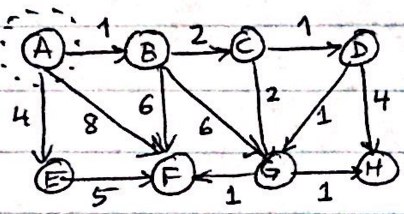
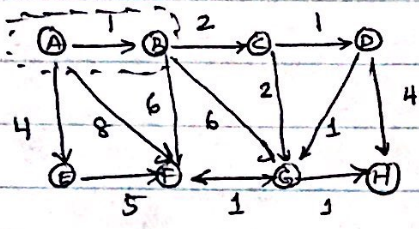
Dijkstra's single-source shortest path Algorithm

Removing edges as we traverse:

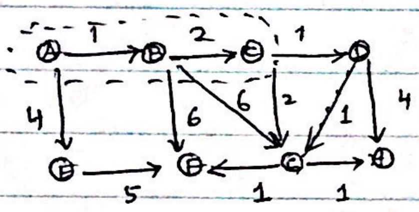


A	B	C	D	E	F	G	H
0	1	∞	∞	4	8	∞	∞



A	B	C	D	E	F	G	H
0	1	3	∞	4	7	7	∞

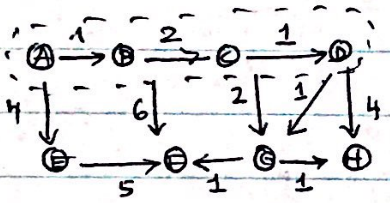
Removed edge AF



A	B	C	D	E	F	G	H
0	1	3	4	4	7	5	∞

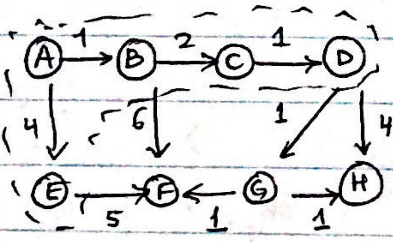
Two choices: choose D

Remove edge BG



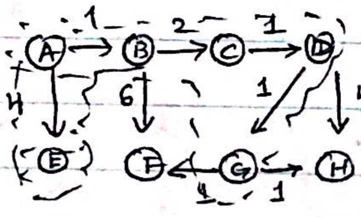
A	B	C	D	E	F	G	H
0	1	3	4	4	7	5	8

Two choices
Remove edge CG



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

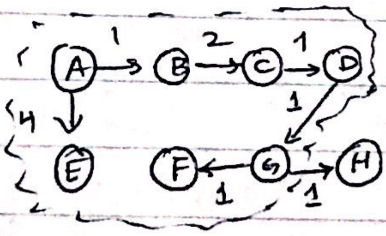
Remove edge EF



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

Two choices: choose G

Remove edges BF and DH



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

↓
one more step

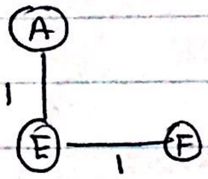
same shortest path

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

Kruskal's Algorithm

Two choices: AE or EF ⇒ next choice will be EF or AE

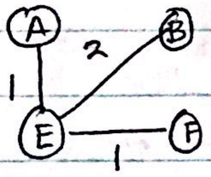
⇒

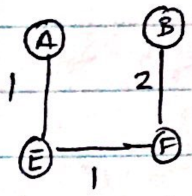


Two choices BE or BF ⇒ both not possible ⇒ will lead to 2 separate options

choose BE

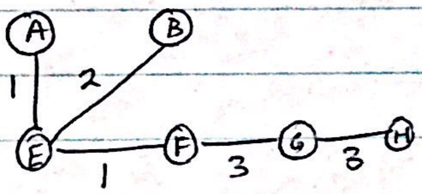
choose BF

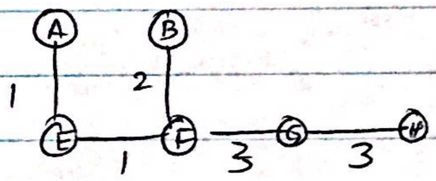




Two choices FG & GH
Both ok

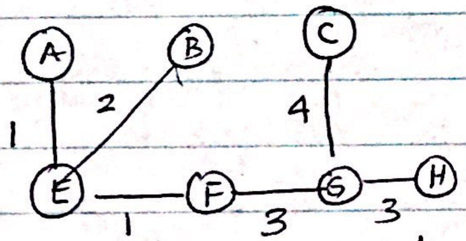
Two choices FG & GH Both ok

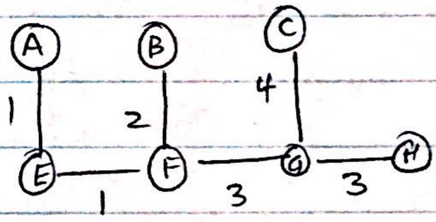




choose CG

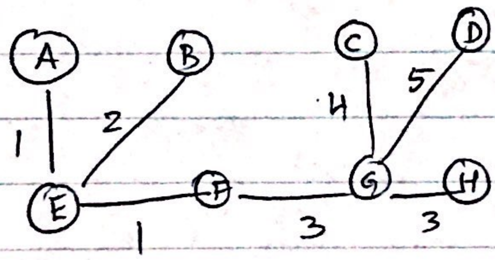
choose CG

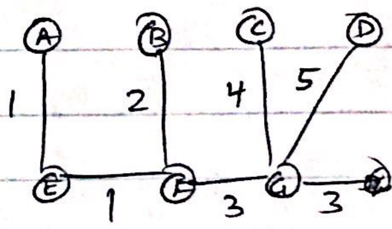




FC, & BC will create cycles
choose GD

FC & BC will create cycle
choose GD



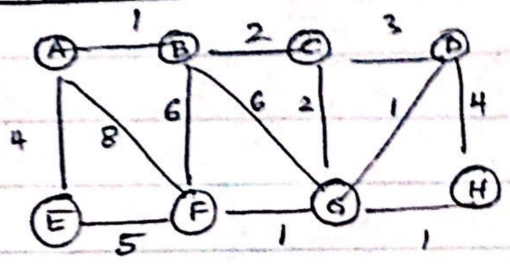


Cost = 1+1+2+3+3+4+5
= 19

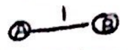
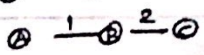
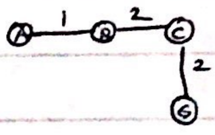
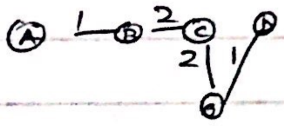
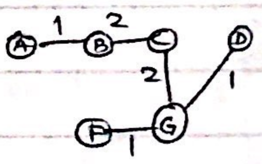
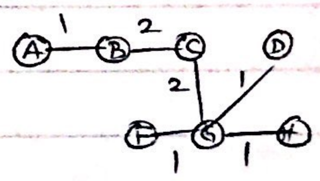
Cost = 1+1+2+3+3+4+5
= 19

2 minimum spanning trees

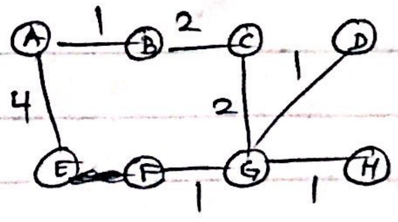
Prim's Algorithm



sets S	A	B	C	D	E	F	G	H
{ }	0/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil
A		1/A	∞/nil	∞/nil	4/A	8/A	∞/nil	∞/nil
A, B			2/B	∞/nil	4/A	6/B	6/B	∞/nil
A, B, C				3/C	4/A	6/B	2/C	∞/nil
A, B, C, G				1/G	4/A	1/G		1/G
A, B, C, G, D					4/A	1/G		1/G
A, B, C, G, D, F					4/A			1/G
A, B, C, G, D, F, H					4/A			

A, B, C, G, D, F, H, E



1+1+1+1+2+2+4 = 12

CSE 102
Homework Assignment 4
Solutions

1. (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 $n - i$.

   Counter Example
   Let $n = 3$ and $p = (p_1, p_2, p_3) = (1, 10, 12)$. Then the densities p_i/i are $(1, 5, 4)$. The greedy strategy first cuts a rod of length 2 having price 10, leaving a rod of length 1 having price 1. The total price for this sequence of cuts is then $10 + 1 = 11$. This sequence is not optimal however, since the whole rod of length 3 (with no cuts) is worth 12. Therefore the greedy strategy does not produce an optimal sequence of rod cuts in this instance. ■

2. 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.

   Solution:

```
GreedyCoinChange($d, N$) (Pre: $1 = d[1] < d[2] < \cdots < d[n]$)
1.  $n = \text{length}[d]$
2.  $G[1 \cdots n] = (0, 0, \ldots, 0)$
3.  sum $= 0$
4.  $i = n$
5.  while  sum $< N$
6.      while  sum $+ d[i] \le N$
7.          sum $+= d[i]$
8.          $G[i]++$
9.      $i--$
10. return $G$
```

   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.

   Theorem
   The greedy strategy produces an optimal solution with this denomination set for any amount N.

Proof:
Let $N \geq 0$, and suppose that $x = (x_1, x_2, \ldots, x_n)$ is an optimal disbursement of *N* units, where x_i is the number of coins of type i to be paid $(1 \leq i \leq n)$. Let $g = (g_1, g_2, \ldots, g_n)$ be the solution produced by the greedy strategy. To show that g is optimal, it suffices to show that $g_i = x_i$ for $1 \leq i \leq n$. Since both solutions disburse *N* units, we have $\sum_{i=1}^{n} x_i b^{i-1} = N = \sum_{i=1}^{n} g_i b^{i-1}$, i.e.

$$x_1 + x_2 b + x_3 b^2 + \cdots + x_n b^{n-1} = g_1 + g_2 b + g_3 b^2 + \cdots + g_n b^{n-1}.$$

Reducing this equation mod b yields the congruence $x_1 \equiv g_1 (\operatorname{mod} b)$. Observe $0 \leq x_1 < b$, since otherwise it would be possible to replace b coins of type 1 by 1 coin of type 2, contradicting that *x* is optimal. Likewise $0 \leq g_1 < b$, since the greedy strategy guarantees that as many coins as possible of type 2 (value b) will be disbursed before any coins of type 1 are. These inequalities, along with the preceding congruence, imply $x_1 = g_1$. Cancel x_1 from both sides of the above equation and divide through by b to obtain

$$x_2 + x_3 b + \cdots + x_n b^{n-2} = g_2 + g_3 b + \cdots + g_n b^{n-2}.$$

A similar argument shows that $x_2 = g_2$. Continuing in this manner, we can show $g_i = x_i$ for $1 \leq i \leq n$, and hence the greedy solution g is in fact optimal, as required. ■

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

Counter Example:
Let $n = 3$, $d = (1, 10, 25)$, and $N = 30$. Observe that $2 \cdot 1 \leq 10$, and $2 \cdot 10 \leq 25$. The greedy solution in this case is $g = (5, 0, 1)$, which uses 6 coins, while the optimal solution $x = (0, 3, 0)$ uses only 3 coins. ■

In this section we present two problems concerning the optimal way to schedule jobs on a single machine. In the first, the problem is to minimize the average time that a job spends in the system. In the second, the jobs have deadlines, and a job brings in a certain profit only if it is completed by its deadline: our aim is to maximize profitability. Both these problems can be solved using greedy algorithms.

6.6.1 Minimizing time in the system

A single server, such as a processor, a petrol pump, or a cashier in a bank, has n customers to serve. The service time required by each customer is known in advance: customer i will take time t_i, $1 \le i \le n$. We want to minimize the average time that a customer spends in the system. Since n, the number of customers, is fixed, this is the same as minimizing the total time spent in the system by all the customers. In other words, we want to minimize

$$T = \sum_{i=1}^{n} (\text{time in system for customer } i).$$

Suppose for example we have three customers, with $t_1 = 5$, $t_2 = 10$ and $t_3 = 3$. There are six possible orders of service.

Order	T	
1 2 3:	$5 + (5 + 10) + (5 + 10 + 3) = 38$	
1 3 2:	$5 + (5 + 3) + (5 + 3 + 10) = 31$	
2 1 3:	$10 + (10 + 5) + (10 + 5 + 3) = 43$	
2 3 1:	$10 + (10 + 3) + (10 + 3 + 5) = 41$	
3 1 2:	$3 + (3 + 5) + (3 + 5 + 10) = 29$	← optimal
3 2 1:	$3 + (3 + 10) + (3 + 10 + 5) = 34$	

In the first case, customer 1 is served immediately, customer 2 waits while customer 1 is served and then gets his turn, and customer 3 waits while both 1 and 2 are served and then is served last; the total time passed in the system by the three customers is 38. The calculations for the other cases are similar.

In this case, the optimal schedule is obtained when the three customers are served in order of increasing service time: customer 3, who needs the least time, is served first, while customer 2, who needs the most, is served last. Serving the customers in order of decreasing service time gives the worst schedule. However, one example is not a proof that this is always so.

To add plausibility to the idea that it may be optimal to schedule the customers in order of increasing service time, imagine a greedy algorithm that builds the optimal schedule item by item. Suppose that after scheduling service for customers $i_1, i_2, \ldots, i_m$ we add customer j. The increase in T at this stage is equal to the sum of the service times for customers i_1 to i_m (for this is how long customer j must wait before receiving service), plus t_j, the time needed to serve customer j. To minimize this, since a greedy algorithm never undoes its previous decisions, all we can do is to minimize t_j. Our greedy algorithm is therefore simple: at each step, add to the end of the schedule the customer requiring the least service among those who remain.

Theorem 6.6.1 *This greedy algorithm is optimal.*

Proof Let $P = p_1 p_2 \cdots p_n$ be any permutation of the integers from 1 to n, and let $s_i = t_{p_i}$. If customers are served in the order P, then the service time required by the i–th customer to be served is s_i, and the total time passed in the system by all the customers is

$$\begin{aligned} T(P) &= s_1 + (s_1 + s_2) + (s_1 + s_2 + s_3) + \cdots \\ &= ns_1 + (n-1)s_2 + (n-2)s_3 + \cdots \\ &= \sum_{k=1}^{n} (n-k+1)s_k. \end{aligned}$$

Suppose now that P does not arrange the customers in order of increasing service time. Then we can find two integers a and b with $a < b$ and $s_a > s_b$. In other words, the a–th customer is served before the b–th customer even though the former needs more service time than the latter; see Figure 6.7. If we exchange the positions of these two customers, we obtain a new order of service P', which is simply P with the integers p_a and p_b interchanged. The total time passed in the system by all the customers if schedule P' is used is

$$T(P') = (n-a+1)s_b + (n-b+1)s_a + \sum_{\substack{k=1 \\ k \neq a,b}}^{n} (n-k+1)s_k.$$

The new schedule is preferable to the old because

$$\begin{aligned} T(P) - T(P') &= (n-a+1)(s_a - s_b) + (n-b+1)(s_b - s_a) \\ &= (b-a)(s_a - s_b) > 0. \end{aligned}$$

The same result can be obtained less formally from Figure 6.7. Comparing schedules P and P', we see that the first $a-1$ customers leave the system at exactly the same time in both schedules. The same is true of the last $n-b$ customers. Customer a now leaves when customer b used to, while customer b leaves earlier than customer a used to, because $s_b < s_a$. Finally those customers served in positions $a+1$ to $b-1$ also leave the system earlier, for the same reason. Overall, P' is therefore better than P.

Thus we can improve any schedule in which a customer is served before someone else who requires less service. The only schedules that remain are those obtained by putting the customers in order of nondecreasing service time. All such schedules are clearly equivalent, and therefore all optimal. ∎

Implementing the algorithm is so straightforward that we omit the details. In essence all that is necessary is to sort the customers into order of nondecreasing service time, which takes a time in $O(n \log n)$. The problem can be generalized to a system with s servers, as can the algorithm: see Problem 6.20.

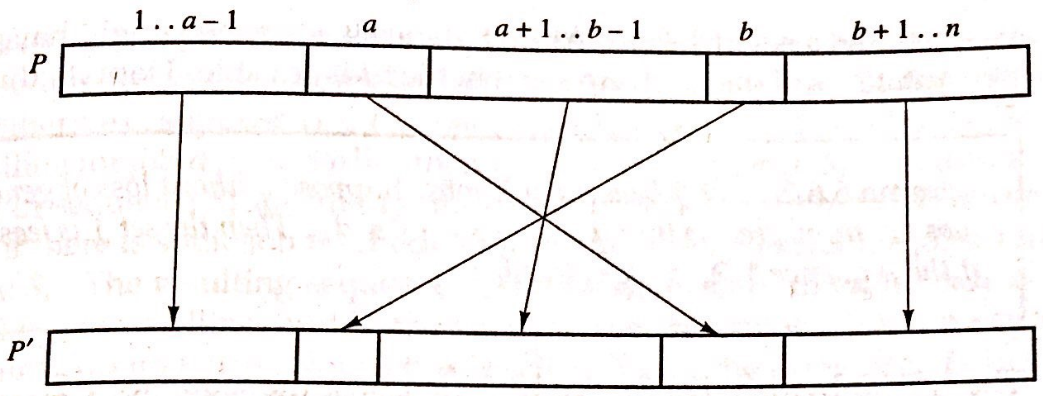


Figure 6.7. Exchanging two customers