CIS 522: Assignment 7

Anubhav Shankar

Q1.) Minimize the Cost of Shipping

Ans.) According to the question,

Company A: $r * s_i$

where s_i -> weight of equipment shipped

r -> rate charged per shipment by Company A

Company B: r = k for four consecutive weeks

Our objective is to minimize the cost of shipment for the PC manufacturing company. The problem is similar to the weighted interval scheduling problem that can be tackled by Dynamic Programming.

This problem belongs to P-Class and can be solved in polynomial time.

Pseudocode ->

Int minC[] //min cost of shipment

Int r // rate

Final Int k // constant rate charged by Company B

Int s_i // weight of equipment shipped

Int weeks // a counter to count the number of weeks

for (i = 1 to min(weeks, k-1)){ //Check which of the freight companies has a cheaper starting rate

int opt[][] // a 2-D matrix to

minC[i] = minC[i-1] + r*si
for(j = 1 to i-1)
{
opt[i][j] = opt[i-1] [1]
opt[i][1] = 'Company A'

```
}
for(i = 4 to weeks)  //Company B has to be selected for 4 continuous weeks
{
minCost[i] = min(minCost[i-1] + si*r, minCost[i-4] + 4*c)
if((minCost[i-1] + si*r) < (minCost[i-4] + 4*c)){ //if the cost of Company B is greater than Company A
        for(j = 1 to i-1)
{
opt[i][j] = opt[i-1][j] // Try to optimize the maximum shipments for Company A before switching
}
opt[i][i] = 'Company A'
}
else
{
for(j = 1 to i-4)
{
opt[i][j] = opt[i-4][j]
}
opt[i][i-3] = opt[i][i-2] = opt[i][i-1] = opt[i][i] = 'Company B'
}
}
}
```

Example:
r = 1; k = 10
s_i = 11, 9, 9, 12, 12, 12, 12, 9, 9, 11
Schedule is:
Week 1: opt[1][1] = 'Company A'
Week 2: opt[2][2] = 'Company A'
Week 3: opt[3][3] = 'Company A'
Week 4: opt[4][4] = 'Company B'
Week 5: opt[5][5] = 'Company B'
Week 6: opt[6][6] = 'Company B'
Week 7: opt[7][7] = 'Company B'
Week 8: opt[8][8] = 'Company A'
Week 9: opt[9][9] = 'Company A'
Week 10: opt[10][10] = 'Company A'

Time Complexity -> As there are two nested for loops, the time complexity will be $O(n^2)$

References -> a.) https://core.ac.uk/reader/24066010

b.) https://courses.cs.washington.edu/courses/cse417/08wi/notes/11-dp-sched-complete.pdf

Q2.) Project selection problem

Ans.) This problem instance can be thought of as a Graph coloring problem with n-colors. So, a graph G is constructed with 'n' nodes representing a set number of jobs. We use the coloring method to represent nodes used by 2 jobs simultaneously.

By its definition, the graph coloring problem is a P class problem if two colors are involved, and that's how we'll approach it. A 3-color graph (or any k>2 coloring problem) problem is an NP-class problem with time complexity of O((n+1)!)

Pseudocode ->

Graph G

n processes/jobs

u,v jobs

while(at least one uncolored node exists)

{

add edge e = (u,v)

if(u requires process n[i] and v requires process n[i])

color u and v with same.

else

color with different colors.

}

Time Complexity -> The time complexity of the above algorithm is -> O(v + e)

References -> https://www.cs.cornell.edu/courses/cs3110/2012sp/recitations/rec21-graphs/rec21.html

Q3.) Calculating Median in a database

Ans.) According to the question, there are two databases with ‘n’ values each. Therefore, there are a total of 2n unique records.

Let ‘n’ be an even number, then the median value of the respective databases will be ceiling((n+1)/2).

Let there be two databases, A and B. A(i) and B(i) are the smallest elements of A and B, respectively. Let ‘k’ be the median value. Then A(k) and B(k) are A and B's two median values, respectively.

Now, let A(k) < B(k), then one can see that B(k) is greater than the first ‘k’ elements of A and the first k-1 elements of B. Therefore, Q(k) is at least the 2kth(2k≤n) element in the combined database. Therefore, all the elements greater than B(k) are greater than the median, and we can eliminate them. Similarly, the first 1/2n elements of A are less than B(k) and A(k). Thus, they are less than n+ 1 elements of the combined database. So, they cannot be the median. Now, we can find the median in the remaining set using recursion. Formally the algorithm is the following. We write recursive function median(n, p, q) that takes integers n, p, q and finds the median of the union of the two segments A[p+1;p+n] and B[q+1;q+n]. Let Q(n) be the number of queries the algorithm median asks (n, p, q).

Pseudocode ->

```
median(n, p, q){
if (n= 1){
return min(A(p+k), B(q+k))
}
else (k=1/2n){
if A(p+k)< B(q+k) then
return median(k, p+q/2n, q)
}
Else{
return median(k, p, q+p/2n)
}
}
```

Time Complexity -> As the median value keeps reducing by a factor of 1/2n, so, the algorithm is O(log n) in complexity.

Reference -> https://stackoverflow.com/questions/40106514/extracting-median-from-databases-of-n-entries-using-olog-n-queries

Q4.) Photocopying Problem

Ans.) This problem is like the weighted interval scheduling problem, which can be tackled using a Greedy approach. Hence, this problem is firmly in the P-class with a polynomial-time algorithm.

Pseudocode ->

n-> number of jobs/customers

s-> starting time of a job i

f-> time taken to complete a job i

w-> weight of the job i //Basis the importance of the customer

wRatio-> array which contains weight of the associated job i

```
for(i = 1 to n)

wRatio[i] = w[i]/t[i]

sortDesc(wRatio)

c[0] = 0

for(i=1 to n)

{

c[i] = c[i-1] + t[i]

totalTime = totalTime + (w * c[i])

}

return totalTime

print sortDesc(wRatio)
```

Time Complexity -> O(n log n)

Reference -> https://www.geeksforgeeks.org/weighted-job-scheduling-log-n-time/

Q5.) Communications Problem

Ans.) This problem can be modeled as a Network Flow problem. So, using the Ford-Fulkerson algorithm we can find the path where no two selected paths share any edge.

This is a P-Space problem for which a polynomial time algorithm can be found.

Input -> G(V,E)

Algorithm -> Let the capacity © of each edge, e ε E be 1. Let 's' and 't' be the source and sink nodes respectively. We make each of these paths carry flow of 1 for each edge on any of the paths and 0 on all the other edges. We use Ford Fulkerson algorithm to find the maximum flow from source s to sink t.

Subsequently, we go about reducing the capacity by 1 leading to the edge, e_i being assigned 0 and thus, rendering it useless. Thus, the max-flow(m) is equal to the maximum number of edge-disjoint paths i.e. the min-cut.

Time Complexity -> O(mE).

References -> a.) https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/

b.) http://www.bcc.bas.bg/BCC_Volumes/Volume_52_Special_A_2020/BCC-52-A-2020-192-196-Sapundzhi-255.pdf

Q6.)

Ans.) Similar to Question 5, this problem can be modeled as a Network Flow problem where each project is a Node, and the utility points are the capacities of each connecting edge. Hence, this problem belongs to the P-space for which a polynomial-time algorithm can be formulated.

From min-flow-max-cut theory, we know that each cut is defined by a subset of nodes. The nodes in this network correspond to a project, except the source and sink nodes. The min-cut will choose the projects near the source. For each project u that depends on a project v, we add an infinite (or no edge) capacity arc from u to v, so that every cut that includes u but not v has infinite capacity, thereby putting it out of contention.

For projects with negative revenue, we set a cost for including them by making a cut with capacity(utility points) equal to the negative revenue from the project node to the sink. We pay this cost if and only if the project node is on the source side, i.e., the project is chosen. For projects with positive revenue, we add an arc from the source to the project node with a capacity equal to revenue. This biases the objective in the sense that the min-cut is no longer equal to the total revenue, but differs by a constant value, and the mechanism has a positive effect by adding its revenue to the min-cut. The max flow is the maximum points that can be utilized and found by executing a BFS, similar to Ford Fulkerson.

Time Complexity -> O(max flow * E)

Q7.)

Ans.) As we need to hire at most ‘k’ counselors and have at least one of them qualified in each sport then we’d need to solve this using a Vertex Cover algorithm which is a NP-Complete class problem.

Let there be a Graph, G = (V,E) such that each sport ‘s’ is an edge and each counsellor ‘c’ are the vertices. According to vertex cover, a counselor ‘c’ is qualified in sport ‘s’ if and only if each edge, a sport ‘s’ in this case has an endpoint on a vertex ‘c’.

If ‘k’ counselors are qualified in all sports then the vertices in G are such that each edge has an end in at least one of them so that they form a vertex cover of size ‘k’.

With an NP-Complete classification, the problem has a polynomial-time certifier.

In this case, the certifier is the subset of all counselors that are qualified in all sports. If the certifier returns a “Yes” that means there exists a vertex cover of size ‘k’.