CSE 102 Spring 2021
Homework Assignment 4

Jaden Liu
University of California at Santa Cruz
Santa Cruz, CA 95064 USA

April 27, 2021

1 HW4

1. *Dijkstra's Single-Source Shortest Path Problem* : We are given a directed graph $G = (V, E)$ with n nodes and m edges. Each edge has a non-negative length $e_l \geq 0$. Length of a path between two nodes is the sum of the lengths of all edges in the path. Single-source shortest path problem is to find the path and the length of the shortest path between a given source (node) to every other node in the graph. If a path does not exist, then the length is inf. This problem is known as single-source shortest path problem.

   Under the assumption that each edge length is non-negative, that is, $e_l \geq 0$, Dijkstra designed a simple greedy algorithm to solve this problem. The time complexity of this algorithm varies depending upon the data structure implementation – array, binary heap, d-ary heap, or Fibonacci heap.

   This is a practical problem arising in many applications where the length represents cost or time.

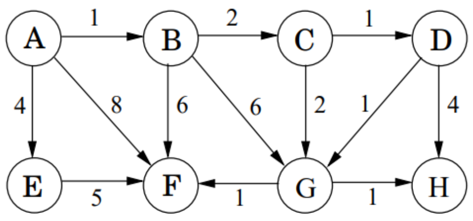
   (a) (1 point): Clearly state the greedy strategy. Explain briefly.

   (b) (2 points): Exercise 4.1 of DPV. This exercise essentially asks you to demonstrate how Dijkstra's algorithm works on a specific graph.

Solution for a. 1. Find the minimum length node
2. For the minimum length node, if there exist other shorter path to its neighbor nodes, then update the minimum length path.

3. Keep the two steps and save the minimum until we reach the end node
4. Calculate the minimum path. □

4.1. Suppose Dijkstra's algorithm is run on the following graph, starting at node A.



(a) Draw a table showing the intermediate distance values of all the nodes at each iteration of the algorithm.

(b) Show the final shortest-path tree.

Solution for b.

A	B	C	D	E	F	G	H
0	1	∞	∞	4	8	∞	∞
0	1	3	∞	4	7	7	∞
0	1	3	4	4	7	5	∞
0	1	3	4	4	7	5	8
0	1	3	4	4	6	5	6

$$\begin{tikzcd}
A \arrow[r, "1"] \arrow[d, "4"'] & B \arrow[r, "2"] & C \arrow[r, "1"] \arrow[d, "2"] & D \\
E & F & G \arrow[l, "1"'] \arrow[r, "1"] & H
\end{tikzcd}$$

□

2. *Minimal Spanning Tree: Kruskal's Algorithm* We are given an undirected connected (there is a path between every pair of nodes) graph $G = (V, E)$ with n nodes and m edges. Each edge has a positive cost $c_e > 0$. The problem is to find a subset of edges so that the resulting graph is connected and the total cost of all the edges combined is the smallest possible. It is easy to observe that the resulting graph must be a tree (a tree is an undirected graph that is connected and acyclic,

because if not then remove one of the edges of the cycle and the remaining graph is still connected and has a lower cost). This problem is referred to as Minimal Spanning Tree problem.

Kruskal designed a greedy algorithm that runs in $O(m \log n)$ time.

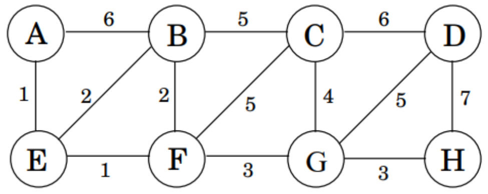
(a) (1 point): Clearly state the greedy strategy. Explain briefly.

(b) (2 points): Exercise 5.1 of DPV. This exercise essentially asks you to demonstrate how Kruskal's algorithm works on a specific graph. You do *not* need to specify the cut.

Solution for a. 1. First sort all edges in the graph according to the edge value from small to large
2. N vertices in a graph are regarded as a forest composed of n independent trees
3. Select the edge according to the weight from small to large, the two vertices connected by the selected edge should belong to two different trees, then become an edge of the minimum spanning tree, and merge the two trees as one tree.
4. Repeat (3) until all vertices are in a tree or have n-1 edges. □

Solution for b.

5.1. Consider the following graph.

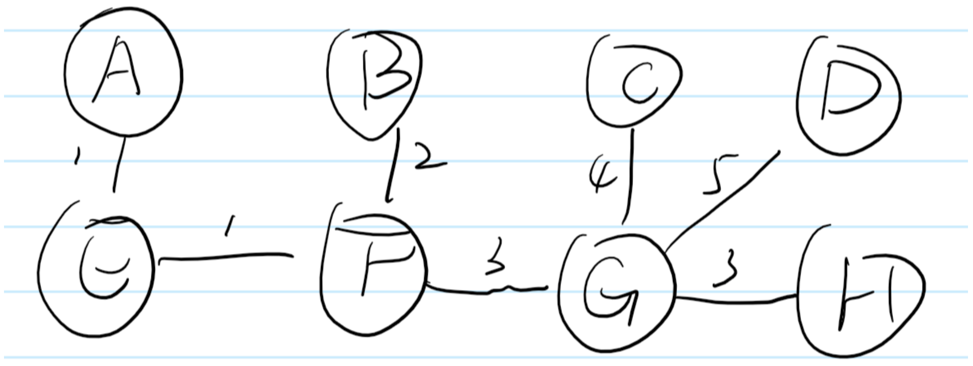


(a) What is the cost of its minimum spanning tree?

(b) How many minimum spanning trees does it have?

(c) Suppose Kruskal's algorithm is run on this graph. In what order are the edges added to the MST? For each edge in this sequence, give a cut that justifies its addition.

(a)



(b)
2
(c)
(A,E),(E,F)$-$ >(B,E)$-$ >(F,G),(G,H)$-$ >(G,C)$-$ >(G,D) □

3. *Minimal Spanning Tree: Prim's Algorithm* Prim designed a different greedy algorithm that also runs in $O(m \log n)$ time.

(a) (1 point): Clearly state the greedy strategy. Explain briefly.

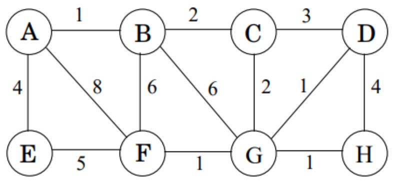
(b) (2 points): Exercise 5.2 (a) of DPV. This exercise essentially asks you to demonstrate how Prim's algorithm works on a specific graph.

Solution for a.

1. First find the minimum length, absorb it into the minimal spanning tree.
2. Find a minimum length that can generated by the set in the existing spanning tree, update it to the node attribute.
3. Add the minimum length again and keep doing (1),(2) until we reach $n-1$ edges in tree or n nodes. □

Solution for b.

5.2. Suppose we want to find the minimum spanning tree of the following graph.



(a) Run Prim's algorithm; whenever there is a choice of nodes, always use alphabetic ordering (e.g., start from node A). Draw a table showing the intermediate values of the `cost` array.

(b) Run Kruskal's algorithm on the same graph. Show how the disjoint-sets data structure looks at every intermediate stage (including the structure of the directed trees), assuming path compression is used.

(a)

A	B	C	D	E	F	G	H
(A,0)	(A,1)	(A,∞)	(A,∞)	(A,4)	(A,8)	(A,∞)	(A,∞)
(A,0)	(A,1)	(B,3)	(B,∞)	(A,4)	(B,6)	(B,6)	(B,∞)
(A,0)	(A,1)	(B,3)	(C,3)	(A,4)	(B,6)	(C,2)	(C,∞)
(A,0)	(A,1)	(B,3)	(G,1)	(A,4)	(G,1)	(C,2)	(G,1)
(A,0)	(A,1)	(B,3)	(G,1)	(A,4)	(G,1)	(C,2)	(G,1)
(A,0)	(A,1)	(B,3)	(G,1)	(A,4)	(G,1)	(C,2)	(G,2)
(A,0)	(A,1)	(B,3)	(G,1)	(A,4)	(G,1)	(C,2)	(G,2)
(A,0)	(A,1)	(B,3)	(G,1)	(A,4)	(G,1)	(C,2)	(G,2)

vertex	**edge**	**cost**	
A	N/A	0	
B	AB	0+1=1	
C	BC	1+2=3	
G	CG	3+2=5	(b)
D	GD	5+1=6	
F	GF	6+1=7	
H	GH	7+1=8	
E	AE	8+4=12	

(A,B),(F,G),(G,H)->(B,C),(C,G)->(C,D)->(A,E)->(E,F) □

4. Consider the following *office hours* problem faced by a professor. n students (numbered 1 to n) show up at the start of office hours with different questions. Each student i has one question that will take time t_i to answer, and will leave as soon as that question is answered. Given the n values of the t_i's, find an order in which to answer the students' questions such that the sum of the times spent by each student in the office is minimized.

   (a) (1 point) Describe a greedy algorithm that correctly finds optimal solutions. Explain briefly.

   (b) (3 points) Prove that your algorithm always returns an optimal solution. [Hint: Use exchange argument. Discussed in Brassard and Bratley's Book.]

Solution for a. Sort students' list by their time to answer t_i. Keep answering students that needs smaller time until no more students has questions. $\square$

Solution for b.

$$\begin{aligned} T(n) &= T(n-1) + t_1 * n \\ &= T(n-2) + t_1 * n + t_2 * (n-1) \\ &\vdots \end{aligned}$$

From the equation, we can find that we can only minimize t_i to get shorter time, since n is a solid number to multiply. Therefore, it's faster that we solved shorter-time-consumed question, the shorter we have in the total time $T(n)$. $\square$

5.

(Read the **Rod-Cutting Problem** in section 15.1 pp. 360-369 of CLRS 3rd edition. This is problem 15.1-2 on p. 370.) Show, by means of a counterexample, that the following "greedy" strategy does not always determine an optimal way to cut rods. Define the *density* of a rod of length i to be p_i/i, that is, its value per inch. The greedy strategy for a rod of length n cuts off a first piece of length i, where $1 \le i \le n$, having maximum density. It then continues by applying the greedy strategy to the remaining piece of length $n - i$.

Solution. Using the sample example in section 15.1 of CLRS 3rd edition:

length i	1	2	3	4	5	6	7
price p_i	1	5	8	9	10	17	17
density	1	2.5	2.666666667	2.25	2	2.833333333	2.428571

For instance that we need to find the maximum price for a rod length n=4. By greedy algorithm that we will find that we first cut the maximum density, which is n=3, then the other part is not dividable anymore. We get the total price, which is 8+1=9.

However, we can easily come up with that we cut the rod half then we get the maximum 5+5=10. Thus, there exist a counterexample. □

6.

Recall the coin changing problem: Given denominations $d = (d_1, d_2, \ldots, d_n)$ and an amount N to be paid, determine the number of coins in each denomination necessary to disburse N units using the fewest possible coins. Assume that there is an unlimited supply of coins in each denomination.

a. Write pseudo-code for a greedy algorithm that attempts to solve this problem. (Recall that the greedy strategy doesn't necessarily produce an optimal solution to this problem. Whether it does or not depends on the denomination set d.) Your algorithm will take the array d as input and return an array G as output, where $G[i]$ is the number of coins of type i to be disbursed. Assume the denominations are arranged by increasing value $d_1 < d_2 < \cdots < d_n$, so your algorithm will step through array d in reverse order. Also assume that $d_1 = 1$ so any amount can be paid.

b. Let $d_i = b^{i-1}$ for some integer $b > 1$, and $1 \le i \le n$, i.e. $d = (1, b, b^2, \ldots, b^{n-1})$. Does the greedy strategy always produce an optimal solution in this case? Either prove that it does, or give a counter-example.

c. Let $d_1 = 1$ and $2d_i \le d_{i+1}$ for $1 \le i \le n-1$. Does the greedy strategy always produce an optimal solution in this case? Either prove that it does, or give a counter- example.

Solution for a.

```

        procedure(array d, amount N)
        result =[]
        N_copy,rem = N.copy()
        for i in reversed(d): # reversed iteratet the input array
                              #so start from biggest one
                N_copy = rem
                coin_num = 0
                while(N_copy - i >= 0): # count for coin maximum
                                        #coin we could use
                        coin_num++
                rem = rem - coin_num*i
                result.append(coin_num)
        return result
```

□

Solution for b. The reason why greedy algorithm did not work well in general case is that if $d_i < d_{i-1} + d_{i-2}$, then we can subsitute amount of d_i with smaller count with d_{i-1} and d_{i-2} combination. For this case, we have always $d_i > d_{i-1} + d_{i-2}$. Therefore, using greedy algorithm is in subset d_i, d_{i-1}, $d_{i-2}, \cdots$, d_1 is better than in subset $d_{i-1}, d_{i-2}, \cdots, d_1$. Thus, this case alwasy produce an optimal solution. □

Solution for c. For this we have a counterexample that:

	d1	d2	d3
denomination	1	10	25

If we want N=30, then by greedy algorithm, we have one 25-coin and five 1-coins, in total 5+1=6. However the optimal solution is three 10-coin. □

References

[1] https://www.chegg.com/homework-help/questions-and-answers/recall-coin-changing-problem-given-denominations-1-2–amount-bepaid-determine-number-coins-q59517636