Algorithms: Homework 3

Li-Yuan Wei

Wednesday 14th June, 2023

Problem 1

Give an $\mathcal{O}(n^2)$-time algorithm to find the longest monotonically increasing subsequence of n numbers.

Solution. Psuedocode for an $\mathcal{O}(n^2)$-time algorithm:

```
LONGEST-INCREASING-SUBSEQUENCE(A, n)
 1: B = MERGE-SORT(A) // B is a sorted copy of A in ascending order
 2: C[n+1, n+1] // create 2-D array C with val, dir fields
 3: for i = 0 to n
 4:    C[i][0] = 0
 5: for j = 0 to n
 6:    C[0][j] = 0
 7: for i = 1 to n
 8:    for j = 1 to n
 9:       if B[i] ≤ A[j]
10:          C[i][j].val = C[i − 1][j − 1].val + 1
11:          C[i][j].dir =↖
12:       else // B[i] > A[j] no increasing
13:          if C[i − 1][j].val < C[i][j − 1].val
14:             C[i][j].val = C[i][j − 1].val
15:             C[i][j].dir =←
16:          else
17:             C[i][j].val = C[i − 1][j].val
18:             C[i][j].dir =↑
19: return C // C[n][n].val is the length of the longest increasing subsequence of n
```

Line 1 has time complexity of merge sort, which is $\mathcal{O}(n \lg n)$. Line 2 has constant time complexity. Line 3-6 has time complexity of $\mathcal{O}(n)$. Line 7-18 has time complexity of $\mathcal{O}(n^2)$. Thus, the overall time comlexity is $\mathcal{O}(n^2)$. ■

Problem 2

Find an optimal solution to the following activity selection problem:

i	1	2	3	4	5	6	7	8	9	10
s_i	1	3	2	3	5	8	7	10	5	11
f_i	3	4	5	6	8	9	11	12	14	15

Solution. The optimal solution is to chose the following events: <1, 2, 5, 6, 8>. ■

Problem 3

Not just any greedy approach to the activity-selection problem produces a maximum-size set of mutually compatible activities. (a) Give an example to show that the approach of selecting the activity of least duration from those that are compatible with previously selected activities does not work. (b) Do the same for the approaches of always selecting the compatible activity that overlaps with the fewest other remaining activities by selecting the compatible remaining activity with the earliest start time.

i	1	2	3	4	5
s_i	1	3	5	7	9
f_i	5	6	9	10	13

Figure 1: (a) select least duration

i	1	2	3	4	5	6	7	8	9	10	11
s_i	0	1	1	1	2	3	4	5	5	5	6
f_i	2	3	3	3	4	5	6	7	7	7	8

Figure 2: (b) select fewest overlaps

Solution. (a) By selecting event that has the least duration, we will pick events 2(3 - 6) and 4(7 - 10). However, if we choose events based on their earliest finish time, we will pick events 1(1 - 5), 3(5 - 9) and 5(9 - 13).
(b) By selecting event that has the fewest overlaps, we will pick events 6(3 - 5) and 1(0 - 2) and 11(6 -8), since these two events do not overlap with other events. Yet, the optimal solution should choose events 1(0 - 2), 5(2 - 4), 7(4 - 6) and 11(6 - 8). ■

Problem 4

Give a dynamic-programming solution to the 0-1 knapsack problem that runs in $\mathcal{O}(nW)$ time, where n is number of items and W is the maximum weight of items that the thief can put in his knapsack. Note that W, as well as all the weights of the involved items are integers.

Solution. Psuedocode for an $\mathcal{O}(nW)$-time algorithm, where A is the array of n items with $val, weight$ fields:

```
KNAPSACK-DP(A, W, n)
 1: C[n+1, W+1] // create 2-D array C with val field
 2: for i = 0 to n
 3:    C[i][0].val = 0 // initialization j from 0 to n
 4: for j = 0 to W
 5:    C[0][j].val = 0 // initialization i from 0 to W
 6: for i = 1 to n
 7:    for j = 1 to W
 8:       if A[i].weight > j // current item's weight is greater than the knapsack capacity
 9:          C[i][j].val = C[i − 1][j].val // get the value of previous i − 1 items
10:       else
11:          // C[i − 1][j].val: value of no additional item
12:          // C[i − 1][j − A[i].weight] + A[i].val: value of add A[i] into knapsack
13:          C[i][j].val = MAX(C[i − 1][j].val, C[i − 1][j − A[i].weight] + A[i].val)
14: return C // C[n][W].val has the maximum value of knapsack
```

Line 1 has constant time complexity. Line 3-4 has time complexity of $\mathcal{O}(n)$, while line 5-6 has time complexity of $\mathcal{O}(W)$. Line 6-13 has time complexity of $\mathcal{O}(nW)$. Thus, the overall time comlexity is $\mathcal{O}(nW)$. ■

Problem 5

Assume there are only 7 characters A, B, C, D, E, F and G in a document, with occurrences of 6, 4, 8, 3, 2, 1 and 5, respectively. Please calculate the total number of bits of this document after applying the Huffman coding.

C: 8 | 10
A: 6 | 01
G: 5 | 111
B: 4 | 110
D: 3 | 001
E: 2 | 0001
F: 1 | 0000

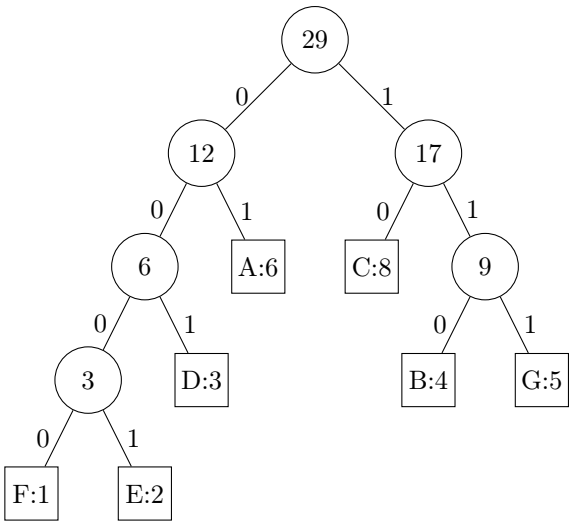

Figure 3: Huffman code tree

Solution. total bits: $1 \cdot 4 + 2 \cdot 4 + 3 \cdot 3 + 4 \cdot 3 + 5 \cdot 3 + 6 \cdot 2 + 8 \cdot 2 = 76$ ■

Problem 6

Represent the graph on the right by adjacency-matrix and adjacency-list.

	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	1	0
3	0	1	0	0	1
4	1	1	0	0	1
5	0	0	1	1	0

Figure 4: adjacency-matrix

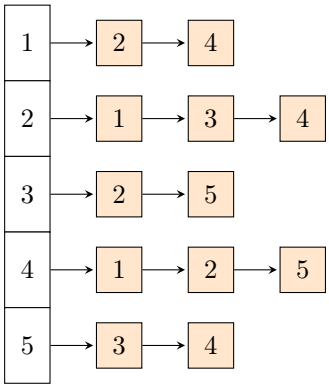

Figure 5: adjacency-list

Solution. ■

Problem 7

Show the d and π values that result from running the breadth-first search on the same graph using vertex 3 as the source.

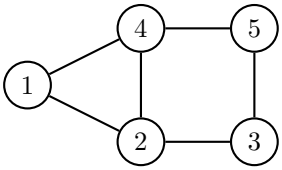

Figure 6: Original Graph

v	1	2	3	4	5
d	2	1	0	2	1
π	2	3	**NIL**	2	3

Solution. v stands for verticies in the graph. d is the search distance, while π is vertex v's parent node. Vertex 3 is the starting point, thus, it has no parent node. ■

Problem 8

(a) Show how depth-first search works on the graph at the bottom. Assume that the for loop of lines 5—7 of the DFS procedure considers the vertices in alphabetical order, and assume that each adjacency list is ordered alphabetically. Show the discovery and finishing time for each vertex, and show the classification of each edge.
(b) According to (a), show the corresponding topological sort of this graph.

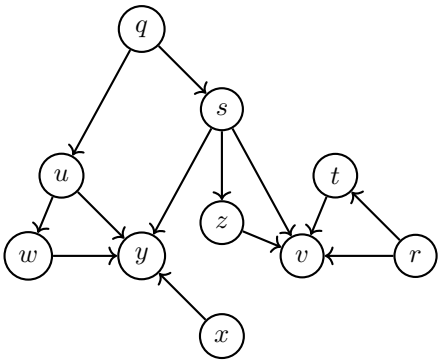

Figure 7: Original Graph

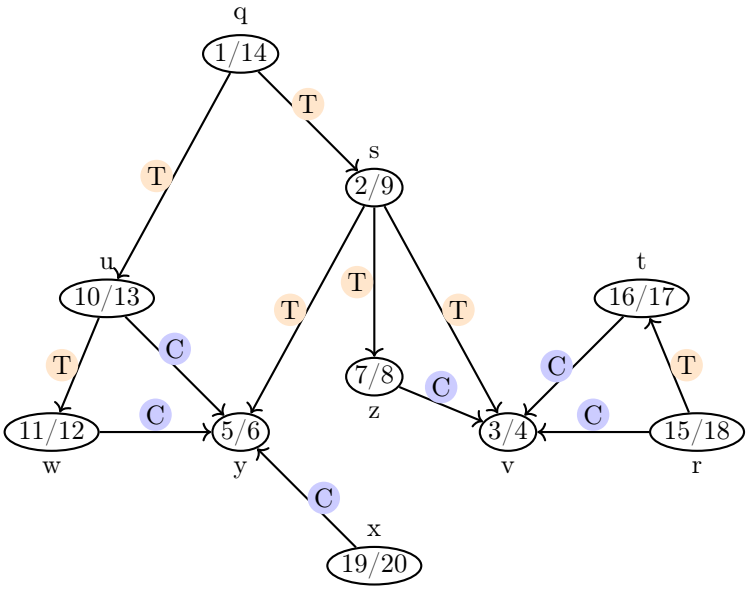

Figure 8: (a) DFS with discovery, finish time and edge classification

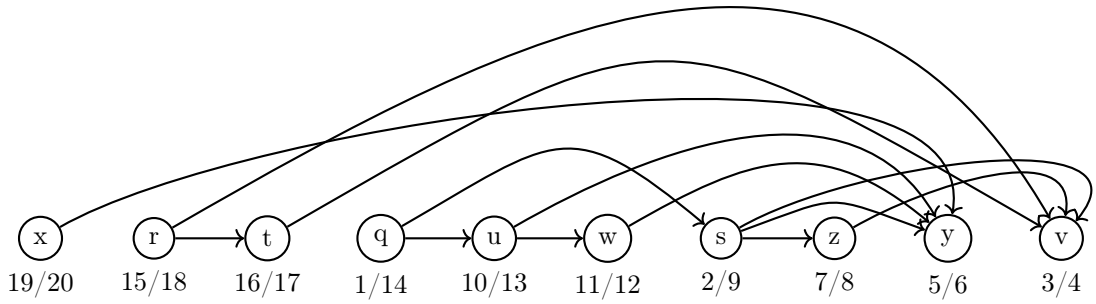

Figure 9: (b) topological sort with decreasing finish time order

Problem 9

Find the strongest components of the following graph:

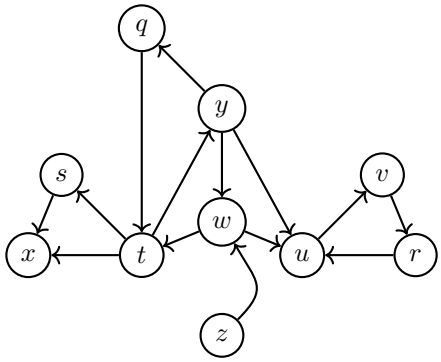

Figure 10: Original Graph

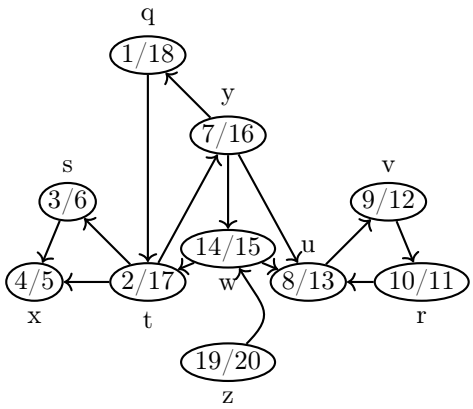

Figure 11: First DFS

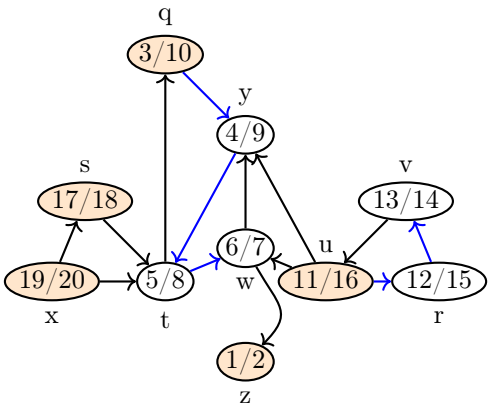

Figure 12: Second DFS with Transpose Graph

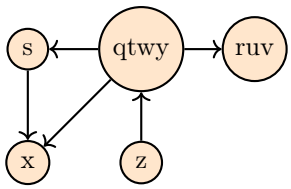

Figure 13: Strongly Connected Components

Solution. Based on the graph in Figure 10, we first call DFS and get the discovery, finish time in Figure 11. After the first DFS call, we transpose our graph to reverse all edges and run DFS again(run with each vertex's finish time in decreasing order. In this example, DFS will run based on this order: z, q, u, s, x), yielding Figure 12. As a result, we have our strongly connected components in Figure 13. ■

Problem 10

Find the minimal spanning tree of the following graph, using (a) Kruskal's algorithm, (b) Prim's algorithm (using q as the starting vertex). Please give the selected edge in order during the spanning tree construction process

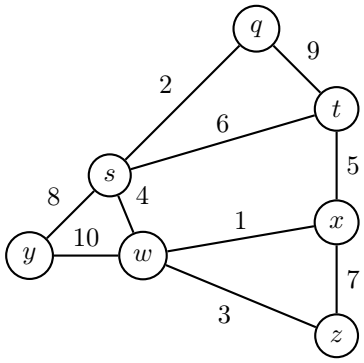

Figure 14: Original Graph

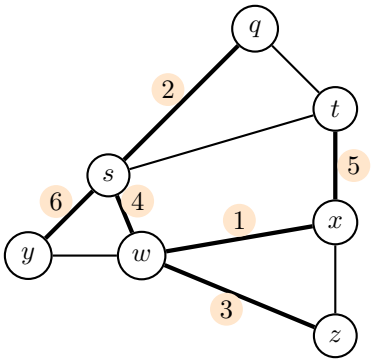

Figure 15: Minimal Spanning Tree: Kruskal's algorithm

Solution. Kruskal's algorithm: first sort all edges in the graph in increasing order, then pick the lowest cost edge every step until every vertex is connected to our tree. The final result is shown in Figure 15. ■