a3

monthon.kra

16 November 2021

1 Question 1

Answer

(a) The maximum flow of the graph is $= 4 + 3 + 2 + 2 = 11$
The minimum cut is the line that cut through $AC \rightarrow AD \rightarrow BC \rightarrow AD$ and the value is also 11

(b)

2 Question 2

Answer

The table below will be using "Ite" as a shorten form of Iteration

	Ite 1(A)	Ite 2(B)	Ite 3(D)	Ite 4(E)	Ite 5(C)	Ite 6(D)
A	0	0	0	0	0	0
B	4	4	4	4	4	4
C	15	15	13	8	8	8
D	∞	5	5	5	5	5
E	∞	∞	∞	12	12	12
F	∞	∞	7	7	7	7

3 Question 3

Answer

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.

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.

```
Algorithms for the above explanation.

    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
```

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

4 Question 4

Answer

We can prove the Below algorithm using contradiction.

Proof: Let assume that our algorithm is not the best, for example let our answer be $s_1 = (g_1, g_2, ...g_k)$ and the better answer be $s_2 = (g'_1, g'_2, ...g'_{k-1})$
We will compare the gas stops and the outcome turnout to have 3 possibilities:

1) $g_i = g'_i$
2) $g_i > g'_i$
3) $g_i < g'_i$

For the 3rd case, g'_i is more than g_i this mean that the car will run out of gas as the previous gas stop is the same place for both answer.

Furthermore, let take a look at $g_i = g'_i$ for all k-1 gas stops, s_1 will makes a stop at g_k so that the car could reach the destination before running out of fuel. However for s_2 the car doesn't stop for gas and couldn't make it which indicate that s_2 is not correct.

We have proven that s_2 is incorrect. Therefore there is no better solution than our algorithm.

```
gasStop_Radar():
currentPos = starting Position;
while ( currentPos < endPos )
    ttt = calculate for the position that car will run out;
    if ( ttt < endPos ){
        find closest gas station before ttt;
        print ( "gas = max" );
        currentPos = current gas station;
        }
    else{
    currentPos = endPos;
    }
```

`The running time of this algorithm is = O(N)`

5 Question 5

Answer

Let S[t] be the sum of the maximum sum contiguous that is a sub sequence

that end at exactly index t.

For all $1 \leq t \leq n-1$:

$$S[t] = A[0]$$
$$S[t+1] = max(S[t] + A[t+1], A[t+1])$$

By using the above formula, we are now able to work out the sum of the optimal sub sequence for array A, which is the maximum for S[i] for all $0 \leq i \leq n-1$. We would need additional array V to store the starting index of the contiguous sub sequence of maximum sum as we needed to output both the starting and ending indices of an optimal sub sequence.

Below is an Algorithm that printed out both the starting and ending indices of an optimal sub sequence which running time is = O(n).

```
V[0] = 0;
S[0] = A[0];
start_Max = 0;
end_Max = 0;
max = S[0];

int i = 1;
while( i<= n-1) {
    if(S[i-1]>0)
        S[i] = S[i-1] + A[i];
        V[i] = V[i-1];

    Else
    S[i] = A[i];
    V[i] = i;
    if(S[i]>max)
        start_Max = V[i];
        end_Max = i;
        max = S[i];
    i++;
}
System.out.println(start_Max);
System.out.println(end_Max);
```

6 Question 6

Answer

Let S[0,n-1] be the length of the longest palindrome sub sequence.
Let X[0,n-1] be the input sequence of the length n.

If the first character and last character of X are same then

$$S(0, n-1) = S(1, n-2) + 2$$

Else

$$S(0, n-1) = max(S(1, n-1), S(0, n-2))$$

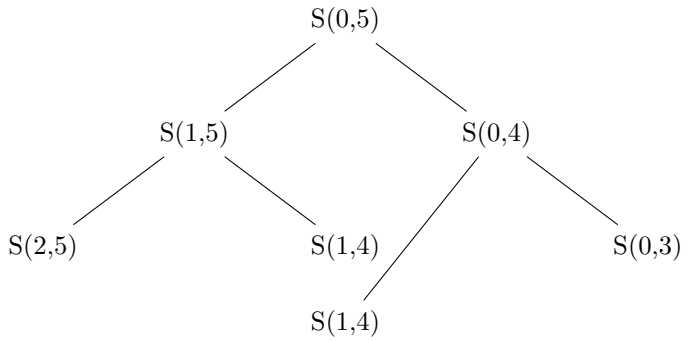
In addition, there are multiple possibility of cases that could occur and need to be handled.

`Recursive Solution`

```
S(i,i) = 1 for all indexes i in given sequence

if (x[i] != x[j]){
    S(i,j) = max(S(i + 1, j), S(i, j -1));
}
Else if ( j = i + 1 ){
    S(i,j) = 2;
}
Else{
    S(i,j) = S(i + 1, j - 1 ) + 2;
}
```

Base on the previous given algorithms, below is the recursion tree for a sequence that n = 6 with no matching character.



The above tree is incomplete. However, as we can see S(1,4) is being solved twice and if we draw a complete tree there will be many sub problems that being solved over and over again.

To prevent the solving of same sub problems over and over, we can create a new array 2d array S in the bottom up form.

This Algorithms has a running time of $O(n^2)$

```
for ( i = 0; i < n; i++){
    S[i][i] = 1;
}

for ( ttt = 2; ttt <= n; ttt++ ){
    for( i = 0; i < n - ttt + 1; i++ ){
        j = i + ttt-1;
        if ( input_str[i] == input_str[j] && ttt == 2){
            S[i][j] = 2;
        }
        else if (input_str[i] == input_str[j]){
            S[i][j] = S[i+1][j-1] + 2;
        }
        else{
            S[i][j] = max(S[i][j+1], S(i+1)[j]);
        }
    }
}
```