CS 161 Final Exam

Do not turn this page until you are instructed to do so!

Instructions: Solve all questions to the best of your abilities. You may cite any result we have seen in class or CLRS without proof. You have **180 minutes** to complete this exam. You may use three single-sided sheets of notes that you have prepared yourself. You may not use any other notes, books, or online resources. Please write your name at the top of all pages. Do not write on the backs of pages.
Advice: If you get stuck on a problem, move on to the next one. Pay attention to how many points each problem is worth. Read the problems carefully.

Special instructions for algorithm design questions (Problems 2-5)

3 out of 4: Choose **THREE** out of the **FOUR** algorithm design questions (problems 2 - 5) to answer. Clearly mark which problems you chose; if it is ambiguous we will take the *worst* 3.

Partial credit: For each problem, design a correct algorithm that is as efficient as possible. If you cannot come up with a correct algorithm, you may describe an incorrect algorithm for a *small partial credit*; in this case, clearly designate that your algorithm is incorrect.

The following is a statement of the Stanford University Honor Code:

1. *The Honor Code is an undertaking of the students, individually and collectively:*
   - *(1) that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;*
   - *(2) that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.*
2. *The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.*
3. *While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.*

By signing your name below, you acknowledge that you have abided by the Stanford Honor Code while taking this exam.

Signature: _____

Name: _____

SUNetID: _____

1 Multiple Choice / Short Answer (25 pts)

No explanation is required for the questions in Section 1. Please clearly mark your answers; if you must change an answer, either erase thoroughly or else make it **very** clear which answer you intend. **Ambiguous answers will be marked incorrect.**

1.1. **(5 pt.)** Which of the following correctly describes $n\log(n)$? **Circle all that apply.**

(A) $\Theta(n)$ (B) $\Omega(n^{3/2})$ (C) $O(n^{3/2})$ (D) $\Theta(n\log(n))$

CD. (C) provides a larger upper bound, and (D) exactly describes the function. (A) requires a tight bound that $n\log(n)$ is larger than, and (B) requires the same lower bound as (C).

1.2. **(5 pt.)** Function f takes as input an array A of n keys and outputs an integer in $\{1,\ldots,L\}$. For every integer ℓ in $\{1,\ldots,L\}$, there is some input A such that $f(A)=\ell$. Use the above information to deduce the strongest lower bound you can in the comparisons model for deterministic algorithms that on input A computes $f(A)$. **Circle all that apply.**

Every deterministic algorithm for computing any function f that satisfies the above properties requires at least _____ comparisons.

(A) $\Omega(n\log(n))$ (B) $\Omega(L\log(L))$ (C) $\Omega(\log(L))$
(D) Same as sorting an array of size L.

C. Every decision tree for computing f must have L leaves, hence the height of the tree is at least $\log(L)$.

1.3. **(5 pt.)** Suppose that we match n doctors to n hospitals using the Deferred Acceptance Algorithm. In which of the following scenarios, is it possible that Doctor i and Hospital j prefer to be matched to each other over the matching produced by the algorithm? **Circle all that apply.**

(A) i ranks j first and j ranks i second.
(B) i ranks j second and j ranks i first.
(C) i ranks j last and j ranks i third.
(D) i ranks j third and j ranks i second.

None of the above
The Deferred Acceptance Algorithm outputs a Stable Matching with no blocking pairs.

1.4. **(5 pt.)** For each of the following concepts, determine if they apply to undirected graphs, directed graphs, both, or none. **Circle one option for each row.**

- Strongly Connected Components

(A) Undirected (B) Directed (C) Both (D) Neither

- Topological Sorting

(A) Undirected (B) Directed (C) Both (D) Neither

- Shortest Path

(A) Undirected (B) Directed (C) Both (D) Neither

- Minimum Spanning Tree

(A) Undirected (B) Directed (C) Both (D) Neither

- Strongly Connected Components

(A) Undirected **(B) Directed** (C) Both (D) Neither

- Topological Sorting

(A) Undirected **(B) Directed** (C) Both (D) Neither

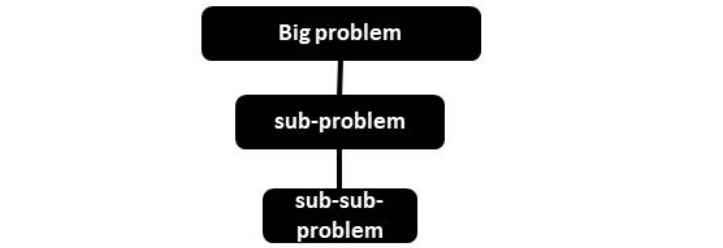
- Shortest Path

(A) Undirected (B) Directed **(C) Both** (D) Neither

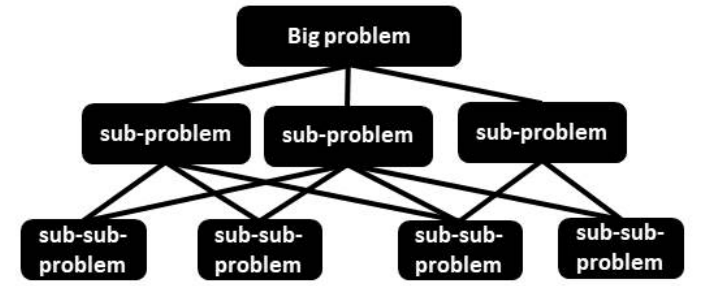
- Minimum Spanning Tree

(A) Undirected (B) Directed (C) Both (D) Neither

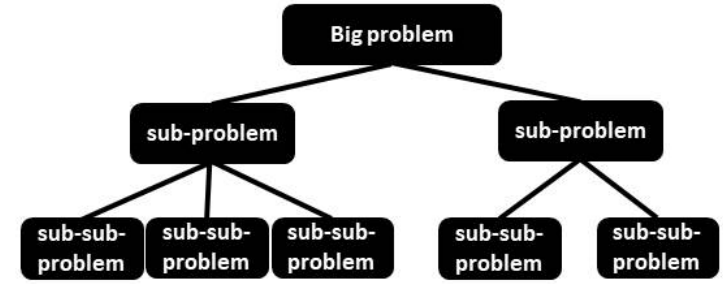
1.5. **(5 pt.)** Match each of the following algorithm design paradigms to the picture that best captures it. **Draw 3 lines.**



Divide-and-Conquer



Greedy



Dynamic Programming

First diagram is greedy, second diagram is dynamic programming, third is divide-and-conquer.

2 Algorithm Design, Part 1 (25 pts)

IMPORTANT: This is one of the questions I want graded (circle one): **(YES)** **(NO)**

Instructions: Design an efficient algorithm for the following problem and analyze its running time. Correct algorithms that use fewer comparisons will receive more credit than less efficient algorithms. You do not need to prove anything. You may use any algorithm seen in lecture. Randomized algorithms that succeed with high probability will be considered correct.

You are running for reelection as president of the United States of Algorithmica. There are k states in the union, each with n_i cities and ℓ_i votes in the electoral college. To make your voters happy, you plan to lay out a new fiber network that connects the n_i cities in the i-th state. (The capitals of different states are already connected, so you only need to worry about connections within each state.) You have a budget of B dollars for this project, so you may not be able to upgrade the network in every state. **For each state, you either upgrade it and secure ℓ_i votes, or don't upgrade it at all. Can you win a majority of the $\sum \ell_i$ electoral college votes?**

Congestion on the new fiber is unlimited but installing in different terrains has different costs. Fiberithms, the company that will be laying down cables provides you with a cost estimate for every pair of cities; the cost for each pair is an integer less than or equal to 10. Also per the contract in Fiberithms' winning bid, in each state they will connect one pair of cities -of your choice- for free.

Problem Description: For each of k states, you are given a number ℓ_i of votes, an array of $n_i \times n_i$ costs. Your task is to wire, for a cost of at most B enough states to have the majority of the electoral votes (or return that this is impossible).

Input: Budget B, and for each $1 \leq i \leq k$: ℓ_i the number of votes, n_i: the number of cities, C_i: an $n_i \times n_i$ array of integer costs in $\{1, 2, \ldots, 10\}$.
Output: Yes/No: Is your budget enough to win the elections?

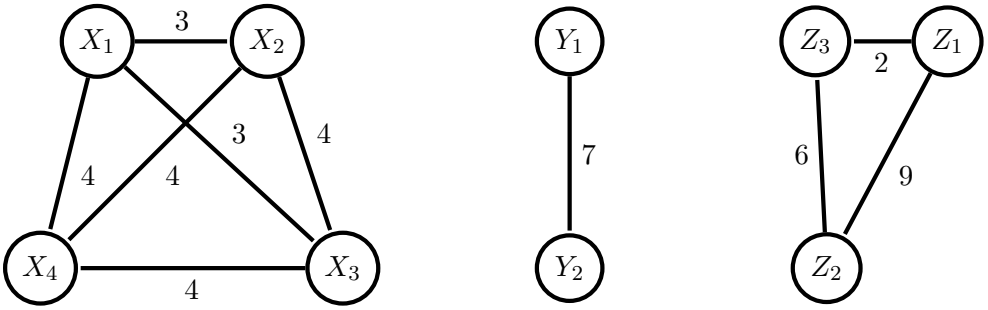


Figure 1: Illustration for Example 1. The cities of states X, Y, and Z are shown above, with their connecting edge weights described by matrices C_X, C_Y, and C_Z, respectively in Example 1

Example 1:

```
Input:
B = 4, k = 3, l = [21, 8, 17], n = [4, 2, 3]

C_X = [[0,3,3,4]    C_Y = [[0,7]    C_Z = [[0,9,2]
       [3,0,4,4]           [7,0]]          [9,0,6]
       [3,4,0,4]                           [2,6,0]]
       [4,4,4,0]]

Output: Yes
(Since we have one free connection per state, we can connect state Y
for free and state Z with cost 2. This total cost is within budget and
achieves a total of 8+17=25 votes, enough to win the election)
```

Example 2:

```
Input:
B = 8, k = 2, l = [9, 11], n = [3, 3]

C_1 = [[0,8,8]    C_2 = [[0,9,9]
       [8,0,8]           [9,0,9]
       [8,8,0]]          [9,9,0]]

Output: No
(We can connect state #1 with cost 8 and state #2 with cost 9. Our
budget only lets us capture the votes in state 1, which leaves us
short of the majority vote)
```

2.1. Provide an English description of your algorithm.

[We are expecting: A clear yet thorough English description of the algorithm. If we use a graph, we highly encourage drawing an example.]

2.1.1. For each state, compute the cost by running **Prim**'s Algorithm to find an MST. Remove the edge with the highest cost.

2.1.2. Given all the state costs, use the 0/1-Knapsack DP algorithm

Alternative correct and only very slightly ($\alpha(n_i)$) slower solution: Solve MST with Kruskal with RadixSort instead of Prim. Notice that we spend more than B time on the Knapsack anyway, so we can assume without loss of generality that costs are reasonably small integers.

Common mistake: Trying to solve Knapsack with a greedy algorithm.

2.2. Is your algorithm correct (Yes/No)?

[We are expecting: A clear answer of YES or NO. If YES, no explanation is necessary, if NO, explain why your algorithm is incorrect.]

2.3. Provide the pseudocode for your algorithm.

[We are expecting: Detailed pseudocode that matches your English description. You are free to use an interface of any of the algorithms covered in lecture.]

```
Input: n, the number of cities; C, list of edge costs
Output: tree T
/* Compute state costs:                                          */
for i : 1 ... k do
    T ← Prim(C_i);
    max ← ∞;
    for e ∈ T do
        if C(e) > max then
            e* ← e;
            max ← C(e);
    StateCosts[i] ← Cost(T) − max
/* 0/1 Knapsack:                                                 */
for b : 0 ... B do
    DP[0][b] ← 0
for i : 1 ... k do
    for b : 0 ... StateCost[i] − 1 do
        DP[i][b] ← DP[i][b]
    for b : StateCost[i] ... B do
        DP[i][b] ← max{DP[i][b], DP[i − 1][b − StateCost[i]] + ℓ_i
if DP[k][B] > (1/2) Σ ℓ_i then
    return YES
else
    return NO
```

2.4. What is the algorithm's running time?

[We are expecting: Clear but succinct analysis of the algorithm's running time as a function of $\sum n_i$, $\sum n_i^2$, k, and B.]

Total running time is $\underline{\sum n_i^2 + Bk}$.

Prim's algorithm for each state takes $O(n_i^2)$. Deciding which edge to remove can be done in $O(n_i)$ time.

The DP takes $O(Bk)$.

3 Algorithm Design, Part 2 (25 pts)

IMPORTANT: This is one of the questions I want graded (circle one): **(YES)** **(NO)**

Instructions: Design an efficient algorithm for the following problem and analyze its running time. Correct algorithms that are (asymptotically) more efficient will receive more credit than less efficient algorithms. You do not need to prove anything. You may use any algorithm seen in lecture. Randomized algorithms that succeed with high probability will be considered correct.

Problem description: You are organizing a trip for a group of n students. You need to assign the students into two buses with $n-1$ seats each. You survey the students for their preferences, and each student submits a list of $\sqrt{n}$ friends they would like to sit with. Each student's *happiness* is the number of friends in their list that are assigned to the same bus as them. Your goal is to assign the students to buses, while maximizing the total happiness of all the students.

Before you begin, two of the students ask *not* to sit on the same bus. This is not a standard request: you will try to accommodate their request, as long as it does not decrease the total happiness.

Input: an $n \times \sqrt{n}$ array F (each student's $\sqrt{n}$-length list of friend requests is a row), and a pair x, y of students who prefer not to sit in the same bus.
Output: A list A of $1 \leq |A| \leq n-1$ students to sit in the first bus.

Example:

```
Input:
n = 4, (x,y) = (1,4)
F = [
[2,3]                    // 1's friend requests
[1,3]                    // 2's friend requests
[2,4]                    // 3's friend requests
[2,3]                    // 4's friend requests
]

Output:
A = [1,2,3]

The total happiness in this assignment is 5, since the happiness of
1, 2, 3, 4 are 2, 2, 1, 0 respectively.
Another possible output is A = [2, 3, 4], in which case 1, 2, 3, 4
have happiness 0, 1, 2, 2 for a total of 5.
Any other assignment has less total happiness.
```

3.1. Give a short but clear English description of your algorithm.

[We are expecting: A clear yet thorough English description of the algorithm. If you use a graph, we highly encourage drawing a clear example.]

Run **Karger-Stein** algorithm for min-cut on the friendship graph. If there are multiple equal-sized min-cuts and one of them separates x, y return that one. Otherwise, return an arbitrary min-cut.
(Notice that Karger-Stein is guaranteed to find any min-cut with good probability. In particular, if there is a min-cut that separates x, y, Karger-Stein will also find it.)

Common alternative solution: Finding a global min-cut with Karger-Stein, and then using Ford-Fulkerson $s - t$ min-cut algorithm to find the min cut that separates x, y.
Using Ford-Fulkerson is clearly redundant, but it actually doesn't hurt your run-time if you use the simplest analysis of $O(m|f|)$ for Ford-Fulkerson: $m = n^{3/2}$ and $|f| \leq n^{1/2}$, so $m|f| \leq n^2$, aka faster than Karger-Stein. (Black box analyses for fattest path or Edmonds-Karp will give you slower run times.)

3.2. Is your algorithm correct (Yes/No)?

[We are expecting: A clear answer of YES or NO. If YES, no explanation is necessary, if NO, explain why your algorithm is incorrect.]

3.3. Provide the pseudocode for your algorithm.

[We are expecting: Detailed pseudocode that matches your English description. You are free to use an interface of any of the algorithms covered in lecture.]

Input: F, the friendship graph
Output: A, the list of students on the first bus
ModifiedKargerStein($[n], F$): In line 9, break ties in favor of cuts that separate x, y.

3.4. What is the algorithm's running time?

[We are expecting: Clear but succinct analysis of the algorithm's running time as a function of n.]

Total running time is $O(n^2 \log^3(n))$.

Same as Karger-Stein ($O(n^2 \log^2(n))$ for constant probability of success).

4 Algorithm Design, Part 3 (25 pts)

IMPORTANT: This is one of the questions I want graded (circle one): **(YES)** **(NO)**

Instructions: Design an efficient algorithm for the following problem and analyze its running time. Correct algorithms that are (asymptotically) more efficient will receive more credit than less efficient algorithms. You do not need to prove anything. You may use any algorithm seen in lecture. Randomized algorithms that succeed with high probability will be considered correct.

Problem Description: You are handed a scenic black-and-white photo of the skyline of Algoville. The photo is n-pixels tall and m-pixels wide, for $n := m \log\log(m)$. In the photo, buildings appear as black and sky background appears as white. In any column, all the black pixels are below all the white pixels. Design an efficient algorithm that finds the height of the tallest building in Algoville.

Example 1:

```
Input:
n=4, m=4
A = [
[1, 0, 1, 1]
[1, 0, 1, 0]
[1, 0, 0, 0]
[0, 0, 0, 0]
]

Output:
4
(Height of second column corresponds to tallest tower. Note that
we give you the array upside down such that the first row of
matrix A corresponds to the highest level and the bottom row of
A corresponds to the ground level).
```

Example 2:

```
Input:
n=32, m=16
A = [
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
[1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1]
[1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1]
[1,1,1,1,1,1,1,1,0,1,1,1,0,1,1,1,1]
[1,1,1,1,1,1,1,1,0,0,1,1,0,1,1,1,1]
[1,1,1,1,1,1,1,1,0,0,0,0,0,1,1,1,1]
[1,1,1,1,1,1,1,1,0,0,0,0,0,1,1,1,1]
[1,1,1,1,1,1,1,1,0,0,0,0,0,1,1,1,1]
[1,1,1,1,1,1,1,1,0,0,0,0,0,1,1,1,1]
[1,1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,1]
[1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,1,1]
[1,0,1,1,1,1,0,0,0,0,0,0,0,1,1,1]
[1,0,0,1,1,1,0,0,0,0,0,0,0,1,1,1]
[1,0,0,1,1,1,0,0,0,0,0,0,0,0,1,1]
[1,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0]
[0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0]
[0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0]
[0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
]

Output:
27
(Height of 12th column corresponds to tallest tower).
```

4.1. Give a short but clear English description of your algorithm.

[We are expecting: A clear yet thorough English description of the algorithm. If we use a graph, we highly encourage drawing an example.]

Divide-and-Conquer algorithm: $O(m\log(n)) = O(m\log(m))$ time
For each column, binary search for the highest black pixel.

Greedy algorithm: $O(n+m) = O(m\log\log(m))$ time
Consider a walk starting from the bottom left corner of the matrix. Whenever we see a black pixel we go up, whenever we see a white pixel we go right. (We record the highest black pixel that we ever saw.) In total we can only go right/up $n+m$ steps, and we must pass through the highest black pixel.

Combined algorithm: $O(m\log(\frac{n}{m})+m) = O(m\log\log\log(m))$ time
We do the walk like in the greedy algorithm, but whenever we need to go up, we use binary search to find how far up we should go.
(On average, on each column we go up by n/m pixels, so binary search on that takes $O\log(\frac{n}{m})$.)

4.2. Is your algorithm correct (Yes/No)?

[We are expecting: A clear answer of YES or NO. If YES, no explanation is necessary, if NO, explain why your algorithm is incorrect.]

4.3. Provide the pseudocode for your algorithm.

[We are expecting: Detailed pseudocode that matches your English description. You are free to use an interface of any of the algorithms covered in lecture.]

```
Input: A, and m × n array of zeros and ones
Output: h, the heighest of the tallest building
h ← 1;
/* Loop over columns from left to right:                    */
for i = 1 ... m do
    j ← 1 if A[h][i] == 0 then
        /* Find largest j = 2^ℓ such that we can increase height
           by j                                              */
        while A[h + j][i] == 0 do
            j ← 2j;
        k ← j/2;
        h ← h + k;
        /* The max height in column i is between h and h + j/2;
           */
        /* find it with binary search:                       */
        for k = j/2, j/4, ..., 1 do
            if A[h + k][i] == 0 then
                h ← h + k;
return h
```

4.4. Provide an analysis of the runtime of your algorithm.

[We are expecting: A detailed analysis of the runtime of your algorithm including the Big-O time in terms of m.]

5 Algorithm Design, Part 4 (25 pts)

IMPORTANT: This is one of the questions I want graded (circle one): **(YES)** **(NO)**

Instructions: Design an efficient algorithm for the following problem and analyze its running time. Correct algorithms that are (asymptotically) more efficient will receive more credit than less efficient algorithms. You do not need to prove anything. You may use any algorithm seen in lecture. Randomized algorithms that succeed with high probability will be considered correct.

Problem description: We want your help coming up with a new algorithm for scheduling office hours next quarter. There are n CAs, d days, and t time slots on each day.
Each CA will submit their availability, i.e. which time slots they can cover on each day. The head CA will provide a list of available time slots on each day. Design an algorithm that uses this information to determine whether it's possible to cover at least $t' \leq t$ slots every day.

Easier variant:
Assume the time slots and CA availabilities are the same on every day. You want a schedule such that no CA holds more than Y office hours on each day. (I.e. you may solve the problem for a single day only).

Harder variant (you will receive more credit if you correctly solve this variant. It may help to solve the easier variant first):

- Each CA should hold at most X office hours during the quarter.
- Each CA should hold at most Y office hours on each day.

Input: Parameters n, d, t, t', X, Y, and an $n \times d \times t$ array A with CA availabilities.
Output: Yes/No (Is it possible to schedule OH given constraints).
Example:

```
Input:
n = 2, d = 2, t = 4
t' = 3, X = 3, Y = 2

(CA Availability on Day 1)
A[:, 1, :] = [[1, 1, 1, 0]
              [0, 1, 1, 0]]

(CA Availability on Day 2)
A[:, 2, :] = [[0, 1, 0, 0]
              [1, 1, 1, 1]]

Output:
Yes
(One possible assignment proceeds as follows. On day 1, CA #1 takes the
first two shifts and CA #2 takes the third shift. On day 2, CA #1 takes
the second shift and CA #2 takes the first and third shift)
```

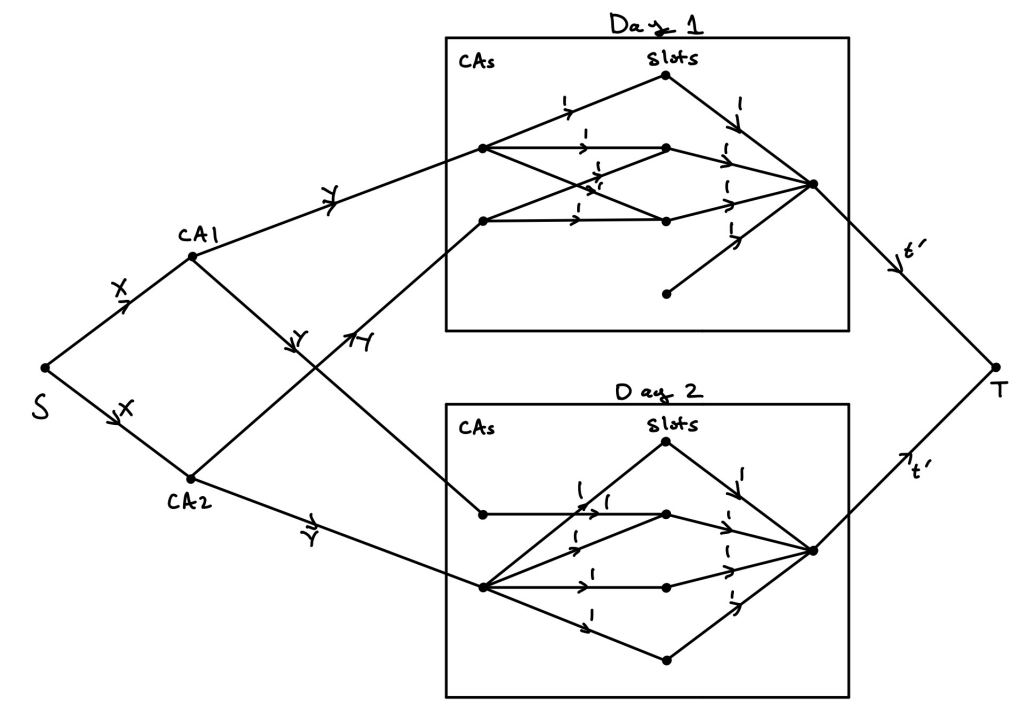
5.1. Give a short but clear English description of your algorithm.

[We are expecting: A clear yet thorough English description of the algorithm. If we use a graph, we highly encourage drawing an example.]

We represent this problem as a graph and use **Ford-Fulkerson Max-Flow algorithm** with Fattest Path rule. We create the following vertices and edges:

- 1 source node S;
- n CA node, and an edge with capacity X from s to each CA node;
- $n \times d$ CA-day nodes, and an edge with capacity Y from each CA node to each corresponding CA-day node;
- $d \times t$ time slot nodes, and an edge with capacity 1 from each CA-day node to any time slot that CA can hold OH;
- d day nodes, and an edge with capacity 1 from each time slot node to the corresponding day node; and
- 1 sink T, and an edge with capacity t' from each day node to the sink node.

Below is the flow graph we obtain for the example.



5.2. Is your algorithm correct (Yes/No)?
Which variant of the problem did you do (Harder/Easier)?

[We are expecting: A clear answer of YES or NO for the first question, and HARDER or EASIER for the second. If YES, no explanation is necessary, if NO, explain why your algorithm is incorrect.]

5.3. Provide the pseudocode for your algorithm.

[We are expecting: Detailed pseudocode that matches your English description. You are free to use an interface of any of the algorithms covered in lecture.]

Input: n, d, t, t', X, Y, and array $A \in \{0,1\}^{n \times d \times t}$ with CA availabilities.
Output: Yes/No
`/* Construct the graph: */`
Create an empty graph G Create a source node S and target node T.;
for $i = 1 \dots n$ **do**
  Create node C_i;
  Create edge (s, C_i) with capacity X;
for $day = 1 \dots d$ **do**
  Create node D_{day};
  for $i = 1 \dots m$ **do**
    Create node $CD_{i,day}$;
    Create edge $(C_i, CD_{i,day})$ with capacity Y;
  for $j = 1 \dots t$ **do**
    Create node $DT_{day,j}$;
    for $i = 1 \dots n$ **do**
      if $A[i][day][j] == 1$ **then**
        Create edge $(CD_{i,day}, DT_{day,j})$ with capacity 1;
    Create edge $(DT_{day,j}, D_{day})$ with capacity 1;
  Create edge (D_{day}, T) with capacity t';
$f \leftarrow$ max-flow(G, s, t) ; `// Using fattest-path rule`
if $|f| = dt'$ **then**
  return YES
else
  return NO

5.4. Provide an analysis of the runtime of your algorithm.

[We are expecting: A detailed analysis of the runtime of your algorithm including the Big-O time in terms of t, d, n.]

$\underline{(tdn)^2 \log^2(td)}$
The graph has $m = O(tdn)$ edges. The total flow is $|f| \leq td$ (since we have at most one unit of flow per time slot). Plugging into $m^2 \log(m) \log(|f|)$ this gives $(tdn)^2 \log(tdn) \log(td)$.
Since the maximum edge capacity is wlog $\leq td$, we can actually improve the running time to $(tdn)^2 \log^2(td)$ using a modified fattest-path algorithm.