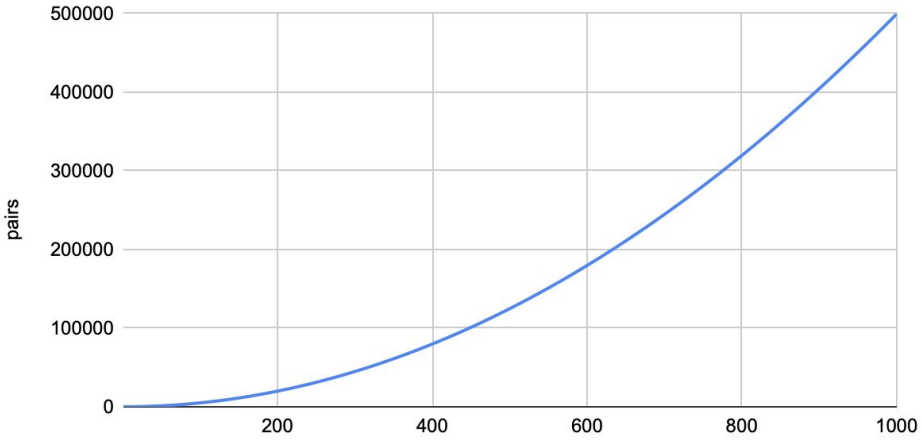
Enumeration versus Ingenuity

Same birthday probability



- The brute force solution has to check all possible pairs:
  - $\frac{n(n-1)}{2}$
- This can be upper bounded to n^2, thus the time complexity will be:
  - $O(n^2)$

A quadratic problem!

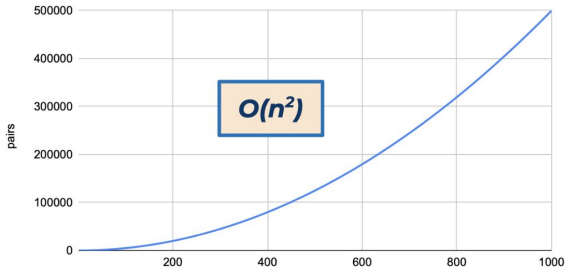


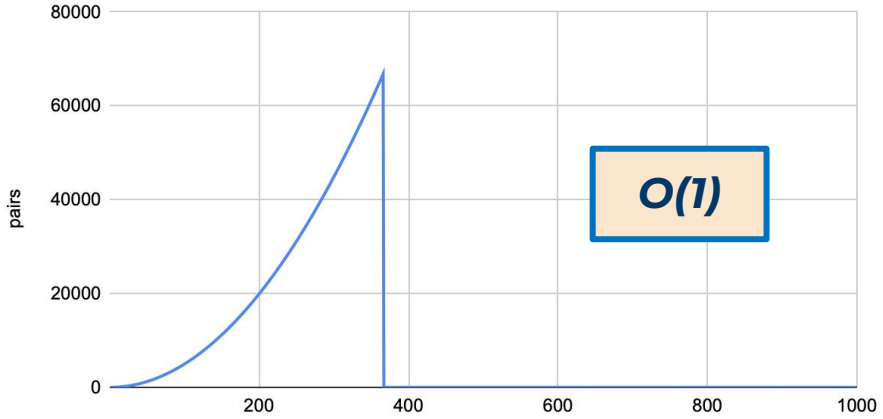
An algorithm with quadratic time complexity is slow.

Enumeration versus Ingenuity

Same birthday probability

- The brute force would be verify to all possible pairs of people if they have the same dob (month and day only), but if we consider that for ***n > 366*** the answer is always ***True***, we actually have:





... and this is a clever way!

An algorithm with constant time complexity is fast.

Why brute force?

Find the cases ...
Enumerate them ...
Then just do it!

We call brute force algorithms the class of algorithms that we do not search for tricks and clever shortcuts, but instead we just try to enumerate the cases to analyze, and we just go one by one, without much wit, just sheer computations, and this relates to brute force solutions.

Brute force, in this sense, is not a lack of sophistication, just a solution relying on sheer processing power.

- Enumeration versus Ingenuity
- **Basics**
  - **Combinatorics and**
  - **Summations (again)**

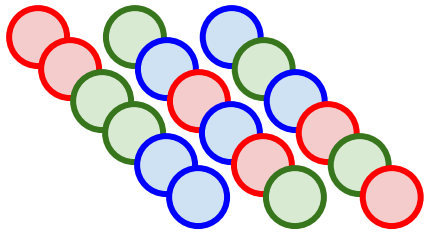
Basic concepts - Combinatorics

Permutations states how many ways to sort the elements of a set.

$$P(n) = n!$$

For example, Three athletes, **Joe**, **Jack**, and **Jeff** are competing in a final round, how many outcomes are possible for the podium?

- There are six possible outcomes!
  - **Joe**, **Jack**, and **Jeff**
  - **Joe**, **Jeff**, and **Jack**
  - **Jack**, **Joe**, and **Jeff**
  - **Jack**, **Jeff**, and **Joe**
  - **Jeff**, **Joe**, and **Jack**
  - **Jeff**, **Jack**, and **Joe**



Wikipedia: Permutation.

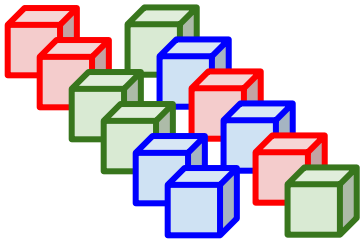
Basic concepts - Combinatorics

Arrangements states how many ways to sort a certain number of elements of a set.

$$A(k,n) = \frac{n!}{(n-k)!}$$

For example, you bought four Boxes, **A**, **B**, and **C**, and now you need pick two of them, one for the kitchen, and one for the garage, how many choices do you have?

- There are six possible choices!
  - **A** for the kitchen and **B** for the garage
  - **A** for the kitchen and **C** for the garage
  - **B** for the kitchen and **A** for the garage
  - **B** for the kitchen and **C** for the garage
  - **C** for the kitchen and **A** for the garage
  - **C** for the kitchen and **B** for the garage



Wikipedia: Partial permutation.

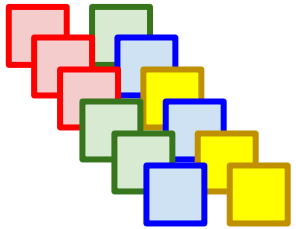
Basic concepts - Combinatorics

Combinations states how many subsets of a given size a set can have.

$$C(k,n) = \frac{n!}{(n-k)!k!}$$

For example, four state teams, **MA**, **NY**, **FL**, and **CA**, are competing in a pool, how many matches are needed to all teams play against each other?

- There are six possible matches!
  - **MA** against **NY**
  - **MA** against **FL**
  - **MA** against **CA**
  - **NY** against **FL**
  - **NY** against **CA**
  - **FL** against **CA**



Wikipedia: Combination.

Basic concepts - Summations

$$\sum_{\text{variable} = \text{initial}}^{\text{final}} \text{term}$$

The **summation** is defined by the Greek letter capital **sigma** with:

- one subscript which states the variable and its initial value;
- one superscript that states the final value; and
- the term expression.

A summation represents the sum of all term expressions with the variable going from the initial to the final value.

$$\sum_{i=1}^{5} i = 1 + 2 + 3 + 4 + 5 = 15$$

$$\sum_{i=1}^{5} 2i = 2 + 4 + 6 + 8 + 10 = 30$$

$$\sum_{i=0}^{5} 2^i = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 = 2^6 - 1 = 63$$

$$\sum_{i=0}^{99} 2i + 17$$

```
acc = 0
for i in range(100):
    acc += 2 * i + 17
print("The sum is:", acc)
```

Wikipedia: Summation.

Brute Force Algorithms Examples

Video: Brute Force Algorithms with real life examples.

Video: Calculating Time Complexity.

Algorithms Examples

- **selection sort**
- bubble sort
- string matching
- closest pair
- breadth first search in graphs (BFS)

Example 1 - Selection Sort

- Giving an array $A = [a_1, a_2, \ldots a_n]$
- Deliver a permutation of A where $a_i <= a_j$ if $i < j$

5 3 4 1 2

Selection Sort

One brute force method is:

- Find the smallest element in the sequence;
  - swap it into slot 1.
- Find the 2nd smallest element in the sequence;
  - swap it into slot 2.
- Find the 3rd smallest element in the sequence;
  - swap it into slot 3.
- … until the before last element of the array.

Starting with [5, 3, 4, 1, 2] and ending with [1, 2, 3, 4, 5].

Example 1 - Selection Sort

- For all elements, but the last:
  - Search the smallest after it;
  - Swap the element with the smallest.

5 3 4 1 2

Selection Sort

```python
def selectionSort(A):
    for i in range(len(A)-1):
        minIndex = i
        for j in range(i+1, len(A)):
            if (A[j] < A[minIndex]):
                minIndex = j
        A[i], A[minIndex] = A[minIndex], A[i]

from random import shuffle

A = list(range(1000))
shuffle(A)
selectionSort(A)
print("A=[{}, {}, ... {}, {}]".format(A[0], A[1], A[-2], A[-1]))
```

Brute Force Implementation... straightforward!

Example 1 - Selection Sort

Quadratic
Polynomial of order 2

5 3 4 1 2

- What is the worst case of it?
  - Line 5 ***if*** is always ***True***.
- What is the Big Oh for it?
  - Asymptotic Analysis:
    - Lines 2, 3, and 7 are executed ***n-1*** times
    - Lines 4, 5 and 6 are executed ***((n-1) * n) / 2***
    - ***$T(n) = c_1 * (n-1) + c_2 * ((n-1) * n) / 2$***
    - ***$T(n) < c * n^2$***

Thus, Selection sort algorithm has time complexity:

- ***$O(n^2)$***

Selection Sort

```
1  def selectionSort(A):
2      for i in range(len(A)-1):
3          minIndex = i
4          for j in range(i+1, len(A)):
5              if (A[j] < A[minIndex]):
6                  minIndex = j
7          A[i], A[minIndex] = A[minIndex], A[i]
```

... not great ... but simple!

Brute Force Algorithms Examples

Algorithms Examples

- selection sort
- **bubble sort**
- string matching
- closest pair
- breadth first search in graphs (BFS)

Example 2 - Bubble Sort

- Giving an array $A = [a_1, a_2, \ldots a_n]$
- Deliver a permutation of A where $a_i <= a_j$ if $i < j$

One brute force method is:

- Walk through the array comparing adjacent elements, exchanging them if they are out of order;
- After the first pass the largest element will be in the last position, so, the next time it is possible to stop before it;
- Thus, Bubble sort places one element in the right position position at each pass;
- In such way, the sorted elements grow in a bubble from the end to the front.

6 5 3 1 8 7 2 4

Starting [6, 5, 3, 1, 8, 7, 2, 4] and ending [1, 2, 3, 4, 5, 6, 7, 8].

Example 2 - Bubble Sort

- For ***n-1*** times:
  - Go from the first to the before last unsorted
    - Check if the element and its next are in the right order, if not swap them.

6 5 3 1 8 7 2 4

```
1   def bubbleSort(A):
2       for i in range(len(A)-1):
3           for j in range(len(A)-i-1):
4               if (A[j] > A[j+1]):
5                   A[j], A[j+1] = A[j+1], A[j]
6
7   from random import shuffle
8   A = list(range(1000))
9   shuffle(A)
10  bubbleSort(A)
11  print("A=[{}, {}, ... {}, {}]".format(A[0], A[1], A[-2], A[-1]))
```

Very intuitive... with comparison of neighbors.

Example 2 - Bubble Sort

- What is the worst case of it?
  - Line 4 ***if*** is always ***True***.

6 5 3 1 8 7 2 4

- What is the Big Oh for it?
  - Asymptotic Analysis:
    - Line 2 is executed ***n-1*** times
    - Lines 3, 4 and 5 are executed ***((n-1) * n) / 2***
    - $T(n) = c_1 * (n-1) + c_2 * ((n-1) * n) / 2$
    - $T(n) < c * n^2$
  - Thus, Bubble sort algorithm has time complexity:
    - $O(n^2)$

```
1    def bubbleSort(A):
2        for i in range(len(A)-1):
3            for j in range(len(A)-i-1):
4                if (A[j] > A[j+1]):
5                    A[j], A[j+1] = A[j+1], A[j]
```

Quadratic
Polynomial of order 2

... not great ... but very intuitive!

Brute Force Algorithms Examples

Algorithms Examples

- selection sort
- bubble sort
- **string matching**
- closest pair
- breadth first search in graphs (BFS)

Example 3 - String Matching

Is "car" in "racecar"?

0	1	2	3	4	5	6
r	a	c	e	c	a	r
c	a	r				
	c	a	r			
		c	a	r		
			c	a	r	
				c	a	r

S = "racecar" - n = 7

R = "car" - m = 3

i = 4

- Giving a string ***S = [s_1, s_2, ... s_n]*** and
  - another string ***R = [r_1, r_2, ... r_m]***
    - with ***n >= m***.
- If there is an index ***i*** where ***s_i = r_1*** and ***s_{i+1} = r_2*** and so on until ***s_{i+m-1} = r_m***
  - It returns ***i***,
  - Otherwise it returns ***None***.

One brute force method is:

- Continuously shift a window of size ***m*** from the beginning of ***S*** and compare it with the elements of ***R***.

Video: Substring search.

Example 3 - String Matching

Is "car" in "racecar"?

0	1	2	3	4	5	6
r	a	c	e	c	a	r
c	a	r				
	c	a	r			
		c	a	r		
			c	a	r	
				c	a	r

S = "racecar" - n = 7

R = "car" - m = 3

i = 4

- For all elements of ***S*** until ***n-m***:
  - Check if ***R*** match the substring of ***S*** starting at the element.

```python
def stringMatching(S, R):
    for i in range(len(S)-len(R)+1):
        j, k = i, 0
        while (k < len(R)) and (S[j] == R[k]):
            j += 1
            k += 1
        if (k == len(R)):
            return i
    return None

S, R = "racecar", "car"
i = stringMatching(S, R)
if (i != None):
    print("The match starts at", i)
else:
    print("There were no match")
```

Example 3 - String Matching

A	T	C	A	A	G	T	T	A	C	C	A	A	T	A
A	T	A												

0	1	2	3	4	5	6
r	**a**	**c**	**e**	**c**	**a**	**r**
c	a	r				
	c	a	r			
		c	a	r		
			c	a	r	
				c	a	r

```
1   def stringMatching(S, R):
2       for i in range(len(S)-len(R)+1):
3           j, k = i, 0
4           while (k < len(R)) and (S[j] == R[k]):
5               j += 1
6               k += 1
7           if (k == len(R)):
8               return i
9       return None
```

- What is the worst case of it?
  - The loop test in Line 4 is always ***True***, but the last time.
- What is the Big Oh for it?
  - Asymptotic Analysis:
    - Lines 2, 3, and 7 are executed ***n-m*** times
    - Lines 4, 5, and 6 are executed ***((n-m) * m)*** times
    - $T(n) = c_1 * (n-m) + c_2 * ((n-m) * m)$
    - $T(n) < c * nm$
  - Thus, it has time complexity:
    - ***O(nm)***

Linear
Over
n *times* ***m***

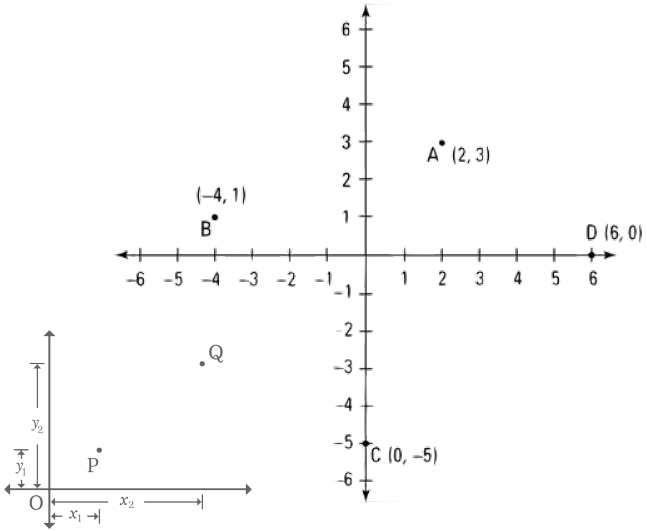
... costly ... but best you can do!

Brute Force Algorithms Examples

Algorithms Examples

- selection sort
- bubble sort
- string matching
- **closest pair**
- breadth first search in graphs (BFS)

Example 4 - Closest Pair



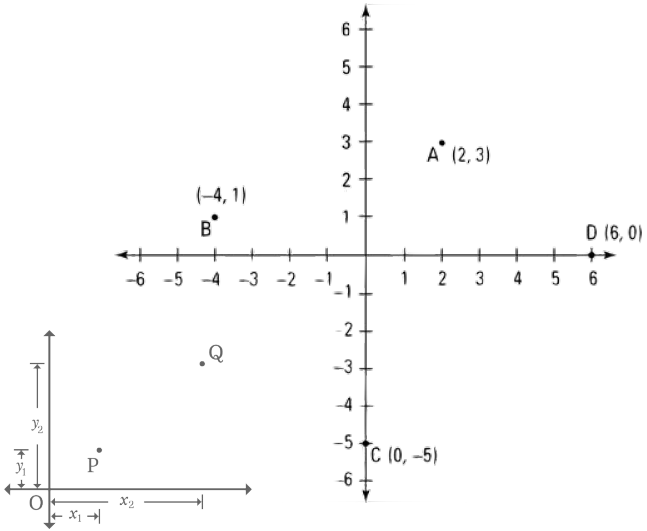
- Giving a set of Points $P = [p_1, p_2, \ldots p_n]$ in a d-dimensional space (each p_i is a coordinate with d values).
- What are the two closest points?

One brute force method is:

- To all distinct pairs of points of P compute the distance between them, and remember the pair with the smallest distance.

Application in astronomy, GPS, and several optimization techniques.

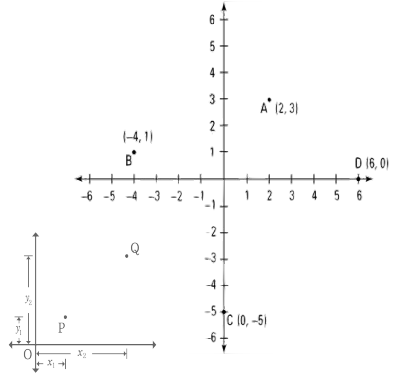
Example 4 - Closest Pair



- For all points of ***P***:
  - Compute the distance between the point and one of the subsequent points:
    - Compare to find the minimum

```python
def closestPair(P):
    minDist, ans = float("inf"), []
    for i in range(len(P)-1):
        for j in range(i+1, len(P)):
            dist = 0
            for k in range(len(P[i])):
                dist += (P[i][k]-P[j][k])**2
            dist = dist ** 0.5
            if (dist < minDist):
                minDist, pair = dist, [i,j]
    return pair[0], pair[1]

P = [ [2,3], [-4,1], [0,-5], [6,0]]
i, j = closestPair(P)
print("The closest pair is", P[i], "and", P[j])
```

Example 4 - Closest Pair



- What is the worst case of it?
  - The if in Line 9 is always ***True***.
- What is the Big Oh for it?
  - Asymptotic Analysis:

```
1   def closestPair(P):
2       minDist, ans = float("inf"), []
3       for i in range(len(P)-1):
4           for j in range(i+1, len(P)):
5               dist = 0
6               for k in range(len(P[i])):
7                   dist += (P[i][k]-P[j][k])**2
8               dist = dist ** 0.5
9               if (dist < minDist):
10                  minDist, pair = dist, [i,j]
11      return pair[0], pair[1]
```

    - Lines 2 and 11 are executed ***1*** time (once)
    - Lines 3 is executed ***n-1*** times
    - Lines 4, 5, 8, 9, and 10 are executed ***((n * n-1)/2)***
    - Lines 6 and 7 are executed ***((n * n-1)/2) * d***
    - $T(n) = c_1 + c_2 * (n-1) + c_3 * ((n * n-1)/2) + c_4 * d ((n * n-1)/2)$
    - $T(n) < c * n^2$
  - Thus, Closest Pair algorithm has time complexity:
    - $O(n^2)$

MERRIMACK COLLEGE

Number crunching … straightforward!

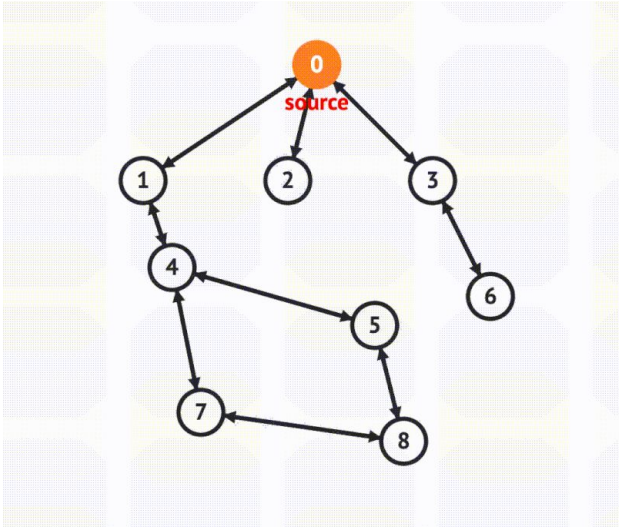
Quadratic
Polynomial of order 2

Brute Force Algorithms Examples

Algorithms Examples

- selection sort
- bubble sort
- string matching
- closest pair
- **breadth first search in graphs (BFS)**

Example 5 - Breadth-First Search in Graphs (BFS)



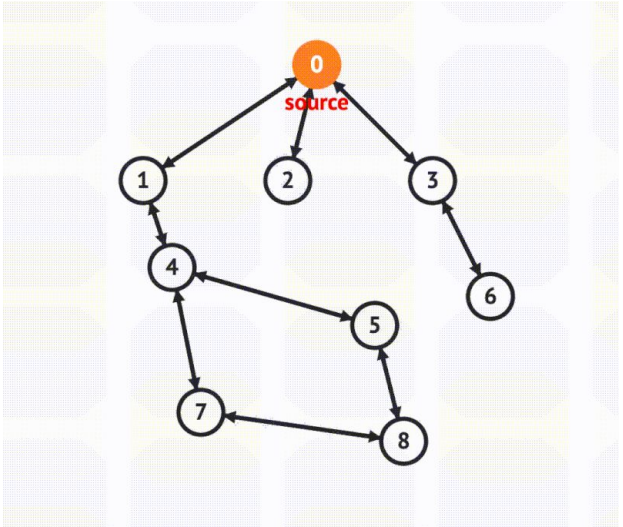
- Given a graph defined by a ***V*** set of ***n*** Vertices and ***E*** a set of ***m*** Edges.
- Starting from the first vertex in ***V*** mark all vertices from ***0*** to ***n-1*** using Breadth-First Search.

One brute force method is:

- Starting with the first vertex ***v***:
  - annotate to visit all its immediate neighbors and visit ***v***
  - repeat the process to all annotated vertices until all vertices are visited.

Application in GPS, Networks, and optimizations.

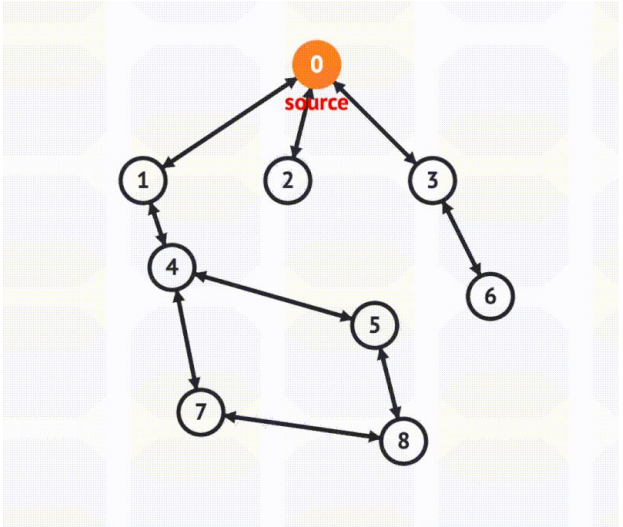
Example 5 - Breadth-First Search in Graphs (BFS)



- The implementation of this algorithm uses:
  - an array ***V*** with all vertices v_i holding information if they are visited or not;
  - an adjacency list of edges ***E*** with triplets $(v_i, v_j, 1)$
  - a counter ***count*** to keep the track of the nodes to visit
  - a queue ***Q*** to memorize the vertices to annotate the immediate neighbors.

This problem has non-brute force approaches too.

Example 5 - Breadth-First Search in Graphs (BFS)

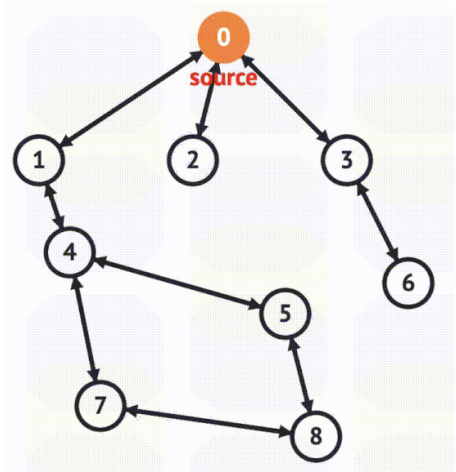



```
1   def BFS(V, E):
2       for i in range(len(V)):
3           V[i] = -1                # all vertices not visited
4       count = 0
5       for i in range(len(V)):   # for all possible sources
6           if (V[i] == -1):
7               Q = [i]                          # enqueue the source
8               V[i], count = count, count+1     # visit it
9               while (len(Q) != 0):             # for all enqueued
10                  for e in E:                      # search neighbors
11                      if (e[0] == Q[0]) and (V[e[1]] == -1):
12                          Q.append(e[1])                    # enqueue it
13                          V[e[1]], count = count, count+1   # visit it
14                      elif (e[1] == Q[0]) and (V[e[0]] == -1):
15                          Q.append(e[0])                    # enqueue it
16                          V[e[0]], count = count, count+1   # visit it
17                  Q.pop(0)                         # dequeue it
18
19  V = [0]*9
20  E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
21       [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
22  BFS(V, E)
23  print(V)
```

undirected graph

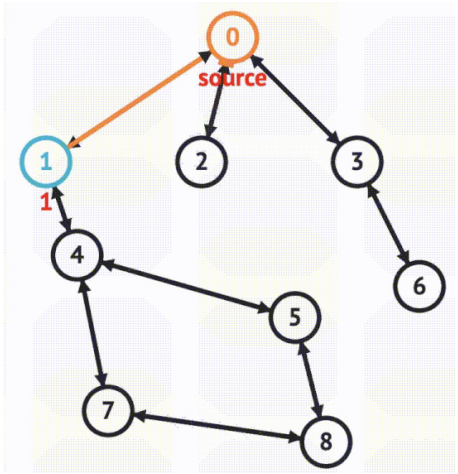
For directed graphs, edges (X, Y, W) != (Y, X, W).

Example 5 - Breadth-First Search in Graphs (BFS)



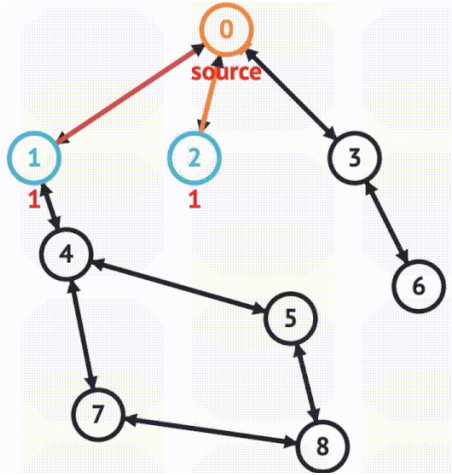

```
1   def BFS(V, E):
2       for i in range(len(V)):
3           V[i] = -1                # all vertices not visited
4       count = 0
5       for i in range(len(V)):   # for all possible sources
6           if (V[i] == -1):
7               Q = [i]                          # enqueue the source
8               V[i], count = count, count+1     # visit it
9               while (len(Q) != 0):             # for all enqueued
10                  for e in E:                      # search neighbors
11                      if (e[0] == Q[0]) and (V[e[1]] == -1):
12                          Q.append(e[1])                    # enqueue it
13                          V[e[1]], count = count, count+1   # visit it
14                      elif (e[1] == Q[0]) and (V[e[0]] == -1):
15                          Q.append(e[0])                    # enqueue it
16                          V[e[0]], count = count, count+1   # visit it
17                  Q.pop(0)                         # dequeue it
18
19  V = [0]*9
20  E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
21       [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
22  BFS(V, E)
23  print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)



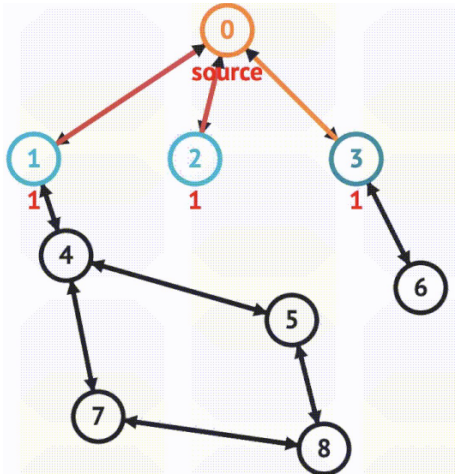

```python
def BFS(V, E):
    for i in range(len(V)):
        V[i] = -1                # all vertices not visited
    count = 0
    for i in range(len(V)):   # for all possible sources
        if (V[i] == -1):
            Q = [i]                          # enqueue the source
            V[i], count = count, count+1     # visit it
            while (len(Q) != 0):             # for all enqueued
                for e in E:                      # search neighbors
                    if (e[0] == Q[0]) and (V[e[1]] == -1):
                        Q.append(e[1])                      # enqueue it
                        V[e[1]], count = count, count+1     # visit it
                    elif (e[1] == Q[0]) and (V[e[0]] == -1):
                        Q.append(e[0])                      # enqueue it
                        V[e[0]], count = count, count+1     # visit it
                Q.pop(0)                         # dequeue it

V = [0]
E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
BFS(V, E)
print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)



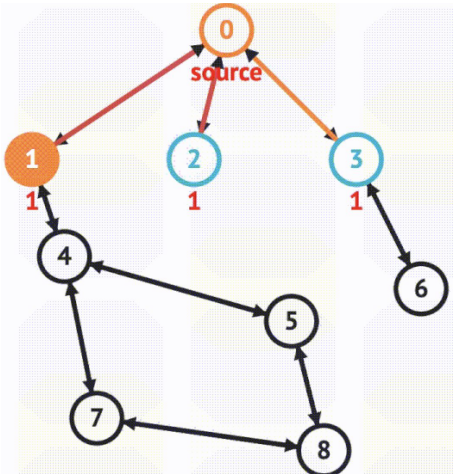

```python
def BFS(V, E):
    for i in range(len(V)):
        V[i] = -1                # all vertices not visited
    count = 0
    for i in range(len(V)):   # for all possible sources
        if (V[i] == -1):
            Q = [i]                          # enqueue the source
            V[i], count = count, count+1     # visit it
            while (len(Q) != 0):             # for all enqueued
                for e in E:                      # search neighbors
                    if (e[0] == Q[0]) and (V[e[1]] == -1):
                        Q.append(e[1])                      # enqueue it
                        V[e[1]], count = count, count+1     # visit it
                    elif (e[1] == Q[0]) and (V[e[0]] == -1):
                        Q.append(e[0])                      # enqueue it
                        V[e[0]], count = count, count+1     # visit it
                Q.pop(0)                         # dequeue it

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
BFS(V, E)
print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)



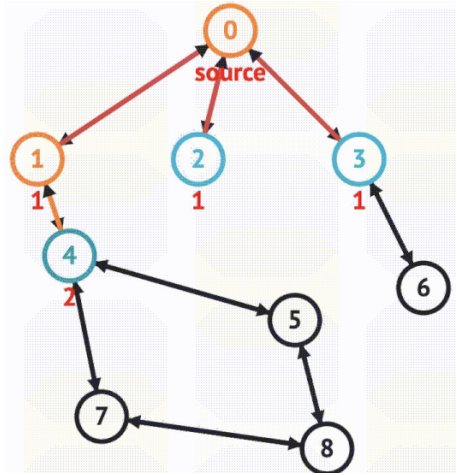

```python
def BFS(V, E):
    for i in range(len(V)):
        V[i] = -1                # all vertices not visited
    count = 0
    for i in range(len(V)):    # for all possible sources
        if (V[i] == -1):
            Q = [i]                          # enqueue the source
            V[i], count = count, count+1     # visit it
            while (len(Q) != 0):             # for all enqueued
                for e in E:                      # search neighbors
                    if (e[0] == Q[0]) and (V[e[1]] == -1):
                        Q.append(e[1])                    # enqueue it
                        V[e[1]], count = count, count+1   # visit it
                    elif (e[1] == Q[0]) and (V[e[0]] == -1):
                        Q.append(e[0])                    # enqueue it
                        V[e[0]], count = count, count+1   # visit it
                Q.pop(0)                         # dequeue it

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
BFS(V, E)
print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)



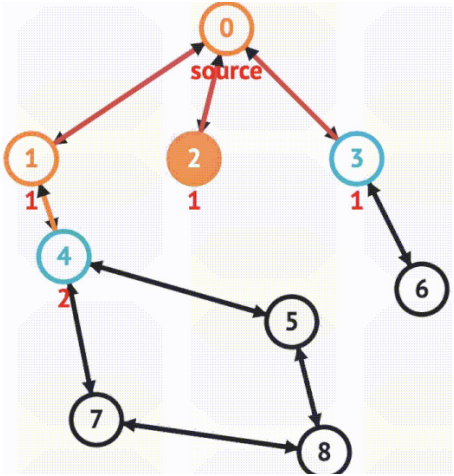

```python
def BFS(V, E):
    for i in range(len(V)):
        V[i] = -1                # all vertices not visited
    count = 0
    for i in range(len(V)):   # for all possible sources
        if (V[i] == -1):
            Q = [i]                          # enqueue the source
            V[i], count = count, count+1     # visit it
            while (len(Q) != 0):             # for all enqueued
                for e in E:                      # search neighbors
                    if (e[0] == Q[0]) and (V[e[1]] == -1):
                        Q.append(e[1])                    # enqueue it
                        V[e[1]], count = count, count+1   # visit it
                    elif (e[1] == Q[0]) and (V[e[0]] == -1):
                        Q.append(e[0])                    # enqueue it
                        V[e[0]], count = count, count+1   # visit it
                Q.pop(0)                         # dequeue it

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
BFS(V, E)
print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)



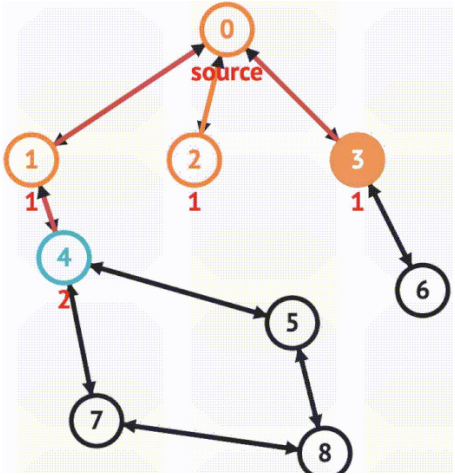

```python
def BFS(V, E):
    for i in range(len(V)):
        V[i] = -1                # all vertices not visited
    count = 0
    for i in range(len(V)):   # for all possible sources
        if (V[i] == -1):
            Q = [i]                          # enqueue the source
            V[i], count = count, count+1     # visit it
            while (len(Q) != 0):             # for all enqueued
                for e in E:                      # search neighbors
                    if (e[0] == Q[0]) and (V[e[1]] == -1):
                        Q.append(e[1])                      # enqueue it
                        V[e[1]], count = count, count+1     # visit it
                    elif (e[1] == Q[0]) and (V[e[0]] == -1):
                        Q.append(e[0])                      # enqueue it
                        V[e[0]], count = count, count+1     # visit it
                Q.pop(0)                                    # dequeue it

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
BFS(V, E)
print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)



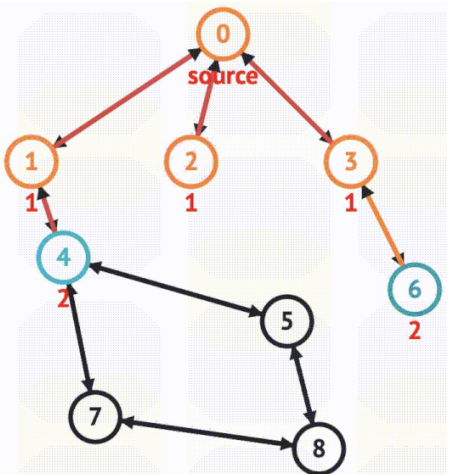

```python
def BFS(V, E):
    for i in range(len(V)):
        V[i] = -1                # all vertices not visited
    count = 0
    for i in range(len(V)):   # for all possible sources
        if (V[i] == -1):
            Q = [i]                          # enqueue the source
            V[i], count = count, count+1   # visit it
            while (len(Q) != 0):             # for all enqueued
                for e in E:                      # search neighbors
                    if (e[0] == Q[0]) and (V[e[1]] == -1):
                        Q.append(e[1])                    # enqueue it
                        V[e[1]], count = count, count+1   # visit it
                    elif (e[1] == Q[0]) and (V[e[0]] == -1):
                        Q.append(e[0])                    # enqueue it
                        V[e[0]], count = count, count+1   # visit it
                Q.pop(0)                         # dequeue it

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
BFS(V, E)
print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)



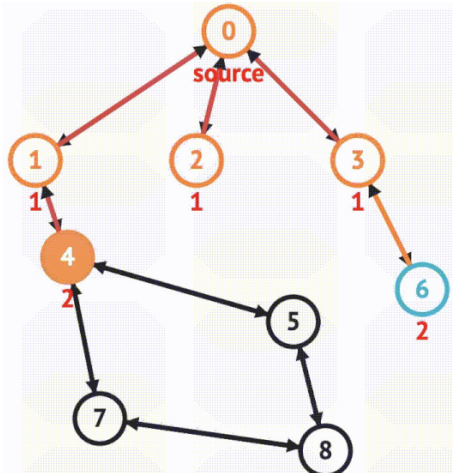

```python
def BFS(V, E):
    for i in range(len(V)):
        V[i] = -1                # all vertices not visited
    count = 0
    for i in range(len(V)):   # for all possible sources
        if (V[i] == -1):
            Q = [i]                          # enqueue the source
            V[i], count = count, count+1     # visit it
            while (len(Q) != 0):             # for all enqueued
                for e in E:                      # search neighbors
                    if (e[0] == Q[0]) and (V[e[1]] == -1):
                        Q.append(e[1])                    # enqueue it
                        V[e[1]], count = count, count+1   # visit it
                    elif (e[1] == Q[0]) and (V[e[0]] == -1):
                        Q.append(e[0])                    # enqueue it
                        V[e[0]], count = count, count+1   # visit it
                Q.pop(0)                         # dequeue it

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
BFS(V, E)
print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)



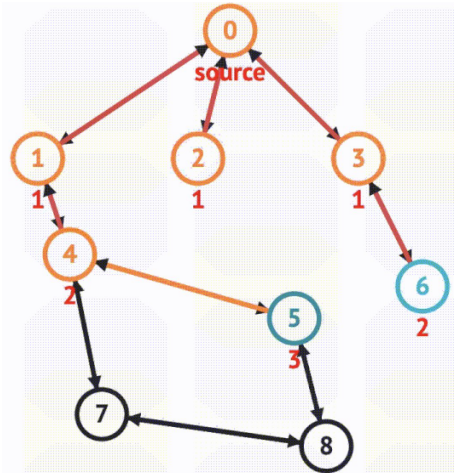

```python
def BFS(V, E):
    for i in range(len(V)):
        V[i] = -1                # all vertices not visited
    count = 0
    for i in range(len(V)):    # for all possible sources
        if (V[i] == -1):
            Q = [i]                          # enqueue the source
            V[i], count = count, count+1     # visit it
            while (len(Q) != 0):             # for all enqueued
                for e in E:                      # search neighbors
                    if (e[0] == Q[0]) and (V[e[1]] == -1):
                        Q.append(e[1])                      # enqueue it
                        V[e[1]], count = count, count+1     # visit it
                    elif (e[1] == Q[0]) and (V[e[0]] == -1):
                        Q.append(e[0])                      # enqueue it
                        V[e[0]], count = count, count+1     # visit it
                Q.pop(0)                             # dequeue it

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
BFS(V, E)
print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)



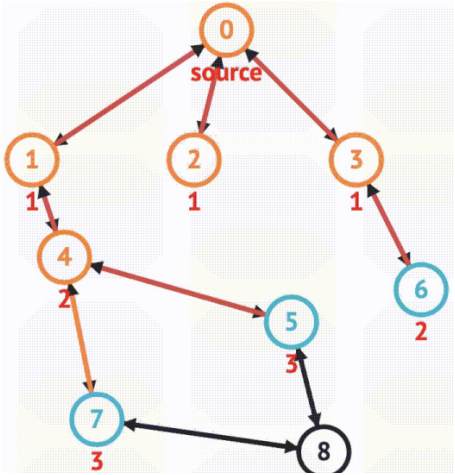

```python
def BFS(V, E):
    for i in range(len(V)):
        V[i] = -1                # all vertices not visited
    count = 0
    for i in range(len(V)):   # for all possible sources
        if (V[i] == -1):
            Q = [i]                          # enqueue the source
            V[i], count = count, count+1     # visit it
            while (len(Q) != 0):             # for all enqueued
                for e in E:                      # search neighbors
                    if (e[0] == Q[0]) and (V[e[1]] == -1):
                        Q.append(e[1])                    # enqueue it
                        V[e[1]], count = count, count+1   # visit it
                    elif (e[1] == Q[0]) and (V[e[0]] == -1):
                        Q.append(e[0])                    # enqueue it
                        V[e[0]], count = count, count+1   # visit it
                Q.pop(0)                         # dequeue it

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
BFS(V, E)
print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)



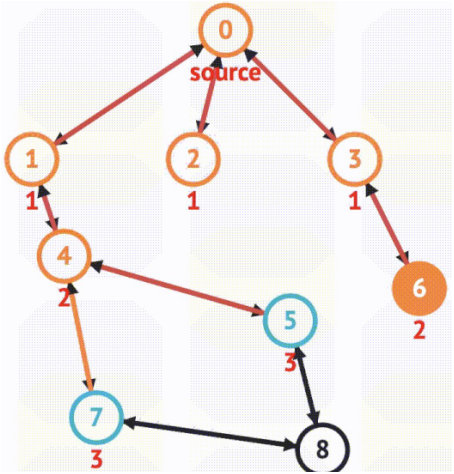

```python
def BFS(V, E):
    for i in range(len(V)):
        V[i] = -1                # all vertices not visited
    count = 0
    for i in range(len(V)):   # for all possible sources
        if (V[i] == -1):
            Q = [i]                          # enqueue the source
            V[i], count = count, count+1     # visit it
            while (len(Q) != 0):             # for all enqueued
                for e in E:                      # search neighbors
                    if (e[0] == Q[0]) and (V[e[1]] == -1):
                        Q.append(e[1])                    # enqueue it
                        V[e[1]], count = count, count+1   # visit it
                    elif (e[1] == Q[0]) and (V[e[0]] == -1):
                        Q.append(e[0])                    # enqueue it
                        V[e[0]], count = count, count+1   # visit it
                Q.pop(0)                         # dequeue it

V = [0 9
E = [[0 ,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
BFS(V, E)
print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)



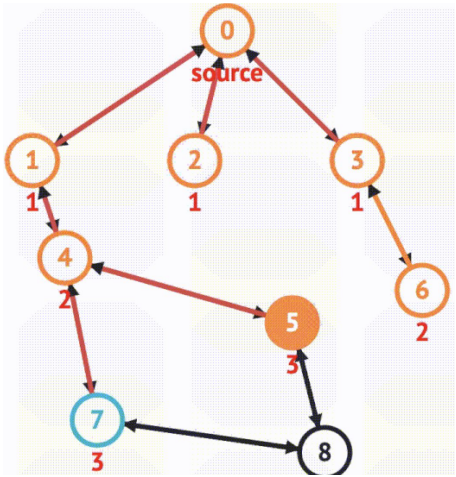

```python
def BFS(V, E):
    for i in range(len(V)):
        V[i] = -1                # all vertices not visited
    count = 0
    for i in range(len(V)):    # for all possible sources
        if (V[i] == -1):
            Q = [i]                          # enqueue the source
            V[i], count = count, count+1     # visit it
            while (len(Q) != 0):             # for all enqueued
                for e in E:                      # search neighbors
                    if (e[0] == Q[0]) and (V[e[1]] == -1):
                        Q.append(e[1])                    # enqueue it
                        V[e[1]], count = count, count+1   # visit it
                    elif (e[1] == Q[0]) and (V[e[0]] == -1):
                        Q.append(e[0])                    # enqueue it
                        V[e[0]], count = count, count+1   # visit it
                Q.pop(0)                         # dequeue it

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
BFS(V, E)
print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)



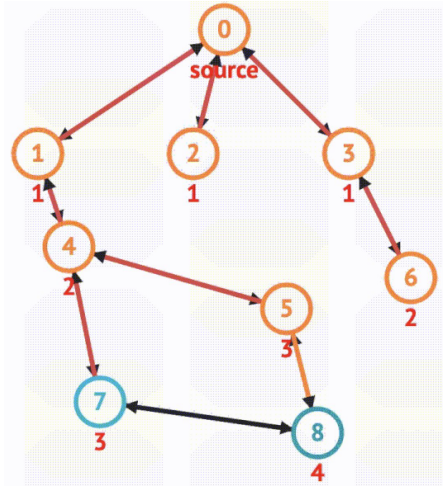

```python
def BFS(V, E):
    for i in range(len(V)):
        V[i] = -1                # all vertices not visited
    count = 0
    for i in range(len(V)):    # for all possible sources
        if (V[i] == -1):
            Q = [i]                          # enqueue the source
            V[i], count = count, count+1     # visit it
            while (len(Q) != 0):             # for all enqueued
                for e in E:                      # search neighbors
                    if (e[0] == Q[0]) and (V[e[1]] == -1):
                        Q.append(e[1])                    # enqueue it
                        V[e[1]], count = count, count+1   # visit it
                    elif (e[1] == Q[0]) and (V[e[0]] == -1):
                        Q.append(e[0])                    # enqueue it
                        V[e[0]], count = count, count+1   # visit it
                Q.pop(0)                         # dequeue it

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
BFS(V, E)
print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)



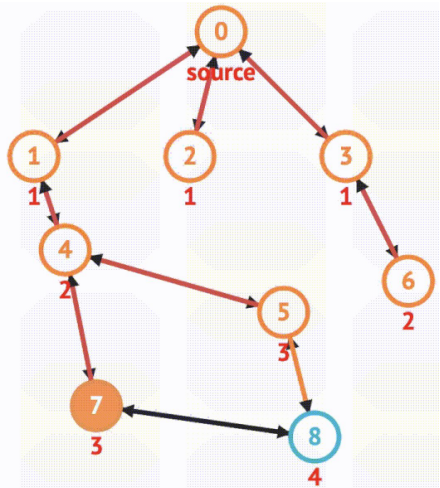

```python
def BFS(V, E):
    for i in range(len(V)):
        V[i] = -1                # all vertices not visited
    count = 0
    for i in range(len(V)):   # for all possible sources
        if (V[i] == -1):
            Q = [i]                          # enqueue the source
            V[i], count = count, count+1     # visit it
            while (len(Q) != 0):             # for all enqueued
                for e in E:                      # search neighbors
                    if (e[0] == Q[0]) and (V[e[1]] == -1):
                        Q.append(e[1])                    # enqueue it
                        V[e[1]], count = count, count+1   # visit it
                    elif (e[1] == Q[0]) and (V[e[0]] == -1):
                        Q.append(e[0])                    # enqueue it
                        V[e[0]], count = count, count+1   # visit it
                Q.pop(0)                         # dequeue it

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
BFS(V, E)
print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)



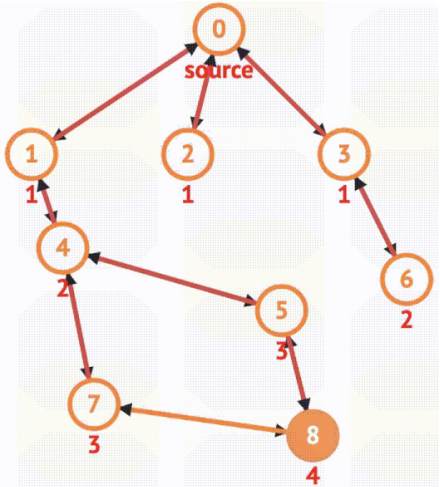

```
1   def BFS(V, E):
2       for i in range(len(V)):
3           V[i] = -1                # all vertices not visited
4       count = 0
5       for i in range(len(V)):   # for all possible sources
6           if (V[i] == -1):
7               Q = [i]                          # enqueue the source
8               V[i], count = count, count+1     # visit it
9               while (len(Q) != 0):             # for all enqueued
10                  for e in E:                      # search neighbors
11                      if (e[0] == Q[0]) and (V[e[1]] == -1):
12                          Q.append(e[1])                    # enqueue it
13                          V[e[1]], count = count, count+1   # visit it
14                      elif (e[1] == Q[0]) and (V[e[0]] == -1):
15                          Q.append(e[0])                    # enqueue it
16                          V[e[0]], count = count, count+1   # visit it
17                  Q.pop(0)                         # dequeue it
18
19  V = [0]*9
20  E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
21       [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
22  BFS(V, E)
23  print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)



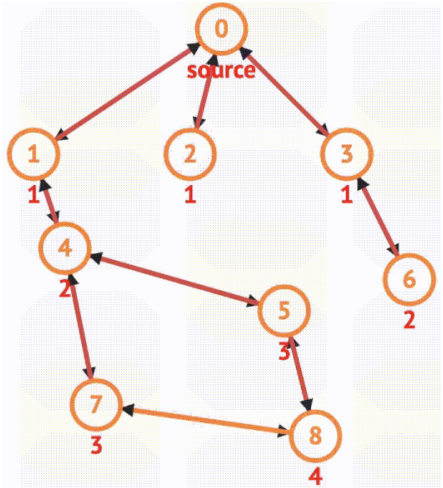

```
1   def BFS(V, E):
2       for i in range(len(V)):
3           V[i] = -1                # all vertices not visited
4       count = 0
5       for i in range(len(V)):   # for all possible sources
6           if (V[i] == -1):
7               Q = [i]                          # enqueue the source
8               V[i], count = count, count+1     # visit it
9               while (len(Q) != 0):             # for all enqueued
10                  for e in E:                      # search neighbors
11                      if (e[0] == Q[0]) and (V[e[1]] == -1):
12                          Q.append(e[1])                    # enqueue it
13                          V[e[1]], count = count, count+1   # visit it
14                      elif (e[1] == Q[0]) and (V[e[0]] == -1):
15                          Q.append(e[0])                    # enqueue it
16                          V[e[0]], count = count, count+1   # visit it
17                  Q.pop(0)                         # dequeue it
18
19  V = [0]*9
20  E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
21       [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
22  BFS(V, E)
23  print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)



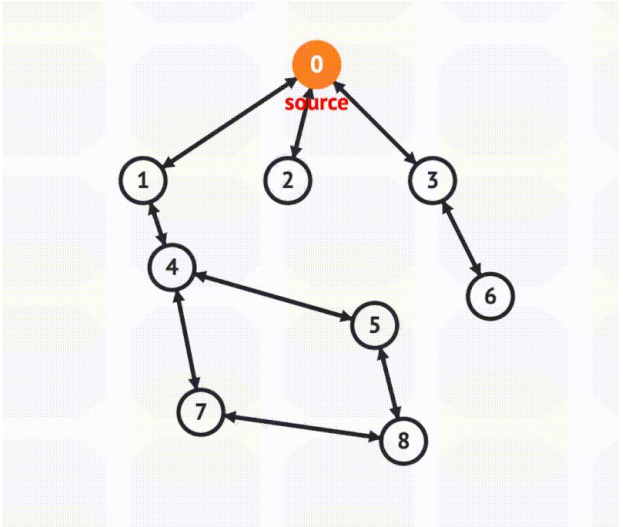

```
1   def BFS(V, E):
2       for i in range(len(V)):
3           V[i] = -1                # all vertices not visited
4       count = 0
5       for i in range(len(V)):   # for all possible sources
6           if (V[i] == -1):
7               Q = [i]                          # enqueue the source
8               V[i], count = count, count+1     # visit it
9               while (len(Q) != 0):             # for all enqueued
10                  for e in E:                      # search neighbors
11                      if (e[0] == Q[0]) and (V[e[1]] == -1):
12                          Q.append(e[1])                    # enqueue it
13                          V[e[1]], count = count, count+1   # visit it
14                      elif (e[1] == Q[0]) and (V[e[0]] == -1):
15                          Q.append(e[0])                    # enqueue it
16                          V[e[0]], count = count, count+1   # visit it
17                  Q.pop(0)                         # dequeue it
18
19  V = [0]*9
20  E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
21       [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
22  BFS(V, E)
23  print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)




```python
def BFS(V, E):
    for i in range(len(V)):
        V[i] = -1                # all vertices not visited
    count = 0
    for i in range(len(V)):    # for all possible sources
        if (V[i] == -1):
            Q = [i]                          # enqueue the source
            V[i], count = count, count+1     # visit it
            while (len(Q) != 0):             # for all enqueued
                for e in E:                      # search neighbors
                    if (e[0] == Q[0]) and (V[e[1]] == -1):
                        Q.append(e[1])                    # enqueue it
                        V[e[1]], count = count, count+1   # visit it
                    elif (e[1] == Q[0]) and (V[e[0]] == -1):
                        Q.append(e[0])                    # enqueue it
                        V[e[0]], count = count, count+1   # visit it
                Q.pop(0)                         # dequeue it

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
BFS(V, E)
print(V)
```

Example 5 - Breadth-First Search in Graphs (BFS)

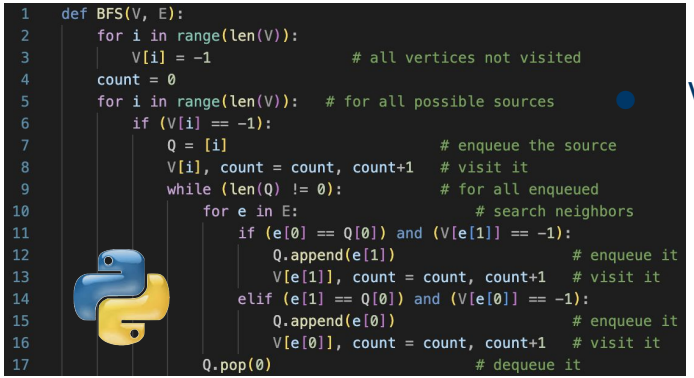



```
1   def BFS(V, E):
2       for i in range(len(V)):
3           V[i] = -1                # all vertices not visited
4       count = 0
5       for i in range(len(V)):   # for all possible sources
6           if (V[i] == -1):
7               Q = [i]                          # enqueue the source
8               V[i], count = count, count+1     # visit it
9               while (len(Q) != 0):             # for all enqueued
10                  for e in E:                      # search neighbors
11                      if (e[0] == Q[0]) and (V[e[1]] == -1):
12                          Q.append(e[1])                    # enqueue it
13                          V[e[1]], count = count, count+1   # visit it
14                      elif (e[1] == Q[0]) and (V[e[0]] == -1):
15                          Q.append(e[0])                    # enqueue it
16                          V[e[0]], count = count, count+1   # visit it
17                  Q.pop(0)                         # dequeue it
18
19  V = [0]*9
20  E = [[0,1,1], [0,2,1], [0,3,1], [1,4,1], [3,6,1],
21       [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
22  BFS(V, E)
23  print(V)
```

undirected graph

For directed graphs, edges (X, Y, W) != (Y, X, W).

Example 5 - Breadth-First Search in Graphs (BFS)



```
1  def BFS(V, E):
2      for i in range(len(V)):
3          V[i] = -1                # all vertices not visited
4      count = 0
5      for i in range(len(V)):   # for all possible sources
6          if (V[i] == -1):
7              Q = [i]                          # enqueue the source
8              V[i], count = count, count+1     # visit it
9              while (len(Q) != 0):             # for all enqueued
10                 for e in E:                      # search neighbors
11                     if (e[0] == Q[0]) and (V[e[1]] == -1):
12                         Q.append(e[1])                   # enqueue it
13                         V[e[1]], count = count, count+1  # visit it
14                     elif (e[1] == Q[0]) and (V[e[0]] == -1):
15                         Q.append(e[0])                   # enqueue it
16                         V[e[0]], count = count, count+1  # visit it
17             Q.pop(0)                         # dequeue it
```

- What is the worst case of it?
  - Actually, it always enqueues ***n*** vertices and to each, it checks the list of ***m*** edges looking for neighbors.

What is the Big Oh for it?

- Asymptotic Analysis:
  - Lines 2 to 8 and 17 are executed ***n*** times
  - Lines 10 to 16 are executed ***n * m*** times
  - $T(n) = c_1 * n + c_2 * n\ m$
  - $T(n) < c * n\ m$
- Thus, BFS algorithm has time complexity:
  - ***O(nm)***

Linear
Over
n *times* ***m***

Straightforward… not a bad choice!

This Week's tasks

- In-class Exercise E#3
- Coding Project P#3
- Quiz Q#3

Tasks

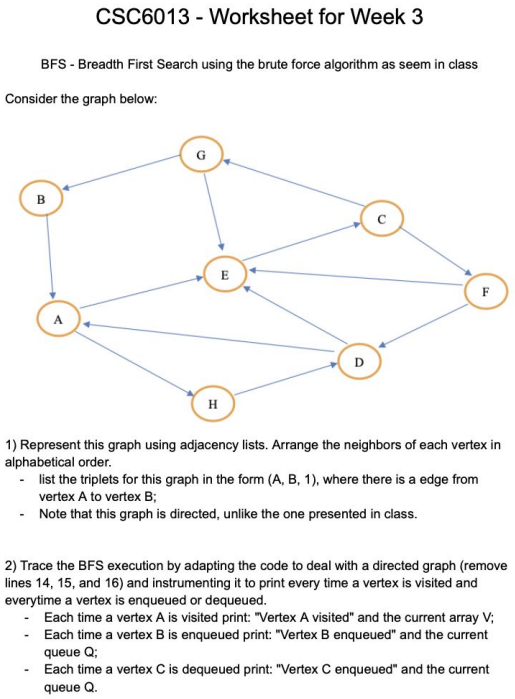
- Fill the worksheet as required.
- Adapt 2 Python programs:
  - selection sort;
  - bubble sort.
- Quiz #3 about this week topics.

In-class Exercise - E#3

Download the pdf (link here also) and perform the following tasks:

- (1) create the adjacency list for the graph presented;
- (2) Trace the algorithm printing every time a vertex is visited, enqueued, or dequeued.

You have to submit a ***.pdf*** file with your answers.



CSC6013 - Worksheet for Week 3

BFS - Breadth First Search using the brute force algorithm as seem in class

Consider the graph below:

1) Represent this graph using adjacency lists. Arrange the neighbors of each vertex in alphabetical order.
   - list the triplets for this graph in the form (A, B, 1), where there is a edge from vertex A to vertex B;
   - Note that this graph is directed, unlike the one presented in class.

2) Trace the BFS execution by adapting the code to deal with a directed graph (remove lines 14, 15, and 16) and instrumenting it to print every time a vertex is visited and everytime a vertex is enqueued or dequeued.
   - Each time a vertex A is visited print: "Vertex A visited" and the current array V;
   - Each time a vertex B is enqueued print: "Vertex B enqueued" and the current queue Q;
   - Each time a vertex C is dequeued print: "Vertex C enqueued" and the current queue Q.

Feel free to type it or hand write it, but you have to submit a single ***.pdf*** with your answers.

MERRIMACK COLLEGE

This task counts towards the In-class Exercises grade and the deadline is This Friday.

Third Coding Project - P#3

1. **Adapt the Selection Sort algorithm** saw in class to sort the elements being the largest ones selected to go to the last positions first, instead of the version presented in class, where the smallest ones where selected to go to the first positions. Trace your algorithm execution printing:
    a. at each iteration of the outer loop count the number of times two array elements are compared and the number of times two array elements were swapped, plus the current status of the array.
    b. test your algorithm for the arrays:
        i. A1 = [63, 44, 17, 77, 20, 6, 99, 84, 52, 39]
        ii. A2 = [84, 52, 39, 6, 20, 17, 77, 99, 63, 44]
        iii. A3 = [99, 84, 77, 63, 52, 44, 39, 20, 17, 6]

To both programs you have to submit the code (***.py*** file) of your adapted algorithm and a ***.pdf*** with the test cases outputs.

Two algorithms to adapt, this is the first.

Third Coding Project - P#3

2. **Adapt the Bubble Sort algorithm** saw in class to stop the outer loop if no swap was made during the last iteration (thus, the array is already sorted). Trace your algorithm execution printing:
    a. at each iteration of the outer loop count the number of times two array elements are compared and the number of times two array elements were swapped, plus the current status of the array.
    b. test your algorithm for the arrays:
        i. A4 = [44, 63, 77, 17, 20, 99, 84, 6, 39, 52]
        ii. A5 = [52, 84, 6, 39, 20, 77, 17, 99, 44, 63]
        iii. A6 = [6, 17, 20, 39, 44, 52, 63, 77, 84, 99]

Your task:

Go to Canvas, and submit your ***.py*** and ***.pdf*** files within the deadline.

This assignment counts towards the Projects grade and the deadline is Next Monday.

Third Quiz - Q#3

- The third quiz in this course covers the topics of Week 3;
- The quiz will be available this Friday, and it is composed by 10 questions;
- The quiz should be taken on Canvas (Module 3), and it is not a timed quiz:
  - You can take as long as you want to answer it (a quiz taken in less than one hour is usually a too short time);
- The quiz is open book, open notes, and you can even use any language Interpreter to answer it;
- Yet, the quiz is evaluated and you are allowed to submit it only once.

Your task:

- Go to Canvas, answer the quiz and submit it within the deadline.

This quiz counts towards the Quizzes grade and the deadline is Next Monday.

Welcome to CSC 6013

- Do In-class Exercise E#3 until Friday;
- Do Quiz Q#3 (available Friday) until next Monday;
- Do Coding Project P#3 until next Monday.

Next Week - Recursive Algorithms

MERRIMACK COLLEGE