# ICS 340

# Spring 2021

# 8:30 PM Monday 03 May – 11:59 PM Wednesday 05 May

# Take Home Portion of Final Exam

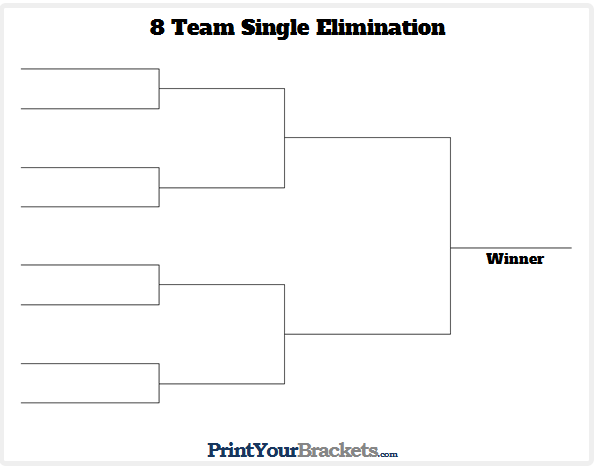
# (100 points)

Nalongsone Danddank Student ID : 14958950 StarID: jf3893pd

Email: [nalongsone.danddank@my.metrostate.edu](mailto:nalongsone.danddank@my.metrostate.edu)\

*Notes:*

1. *Please answer the question on the page containing the problem, and any blank page after that page (for problem 2). If your answer doesn’t fit in the space provided, it might be too long. (Exception: If you want to illustrate your solution with a drawing, it may be reasonable to break into another page. But remember that one example does not serve to prove a general theorem.)*
   * *In most cases, the problem is defined on one page, and each part of the problem has a separate page on which I want you to write your answer.*
2. ***It is vital to show your work****. Correct answer without work shown will lose substantial credit.*
3. *If you quote (or nearly quote) any sources, reference those sources (for the textbooks and my slides, just say “CLRS Section m”, “P&M, Section n”, or “Topic 3 slides”).*
4. *The due date is 11:59 PM on Wednesday, May 5th. The dropbox remains open until 12:15 AM on May 6th. This is because I don’t want anyone to get zero points by being a couple minutes late. You lose 1 point per minute late. If you are 15+ minutes late, you don’t get credit.*
5. *Remember that the textbooks are available freely online:*
   * *Poole: https://artint.info*
   * *Cormen:* [https://ebookcentral.proquest.com/lib/metrostate/detail.action?docID=3339142](https://nam02.safelinks.protection.outlook.com/?url=https%3A%2F%2Febookcentral.proquest.com%2Flib%2Fmetrostate%2Fdetail.action%3FdocID%3D3339142&data=02%7C01%7Cmichael.stein%40metrostate.edu%7C6dc02dfa117644bfd61e08d7db164a67%7C5011c7c60ab446ab9ef4fae74a921a7f%7C0%7C0%7C637218761090371909&sdata=BIfi9UVjoWCY3MhT1Ip2yLsq000y2Ya5Ed3K6db%2BZ24%3D&reserved=0)
6. *[25 pts]* Suppose that you are in charge of a single elimination, 2-person, head-to-head tournament for some game with *n* players. If *n* is an exact power of 2, this is not a problem. You have n/2 games in the first round, the winners play in n/4 games in the second round, etc. But if the number of players is not a power of 2, it’s not this simple. Someone, somewhere needs to get a bye.[[1]](#footnote-0) See samples below from printyourbrackets.com. In the right-hand 10-team tournament, 6 teams get byes in the first round.[[2]](#footnote-1)

****

****

.

Part A: [10 pts] One way to address this problem is to give some number of players a bye in the first round, so that after the first round the number of players remaining is an exact power of 2. How many byes need to be given out in the first round for a tournament of size *n* players? Write your answer as a function of n.[[3]](#footnote-2) In case you get a wrong answer, you need to explain your reasoning to get partial credit.

Answer:

when the number of size n player, is not an exact power of 2, like n = 10 = 23 + 2, so we have:

If | log2 n - floor(log2 n) | > 0 :

the 0th round: zero\_round(n) = 2 \* (n - 2 floor(log2n))

the first round: first\_round(n) = 2 floor(log2n) - zero\_round(n) / 2

= 2 floor(log2n) - (n - 2 floor(log2n))

= 2 floor(log2n)+1 - n

**= 2 ceiling(log2n)  - n**

Example:

zero\_round(10) = 2 \* (10 - 2 floor(log210)) = 4 players

first\_round(10) = 2 ceiling(log210)  - 10 = 6 players get byes.

Part B: [15 pts] As an alternative, suppose you decide that you want to minimize the number of byes by spreading the byes out among all the rounds of the tournament. What is the minimum number of byes you will need to give out in the tournament? How can you do that? Explain.[[4]](#footnote-3)

Solution:

I like to use recursive function to find out the minimum number of byes that will need to give out in the tournament. For each round which number of team is odd we have to left 1 team have a bye.

Example: 10 => 10/2 = 5 (1 team have a bye for second round) => 4/2 +1 = 3 (again 1 team have a bye for 3rd round) => 2/2 + 1 then the last one is => 1+1 = 2 => 2/2=1. we can see that we need 1 bye for each round is odd number like 5 and 3 when divide by 2. So the total number is 2 byes.

Pseudo-code (Java syntax):

**int** *total\_min\_byes* = 0;// count all of the byes of each round that need bye.

// take recursion for find out the total number of minimum of byes.

**void** find\_byes(**int** n, **int** b) {

**if** (n == 1) **return**;

**else** **if** (n % 2 == 0) {

n += b;

**if** (n % 2 == 1) {

//when the result of division is odd count for minimum of bye

*total\_min\_byes*++;

*find\_byes*(n / 2, 1);

} **else** *find\_byes*(n / 2, 0);

} **else** **if** (n % 2 == 1) {

n += b;

**if** (n % 2 == 1) {

//when the result of division is odd count for minimum of bye

*total\_min\_byes*++;

*find\_byes*(n / 2, 1);

} **else** *find\_byes*(n / 2, 0);

}

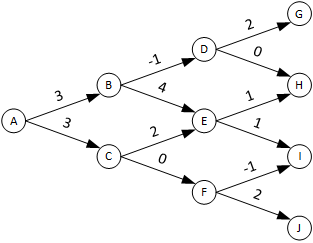
}

System.***out***.println("the total number of minimum of bye: "+*total\_min\_byes*);

Example:

find\_byes(**10**, **0**) -> the total number of minimum of bye: 2

1. [25 pts] Consider a directed acyclic graph like the following. Call the node on the left the “root” node of the graph and the nodes on the right the “leaf” nodes, even though it’s not a tree. Describe a polynomial time algorithm that finds the minimum edge length to get from the root to any of the leaf nodes For instance, in the graph below your algorithm should identify ABDH or ACFI as the minimum cost path. Use dynamic programming, and as part of your algorithm describe the recurrence that you use in your algorithm (compare this to the Coin Changing or Edit Distance recurrences from the Topic 3 slides). Write solution on the bottom of this page and on the next page.

** Solution: I would like to use Single Source shortest Paths in directed acyclic graph, it similar to Bellman Ford algorithm. Because there have no negative weight cycle in directed acyclic graph. we just only run one time instead of run V time(total number of Node in Graph), so The time complexity is Θ(V+E) = Θ(n) is the linear time. And the Start node is A then go to the leaf nodes like, G, H, I, J. Finally, Comparing all these path of the leaf to get a shortest path of them to be the answer.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Node | Start | | Rel. AB | | Rel. AC | | Rel. BD | | Rel. BE | | Rel. CE | | Rel. CF | | Rel. DG | | Rel. DH | | Rel. EH | | Rel. EI | | Rel. FI | | Rel. FJ | |
| s.d | π | s.d | π | s.d | π | s.d | π | s.d | π | s.d | π | s.d | π | s.d | π | s.d | π | s.d | π | s.d | π | s.d | π | s.d | π |
| A | 0 | ∅ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| B | ∞ | ∅ | 3 | A |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| C | ∞ | ∅ |  |  | 3 | A |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D | ∞ | ∅ |  |  |  |  | 2 | B |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| E | ∞ | ∅ |  |  |  |  |  |  | 7 | B | 5 | C |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| F | ∞ | ∅ |  |  |  |  |  |  |  |  |  |  | 3 | C |  |  |  |  |  |  |  |  |  |  |  |  |
| **G** | ∞ | ∅ |  |  |  |  |  |  |  |  |  |  |  |  | 4 | D |  |  |  |  |  |  |  |  |  |  |
| **H** | ∞ | ∅ |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 | D | 6 | E |  |  |  |  |  |  |
| **I** | ∞ | ∅ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 6 | E | 2 | F |  |  |
| **J** | ∞ | ∅ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 5 | F |

|  |  |  |
| --- | --- | --- |
| Node | Start | |
|  | |
| s.d | π |
| A | 0 | ∅ |
| B | 3 | A |
| C | 3 | A |
| D | 2 | B |
| E | 5 | C |
| F | 3 | C |
| **G** | **4** | **D** |
| **H** | **2** | **D** |
| **I** | **2** | **F** |
| **J** | **5** | **F** |

In the final table we can see the final shortest path from node A to G, H, I, and J.

Then we sort the length of path :

ABDH = 2 ; ACFI = 2 ; ABDG = 4 ; ACFJ = 5

Then we get the shortest path is ABDH or ACFI.

As above table which following from Single Source shortest Paths in directed acyclic graph, it similar to Bellman Ford algorithm but running only one time like below:

Relax(u, v, w):

if **(v.d > u.d + w(u, v):**

v.d = u.d + w(u, v)

v.Π = u

DAG\_shortest-Paths(G, w, s):

topologically sort the vertices of G

initialize\_single\_source(G, s)

for each vertex u, taken in topologically sorted order

for each vertex *v* in G.Adj[u]

Relax(u,v,w)

Answer : min(DAG\_shortest-Paths(G, w, s) of the leaf nodes)

This idea is similar to solving the Coin Changing or Edit Distance recurrences that using Dynamic programming bottom up solution. If we describe the coin changing or edit distance by using graph to present the problem. It would like similar with this graph, Single Source shortest Paths in directed acyclic graph, that graph has start with a single node and have the leaf nodes so we could find all solution following by condition by using DAG shortest-Paths to solve them.

1. *[24 pts] NP Completeness Proof:* Define the Exact 0-1 Knapsack Problem X01K(W,V) as follows: You are given *n* items with weights w­1­, w­2­, …, w­­­n; and values v­1, v2, …, vn; the capacity of the knapsack is W. Is there a subset S of the items such that exactly and exactly?[[5]](#footnote-4)

The X01K(W,V) problem is NP-Complete. Show this in 4 parts. To save writing, you may refer to this problem as X(W,V) in your solution.[[6]](#footnote-5)

Part A: *[6 pts]* Show that X01K(W,V) ∈ NP by describing an algorithm that verifies a purported solution to the X01K(W,V) problem. Show that your algorithm runs in polynomial time.

Solution: Using Dynamic Programming Algorithm like below pseudocode:

Result: S[V, W]

S[0, w] := 0, ∀ wi to wn

S[i, 0] := 0, ∀ i = 0 to V

for i := 1 to V do:

for w := 1 to W do:

if wi <= w then:

S[i, w] = max(S[i-1, w - wi] + vi, S[i - 1, w])

else:

S[i , w] = S[i - 1, w]

end

end

end

return S[V, W]

The algorithm create a matrix of size( (V+1) \* (W+1)). and we can see that it run with only 2 loop, V \* W so O(V\*W), there are n number in V and W and n == V == W, then loop through the V and W in n time. And this is the Exact 0-1 knapsack problem it is clear here the time complexity is polynomial time.

Part B: *[7 pts]* Show how to reduce one of the following NP-Complete Problems (Circuit-Sat, Sat, Clique, Vertex Cover, Dominating Set, Hamiltonian Cycle (on a directed or undirected graph), Set Cover, or Subset Sum) to the X01K(W,V) problem. That is, show how to map the known NP-Complete problem to a set of items and a knapsack on which you want to solve the X01K(W,V) problem. Show that you can do this mapping in polynomial time.

Solution:

I would to reduce Subset Sum problem to the X01K(W,V) problem: Recall subset-sum is defined as follows: Given a set A of integers and a target number t, find a subset B is subset of A such that the members of B add up to exactly t. Let s be the sum of member of A. Feed A’ = A ∪{ s - 2t} into X01K(W,V). Accept if and only if X01K(W,V) accepts.

This reduction clearly works in polynomial time.Part C: *[6 pts]* Given a solution to your known NP-Complete problem, show how to use that solution to generate a solution to the associated X01K(W,V) problem.

Solution:

I would like to use Double Knapsack problem that solve by Dynamic programming in NP-complete to generate a solution to the associated X01K(W, V) problem.

Using a recursive solution to try out all the possible ways of filling the two knapsacks and choose the one give the maximum weight. Them, to optimize the above idea, we need to determine the states of dynamic programming that we will build up out solution upon.

This can be represented like below:

DKP(i, w1\_r, w2\_r) = max( DKP[i+1][w1\_r][w2\_r], arr[i] + DXP[i+1][w1\_r - arr[i]] [w2\_r],

arr[i] + DXP[i+1][w1\_r][w2\_r - arr[i]])

The time complexity is : O(n\*W1\*W2).

We found that the Double Knapsack problem can be solved using a 3 dimensional dynamic programming with a recurrence relation. And we also know that X01K(W, V) problem is can be solved by using a 2 dimensional dynamic programming and time complexity is O(n\*W) or O(V\*W).

=> DKP(i, w1\_r, w2\_r) > X01K(W, V)

Thus, Double Knapsack problem that solve by Dynamic programming in NP-complete to generate a solution to the associated X01K(W, V) problem.

Part D: *[6 pts]* Given a solution to the X01K(W,V) problem, show how to use that solution to generate a solution to your known NP-Complete problem.

Solution:

I would like to use Fractional Knapsack problem that when giving a solution to the X01K(W, V) solve by Dynamic programming in NP-complete to generate a solution to this problem.

In Fractional Knapsack, we can break items for maximizing the total value of knapsack. But in the 0-1 Knapsack problem, we are not allowed to break items and we either take the whole item or don’t take it.

Then we can use an efficient solution to solve the Fractional Knapsack problem like using Greedy approach. The idea is to calculate the ratio value/weight for each item and sort the item on basis of this radio. Then take item with the highest ratio and add them until we can not add the next item as a whole and at the end add the next item as much as we can. Which will always be the optimal solution to this problem.

Because the main time taking step is sorting, so the time complexity is O(n log n)

Finally, as we solved and proof on above part of the question, we found that X01K(W,V) problem of time complexity is O(V\*W) or O(n2), polynomial time that is also NP-Complete problem. Then Fractional Knapsack problem time complexity is O(n log n) is less them O(n2).

Hence, we can show that, when a solution to the X01K(W,V) problem can generate a solution of Fractional Knapsack problem.4. [*25 pts*] Programming problem

I have uploaded a file called DelivFX.java to D2L. This is a scaffolding (just like the initial DelivA, B, C, and D files) for the final exam program. I have also uploaded a modified version of Prog340.java that includes this file in its menus.

Using **your** latest code, write DelivFX to find every pair of Nodes that in a graph that have Edges going directly in both directions between them. For each such pair of Nodes, print

**There are edges in both directions between *<name of one node>* and *<name of other node*>.**

See Appendix A for possible output form running this on the F2dd.txt test file (the order of the lines is not relevant). Do all your work in DelivFX.java (with one possible exception, see below). Submit only the source file DelivFX.java to the Final Exam Part B dropbox. I will combine it with my modified Prog340.java and the other files from the last running program you sent me and run this program.[[7]](#footnote-6) [[8]](#footnote-7)

***Exception***: If you modified your Prog340.java file, you can send me that, too. You’ll have to put in the modifications for it to run DelivFX.java yourself. Appendix B lists the changes I made to the original Prog340.java file.

Solution:

I know that you have a mistake to give the solution on D2L file.

However, I have figure out my solution of finding pairs of edges implement by Stack data structure. With only 2 loop and Time complexity is O(E^2). Then, it print out the same result but the order of the line is reverse from your solution.

If interesting, please go to check on my “DelivFX.java” which I submit.

And I show the part of codes on next page:

// FInd pairs of edges implement by Stack. Time complexity is O(E^2)

**public** **void** findPairsOfEdges\_ping() {

// convert edges ArrayList to stack data structure

Stack<Edge> edges = **new** Stack<Edge>();

edges.addAll(g.getEdgeList());

// Iterate through each Edge in the graph.

**while** (!edges.empty()) {

Edge edge = edges.pop();

Node head = edge.getHead();

Node tail = edge.getTail();

**if** (!edges.empty()) {

// Look for Edge in opposite direction.

**for** (Edge e : edges) {

**if** (head.getName().contentEquals(e.getTail().getName())

&& tail.getName().contentEquals(e.getHead().getName())) {

// Print out the result.

String printString = "There are edges in both directions between " + e.getHead().getName()

+ " and " + e.getTail().getName();

System.***out***.println(printString);

output.println(printString);

// if found it break this loop.

**break**;

}

}

}

}

}

***Appendix A: Output from running this program on F2dd.txt***

Recall that the order of the lines printed is not important.

**There are edges in both directions between Chicago and Boston.**

**There are edges in both directions between Dallas and Atlanta.**

**There are edges in both directions between Dallas and Chicago.**

**There are edges in both directions between Denver and Chicago.**

**There are edges in both directions between Denver and Dallas.**

**There are edges in both directions between LosAngeles and Dallas.**

**There are edges in both directions between LosAngeles and Denver.**

**There are edges in both directions between Miami and Atlanta.**

**There are edges in both directions between Miami and Boston.**

**There are edges in both directions between Miami and Dallas.**

**There are edges in both directions between Minneapolis and Boston.**

**There are edges in both directions between Minneapolis and Chicago.**

**There are edges in both directions between Minneapolis and Dallas.**

**There are edges in both directions between Minneapolis and Denver.**

**There are edges in both directions between NewYork and Boston.**

**There are edges in both directions between NewYork and Chicago.**

**There are edges in both directions between NewYork and Miami.**

**There are edges in both directions between SanFrancisco and Denver.**

**There are edges in both directions between SanFrancisco and LosAngeles.**

**There are edges in both directions between Seattle and Denver.**

**There are edges in both directions between Seattle and LosAngeles.**

**There are edges in both directions between Seattle and Minneapolis.**

**There are edges in both directions between Seattle and SanFrancisco.**

**There are edges in both directions between Washington and Atlanta.**

**There are edges in both directions between Washington and Chicago.**

**There are edges in both directions between Washington and NewYork.**

***Appendix B: Changes made to the original Prog340.java file.***

If you didn’t change Prog340.java, don’t submit it! I will run it with my own version of the file. Changes are in **bold maroon**

Line 25: Declared a new MenutItem

JMenuItem delivA, delivB, delivC, delivD**, delivFX**;

Limes 108 & 109: Added a new MenuItem

delivA = new JMenuItem("Run Deliv A");

runMenu.add(delivA);

delivB = new JMenuItem("Run Deliv B");

runMenu.add(delivB);

delivC = new JMenuItem("Run Deliv C");

runMenu.add(delivC);

delivD = new JMenuItem("Run Deliv D");

runMenu.add(delivD);

**delivFX = new JMenuItem("Run Deliv FX");**

**runMenu.add(delivFX);**

Around line 150 or so: Added a new Action listener to the list of the action listeners.

// Here are the Action Listeners for the Menu Items

readFile.addActionListener( new ActionListener() {

public void actionPerformed( ActionEvent e ) {

g = new Graph();

readGraphInfo(g);

}

});

exit.addActionListener( new ActionListener() {

public void actionPerformed( ActionEvent e ) {

System.out.println("Goodbye");

System.exit(0);;

}

});

delivA.addActionListener( new ActionListener() {

public void actionPerformed( ActionEvent e ) {

new DelivA( inputFile, g);

}

});

delivB.addActionListener( new ActionListener() {

public void actionPerformed( ActionEvent e ) {

new DelivB( inputFile, g);

}

});

delivC.addActionListener( new ActionListener() {

public void actionPerformed( ActionEvent e ) {

new DelivC( inputFile, g);

}

});

delivD.addActionListener( new ActionListener() {

public void actionPerformed( ActionEvent e ) {

new DelivD( inputFile, g);

}

});

**delivFX.addActionListener( new ActionListener() {**

**public void actionPerformed( ActionEvent e ) {**

**new DelivFX( inputFile, g);**

**}**

**});**

this.add(menuBar);



1. If you wonder “what does this have to do with Computer Science?”, this problem was studied extensively by Donald Knuth and written about in his classic 1990s 4-volume work *The Art of Computer Programming*. [↑](#footnote-ref-0)
2. As another example, the NCAA basketball “March Madness” tournament has 68 teams in it. 8 of them play in the 0th round, the so-called “play-in round”. I think of this as the first round for the purposes of this problem. Essentially, the NCAA gives out 60 byes in the beginning of the tournament. [↑](#footnote-ref-1)
3. You might find the use of floor and ceiling notations helpful in your answer. [↑](#footnote-ref-2)
4. I am ***not*** asking for a formal proof that you have a minimum number of byes. I want you to figure out the minimum, and then explain an algorithm that gives you some small number *x* of byes in a tournament with *n* players. [↑](#footnote-ref-3)
5. Note that I am asking for strict equality here, not weight less than or equal to W and/or value greater than or equal to V. Asking for exact equality makes the problem a lot easier. [↑](#footnote-ref-4)
6. But it’s important to remember that this is for the 0-1 Knapsack problem. The fractional Knapsack problem is in P. [↑](#footnote-ref-5)
7. I did this problem in about 20 minutes. I found it pretty easy. And it’s independent of what you did for every other Deliv. [↑](#footnote-ref-6)
8. Hint: It may help you to remember that the default String comparator (**string1.compareTo( string2 )** ) compares Strings lexicographically already. [↑](#footnote-ref-7)