Joyce Chen 10/11/2023

CS180 HW1

1) Exercise 3, Page 22

Example of an unstable schedule:

- Network A – 1, 2, 3, 4, 6; Network B – 5, 7, 8, 9, 10
- Network B can win all the time slots if its 5-rated show doesn't compete in the same time slot as Network A's 6-rated show.
- Network A can win a maximum of 1 time slot, if its 6-rated show is competing for the same time slot as Network B's 5-rated show.
- This is an unstable schedule, because no matter how the schedules are arranged, A and B are able to win more time slots if they change their schedules unilaterally.

2) Exercise 4, on Page 22

Problem definition:

- Match *n* students with *m* hospitals, where $n > m$ (guaranteed).
- Each hospital wants a certain number of students c_1 - c_m (varies by hospital), so the match is *c* students → 1 hospital.
- Each hospital has a ranking of the students ($s_1 \ldots s_n$), and each student has a ranking of the hospitals ($h_1 \ldots h_m$).
- $m * \sum_{i=1}^{m} (c_i)$ *(total slots available)* $< n$ *(# of students)*, so some students are unmatched with any hospital at the end.
- Instabilities:
  - s' is unmatched to any hospital, but some h matched with s prefers s'.
  - (s, s', h, h') where s → h and s' → h' but s prefers h' and h' prefers s.
- Summary: Stable Matching Problem, except that (i) hospitals generally want more than one resident, and (ii) there is a surplus of medical students.

Algorithm:

```
Initially all students s ∈ S and hospitals h ∈ H are free
While there is a hospital h that has a free slot
    Choose such a hospital h
    For each free slot in h
        Let s be the highest-ranked in h's preference list that h hasn't proposed to
        If s is free
            (h, s) become a match
        Else s is currently matched with h'
            If s prefers h' to h
                h's slot remains free
            Else s prefers h to h'
                (h, s) become a match
                slot in h' becomes free
Return the set of engaged pairs
```

Proof of termination:

- Let each iteration for finding the best student match for a hospital be constant time O(1).
- Then, worst case time would be O(*m*n*) where each of the *m* hospitals goes through the entire list of *n* students before finding their stable match.
- Best case time would be O(*m*), where each of the *m* hospitals finds their student match on the first try.

Proof of stable matching:

- Assume that the algorithm produces an unstable match: (*h, s*) and (*h', s'*), but *h* prefers *s'* and *s'* prefers *h*.
    - Therefore, *h* hasn't proposed to *s'* at some point, because *h* is matched with *s* and *s'* would have been matched with *h* since *h* is higher in its priority list.
    - Therefore, *s'* comes after *s* in *h*'s priority list, since *h* proposes to students in order.
    - However, *s'* is higher than *s* in priority, because *h* prefers *s'* to *s*.
    - >< contradiction!
- Assume that the algorithm has the instability where *s'* is unmatched to any hospital, but some *h* matched with *s* prefers *s'*.
    - Therefore, no hospital has proposed to *s'* before. This is because once a hospital has proposed to a student *s*, *s* will remain matched and their hospital preference will get better with time.
    - Therefore, *h* has proposed to *s* at some point, causing *h* and *s* to become a match.
    - However, *s'* was unmatched at the end of the algorithm, meaning *s'* must have come after *s* in *h's* priority list, which isn't true.
    - >< contradiction!

3) Exercise 6 on page 25

Algorithm:

```
Create a structure that keeps track of the ports already taken by a ship
While there is a ship s that isn't at a port
    From s's last considered day for truncation, find the next port p it arrives at
    If port p is free
        s stays at port p
    If port p is already taken by some ship s'
        If s' arrived later than s to p
            s' stays at port p
        Else if s' arrived earlier than s to p
            s stays at port p
            s' leaves port p
Return the set of all truncations
```

Proofs:

All truncations upon termination are valid: no 2 ships are ever in the same port on the same day.

- Suppose there exists some ship *s* at port *p* on day *d* after *s'* has been truncated at *p* on an earlier day *d'*.
- Since the algorithm has terminated, *s* must have been truncated on a day after *d*.
- This means that *s* has checked whether the port on day *d* was free or not, and day *d'* must have been later than *d* because *s* did not end up truncating at *d*.
- However, we established that *d'* should be earlier than *d* in the assumption.
- >< contradiction.

All schedules can be truncated at the end of the algorithm.

- Suppose that by the end of the algorithm, the schedule of ship *s* could not be truncated.
- This means that *s* must have considered all days in its schedule.
- Once a port *p* is taken by a ship, it remains taken until the end of the algorithm, only swapping ships that come to *p* at a later date.
- Therefore, no ship has ever considered truncating at *p*. However, *s* considered everyday in its schedule, meaning it must have considered truncating at *p*. This is because there are *m* days in a schedule and *n* total ports. The problem established that *m* > *n*.
- >< contradiction!

Time complexity

- Worst case scenario: O(n^2) runtime where *n* = # of ships = # of ports. For each ship, we'd have to search through all the possible ports before arriving at a stable port.
- Best case scenario: O(*n*) runtime where *n* = # of ships. For each ship, we just choose the first port, assuming that they are all at different ports on the same day.

4) Exercise 4 on page 67

- To find the greater function: Let f and g be two functions that limn→∞ f(n)/g(n) exists and is equal to 0. Then f(n) = O(g(n)), or f(n) < g(n).
- In general, 1 < logn < sqrt(n) < n < nlogn < n^2 < n^3 < … < 2^n < 3^n < … < n^n
- The ranking of functions in ascending order is: 1 < 3 < 4 < 5 < 2 < 7 < 6.

$$g_1(n) = 2^{\sqrt{\log n}}$$

$$g_2(n) = 2^n$$

$$g_4(n) = n^{4/3}$$

$$g_3(n) = n(\log n)^3$$

$$g_5(n) = n^{\log n}$$

$$g_6(n) = 2^{2^n}$$

$$g_7(n) = 2^{n^2}$$

5a) Prove (by induction) that sum of the first n integers (1+2+....+n) is n(n+1)/2

- check the base case n = 1:
  - sum = 1
  - 1(1+1)/2 = 1
  - 1 = 1, so the base case works.

- prove by induction:
    - sum of first n+1 integers = 1 + 2 + … n + 1 = (n+1)(1+(n+1))/2 = (n+1)(n+2)/2
    - plug in n+1 → (n+1)(n+1+1)/2 = (n+1)(n+2)/2
    - (n+1)(n+2)/2 = (n+1)(n+2)/2
    - by the inductive hypothesis, the sum of the first n integers is n(n+1)/2

5b) What is $1^3 + 2^3 + 3^3 + ... + n^3$ = ? Prove your answer by induction.

- the sum of $1^3 + … + n^3 = [n(n+1)/2]^2$
- check the base case n = 1:
    - sum = $1^3 = 1$
    - $[1(1+1)/2]^2 = 1$
    - 1 = 1, so the base case works.
- prove by induction:
    - $1^3 + … + n^3 + (n+1)^3 = [n(n+1)/2]^2 + (n+1)^3 = [(n)(n+1)]^2/4 + 4*[(n+1)^2(n+1)]/4 = (n+1)^2[n^2 + 4(n+1)]/4 = [(n+1)^2(n+2)^2]/4$
    - plug in n+1 → $[(n+1)(n+2)/2]^2 = [(n+1)^2(n+2)^2]/4$
    - $[(n+1)^2(n+2)^2]/4 = [(n+1)^2(n+2)^2]/4$
    - by inductive hypothesis, the sum of $1^3 + … + n^3 = [n(n+1)/2]^2$

6) Given an array A of size N. The elements of the array consist of positive integers. You have to find the largest element with minimum frequency.

Algorithm:

```
create a dictionary D that maps elements to their frequencies
for each element a in A
    if a doesn't exist in D
        add the pair (a, 1) in D
    else if a already exists in D
        increment a's frequency by 1 in D
initialize the variable minFrequency to 1, to keep track of the minimum frequency
create the variable largestElement to keep track of the corresponding element
for each element d in D
    if its corresponding frequency f is smaller than or equal to minFrequency
        if d is greater than largestElement
            set largestElement to d
        else
            set minFrequency to f
return largestElement
```

Proof of termination:

- In all cases, we will have to first traverse A in its entirety to map its elements with their frequencies → O(N).
- We will then have to traverse D to find the largest element with the minimum frequency → O(N) because there are N elements in the dictionary.
- Total time: O(N) + O(N) = O(2N), which can be simplified to O(N) runtime.