**COMP3121 Sample Exam Practise**

**Trig Table:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Radian** | **0** |  |  |  |  |  |  |  |  |  | **2** |
| **Degree** | **0** | **30** | **45** | **60** | **90** | **120** | **135** | **150** | **180** | **270** | **360** |
| **Sin** | 0 |  |  |  | 1 |  |  |  | 0 |  | 0 |
| **Cos** | 1 |  |  |  | 0 |  |  |  |  | 0 | 1 |
| **Tan** | 0 |  |  |  | Undefined |  |  |  | 0 | Undefined | 0 |

**In form:**

1a)

1b)

--------------------------------------------------------------------------------------------------------------------------------------**Final Exam Answers:**

1)

1. Let A and B be two sequences of numbers such that A\*A = B where \* denotes the convolution of sequences.

(a) Let the length of sequence A be equal to a and the length of sequence B be equal to b. Express a in terms of b. (5 pts)  
(b) Find ALL sequences A such that A\*A = (4,4,-3,-2,1). (20 pts)

**ANS:**

We are given two sequences of numbers A and B where it is given that the convolution of sequence A with itself gives the sequence B.

1. If we let the length of sequence A = a, and the length of sequence B = b, we can express the length a in terms of b quite simply. In other words, if a = 2 for example then one possible sequence that A can be expressed is {1, 1} which would form the polynomial:

P(x) = 1 + x

The convolution of that with itself would be:

(1 + x ) \* (1 + x ) = (1 + x)^2 = 1 + 2x + x^2 which is length 3.

Hence, we can observe that the length a determines the highest degree power of x which would be x^(a – 1) and since we are doing the convolution of sequence A with itself the resulting degree power in sequence B would be x^[(a – 1) + (a – 1)] = x^(2a – 2). We then simply add 1 to the highest power of x in B to get the length b which would be:

b = 2a – 2 + 1 which can be simplified to: **b = 2a – 1**.

We are required to find all sequences A such that the convolution of A with itself will result in B being {4, 4, -3, -2, 1} which has length b = 5.

Expressing B as a polynomial: P(x) = 4 + 4x – 3x^2 – 2x^3 + x^4

Rearranging our equation in part 1a, we can get the length a = (b + 1)/2 and substituting our value b = 5 we get a = 3. Hence our sequence A has to be of the form {a, b, c} for some values x, y, z such that they result in B = {4, 4, -3, -2, 1}.

We can express our A as a polynomial P(x) = a + b\*x + c\*x^2 and the convolution of itself would be:

(a + bx + cx^2) \* (a + bx + cx^2) = (4 + 4x – 3x^2 – 2x^3 + x^4)

=> (a^2 + 2abx + 2acx^2 + b^2x^2 + 2bcx^3 + c^2x^4) = (4 + 4x – 3x^2 – 2x^3 + x^4)

=> a^2 + x(2ab) + x^2(2ac + b^2) + x^3(2bc) + x^4(c^2) = (4 + 4x – 3x^2 – 2x^3 + x^4)

Hence, we must solve for the above coefficients:

a^2 = 4, 2ab = 4, 2ac + b^2 = -3, 2bc = -2 and c^2 = 1.

We can derive the coefficients from the above values by setting up a system of linear equations as it is in the form of Ax = b (i.e. A is a Vandermonde matrix). Note that this Vandermonde matrix is appropriate as it can result in more than one solution (which would be our possible forms of sequence A). This can be solved by inverting a constant matrix and then multiplying the matrix by the vector formed from the pointwise multiplications, which again only multiplies these results by scalars, to give the final polynomial and resultant coefficients which would be our sequence A.

2)

You are late with n assignments a(1), a(2),..., a(n)  that were all due today. Assignment a(i) accrues a penalty of p(i) points per day and takes t(i) days to finish. At any moment, you can work on one assignment only. Determine the order in which you should work on your assignments in order to minimise the total number of points lost. Justify the correctness of your algorithm. Your algorithm should run in time O(n log n) and you should explain why your algorithm runs in time O(n log n). (25 pts)

**ANS:**

We are given n assignments that are all due today. An assignment a(i) accrues a given penalty of p(i) points per day and takes t(i) days to finish. We are required to minimise the total number of points lost and need to justify the correctness of our algorithm, as well as explaining why it runs in time O(n\*log(n)).

In order to minimise the total number of points lost overall, we need to find how long each project will take AND assuming that we can only complete 1 assignment at a time, the optimal order in which we need to complete the assignments. The first order of business would be to obtain the overall penalty m(i) over the assignement’s lifetime which would be the penalty p(i) per day multiplied by its respective duration t(i).

i.e. m(i) = p(i) x t(i)

Once we have obtained all m(i) for each a(i), we can then sort this in decreasing order and commence with the assignment a(i) that results in the highest penalty p(i) over the time duration t(i) (which would be the first element in the list). We do this until we have gone through all of our assignments.

Optimality and correctness is easy to prove. We are sorting both our p(i) and t(i) lists such that we are obtaining the assignment with the highest penalty which has a long duration (days) to complete **compared to the other assignments**. The key here is that every time we choose to complete an assignment in the m(i) list, we are minimising the overall penalty as we go down through the m(i) list. Assuming we did not do this, if we simply chose the project with the largest penalty or longest duration, we will most likely be missing a project that lies higher on our m(i) list such that the overall penalty accumulated will be larger.

This algorithm runs in time O(n\*log(n)) as we are multiplying our p(i) and t(i) for each a(i) which will take O(n) time. We are then sorting our m(i) list in decreasing order which will have time complexity O(n\*log(n)) if we use mergesort to sort the list. Hence, the overall time complexity will be O(n\*log(n)) as required.

3)

Solve the following problem using Dynamic Programming. You are travelling along the Elbonian coast with cities c(0), c(1), c(2),..., c(n) on the shore, in that order. You are starting in city c(0) where a famous spa is, and need to reach the airport situated in city c(n); thus, you will be going through all cities c(0), c(1),..., c(n-1), c(n) in that order. In each city you must swap the animal you are riding on and the choices are a camel, a horse, a mule and a donkey, denoted C,H,M,D respectively. However, each city has its own rules what kind of animal exchanges are allowed. For example, in some of the cities you can swap a horse only for a donkey or a mule, in some of the cities you can swap a camel only for another camel or a horse, and so on. You know all the rules of all the cities c(1),...,c(n), expressed by a function R(i,a,b) given by R(i,a,b)=1 if in city c(i) one can swap animal a for animal b and zero otherwise (a and b belong to the set {C,H,M,D}). You also know the speed v(a) (a = C,H,M,D) of each of the four animals, as well as the distances d(i) between cities c(i-1) and c(i) for all i = 1,2,...,n. You have to design an algorithm for computing the minimal amount of travel time needed to travel from city c(0) all the way to city c(n) as well as an animal swapping strategy which allows you to travel in such a minimal amount of time. In the starting city c(0) you can choose to start your journey riding any of these four animals. To solve this problem do the following tasks in this order:  
(a) Formulate precisely the subproblems you are going to solve. (10 pts)  
(b) Write the exact recursion equations. (12 pts)  
(c) Explain how the solution to the original problem is obtained from the solutions of the subproblems you have defined. (2 pts)  
(d) Estimate the asymptotic run time of your algorithm in terms of the number n of cities. (1 pt)

**ANS:**

We are required to design an algorithm for computing the minimal amount of travel time needed to travel from city c(0) all the way to city c(n) utilising an animal swapping strategy which will allow us to achieve this minimal time. We are required to swap our animal at every city however we are told that we can swap for the same animal e.g. horse for a horse if the given city allows for it.

a)

Let t(c(i), c(j)) be the minimum time taken to travel from city i to city j (next to each other) where c(i) and c(j) lie between the path from c(0) to c(n).

The subproblems that we need to solve will be of the form of finding the minimum time spent travelling from city i to city j, given we know the rules R(i, a, b) and R(j, a, b) for c(i) and c(j) in order to achieve opt(c(i), c(j)) which is the optimal time taken to travel between a given city i to city j. Also, we need to know the optimal time taken to the previous city (explained in b).

b)

Clearly, the base case opt(c(0), c(1)) = t(c(0), c(1)) and the recursion is given by:

Opt(c(0), k) = min{opt(c(0), c(k – 1)) + t(c(k – 1), c(k)) : k – 1 is the previous city and k is the city we want to get to}

By taking R(i, a, b) for each city, we know the animals that are available there and need to start with the animal at city c(0) such that we can achieve our optimal time. Hence, we need to construct a table and add in the animals available to use in real-time at every city and choose the optimal animals at EACH city before a given city k when knowing R(k, a, b) (which will allows us to achieve the most optimal time in the end).

c)

Let OPT be the overall optimal value of the sum of the times taken to travel between the cities and let OPT(R(i, a, b)) be our table of optimal animals to travel on then

OPT = min{opt(c(0), c(n))}

OPT(R(i, a, b)) = Order{opt(R(0, a, b)) …. opt(R(n, a, b))}

I.e. the sequence of animals OPT(R(i, a, b)) is obtained by backtracking starting from c(n) and obtaining each previous city c(n – 1) by applying recursion for all i = 0 … n.

This will hence result in the most optimal order available and essentially the minimal time taken to travel all the way from city 0 to city n.

d)

The overall asymptotic runtime would be O(n) as we are traversing through the list of cities and every time we input our chosen animal into the table, it is done in linear time O(1).

4)

(a) Describe in detail how max flow algorithms can be used to find a maximum matching in bipartite graphs. (5 pts)  
(b) You are given a sequence A(1),..., A(N) of N distinct positive integers and an integer M. You have to determine if it is possible to assign to each element A(i) a distinct integer B(i) larger or equal to 2 and smaller or equal than M such that for all i between 1 and N integer A(i) is divisible by integer B(i). Different A(i) must be assigned different B(i). (20 pts)

**ANS:**

a)

We can use max flow algorithms such as the Edmonds-Karp algorithm to find the maximum matching in bipartite graphs quite easily by consider the bipartite graph to be a network of vertices on the left and right side and the edges connecting those vertices as either having a constant capacity or own capacities (depending on the problem). We can add a super source on the left vertices and super sink on the right vertices which would have either infinite edge capacities (or constant capacities depending on the actual problem) connecting to those vertices from the super source or super sink. We can then apply our max flow algorithm to solve the problem.

b)

We are given a sequence of N distinct positive integers and an integer M. We have to determine if it is possible to assign each element A(i) in our sequence a distinct integer B(i) larger or equal to 2 and smaller or equal to M such that for all i between 1 and N, integer A(i) is divisible by integer B(i). Additionally, different A(i) must be assigned different B(i).

We can construct a bipartite graph with our different A(i) as being vertices in the graph and the vertices on the right as being the respective B(i). We can then add edges with