**CS251 Homework 4: Graphs and Strings**

**Out: March 30, 2018 @ 9:00 PM**

**Due: April 6, 2018 @ 11:59 PM**

**Submission Instructions: Please submit a typeset PDF on blackboard. For multiple choice questions, you must provide an explanation along with your answer. Answers without explanations will receive 0 points, even if correct.**

1. For each of the following scenarios, what graph representation should you choose, and why? Assume no self-loops or parallel edges.
   1. An undirected graph G with n vertices and m edges, where n >> m, which must frequently add and remove vertices.

Linked List

In this case, there will be a lot of empty spaces with application of adjacency matrix so linked list is more efficient to save information about edges.

* 1. A very large, dense, weighted undirected graph that doesn't change, for which you must compute all-pairs shortest paths.

Adjacency matrix.

With a very dense graph, space wasting will not be severe and adjacency matrix has its advantage in finding edge in O(1) times which linked list need to go through one by one.

* 1. A weighted undirected graph for which you wish to generate a minimum spanning tree using Prim-Jarnik's algorithm.

linked list

MST with linked list application only takes O(ElogV) time complexity while one with adjacency matrix will take O(V^2) time complexity

1. Match the algorithms to the goals below. Note that this can be a many-to-many mapping. (put the goal number next to the algorithm)

**Algorithms:**

DFS …………………………………………….. 1,4,7

BFS ………………………………………………. 1,4,6

Floyd-Warshall ………………………………….. 2,5

Prim-Jarnik ………………………………………3

Bellman-Ford …………………………………… 6

Dijkstra ………………………………………….. 6

Kruskal …………………………………………... 3

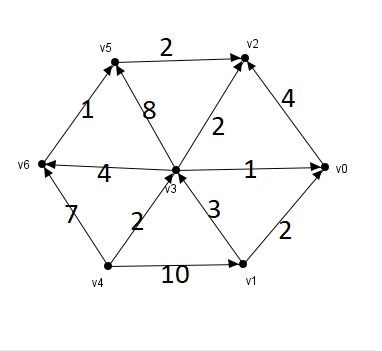
**Goals:**

1. Graph traversal
2. Transitive closure
3. Minimum spanning tree
4. Find cycles
5. All-pairs shortest paths
6. Single-source shortest paths
7. Get connected components

**A**

1. Assume a directed graph G with n vertices and m edges, in which the weight of all edges are equal. Which of the following time complexities is needed to compute the shortest path from a vertex V to every other vertex in G?
   1. **O(nm) Dijkstra cant handle negative weight which may happen, so we use Bellman, with Bellman-ford algorithm, we can achieve O(nm)**
   2. O(n + m)
   3. O(n^2)
   4. O(n log m)
   5. None of the above

The following weighted graph is used in the questions below.



**C**

1. What is the weight of the shortest path from v​4 to v​ ​5? Show your work.​
2. 2
3. 4
4. **7 the shortest path will be v4->v3->v6->v5**
5. 8

**B**

1. If Dijkstra is started from v​4, which vertex would be the last one to be visited? Show​  your work.
   1. v​0
   2. **v​1**
   3. v​2
   4. v​5

visit V3, which is 2, set v3 to 2, set v6 to 6, v0 to 3, v5 to 10, v2 to 4

visit v0, which is 3

visit v2, which is 4

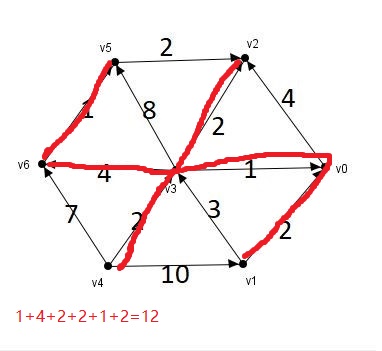
visit v6, which is 6, set v5 to 7

visit v5, which is 7

visit v1, which is 10

**D**

1. If the directions of the edges are removed, what would be the weight of the minimum spanning tree? Show your work.
   1. 8
   2. 10
   3. 11
   4. **12**



**B**

1. How many topological orders does the graph have? Show your work.
   1. 1
   2. **3**
   3. 4
   4. 6

1st: V4,v1,v3,v6,v5,v0,v2

2nd: v4,v1,v3,v0,v6,v5,v2

3rd: v4,v1,v3,v6,v0,v5,v2

1. The brute-force pattern matching algorithm on the slides runs until either
   * 1. A match is found, or
     2. All placements of the pattern have been tried

Write the pseudocode for an algorithm such that it reports all occurrences of a pattern P with the number of shifts needed to achieve that.

1. Show the comparisons the naive string matcher makes for the pattern P = 0001 in the text T = 000010001010001

0001

000010001010001

Compare P[0] & T[0] && P[1] & T[1] && P[2] & T[2] && P[3] & T[3]

Right shift 1 position & compare P[0] & T[1] && P[1] & T[2] && P[2] & T[3] && P[3] & T[4]

Right shift 2 positions & compare P[0] & T[2] && P[1] & T[3] && P[2] & T[4]

Right shift 3 positions & comparison done on P[0] & T[3] && P[1] & T[4]

Right shift 3 positions & comparison stops on P[0] & T[4]

Right shift 5 positions & comparison done on P[0] & T[5] && P[1] & T[6] && P[2] & T[7] && P[3] & T[8]

Right shift 6 positions & comparison done on P[0] & T[6] && P[1] & T[7] && P[2] & T[8]

Right shift 7 positions & compare P[0] & T[7] && P[1] & T[8]

Right shift 8 positions & compare P[0] & T[8]

Right shift 9 positions & compare P[0] & T[9] && P[1] & T[10]

Right shift 10 positions & comparison done on P[0] & T[10]

Right shift 10 positions & comparison done on P[0] & T[11] && P[1] & T[12] && P[2] & T[13] && P[3] & T[14]

1. Suppose that all characters in the pattern P are different. Show the pseudocode of an algorithm that accelerate the brute-force algorithm to run in time O(n) on an n-character text T.

Algorithm stringmatch(T, P)

i, s <- 0

n <- T.length

m <- p.length

while s do

if (i==m) then

i <- 0

if P[i] == T[s] then

i <- i+1

s <- s+1

else

if (i == 0) then

s <- s+1

i <- 0

1. Using Boyer-Moore's algorithm and the text "puppy puppet looks happy", find the substring “puppet”. Compute first the last occurrence function. For the alphabet assume it is the letters that compose the text. Discard blank spaces.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| p | u | p | p | y | p | u | p | p | e | t | l | o | o | k | s | h | a | p | p | y |
| p | u | p | p | e | t |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Mismatch

P has “p”, Shift P to last occurrence of “p” in P

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| p | u | p | p | y | p | u | p | p | e | t | l | o | o | k | s | h | a | p | p | y |
|  |  | p | u | p | p | e | t |  |  |  |  |  |  |  |  |  |  |  |  |  |

Mismatch

P has “p”, Shift P to last occurrence of “p” in P

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| p | u | p | p | y | p | u | p | p | e | t | l | o | o | k | s | h | a | p | p | y |
|  |  |  |  | p | u | p | p | e | t |  |  |  |  |  |  |  |  |  |  |  |

Mismatch

P has “e”, Shift P to last occurrence of “e” in P

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| p | u | p | p | y | p | u | p | p | e | t | l | o | o | k | s | h | a | p | p | y |
|  |  |  |  |  | p | u | p | p | e | t |  |  |  |  |  |  |  |  |  |  |

Now we match!

1. Following the Boyer-Moore algorithm, what is the number of comparisons required for finding the the previous pattern in the given text? Help yourself with a drawing similar to the examples showed in class and paste it with your answer.

9 times

**Extra Credit Questions:**

1. Suppose we want to perform *sequence*​ ​ pattern matching. For example, given a text “hello” and a pattern “eo”, the matcher should return “true” (because the sequence “eo” exists in “hello”). Can an unmodified Boyer-Moore support sequence pattern matching? Justify your answer.

1. Write an algorithm to perform sequence pattern matching. The algorithm should run in O(n) time and O(n) space, n is the length of the input text. The algorithm takes as input the text T and the pattern P. The output should be true if there is a sequence S in T that matches P.