1. (•) ***I understand the course policies.***

   Describe and analyze an algorithm to nest the boxes so that the number of visible boxes is as small as possible.

   Solution: We assume one box can only contain *exactly* one another box, i.e., two boxes cannot be placed into the same box simultaneously. We reduce this problem to bipartite matching problem, and then use Ford-Fulkerson algorithm to solve it. We construct a bipartite graph G with vertex set $U \cup W$ as follows:

   - Each box is a node in both U and W.
   - For a pair of node $u \in U$ and $v \in W$, if u can be rotated so that it can be contained in v, we add an edge $e = (u, v)$.

   Call two boxes are nested if one contains another. Let N be the maximum number of nested boxes. Then, the number of visible boxes can be simply computed as $n - N$.

   Theorem 1. *The maximum number of nested boxes equals to $|M|$, where $|M|$ is the value of the size of the maximum matching in G.*

   Proof: If we know the maximum number of nested boxes and how the boxes contain each other, we can transfer to a maximum matching in G as follows: if u contains v, then we select $e = (u, v)$ in G, where $u \in U$ and $v \in W$. Since each box can contain exactly one another box, this is a valid matching. Each time we place a box into another, the number of visible boxes is decreased by one. Let $|M|$ be the final number of edges selected. It is maximized according to the above procedure. Otherwise, if there is a matching that has larger size, there will be a better way to place the boxes. This contradicts with the assumption. By contradiction, M is a maximum matching. Moreover, the number of the nested boxes and the size of the maximum matching is $|M|$.

   Conversely, if we know a maximum matching, we can find how to place the boxes into each other by placing u into v for each $e = (u, v)$ in M. Each $u \in U$ is matched exactly once, which means each box can be placed in another box only once. Thus this is a valid solution for the problem. Furthermore, this scheme must has the maximum number of nested boxes. Otherwise, if there is a better way to place the box, we can transform it to a matching that has a larger size. By contradiction, this is the best way to place the box.

   Finally, the size of the maximum matching is $|M|$, and equals to the number of nested boxes. □

   There are at most six permutation of width, height and depth. Hence for a pair of boxes, we can determine whether they can contain one another in constant time. Let there be n boxes. Then, there are $2n$ nodes and at most $O(n^2)$ edges in G. Hence, the maximum flow has value at most $O(n)$, and the Ford-Fulkerson algorithm runs in $O(n^3)$ time.

   ■

2. (•) ***I understand the course policies.***

Describe an efficient algorithm that either rounds A in this fashion, or reports correctly that no such rounding is possible.

Solution: We can solve this problem by reducing it to maximum flows problem with edge demands. The capacities and demands are all integers. Before we do that, we first check that if the sum of each row (or column) is an integer. If not, we report that no such rounding is possible. This is because after rounding, the sum of each row (or column) must be an integer.

Now we assume the sum of each row (or column) is an integer. Let $R[1..m]$ be the sum of each row, and $C[1..n]$ be the sum of each column. We construct a graph G with vertex set $\{s,t\} \cup U \cup V$ as follows:

- Each row i is a node r_i in U.
- Each column j is a node c_i in V.

The edges consists of:

- an edge $e : s \to r_i$ with capacity $c(e) = d(e) = R[i]$ for each $r_i \in U$.
- an edge $e : c_j \to t$ with capacity $c(e) = d(e) = C[j]$ for each $c_j \in V$.
- an edge $e : r_i \to c_j$ with capacity $c(e) = \lceil x_{ij} \rceil$ and demand $d(e) = \lfloor x_{ij} \rfloor$ for each pair of $r_i \in U$ and $c_j \in V$.

Theorem 2. *The rounding exists if and only if there is a maximum flow that saturates every edge leaving from s. The rounding scheme consists of the flow values of edges $r_i \to c_j$.*

Proof: If we know a correct rounding scheme, we can transform it to a feasible maximum flow in G. For each $x_{ij} = A[i][j]$, we push the same amount of flow to edge $r_i \to c_j$. Since this rounding scheme is correct, the sum of rows and columns are correct, which means the flow conservation is satisfied and the edges leaving s and the edges entering t are all saturated. Note that $\sum_i R[i] = \sum_j C[j]$. Clearly, it is a feasible maximum flow.

Conversely, if we have a maximum flow that saturates every edge leaving s, we can round each element $A[i][j] = f(e)$, where e is the edge $r_i \to c_j$. The flow value $f(e)$ satisfies wireless.illinois.edu$d(e) \le f(e) \le c(e)$. Moreover, the capacities and demands are all integer values, which means Ford-Fulkerson will compute an integer flow. Thus, each entry $x_{ij} = A[i][j]$ is rounded correctly to $\lfloor x_{ij} \rfloor$ or $\lceil x_{ij} \rceil$. Finally, the sum of rows and columns are the same as the corresponding capacity (or demand) values in edges $s \to r_i$ and $c_j \to t$. Since all these edges are saturated, the sum of entries in any row or column are unchanged. Thus, it is a valid rounding scheme. □

Clearly, there are $O(m+n)$ vertices and $O(mn)$ edges in G. Computing the sums and constructing the graph can be done in $O(mn)$ time. If we use Edmonds-Karp fat-pipe algorithm, we get an overall running time of $O(VE^2) = O((m+n) \cdot m^2 n^2)$. ■

3. (•) ***I understand the course policies.***

   Describe and analyze an algorithm to compute a donation schedule, describing how much money each voter should send to each candidate on each day, that guarantees that every candidate gets enough money to win their election. The schedule must obey both Federal laws and individual voters' budget constraints. If no such schedule exists, your algorithm should report that fact.

Solution: Suppose there are k days left from now to the election, there are c candidates and v voters. Let $M[1..c]$ be the money that must be spent on each candidate, $W[1..v][1..c]$ be the amounts that each voter is willing to donate to each candidate $A[1..v]$ be the total amounts that each voter is able to donate to all candidates. We can model this problem as a maximum flow problem. We construct a flow network $G = (V, E)$ with vertices $X \cup Y \cup \{s, t\}$, where each node in X and Y denotes a voter and a candidate, respectively. The edges consist of:

- an edge $s \to x$ with capacity $\min\{100, A[x]/k\}$ for each $x \in X$.
- an edge $y \to t$ with capacity $M[y]/k$ for each $y \in Y$.
- an edge $x \to y$ with capacity $W[x][y]/k$ for each $x \in X$ and $y \in Y$.

Theorem 3. *The donation schedule exists if and only if there is a maximum flow that saturates every edge entering t. The money that each voter should spend to each candidate on every day is denoted by the edges $x \to y$ for each $x \in X$ and $y \in Y$.*

Proof: Given a maximum flow computed in G, we can convert it to a donation schedule as follows. For each edges $x \to y$, it denotes the amount that voter x will send to candidate y each day. For each edge $y \to t$ for each $y \in Y$, it denotes the amount that every candidate must receive each day to meet the campaign requirement. For each edge $s \to x$ for each $x \in X$, clearly it satisfies both Federal lasw and individual voters' budget constraints. If the maximum flow saturates all the edges entering t, then following this schedule, after k days each candidate can receive exactly $M[y]$ money for all $y \in Y$. Hence, the maximum flow we compute can be converted to a correct schedule.

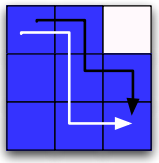
Conversely, if we have a correct donation schedule, we can convert it to a maximum flow in G by pushing the same amount of flow as $W[x][y]/k$ on edge $x \to y$ for each $x \in X$ and $y \in Y$. The donation schedule is correct, hence it satisfies the constraint on edges $s \to x$ for each $x \in X$. Hence this is a feasible flow. Furthermore, the schedule can meet each candidate's requirement and thus all the edges entering t will be saturated. Hence it is a maximum flow. □

The maximum flow has value at most $S = \sum_y M[y]$. There at most $v + v \cdot c + c = O(vc)$ edges. Thus, Ford-Fulkerson algorithm runs in $O(Svc)$ time. ■

4. (•) ***I understand the course policies.***

(a) Prove that this greedy strategy does not always compute an optimal solution.

Solution: We prove by giving a counterexample. As in the following figure, the blue cells are marked. Both of the 2-bend and 3-bend paths contains maximum number of marked cells. However, if we choose the 2-bend path at the first iteration, we need another two monotone path to cover all the cells. On the other hand, choosing the 3-bend path at first can finish marking the cells with only two paths.



■

(b) Describe and analyze an efficient algorithm to compute the smallest set of monotone paths that covers every marked cell. The input to your algorithm is an array $M[1..n, 1..n]$ of booleans, where $M[i, j]$ = True if and only if cell (i, j) is marked.

Solution: We can solve this problem by reducing it to a minimum-cost flow problem with demands. We construct a network G as follows:

- For each cell (i, j), we create two vertices $u_{i,j}$ and $v_{i,j}$. There is an edge $e : u_{i,j} \to v_{i,j}$ with cost 0. If $M[i, j] = \text{TRUE}$, then $d(e) = 1$. Otherwise $d(e) = 0$. $u_{1,1}$ is the source vertex. $v_{n,n}$ is the target vertex.
- For each $v_{i,j}$, there are edges from $v_{i,j}$ to $u_{i+1,j}$ if $i + 1 \le n$, and $u_{i,j+1}$ if $j + 1 \le n$. All of these edges have demands 0. For edges $v_{1,1} \to u_{2,1}$ and $v_{1,1} \to u_{1,2}$, they have cost 1. The rest has cost 0.

Let the number smallest set of monotone paths that covers every marked cell be k.

Theorem 4. *k equals to the cost of the minimum-cost feasible flow in G. The set of monotone paths are the corresponding directed paths in G.*

Proof: If we know the smallest set of monotone paths, we can convert it to a feasible flow in G with cost k by following the corresponding edges in G according to the paths. Since every marked cell is visited, the edges with demand 1 are satisfied and thus the flow is feasible. The cost must be minimum. Otherwise, if there is a feasible flow with smaller cost, we can convert it to a set of monotone paths that has smaller size.

Conversely, if we know a minimum-cost feasible flow with cost k, we can convert it to a set of monotone paths. Since this flow is feasible, the demands are all satisfied, which means the marked cells are all visited. k is minimized. Otherwise, if there is a better way to construct the set of paths, we can convert it back to find a feasible flow that has smaller cost. □

There are $O(n^2)$ vertices and edges in G. G can be constructed in $O(n^2)$ time. By running Sleator and Tarjan's network flow algorithm, our algorithm runs in $O(VE \log V) = O(n^4 \log n)$ time. ■

5. (•) ***I understand the course policies.***

   (a) Prove that if Paul uses a deterministic strategy, and Sally knows his strategy, then Sally can guarantee that she wins.

   Solution: Since r is a positive integer, $|S| > 0$. If Sally knows Paul's strategy, then she knows what path Paul will choose given s and t. She can simply choose the edge $e = (s,u)$ in this path and win. ■

   (b) Describe a deterministic strategy for Sally that guarantees that she wins when $r \leq M$, no matter what strategy Paul uses.

   Solution: Sally can just include all the M edges that across the (s,t)-cut in S. Imagine the path that Paul chooses as a flow from s to t. Then, it must go through one of the M edge in the minimum cut. Since S includes all these edges, $P \cap S$ must not be empty. ■

   (c) Prove that if Sally uses a deterministic strategy, and Paul knows her strategy then Paul can guarantee that he wins when $r < M$.

   Solution: Let M' be the number of edges in an arbitrary (s,t)-cut. It follows $M \leq M'$, and thus $r < M'$. Thus, Sally cannot choose all the edges that P can possibly go through. Conversely, assume that there is no way for Paul to gurantee he wins. It implies that all the edges in one of the minimum cut are included in S, and thus $r \geq M$. This contradicts with the $r < M$. By contradiction, Paul can gurantee he wins. ■

   (d) Describe a randomized strategy for Sally that guarantees that she wins with probability at least $\min\{r/M, 1\}$, no matter what strategy Paul uses.

   Solution: Sally can use the following strategy: choose r edges uniformly at random from the M edges of the minimum (s,t)-cut .

   - $r \geq M$: Sally is essentially using the strategy in (b) and thus she wins with probability 1.
   - $r < M$: No matter what strategy Paul uses, P must use one of the M edges in the minimum (s,t)-cut. Each of these edges will be selected with probability $1/M$ in Sally's strategy, and there are r edges will be selected. Hence, the probability that P uses one of these r edges is r/M. In this case, Sally wins with probability at least r/M.

   In conclusion, Sally wins with probability at least $\min\{r/M, 1\}$ when using the above strategy. ■

   (e) Describe a randomized strategy for Paul that guarantees that he loses with probability at most $\min\{r/M, 1\}$, no matter what strategy Sally uses.

   Solution: Paul can use the following strategy: choose a path that uses an edge from the M edges of the minimum (s,t)-cut uniformly at random.

   - $r \geq M$: Sally is essentially using the strategy in (b) and thus Paul loses with probability 1.
   - $r < M$: No matter what strategy Sally uses, P must use one of the M edges in the minimum (s,t)-cut. Each of these edges will be selected with probability $1/M$ in Paul's strategy, in which there are at most r edges will be selected by Sally. Hence, the probability that P uses an edge that is in S is r/M. In this case, Paul loses with probability at most r/M.

   In conclusion, Paul loses with probability at most $\min\{r/M, 1\}$ when using the above strategy. ■