CS456: Algorithm Design and Analysis

Elikem Asudo Tsatsu Gale-Zoyiku

March 27, 2024

Assignment 4

Problem 1 (Prim's Algorithm)

Tree Vertices	**Remaining Vertices**
a(-,-)	b(a,3) c(a,5) d(a,4) $e(-,\infty)$ $f(-,\infty)$ $g(-,\infty)$ $h(-,\infty)$ $i(-,\infty)$ $j(-,\infty)$ $k(-,\infty)$ $l(-,\infty)$
b(a,3)	e(b, 3) f(b, 6) $c(a,5)$ $d(a,4)$ $g(-,\infty)$ $h(-,\infty)$ $i(-,\infty)$ $j(-,\infty)$ $k(-,\infty)$ $l(-,\infty)$
e(b,3)	f(e, 2) d(e, 1) i(e, 4) c(a,5) $g(-,\infty)$ $h(-,\infty)$ $j(-,\infty)$ $k(-,\infty)$ $l(-,\infty)$
d(e,1)	f(e,2) i(e,4) c(d,2) $g(-,\infty)$ h(d, 5) $j(-,\infty)$ $k(-,\infty)$ $l(-,\infty)$
c(d,2)	f(e,2) i(e,4) g(c, 4) h(d,5) $j(-,\infty)$ $k(-,\infty)$ $l(-,\infty)$
f(e,2)	i(e,4) g(c,4) h(d,5) j(f, 5) $k(-,\infty)$ $l(-,\infty)$
i(e,4)	g(c,4) h(d,5) j(i,3) l(i, 5) $k(-,\infty)$
j(i,3)	g(c,4) h(d,5) l(i,5) $k(-,\infty)$
g(c,4)	h(g,3) l(i,5) k(g, 6)
h(g,3)	l(i,5) k(g,6)
l(i,5)	k(g,6)
k(g,6)	

Table 1: Prim's Algorithm to produce minimum spanning tree

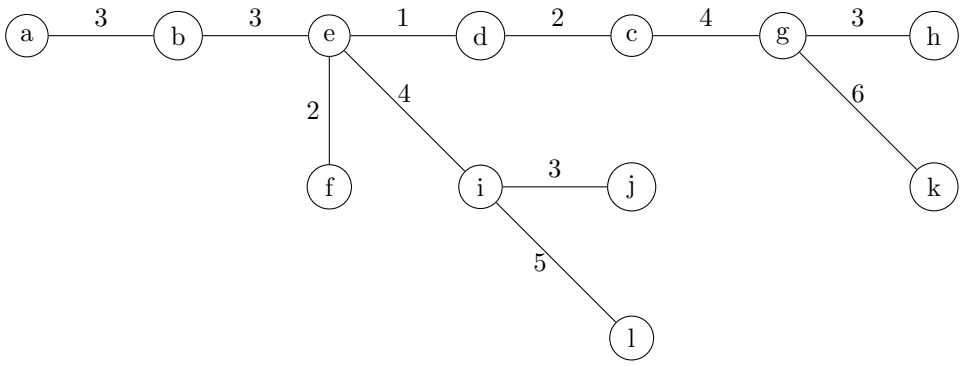

Figure 1: Minimum Spanning Tree generated using Prim's Algorithm

Problem 2 (Kruskal's Algorithm)

Tree Edges	Sorted List of Edges
	de(1) cd(2) ef(2) ab(3) be(3) gh(3) ij(3) ad(4) cg(4) ei(4) ac(5) dh(5) fj(5) il(5) bf(6) gk(6) hi(6) hk(7) kl(8) jl(9)
de(1)	cd(2) ef(2) ab(3) be(3) gh(3) ij(3) ad(4) cg(4) ei(4) ac(5) dh(5) fj(5) il(5) bf(6) gk(6) hi(6) hk(7) kl(8) jl(9)
cd(2)	ef(2) ab(3) be(3) gh(3) ij(3) ad(4) cg(4) ei(4) ac(5) dh(5) fj(5) il(5) bf(6) gk(6) hi(6) hk(7) kl(8) jl(9)
ef(2)	ab(3) be(3) gh(3) ij(3) ad(4) cg(4) ei(4) ac(5) dh(5) fj(5) il(5) bf(6) gk(6) hi(6) hk(7) kl(8) jl(9)
ab(3)	be(3) gh(3) ij(3) ad(4) cg(4) ei(4) ac(5) dh(5) fj(5) il(5) bf(6) gk(6) hi(6) hk(7) kl(8) jl(9)
be(3)	gh(3) ij(3) ad(4) cg(4) ei(4) ac(5) dh(5) fj(5) il(5) bf(6) gk(6) hi(6) hk(7) kl(8) jl(9)
gh(3)	ij(3) ad(4) cg(4) ei(4) ac(5) dh(5) fj(5) il(5) bf(6) gk(6) hi(6) hk(7) kl(8) jl(9)
ij(3)	ad(4) cg(4) ei(4) ac(5) dh(5) fj(5) il(5) bf(6) gk(6) hi(6) hk(7) kl(8) jl(9)
cg(4)	ad(4) ei(4) ac(5) dh(5) fj(5) il(5) bf(6) gk(6) hi(6) hk(7) kl(8) jl(9)
ei(4)	ad(4) ac(5) dh(5) fj(5) il(5) bf(6) gk(6) hi(6) hk(7) kl(8) jl(9)
il(5)	dh(5) fj(5) ac(5) bf(6) gk(6) hi(6) hk(7) kl(8) jl(9)
gk(6)	hi(6) hk(7) ac(5) bf(6) kl(8) jl(9)

Table 2: Kruskal's Algorithm to produce minimum spanning tree

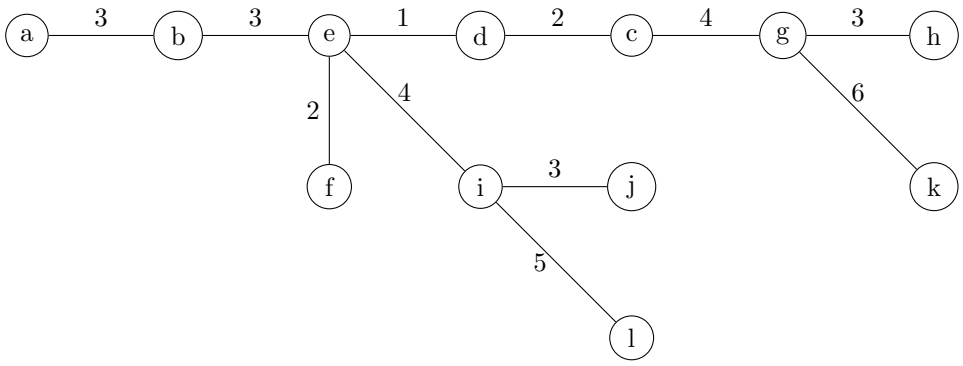

Figure 2: Minimum Spanning Tree generated using Kruskal's Algorithm

Problem 3 (Djikstra's Algorithm)

Tree Vertices	Remaining Vertices
a(-,0)	b(a,3) c(a,5) d(a,4) e(-,∞) f(-,∞) g(-,∞) h(-,∞) i(-,∞) j(-,∞) k(-,∞) l(-,∞)
b(a,3)	e(b,3+3) f(b,3+6) c(a,5) d(a,4) g(-,∞) h(-,∞) i(-,∞) j(-,∞) k(-,∞) l(-,∞)
d(a,4)	e(d,4+1) f(b,9) c(a,5) g(-,∞) h(d,4+5) i(-,∞) j(-,∞) k(-,∞) l(-,∞)
c(a,5)	e(d,5) f(b,9) g(c,5+4) h(d,9) i(-,∞) j(-,∞) k(-,∞) l(-,∞)
e(d,5)	f(e,5+2 == 7) g(c,9) h(d,9) i(-,5+4 == 9) j(-,∞) k(-,∞) l(-,∞)
f(e,7)	g(c,9) h(d,9) i(e,9) j(f,7+5) k(-,∞) l(-,∞)
g(c,9)	h(d,9) i(e,9) j(f,12) k(g,9+6) l(-,∞)
h(d,9)	i(e,9) j(f,12) k(g,15) l(-,∞)
i(e,9)	j(f,12) k(g,15) l(i,9+5)
j(f,12)	k(g,15) l(i,14)
l(i,14)	k(g,15)
k(g,15)	

Table 3: Djikstra's Algorithm to find shortst path to each vertex from source a

Shortest Paths from a to each vertex:

- from a to b : a $\rightarrow$ b of length 3
- from a to d : a $\rightarrow$ d of length 4
- from a to c : a $\rightarrow$ c of length 5
- from a to e : a $\rightarrow$ d $\rightarrow$ e of length 5
- from a to f : a $\rightarrow$ d $\rightarrow$ e $\rightarrow$ f of length 7
- from a to g : a $\rightarrow$ c $\rightarrow$ g of length 9
- from a to h : a $\rightarrow$ d $\rightarrow$ h of length 9
- from a to i : a $\rightarrow$ d $\rightarrow$ e $\rightarrow$ i of length 9
- from a to j : a $\rightarrow$ d $\rightarrow$ e $\rightarrow$ f $\rightarrow$ j of length 12
- from a to l : a $\rightarrow$ d $\rightarrow$ e $\rightarrow$ i $\rightarrow$ l of length 14
- from a to k : a $\rightarrow$ c $\rightarrow$ g $\rightarrow$ k of length 15

Problem 4

a

No adjustments are needed to Djikstra's Algorithm to find the shortest path in a directed graph. All nodes in the directed graph would have edges, and costs, to and from them. The algorithm would work as expected by exploring those edges and costs. In implementing it, just like with undirected graphs, the graph would be represented with each node having a list of connections to other nodes and the costs of those connections. However, unlike in an undirected graph where each edge is essentially a two-way connection between nodes, in a directed graph, connections are not duplicated. This means that an edge from node A to node B does not imply an edge from node B to node A.

b

The algorithm would have to be adjusted to start at the source vertex as usual. Instead of terminating when all vertices have been visited, terminate the algorithm when the target vertex is reached, and there no more paths ending at the target vertex. Determine the shortest path to the target vertex. Every other part of the algorithm would work the same (calculating the displacement from the source vertex). However, early pruning could be factored in to avoid exploring nodes that are not on the path to the target vertex (e.g. dead ends).

c

The algorithm would have to be adjusted to start at the source vertex as usual, and terminate when there are no more edges to the target vertex. The algorithm would also have to be adjusted to iterate over all the vertices in the graph, setting each one as the source, and for each vertex, find the shortest path to the target vertex. For each complete run through, the shortest path is determined.

d

To adapt Dijkstra's algorithm for solving the single-source shortest-paths problem in a graph with nonnegative numbers assigned to its vertices, where the length of a path is defined as the sum of the vertex numbers on the path, the following adjustments are needed:

1. Instead of traditional edge weights, use the sum of vertex numbers along the path as the weight of each edge.

2. Initialize the algorithm by setting the source vertex and assigning a distance of zero to it. Set the distances of all other vertices to infinity.

3. Select the vertex with the smallest cumulative vertex number sum as the next vertex to explore.

4. Update the distance to each vertex based on the cumulative vertex number sum along the path to that vertex. If the cumulative sum through a vertex is smaller than the current known distance to that vertex, update the distance accordingly.

5. Termination: Terminate the algorithm when all vertices have been visited. The shortest path to each vertex will be the path with the smallest cumulative vertex number sum.

Problem 5

Bottom-Up Approach to the Knapsack Problem

	Capacity j								
Item i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	0	0	0	0	0	0	0	0	0
Item 1 — $w_1 = 1, v_1 = 2$	0	2	2	2	2	2	2	2	2
Item 2 — $w_2 = 2, v_2 = 4$	0	2	4	6	6	6	6	6	6
Item 3 — $w_3 = 3, v_3 = 5$	0	2	4	6	7	9	11	11	11
Item 4 — $w_4 = 4, v_4 = 5$	0	2	4	6	7	9	11	11	12
Item 5 — $w_5 = 2, v_5 = 2$	0	2	4	6	7	9	11	11	13
Item 6 — $w_6 = 1, v_6 = 3$	0	3	5	7	9	10	12	14	14

Table 4: Table for the Bottom-Up Approach to the Knapsack Problem

a

The optimal solution to this instance of the knapsack problem is 14. The optimal subset can be found by:

- 14 - Profit of item 6 $\rightarrow$ 14 - 3 = 11
- 11 - Profit of Item in row of first appearance of 11 $\rightarrow$ 11 - 5 = 6
- 6 - Profit of Item in row of first appearance of 6 $\rightarrow$ 6 - 4 = 2
- 2 - Profit of Item in row of first appearance of 2 $\rightarrow$ 2 - 2 = 0

The optimal subset is $\{Item1, Item2, Item3, Item6\}$ with cooresponding weights $\{1, 3, 2, 1\}$ and corresponding profits $\{2, 4, 5, 3\}$.

b

To determine if there are multiple optimal subsets, we can look at the table generated by the algorithm. The value in the cell at the last row and last column of the table is the optimal value of the knapsack problem. You would begin backtracking from this cell to determine the optimal subset, i.e. subtract the profit of the item in the row of the cell from the optimal value and move to the cell in the row of the first appearance of the result. Repeat this process until you reach the first row of the table. If there are multiple optimal subsets, you would find that the backtracking process would yield different subsets that have the same optimal value, i.e. there would be multiple paths to the optimal value. This could occur if there are multiple cells with the same profit value in the last column, i.e. mulitiple items gave the same profit value. If there is only one optimal subset, the backtracking process would yield only one subset.

c

	Capacity j								
Item i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	0	0	0	0	0	0	0	0	0
Item 1 — $w_1 = 1, v_1 = 2$	0	2	2	2	2	2	2	2	2
Item 2 — $w_2 = 2, v_2 = 4$	0	X	4	6	6	6	6	6	6
Item 3 — $w_3 = 3, v_3 = 5$	0	X	X	6	7	9	11	11	11
Item 4 — $w_4 = 4, v_4 = 5$	0	X	X	X	X	X	X	11	12
Item 5 — $w_5 = 2, v_5 = 2$	0	X	X	X	X	X	X	11	13
Item 6 — $w_6 = 1, v_6 = 3$	0	X	X	X	X	X	X	X	14

Table 5: Table for the Bottom-Up Approach to the Knapsack Problem with values never computed marked values never recomputed marked as X

Problem 6

a

Tree Vertices	Remaining Vertices
A(-,0)	B(A,2) C(A,7) E(A,12) D(-,∞) F(-,∞) G(-,∞)
B(A,2)	D(B,2+2) C(A,7) E(A,12) F(-,∞) G(-,∞)
D(B,4)	C(A,7) E(A,12) F(D,4+2) G(-,∞)
F(D,6)	C(A,7) E(A,12) G(F,8)
C(A,7)	E(C,9) G(F,8)
G(F,8)	E(C,9)
E(C,9)	

Table 6: Djikstra's Algorithm to find shortst path to each vertex from source a

c

Edge E-G

Problem 7

Algorithm 1 Rod Cutting Algorithm

```
1: procedure RODCUTTING(n, prices)
2:     Initialize memoization table memo[0...n] with zeros
3:     for i from 1 to n do
4:         max_price ← −1
5:         for j from 1 to i do
6:             // Update max_price considering all possible cuts
7:             max_price ← max(max_price, prices[j − 1] + memo[i − j])
8:         end for
9:         memo[i] ← max_price
10:    end for
11:    return memo[n] // Return the maximum sale price for a rod of length n
12: end procedure
```

Time Complexity

The time complexity of this algorithm is $O(n^2)$. The outer loop runs n times (as many times as the length of the rod), and the inner loop runs i times for each iteration of the outer loop, and i ranges from 1 to n, so in the worst case, $i = n$. The operation being carried out is a comparison and an addition operation, which are both $O(1)$ operations. This gives a total of $T(n) = \sum_{i=1}^{n} \sum_{i=1}^{i} O(1) = \sum_{i=1}^{n} \sum_{i=1}^{n} O(1) = \sum_{i=1}^{n} (n - 1 + 1) = (n + 1 - 1)(n) = O(n^2)$ iterations of the inner loop.

Space Complexity

The space complexity of this algorithm is $O(n)$ as the memoization table $memo[0...n]$ has a size of n where n is the length of the rod. As n grows, the size of the memoization table grows linearly.

Problem 8

Graph 1

- Tour Edges: ('B', 'C'), ('A', 'B'), ('A', 'D'), ('B', 'D') (using multi-fragment heuristic)
- Minimum tour: ('C', 'B', 'A', 'D') with a total cost of 10
- Accuracy ratio = $AR = \sum(G[src][dest])/Wmin, forall(src, dest)inE = \frac{3+2+5+4}{10} = 1.4$

Graph 2

- Tour Edges: ('A', 'B'), ('B', 'D'), ('A', 'C'), ('A', 'D') (using multi-fragment heuristic)
- Minimum tour: ('C', 'A', 'B', 'D') with a total cost of 50
- Accuracy ratio = $AR = \sum(G[src][dest])/Wmin, forall(src, dest)inE = \frac{10+25+15+20}{50} = 1.4$