Python IO

```
def read():
     try:
          result = input.rstrip('\n')
     except EOFError:
          result = None
     finally:
          return result
```

DFS

```
def dfs(graph, start):
    visited, stack = set(), [start]
    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            visited.add(vertex)
            stack.extend(graph[vertex] - visited)
    return visited

dfs(graph, 'C') # {'E', 'D', 'F', 'A', 'C', 'B'}
---------------------------------------------------------

def dfs_paths(graph, start, goal):
    stack = [(start, [start])]
    while stack:
        (vertex, path) = stack.pop()
        for next in graph[vertex] - set(path):
            if next == goal:
                yield path + [next]
            else:
                stack.append((next, path + [next]))

list(dfs_paths(graph, 'C', 'F')) # [['C', 'F'], ['C', 'A', 'B', 'E',
'F']]
```

BFS

```
def bfs(graph, start):
    visited, queue = set(), [start]
    while queue:
        vertex = queue.pop(0)
        if vertex not in visited:
            visited.add(vertex)
            queue.extend(graph[vertex] - visited)
    return visited

bfs(graph, 'A') # {'B', 'C', 'A', 'F', 'D', 'E'}
-----------------------------------------------------

def bfs_paths(graph, start, goal):
    queue = [(start, [start])]
    while queue:
        (vertex, path) = queue.pop(0)
        for next in graph[vertex] - set(path):
            if next == goal:
                yield path + [next]
            else:
                queue.append((next, path + [next]))

list(bfs_paths(graph, 'A', 'F')) # [['A', 'C', 'F'], ['A', 'B', 'E',
'F']]
-----------------------------------------------------

def shortest_path(graph, start, goal):
    try:
        return next(bfs_paths(graph, start, goal))
    except StopIteration:
        return None

shortest_path(graph, 'A', 'F') # ['A', 'C', 'F']
```

Quicksort

```
def partition(arr,low,high):
    i = ( low-1 )
    pivot = arr[high]

    for j in range(low , high):

        if  arr[j] <= pivot:
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]

    arr[i+1],arr[high] = arr[high],arr[i+1]
    return ( i+1 )

def quickSort(arr,low,high):
    if low < high:
        pi = partition(arr,low,high)

        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)

# Driver code to test above
arr = [10, 7, 8, 9, 1, 5]
n = len(arr)
quickSort(arr,0,n-1)
print ("Sorted array is:")
for i in range(n):
    print ("%d " %arr[i])
```

Sort in Python

```
Sort dictionary:
      sorted(a.items(), key = lambda x:x[0], reverse=False)- by key
      sorted(a.items(), key = lambda x:x[1], reverse=True) - by value
```

Dijkstra

```
class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def printSolution(self, dist):
        print "Vertex tDistance from Source"
        for node in range(self.V):
            print node, "t", dist[node]

    def minDistance(self, dist, sptSet):

        min = sys.maxint

        for v in range(self.V):
            if dist[v] < min and sptSet[v] == False:
                min = dist[v]
                min_index = v

        return min_index

    def dijkstra(self, src):

        dist = [sys.maxint] * self.V
        dist[src] = 0
        sptSet = [False] * self.V

        for cout in range(self.V):
            u = self.minDistance(dist, sptSet)
            sptSet[u] = True

            for v in range(self.V):
                if self.graph[u][v] > 0 and sptSet[v] == False and \
                dist[v] > dist[u] + self.graph[u][v]:
                        dist[v] = dist[u] + self.graph[u][v]

        self.printSolution(dist)
```

```
# Driver program
g = Graph(9)
g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],
           [4, 0, 8, 0, 0, 0, 0, 11, 0],
           [0, 8, 0, 7, 0, 4, 0, 0, 2],
           [0, 0, 7, 0, 9, 14, 0, 0, 0],
           [0, 0, 0, 9, 0, 10, 0, 0, 0],
           [0, 0, 4, 14, 10, 0, 2, 0, 0],
           [0, 0, 0, 0, 0, 2, 0, 1, 6],
           [8, 11, 0, 0, 0, 0, 1, 0, 7],
           [0, 0, 2, 0, 0, 0, 6, 7, 0]
          ];

g.dijkstra(0);
```

Output

```
Vertex tDistance from Source
0 t 0
1 t 4
2 t 12
3 t 19
4 t 21
5 t 11
6 t 9
7 t 8
8 t 14
```

Prim

```
class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def printMST(self, parent):
        print "Edge \tWeight"
        for i in range(1, self.V):
            print parent[i], "-", i, "\t", self.graph[i][ parent[i] ]

    def minKey(self, key, mstSet):
        min = sys.maxint

        for v in range(self.V):
            if key[v] < min and mstSet[v] == False:
                min = key[v]
                min_index = v

        return min_index

    def primMST(self):

        key = [sys.maxint] * self.V
        parent = [None] * self.V # Array to store constructed MST

        key[0] = 0
        mstSet = [False] * self.V

        parent[0] = -1 # First node is always the root of

        for cout in range(self.V):
            u = self.minKey(key, mstSet)
            mstSet[u] = True

          for v in range(self.V):
                if self.graph[u][v] > 0 and mstSet[v] == False and
key[v] > self.graph[u][v]:
                        key[v] = self.graph[u][v]
                        parent[v] = u
```

```
        self.printMST(parent)

# Driver program
g = Graph(5)
g.graph = [ [0, 2, 0, 6, 0],
            [2, 0, 3, 8, 5],
            [0, 3, 0, 0, 7],
            [6, 8, 0, 0, 9],
            [0, 5, 7, 9, 0]]

g.primMST();
```

Output

```
Edge   Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5
```

Dynamic Programming

https://www.geeksforgeeks.org/dynamic-programming/

LIS

```
def lis(arr):
    n = len(arr)

    lis = [1]*n

    for i in range (1 , n):
        for j in range(0 , i):
            if arr[i] > arr[j] and lis[i]< lis[j] + 1 :
                lis[i] = lis[j]+1

    maximum = 0

    for i in range(n):
        maximum = max(maximum , lis[i])

    return maximum
```

LCS

```
def lcs(X , Y):
    m = len(X)
    n = len(Y)

    L = [[None]*(n+1) for i in xrange(m+1)]

    for i in range(m+1):
        for j in range(n+1):
            if i == 0 or j == 0 :
                L[i][j] = 0
            elif X[i-1] == Y[j-1]:
                L[i][j] = L[i-1][j-1]+1
            else:
                L[i][j] = max(L[i-1][j] , L[i][j-1])

    return L[m][n]
```

Solutions

https://hc3.seas.harvard.edu/hc3/bospre/2018/private/AbluJsabrd/

https://hc3.seas.harvard.edu/hc3/bospre/2017/private/sight38seer92/

https://hc3.seas.harvard.edu/hc3/bospre/2016/private/BogglonimetricWhatarian/