Algo-quiz 3

monthon.kra

November 2021

1 Question 1

Ans

(a)
i True
ii True
iii True
iv True

(b)
i
ii The maximum flow f $= 5 + 6 + 1 + 1 = 13$
iii The minimum cut is same as the maximum flow = 13, it cut from AC $\rightarrow BC \rightarrow BD$

2 Question 2

Ans

(a) There is an algorithm that can make the customer who has a minimum time to be serve first. Which is the number of customers is sorted by time "t" and serve them in order. The Algorithm is called "Greedy Algorithms"

The total time can be calculated as

$$T = \sum_{i=l}^{n}(t)$$

t = number of customer still waiting

This can minimize the waiting time of customer.

```
Algorithms for the above description.

    BestOrder(n, T)
        Input:  Number of customers "n"
                Set of service time t(i) : i for all (1,2,...,N) T
        Output: System.out.printf( best order for customer process );

        Sort the set of service time T:
            Increasing order of time t(i)
        Return T
```

(b) Let assume there is some optical solution that is better than our solution. Our solution is used to sort the customer by order of time. However, for the optimal solution, it contain one pairs of consecutive customers. The second customer takes lesser time to server.

Let make the pair of customers c_i and c_{i+1} and the service time as t_i and t_{i+1}.
Let assume that the time take to service first customer is more than the second customer ($t_i > t_{i+1}$).
By swapping the order of these two customers will create a better ordering system. In addition it also doesn't affect the waiting time of the other customers.

The waiting time for c_i will increase by t_{i+1} and the same goes to customer c_{i+1} will increase by t_i. Which make the assumption of $t_i > t_{i+1}$ reduce the overall waiting time.

Thus, by swapping the order of the customer it reduced the total waiting time.

The running time of this algorithm = O(n log n).
Therefore, Greedy solution is the optimal solution.

3 Question 3

Ans

(a)

Iteration	Active node	d['A']	d['B']	d['C']	d['D']
0	-	0	∞	∞	∞
1	A	0	7	5	∞
2	C	0	6	5	25
3	B	0	6	5	9
4	D	0	6	5	9

(b) The running time is O (M log N)

(c)

Iteration	d['A']	d['B']	d['C']	d['D']
0	0	∞	∞	∞
1	0	6	5	10
2	0	6	5	9
3	0	6	5	9
4	0	0	0	0

(d) The running time is O(MN)

(e) $A \to B \to D \to C$

(f) Bellman Ford Algorithm

(g) It's a graph that the overall sum of the cycle becomes negative. Best way to detect if there is a negative weight cycle reachable from the given source is to use the Bellman Ford Algorithm.
First calculate the shortest distance which have at most one edge in the path. Then, calculates the shortest path with at most 2 edges and continue. After the i-th iteration of outer loop, the shortest path with most edges will be calculated. It can be a maximum V -1 edge on the simple path, so the other loop would runs V -1 time. If there is negative weight cycle, then another iteration would show the short route.

4 Question 4

Ans

(a) Dynamic programming is a technique that used to solve problems within $O(n^2)$ or $O(n^3)$ running time. It's also a general way to solve problem such like "divide and conquer", however, the sub problem will most likely get overlap. In the other word if the problem can be solved by combining optimal solution to non-overlapping sub-problem, then it's called divide and conquer. That is the reason quick sort and merge sort is not part of dynamic programming.

(b)

	Ø	M	A	N	I	A	C
Ø	0	0	0	0	0	0	0
M	0	1	1	1	1	1	1
U	0	1	1	1	1	1	1
I	0	1	1	1	2	2	2
C	0	1	1	1	2	2	3

(c)