

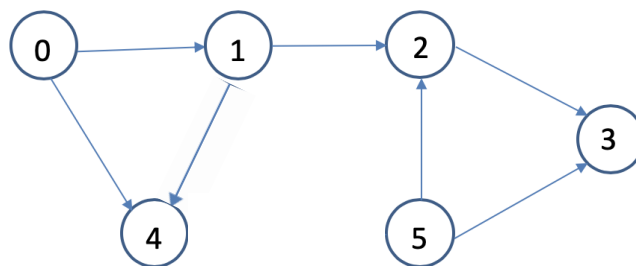
Data Structures - Fall 2019

Exam 3

Section 1 – Tracing (25 points)

s1_tracing.py

Problem 1 (5 points): Trace the execution of the topological sort algorithm using the graph below as input. Break ties by always enqueueing the vertex with the **smallest index first**. The result of topological sort is an array. After tracing, return the contents of this array in `get_problem_1_answer()`



```
def topological_sort(graph):

    all_in_degrees = compute_indegree_every_vertex(graph)
    sort_result = []

    q = Queue()

    for i in range(len(all_in_degrees)):
        if all_in_degrees[i] == 0:
            q.put(i) # enqueue

    while not q.is_empty():
        u = q.put() # dequeue

        sort_result.append(u)

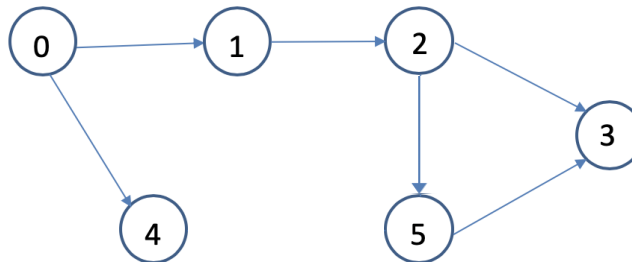
        for adj_vertex in graph.get_adj_vertices(u):
            all_in_degrees[adj_vertex] -= 1

            if all_in_degrees[adj_vertex] == 0:
                q.put(adj_vertex)

    if len(sort_result) != graph.num_vertices: # Cycle found
        return None

    return sort_result
```

Problem 2 (5 points): Trace the execution of breadth-first search using the graph below as input. Start from vertex 0 and break ties by always pushing/enqueuing the vertex with the smallest index first. After tracing, return the contents of the visited_array (the array that stores the order in which the vertices are visited) in get_problem_2_answer()



```

def breath_first_search(graph, start_node):
    visited_order = []
    visited = [False] * graph.num_vertices()
    path = [-1] * graph.num_vertices()

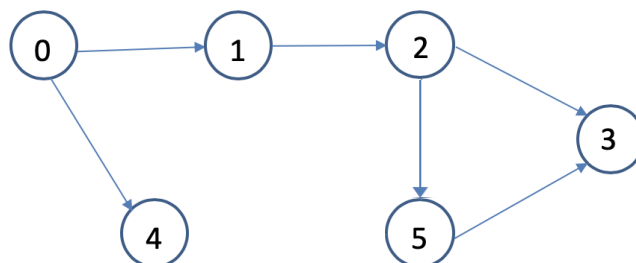
    q = Queue()
    q.put(start_node)
    visited[start_node] = True
    visited_order.append(start_node)

    while not q.empty():
        u = q.get()
        for adj_vertex in graph.vertices_reachable_from(u):
            if not visited[adj_vertex]:
                visited[adj_vertex] = True
                visited_order.append(adj_vertex)
                path[adj_vertex] = u
                q.put(adj_vertex)

    return path, visited_order

```

Problem 3 (5 points): Trace the execution of depth-first search using the graph below as input. Start from vertex 0 and break ties by always pushing/enqueuing the vertex with the smallest index first. After tracing, return the contents of the visited_array (the array that stores the order in which the vertices are visited) in get_problem_3_answer()



```

def depth_first_search(graph, start_node):
    visited_order = []
    visited = [False] * graph.num_vertices()
    path = [-1] * graph.num_vertices()

    stack = [] # A list can be used as a stack

    stack.append(start_node)

    while len(stack) > 0:
        u = stack.pop()

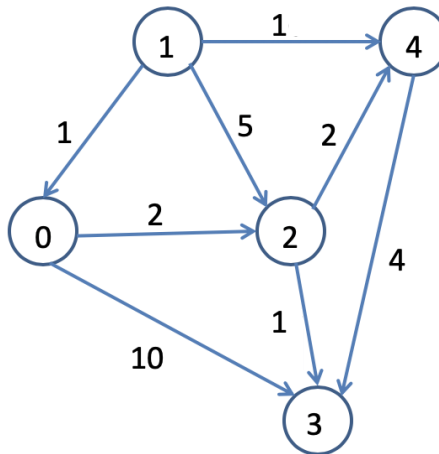
        if not visited[u]:
            visited[u] = True
            visited_order.append(u)

            for adj_vertex in graph.vertices_reachable_from(u):
                if not visited[adj_vertex]:
                    path[adj_vertex] = u
                    stack.append(adj_vertex)

    return path, visited_order

```

Problem 4 (5 points): Trace the execution of Dijkstra's algorithm to determine the shortest path from vertex 1 to every other vertex in the graph. After tracing, return the contents of the *dist* and *path* arrays in `get_problem_4_answer()`



```

def dikstra(graph, src):
    known = [False] * graph.num_vertices
    path = [-1] * graph.num_vertices
    dist = [math.inf] * graph.num_vertices

    dist[src] = 0
    known_vertices = 0

    while known_vertices < graph.num_vertices:
        u = get_unknown_vertex_smallest_dist(graph, known, dist)

```

```
known[u] = True
```

```
known_vertices += 1
```

```
for v in get_unknown_neighbors(u, graph, known): # unknown neighbors of u
    if dist[v] > dist[u] + graph.edge_weight(u, v):
        path[v] = u
        dist[v] = dist[u] + graph.edge_weight(u, v)
```

```
return path, dist
```

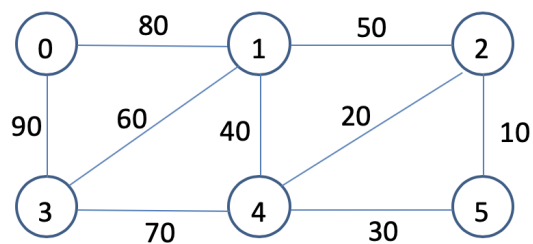
Problem 5 (5 points): Apply the edit distance algorithm based on dynamic programming to determine the minimum sequence of elementary character operations needed to convert the string "HAP" to the string "APP". Return the contents of the 2D array in `get_problem_5_answer()`

	"	H	A	P
"				
A				
P				
P				

Section 2 - Multiple Choice (31 points)

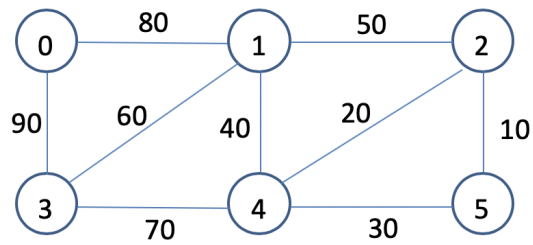
s2_multiple_choice.py

Problem 6 (5 points): What is the **last** edge that Kruskal's algorithm would add to the graph's minimum spanning tree?



- Option 0: Edge connecting 0 and 1
- Option 1: Edge connecting 5 and 2
- Option 2: Edge connecting 0 and 3
- Option 3: Edge connecting 3 and 1
- Option 4: None of the above

Problem 7 (5 points): What is the **last** edge that Prim's algorithm would add to the minimum spanning tree if **vertex 0** is used as the start node?



- Option 0: Edge connecting 0 and 1
- Option 1: Edge connecting 5 and 2
- Option 2: Edge connecting 0 and 3
- Option 3: Edge connecting 3 and 1
- Option 4: None of the above

Problem 8 (3 points): What data structure does breadth-first search use to traverse a graph:

- Option 0: Min-Heap
- Option 1: Queue
- Option 2: Stack
- Option 3: Disjoint Set Forest
- Option 4: None of the above

Problem 9 (3 points): What data structure does depth-first search use to traverse a graph:

- Option 0: Min-Heap
- Option 1: Queue
- Option 2: Stack
- Option 3: Disjoint Set Forest
- Option 4: None of the above

Problem 10 (3 points): Which of the following algorithms uses divide and conquer?

- Option 0: Linear Search
- Option 1: Adding 1 to every element in an array using a loop
- Option 2: Merge Sort
- Option 3: All of the above
- Option 4: None of the above

Problem 11 (3 points): How does the randomized version of quick sort work?

- Option 0: It calls itself recursively at random locations in the array
- Option 1: It uses randomly generated numbers to traverse the unsorted array

- Option 2: It randomly selects a pivot from the array
- Option 3: It randomly swaps elements in the array before sorting it
- Option 4: None of the above

Problem 12 (3 points): Which of the following algorithm(s) use dynamic programming:

- Option 0: Bubble Sort
- Option 1: Merge Sort
- Option 2: Finding the maximum value in an array
- Option 3: Edit Distance
- Option 4: Traversing a dictionary
- Option 5: Options 0 and 1
- Option 6: Options 2 and 3
- Option 7: None of the above

Problem 13 (3 points): Which of the following statements is true?

- Option 0: All greedy algorithms find the optimal solution to the problem they attempt to solve.
- Option 1: Greedy algorithms never find the optimal solution to the problem they attempt to solve.
- Option 2: Some greedy algorithms find the optimal solution to the problem they attempt to solve.
- Option 3: None of the above

Problem 14 (3 points): Which of the following statements is false?

- Option 0: All NP-Complete problems are also NP problems
- Option 1: All NP-Complete problems are also NP-Hard problems
- Option 2: A problem x is said to be NP-Hard if all problems from the class NP can be reduced to it.
- Option 3: It has been proven that all P problems are also NP
- Option 4: It has been proven that all NP problems are also P
- Option 5: We have algorithms to solve NP-hard problems

Section 3 - Graphs - Adjacency List Representation (28 points)

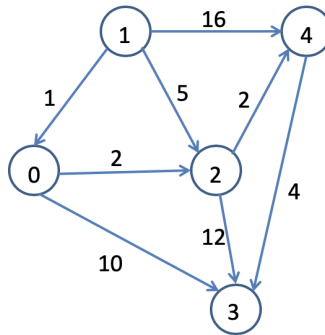
s3_graph_al.py

Problem 15 (7 points): Complete the implementation of the *num_edges* method. This method should return the number of edges in the graph.

Problem 16 (7 points): Complete the implementation of the *compute_in_degree* method. The *in degree* of a vertex v in a directed graph $G = (V, E)$ is the number of edges that point to v . The method *compute_in_degree* should receive a vertex v and return the *in degree* of v .

Problem 17 (7 points): Complete the implementation of the *num_isolated_vertices* method. An isolated vertex is a vertex with no incoming or outgoing edges. The method *num_isolated_vertices* returns the number of isolated vertices in the graph.

Problem 18 (7 points): Complete the implementation of the *highest_in_degree_vertex* method. This method returns the vertex that has the highest in degree. For example, your method should return **3** (integer) if the following graph is passed as input. Assume there are no ties.



Section 4 - Graphs - Adjacency Matrix Representation (28 points)

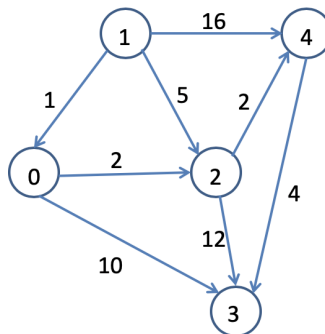
s4_graph_am.py

Problem 19 (7 points): Complete the implementation of the *num_edges* method. This method should return the number of edges in the graph.

Problem 20 (7 points): Complete the implementation of the *compute_in_degree* method. The *in degree* of a vertex v in a directed graph $G = (V, E)$ is the number of edges that point to v . The method *compute_in_degree* should receive a vertex v and return the *in degree* of v .

Problem 21 (7 points): Complete the implementation of the *num_isolated_vertices* method. An isolated vertex is a vertex with no ingoing or outgoing edges. The method *num_isolated_vertices* returns the number of isolated vertices in the graph.

Problem 22 (7 points): Complete the implementation of the *highest_in_degree_vertex* method. This method returns the vertex that has the highest in degree. For example, your method should return 3 (integer) if the following graph is passed as input. Assume there are no ties.



Section 5 – Disjoint Set Forest (28 points)

s5_dsf.py

Problem 23 (5 points): Complete the implementation of the *get_num_sets* method. This method is to return the number of sets in the disjoint set forest

Problem 24 (9 points): Complete the implementation of the *group_singletons* method. This method groups together all singletons in a set. That is, your method should identify all sets of size 1, and join them together in a single set. It does not matter what singleton ends up being the root. Modify the contents of self.dsf.

Problem 25 (7 points): Complete the implementation of the *is_compressed* method. This method should return True if and only if all the paths from a leaf to a root have length at most one. Assume the standard union operation was used (no path compression, no union by height).

Problem 26 (7 points): Complete the implementation of the *create_dsf* method. This method receives integers n and k and builds and returns an array (that represents a disjoint set forest) of size n. This disjoint set forest should encode 1 set. The root of this set should be k, and all other elements need to point to it. That is, all other members need to point to k. For example, if n = 5, and k = 2, your method should return the following: [2, 2, -1, 2, 2]

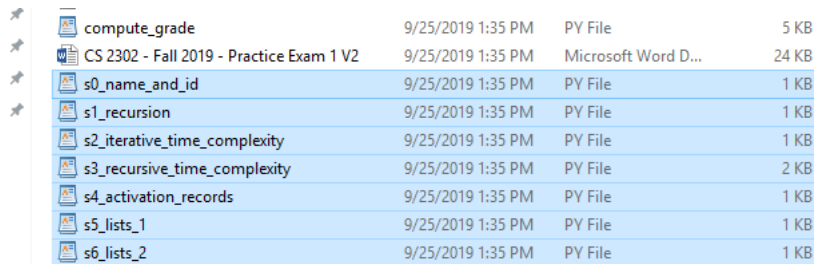
There are 140 points on this exam. This exam is graded out of 100 points.

----- HOW TO UPLOAD YOUR EXAM-----

Make sure compute_grade runs! If you have an infinite loop or if the code fails to compute your grade, you will automatically get a 0

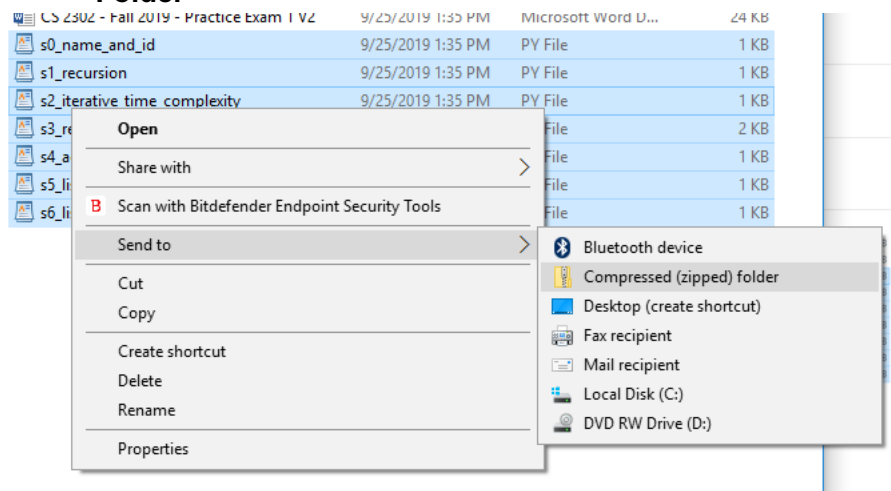
Windows 10:

1. Select the 6 section files (from s0 to s5). The files on the following image do not match the ones you downloaded, but the image is there to illustrate how you would do this.

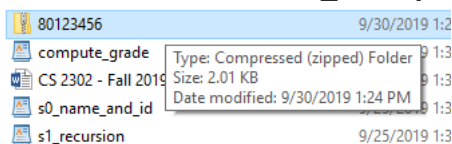


compute_grade	9/25/2019 1:35 PM	PY File	5 KB
CS 2302 - Fall 2019 - Practice Exam 1 V2	9/25/2019 1:35 PM	Microsoft Word D...	24 KB
s0_name_and_id	9/25/2019 1:35 PM	PY File	1 KB
s1_recursion	9/25/2019 1:35 PM	PY File	1 KB
s2_iterative_time_complexity	9/25/2019 1:35 PM	PY File	1 KB
s3_recursive_time_complexity	9/25/2019 1:35 PM	PY File	2 KB
s4_activation_records	9/25/2019 1:35 PM	PY File	1 KB
s5_lists_1	9/25/2019 1:35 PM	PY File	1 KB
s6_lists_2	9/25/2019 1:35 PM	PY File	1 KB

2. Right click on any of the selected files and do “Send to -> Compressed (zipped) Folder”



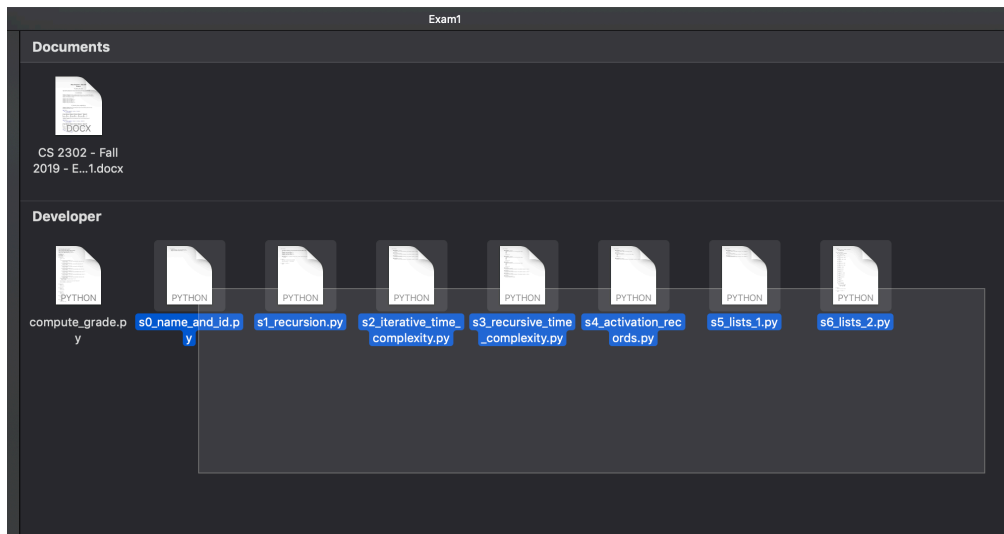
3. A zip file will be created. Use your UTEP ID to rename this file. The final name must be: <UTEP_ID>.zip



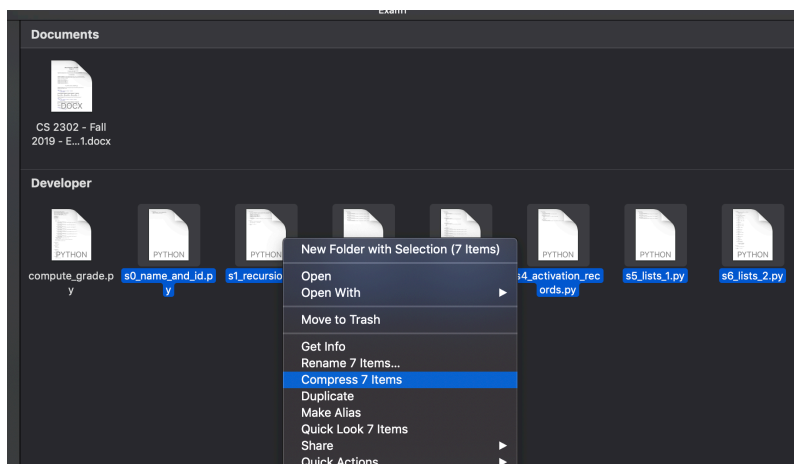
4. Upload the zip file (Blackboard).

macOS:

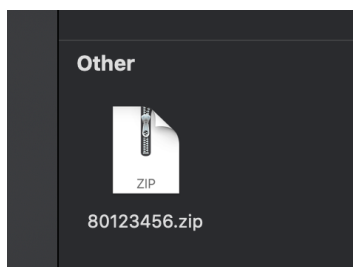
1. Select the 6 section files (from s0 to s6). The files on the following image do not match the ones you downloaded, but the image is there to illustrate how you would do this.



2. Right click on any of the selected files and click “Compress 6 items”



3. An “Archive.zip” file will be created. Use your UTEP ID to rename this file. The final name must be: <UTEP_ID>.zip



4. Upload the zip file (Blackboard)