Owen McGee Design and Analysis of Algorithms Midterm Exam

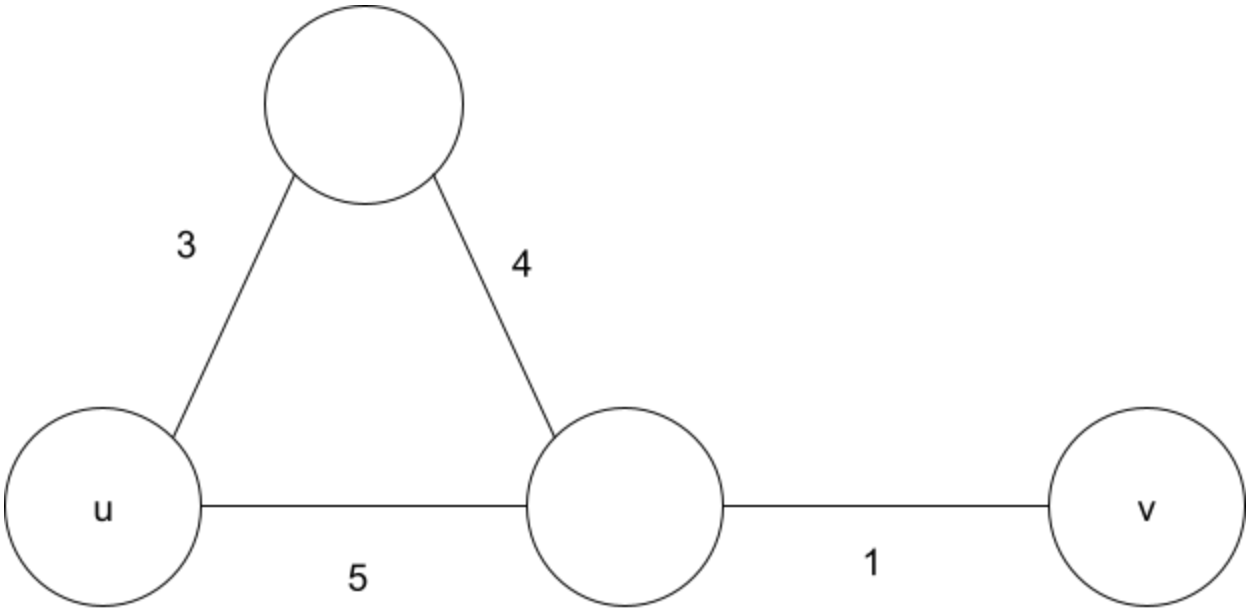
1:

On my honor, I have neither given nor received any unauthorized aid on this exam.

Signed: Owen Daniel McGee

2.A:

False. Counterexample:

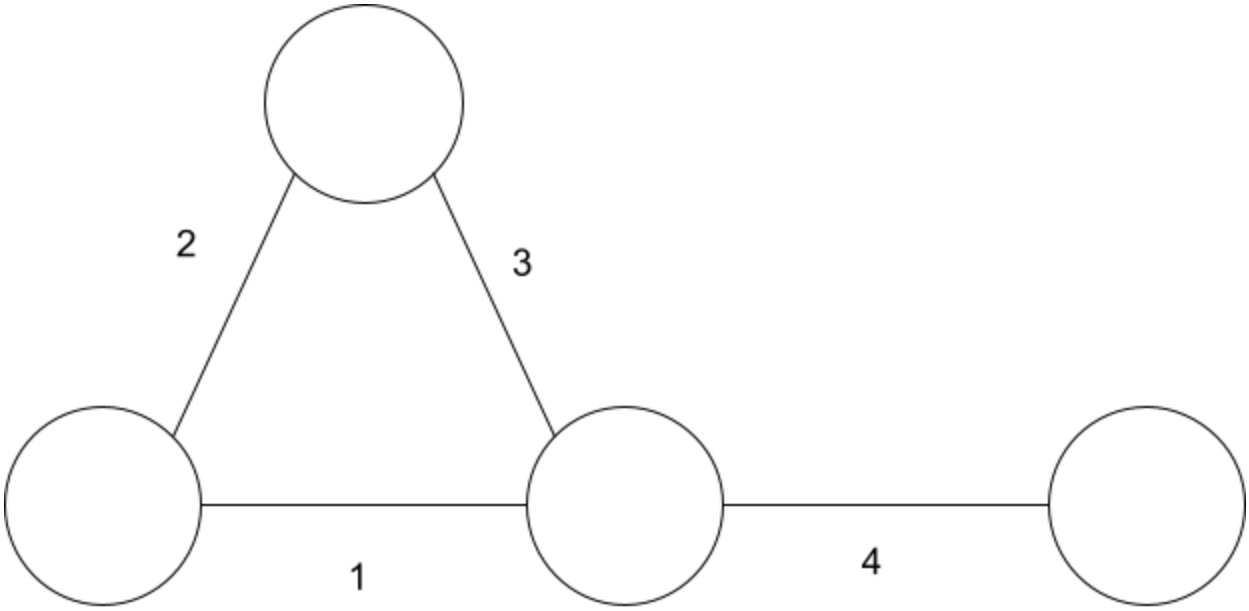


Consider graph G (drawing above). The cheapest path *P* connecting *u* and *v* consists of the two edges with weights 1 and 5 (the only alternative path consisting of edges with weights 1, 3, and 4, for a total of 8 which is more expensive than the total of 6 from the 1 and 5 path). However by Kruskal's algorithm we can clearly see that the MST of G would be built by first adding the edge with weight 1, then the edge with weight 3, then finally the edge with weight 4, which does *not* include the path from *u* to *v* (not in its entirety, at least). This is a contradiction of the statement that "Path *P* must be part of some minimum spanning tree".

(I'm operating under the assumption that "Path *P* must be part of some minimum spanning tree" means that regardless of what the rest of graph G consists of, the entirety of *P* must be part of the MST for the *entirety* of G. Rather than something like, "for some collection of nodes chosen out of G (but not necessarily all of G), path *P* is included in a valid MST for those nodes" (which would of course be true if you chose only the nodes already in path *P*).

2.B:

False. Counterexample:



Consider graph G (drawing above). Let edge *f* be the edge with weight 4, and edge *e* be the edge with weight 3. Using Kruskal's algorithm we can form the MST for G. First we add the edge with weight 1, then the edge with weight 2, then we *skip* the edge with weight 3 (edge *e*), as including it would form a cycle, then we add the edge with weight 4 (edge *f*). Now we can see that edge *f* belongs to G's MST, despite *e* (which is less expensive than *f*) not belonging to G's MST. This is a contradiction of the statement that "Edge *f* will never belong to a minimum spanning tree unless *e* is also in it".

3:

F[i] = maximum amount of fun had on days i through n.

Recurrence: F[i] = max({ if(n - i < x_i): 0 else: f_i + F[i+1+x_i] }, F[i+1])

Justification:

Case 1: Go to the concert on day i, and cannot go again until day i + 1 + x_i . The total fun in this scenario is the fun had on day i (f_i) plus the total possible fun with the remaining days after you are rested (F[i+1+x_i]).
However, no fun can be had on day i if there are not enough days left to get sufficient rest (n - i < x_i), (n - i being days left, and x_i being days of rest needed). If this happens, this case will not yield any fun, and the other case will be chosen (unless it's also 0 fun, in which case the distinction is moot)

Case 2: Do not go to the concert on day i, and so the total fun possible is the same as the total fun starting the next day (represented F[i+1]).

Whichever case results in more fun is desired, and so we take the maximum of the two cases.

(I wasn't sure if an x_i value of 0 was possible, the problem just says "nonnegative" but the examples don't show any 0s. I'm operating under the assumption it is possible, hence taking it into account in the algorithm's base cases below)

Algorithm:
if $x_n \neq 0$
    F[n] = 0
else
    F[n] = f_n
loop i from n-1 to 1
    F[i] = recurrence
return F[1]

Running time: **O(n)**. There are n subproblems, and each subproblem takes constant time to calculate.

4:

F[i, j] = number of fish total you can catch on days i through n, if you start day i with j fishing skill

Recurrence: F[i, j] = max{ min(j, x_i) + F[i+1, j],
F[i+1, j+2] }

Justification:

Case 1: Go fishing on day i. The total number of fish received will then be the number of fish caught on day i (represented as min(j, x_i)) plus the number of fish caught on the remaining (i+1 through n) days, with the same skill j (represented as F[i+1, j]).

Case 2: Do not go fishing on day i, but instead train. As a result, the total number of fish received will just be the number of fish caught on the remaining (i+1 through n) days, now with a fishing skill increased by 2 (so j+2 skill), represented as F[i+1, j+2].

Whichever case results in the higher number of fish is more desired, and so we take the maximum of the two cases.

Algorithm:
F[i, 0] = 0 for all i
F[n, j] = min(j, x_i) for all even j such that: $2 \leq j \leq 2*n - 2$
loop i from n-1 to 2
    loop j from 2 to 2*n - 2, in intervals of 2
        F[i, j] = recurrence
return F[2, 2]

Justification: We know that we will train on day 1, and that we will fish on day n (as fishing with 0 skill is useless, and training with no days remaining is useless). At most we train on every day except the last, resulting in a skill of 0 + 2*(n-1 days), also written as 2*n - 2.

Running time: **$O(n^2)$**, as there are n by (2*n - 2) subproblems, resulting in $2*n^2 - 2n$ total, which can be reduced to $2*n^2$, and further reduced to n^2. Each subproblem can be computed in constant time, so this is the final runtime of the algorithm.