1. Study Group

Jialiang Tang (myself), SID: 3039758308

Yuanzhe Wang, SID: 3039749520

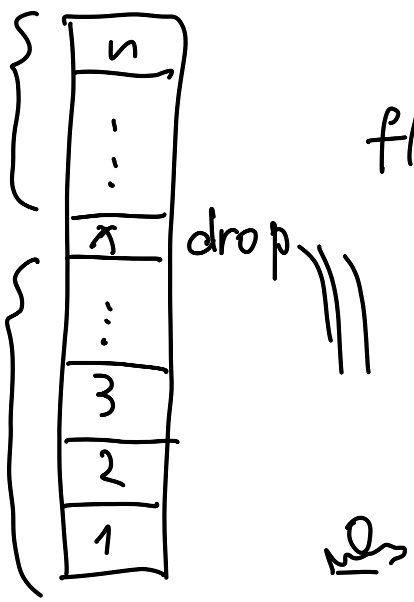
4. (a) $f(1,m)=1$ Because we only need to drop an egg on first floor to decide the l.

$f(0,m)=0$ Because we don't need to drop any egg.

$f(n,1)=n$ Because we need to drop the egg on 1-st、2-nd、3-rd、…、n-th floor successively to determine l.

$f(n,0)=0$ Because we have no egg to drop, we can never determine l! So we define $f(n,0)$ as 0.

(b)



If egg breaks, we need to consider floor $1, 2, \cdots, x-1$ and we have $m-1$ eggs remained to drop.

So we need $f(x-1, m-1)$

If egg doesn't break, we need to consider floor $x+1, x+2, \cdots, n$ and we have m eggs still.

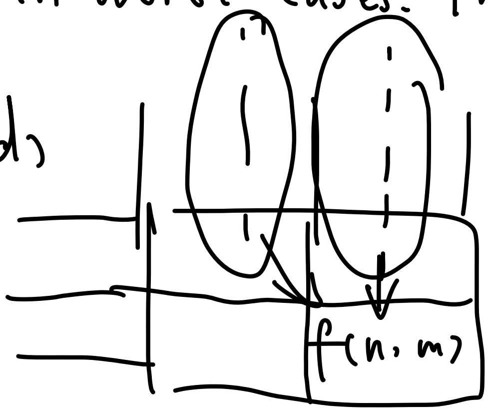
So we need $f(n-x, m)$

(c) $f(n,m) = \min\limits_{1 \le i \le n} \left(\max\left(f(i-1, m-1), f(n-i, m) \right) + 1 \right)$

"i" means we choose i-th floor to drop

So we need to choose the minimum among floors in worst cases. That's where "min-max" comes from.

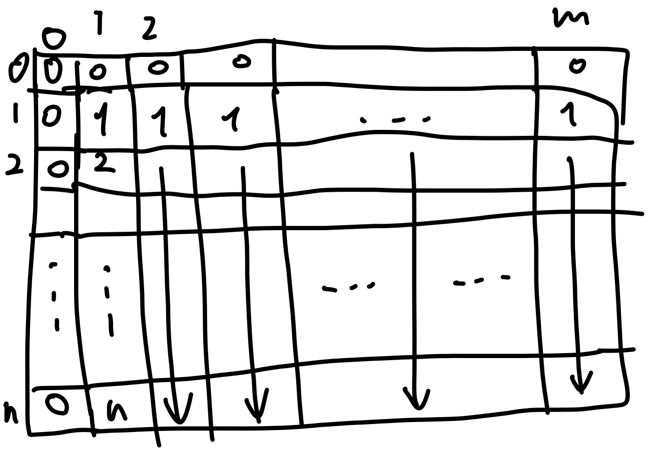
(d)



To compute $f(n,m)$, we need to compute all $f(i-1, m-1)$ and $f(n-i, m)$.

So the order is: $f(2,2), f(3,2), \dots, f(n,2)$,

$f(2,3), f(3,3), \dots, f(n,3), \dots, f(2,m), f(3,m), \dots, f(n,m)$



(We already have $f(i,1) = i$ and $f(1,j) = 1$ and $f(k,0) = f(0,w) = 0$)

(e) $O(mn)$ subproblems

Each needs $O(n)$ to find value

Total runtime is $O(n^2 m)$

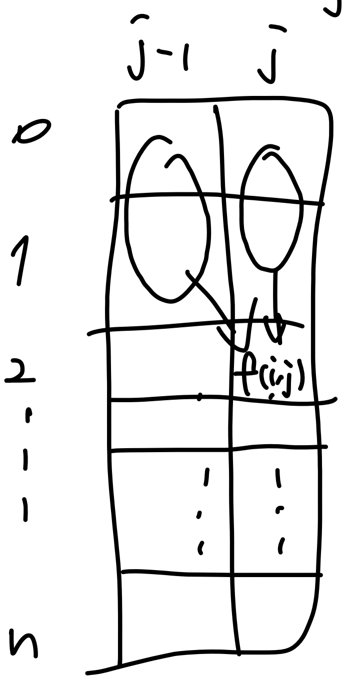
(f) Need a $n \times m$ matrix to store results. Total

Space complexity is $O(nm)$

(g) Possible.

Modification: When running dp, we only need to store the nearest 2 lines of $f(i,j)$ in dp matrix to compute it.

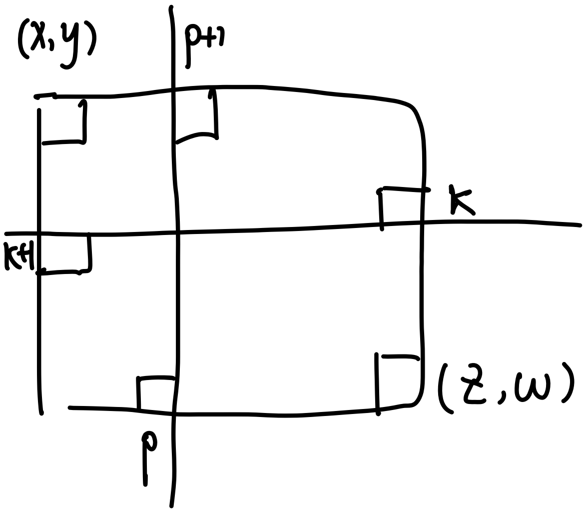
So we only need $O(2n) = O(n)$ space.



5. (a) After any arbitrary cut, we have 2 pieces left and subproblem is to find the smallest cuts on each pieces.

(b) dp[x][y][z][w] : The minimum cut of matrix with left-upper grid (x, y) and right-bottom grid (z, w).

$$dp[x][y][z][w] = \min\Big(\min_{x \le k \le z-1} \big(dp[x][y][k][w] + dp[k+1][y][z][w]\big), \min_{y \le p \le w-1} \big(dp[x][y][z][p] + dp[x][p+1][z][w]\big)\Big) + 1.$$



Base cases : dp[x][y][z][w] = 0 for all matrix having all 0s or 1s.

(c) Firstly every 1×1 grid is in base cases.

Then we solve every 1×2 and 2×1 rectangles

Then 1×3, 3×1, 2×2 rectangles.

⋮ (Each step, we solve cases with 1 more column or 1 more row, but ensure columns ≤ m, rows ≤ n).

Then m×n rectangle (which we want to solve)

(d) Subproblems : $O(m^2n^2)$ (The number of possible rectangles is $C_m^2 \cdot C_n^2 = O(m^2n^2)$)

For each subproblem, we need $O(m+n)$ ~~to~~ determine the dp value. But still need $O(mn)$ to judge whether it's a base case.

Total Runtime : $O(m^2n^2 \cdot mn) = O(m^3n^3)$.

(e) All we need is a 4-dim matrix to store the result.

Space complexity is $O(m^2n^2)$

2. (a) Algorithm Description:

① Subproblems

Assume $J(s)$: True if s is possible to be interpreted.

$s = s_1 \circ s_2$

Subproblem is to judge whether sub-string s_1 and s_2 are able to be interpreted.

② Recurrence

$$J(s) = \left(\bigvee_{\substack{s = s_1 \circ s_2 \\ s_1 \in d[\,]}} J(s_2) \right) \vee \text{False} \quad (\text{when } s \notin \text{base case})$$

Base cases: $J(s) = \text{True}$ iff $s \in d[\,]$. (Can be directly interpreted)

③ Ordering: For $1 \leq i \leq n$, judge $J(s[n-i : n-1])$

Proof of correctness:

If s is interpretable, then $s = s_1 \circ s_2 \circ \cdots \circ s_k$, $s_i \in d[\,]$.

So $J(s_k) = \text{True}$, then we have $J(s_{k-1} \circ s_k) = (s_{k-1} \in d[\,]) \wedge J(s_k) = \text{True}$, $\cdots$, then $J(s) = J(s_1 \circ \cdots \circ s_k) = (s_1 \in d[\,]) \wedge J(s_2 \circ \cdots \circ s_k)$

= True, so we'll always return True.

If s isn't interpretable. Assume by contradiction that we return True on s. Then $J(s) = \text{True}$. s is never a base case. So $J(s) = \underset{\text{(for some } s_1)}{(s_1 \in d[i]) \wedge J(s_2)} = \underset{\text{(for some } s_1, \cdots, s_k)}{(s_1 \in d[i]) \wedge \ldots \wedge (s_k \in d[i])} = \text{True}$. So $s = s_1 \circ s_2 \circ \cdots \circ s_k$, each $s_i \in d[i]$. So s is interpretable.

Contradiction!

So our algorithm returns True iff s is interpretable.

<u>Runtime Analysis:</u> We have $O(n)$ subproblems.

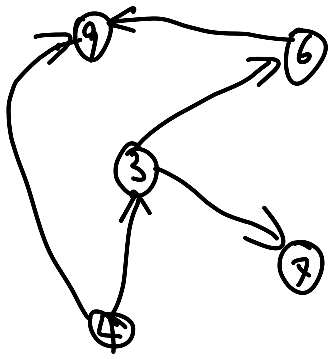
In each subproblem, we make $O(k)$ comparisons of all elements in $d[i]$ and the prefix of current string.

Each comparison takes $O(l)$ time.

So total runtime is $O(nkl)$.

<u>Space Analysis:</u> Only need a n-length space to store our results. So ~~total~~ space complexity is $O(n)$.

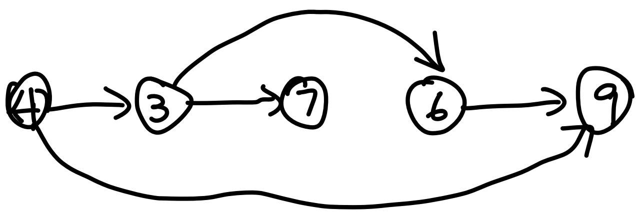
3.



Algorithm Description: Topological sort the graph.

Compute every T[v] in reverse order of topo-sort.

(T[v] : target of v)



① Subproblem: When we consider a vertex v, it only has targets behind it in the topo-order. So we use the nodes behind it as subproblems to compute it (precisely, nodes behind it and have edges with it)

② Recurrence

Use an array M[v] to store s[T[v]].

$$T[v] = \begin{cases} \underset{(v,u)\in E}{\operatorname{argmax}} \{M(u)\}, & \text{if } \max(M(u)) > s[v] \text{ and there exists } (v,u) \in E \\ v, & \text{otherwise} \end{cases}$$

T[u] = u. (Base Case)

(u is the last vertex in topo-order)

③ Ordering: Compute $T[v]$ by the reverse topo-order.

Runtime Analysis: Topological Sort : $O(n+m)$

Subproblems : $O(n)$

Each subproblem we take $O(\text{out-degree}(v))$ time

So total runtime is $O(n+m)$, which is linear.

$(n = |V|,\ m = |E|)$.