Saransh - Agarwal 2019mT60763

Ques) Input : set of 3-D boxes with dimensions h_i, w_i, d_i
Output : max. height of stack.

Operations allowed
1) We can rotate any box to get base.
2) Allow to use multiple instances of a box.

Constraints
We can only stack a box on top of another box if dimensions of the 2-D base of lower base are each strictly Larger than upper box.

Algo)
1) There will be $3n$ combination of h, w, d possible by rotating each box. store all these in array.

2) Sort the array in order of base area. i.e $(w_i d_i)$.

3) Now we have to remove such boxes which violates the above constraint.

4) Now let max possible height of stack with i^{th} box at top be $mPS(i)$ Then

$$mPS(i) = \max[mPS(j), mPS(j) + h(i)] \text{ st } j<i \ \& \ w_j > w_i \ \& \ d_j > d_i$$

5) return $\max[mPS(i)]$ for $0 \le i < 3n$.

Running Time Analysis) Sorting will take $O(n \log n)$, then for removing conflicts we have two iterate through two for loops which takes at worst $O(n^2)$

Total time = $O(n \log n) + O(n^2) \sim \boxed{O(n^2)}$ Ans

Saransh - Agarwal 2019mT60763

Proof of correctness) By Induction

Base case) When $i=1$

Then $mps[i] = h[i]$ which is trivial because if box has Largest Dimensions so it will act as Base box and it can't be kept on any other box.

IH)

$mps[j]$ = max. possible height of stack with j^{th} box at top.

IS) Let for some k such that it satisfies The constraints mentioned above and $mps(k)$ is maximum. — (1)

If for some j st $j<i$ and $w_j > w_i$ and $d_j > d_i$ i^{th} box is stack over j^{th} box.

$\Rightarrow$ Then height of stack $(H)^{max}$ = $h[i]$ + height of the stack with box j at top.

$\Rightarrow$ But we know by IH
$mps[j]$ = max. possible height of stack with j^{th} at top.

So $H = h[i] + mps[j]$

by Eq (1)

$H = h[i] + mps[j] \leq h[i] + mps[k]$

so $mps[i] = h[i] + mps[k]$

Hence $mps[i]$ = max. possible height of stack with box i^{th} at top.

Hence proved

Saransh - Agarwal 2019mT60763

Ques 2) Problem) : partition given integers into 2 subsets S_1, S_2.
Input : set of n-integers in range $[0,K]$.
Output : To minimize $|S_1 S_2|$

Where S_1, S_2 = sum of elements in each subset.

Algo:

1) find minimum no. among given integers. let it be v_1.

2) Then $S_1 = \{v_1\}$, $S_2 = \{$ remaining elements $\}$

3) return S_1, S_2.

4) find ~~product~~ Sum of all the integers Then subtract v_1 Let sum = X

5) return $|v_1 X|$

Running Time analysis
In step-1, we have to find min. element among given n-integers for which we have to iterate the array ones which takes O(n) time. Suming also takes O(n) Time.

Total Time = $\boxed{O(n)}$ Ans

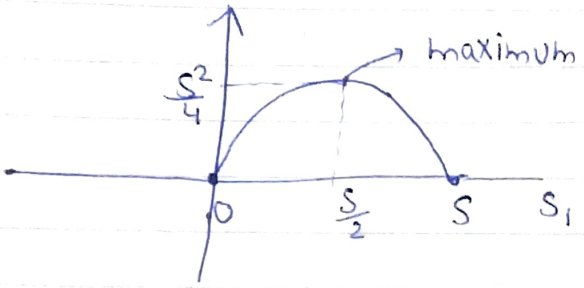
Proof of correctness)
We have to partition n-numbers in such a way that minimizes $|S_1 S_2|$. S_1, S_2 = Sum of elements in a partition.

Let numbers by $a_1, a_2 \text{ ---- } , a_n$
Let Total sum be $S = \sum_{i=1}^{n} a_i$

Then problem reduces to minimizing

$\Rightarrow |S_1(S - S_1|$ as $S_1 + S_2 = S$

$|S_1 S - S_1^2|$ = This will depend on S_1 as S is fixed This is downward parabola.



At $\frac{S}{2}$ we get maxima, By above graph we can see than $|S_1 S - S_1^2|$ is minimum when $S_1 = 0$ or S But $S_1 \neq 0$ as $a_i > 0$ and $S_1 \neq S$ otherwise $S_2 = 0$ which is also not possible.

→ By graph we can observe that S_1 must be taken close to zero or S to get minima.

→ given $a_i > 0$ The minima S_1 we can choose is having only one element in S_1 which is minimum among all the elements of given n-integers.

→ so $S_1 = \{ \min_{i=(1..n)} a_i \}$

$S_2 = \{$ remaining elements $\} = S - S_1$

Thus by choosing S_1, S_2 in such a way we minimizes the product $|S_1 S_2|$.

Hence proved

Saransh Agarwal 2019mt60763

Ques 3)

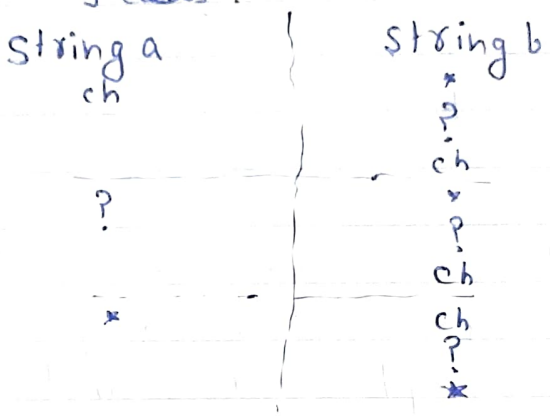
Input : given two strings a, b s.t. len(a) = m, len(b) = n both contain wildcard characters.

Output : whether the string matches.

Problem) wild card characters are '*', '?'

Suppose we have analysed string a till i^{th} length and string b till j^{th} length then at ~~(i+1)th and (j+1)th~~ any pos. we can have Total 9 cases.

↳ a[i], b[j]



Let Lookup[m][n] be DP Table which stores T/F.

We have to explore all 9 cases,

Case I) a[i-1] = 'ch' & b[j-1] = '*'

Then Lookup [i][j] = Lookup [i][j-1]

Because * can be matched with any character

Case II) a[i-1] = 'ch' & b[j-1] = '?'

Then Lookup [i][j] = Lookup [i-1][j-1]

Because ? matches any character so we check at i-1, j-1.

Case III) a[i-1] = 'ch' & b[j-1] = 'ch'

Then If a[j-1] != b[j-1]

Lookup [i][j] = false

Else

Look [i][j] = Lookup [i-1][j-1]

Case IV) If a[i-1] = '?' & b[j-1] = 'ch'
Then Lookup[i][j] = Lookup[i-1][j-1]
By same Logic as in case II.

Case V) If a[i-1] = '?' & b[j-1] = '?'
Then Lookup[i][j] = Lookup[i-1][j-1]
By same Logic as case II.

Case VI) If a[i-1] = '?' & b[j-1] = '*'
Then
Lookup[i][j] = Lookup[i-1][j]

Case VII) If a[i-1] = '*' & b[j-1] = *, ?, ch
Now if * matches with none.
Then if Lookup[i-1,j] = T
Lookup[i][j] = Lookup[i-1][j]
Else
Lookup[i][j] = Lookup[i][j-1]

given * can be matched with any character of other string.

Now we have derived all the recurrences, it is easy now.

1) Create a Table Lookup[m+1][n]

2) Run two loops
for (i=0 ; i<m ; i++)
for (j=0 ; j<n ; j++)
calculate LOOKUP[i][j] using recurrence derived.

3) return Lookup[m][n].

Saransh - Agarwal 2019mT60763

proof of correctness) By Induction

Lookup[0][0] = True If we consider len upto to zero of both string then they will match, so Lookup[0][0] = is true is trivial Base case.

Now If a[i-1] = (*)
Then using above recurrence we have derived we got,

Look[i][0] = Lookup[i-1][0]

Similarly If b[j-1] = '*'
Then using above recurrence we have derived we got,

Lookup[0][j] = Lookup[0][j-1]

Now) The algorithim has already computed all the subproblems which are smaller are computed correctly so Lookup[i][j] is computed correctly using recursion.

Hence proved

Running - Time Analysis)

~~Time complexity~~

for fixed i,j Lookup[i][j] takes O(1) time we have total mn queries so

Total-time = $\boxed{O(mn)}$ Ans

Saransh - Agarwal 2019mT60763

Ques4) Input : A coins with values let say $c_1, c_2 \dots c_A$.
Output : max. amount of money you can win.

Algo

1) By using DP we have to pick a coin in such a way that maximizes the amount.
2) Let $OPT(i,j)$ = maximum amount which can be winned when coins i^{th} - j^{th} are remaining.
$c_i, c_{i+1} \dots, c_j$.
3) At any stage we have 2 choices, we can pick c_i or c_j from either end.
4) Case I) If I pick c_j, second player will choose c_i or c_{j-1} which ever is max. Then
$OPT_1(i,j) = c_j + \min[OPT(i+1, j-1), OPT(i, j-2)]$.
5) Case II) If I pick c_i, second player will choose c_j or c_{i+1} which ever is max. Then
$OPT_2(i,j) = c_i + \min[OPT(i+2, j), OPT(i+1, j-1)]$.
6) Thus, we will get

$$OPT(i,j) = \max(OPT_1(i,j), OPT_2(i,j))$$

i.e

$$OPT(i,j) = \max\Big[c_i + \min(OPT_2(i+2,j), OPT_2(i+1,j-1)),\ c_j + \min(OPT_1(i+1,j-1), OPT_1(i,j-2))\Big]$$

$$0 \le i, j \le A$$

7) Return $OPT(0, A-1)$.

Running Time Analysis →
for fixed i, j $OPT(i,j)$ takes $O(1)$ Time.
Total - Time → proportional to the size of table
Which is $\boxed{O(A^2)}$ at worst.

Saransh- Agarwal 2019mT60763

proof of correctness)

By Induction we will prove.

Base case : When there are only 2 coins.

Then $j - i = 1$

Then $OPT(i,j) = \max(c_1, c_2)$

Now for computing $OPT(i,j)$ it uses computing $OPT_1(i,j)$ and $OPT_2(i,j)$.

Where $OPT_1(i,j)$ uses computing at $OPT(i+1, j-1)$ and $OPT(i, j-2)$

By induction on $|j-i|$ it is correct.

Similarly for $OPT_2(i,j)$ uses computing at $OPT(i+2, j)$ and $OPT(i+1, j-1)$.

By induction on $|j-i|$ it is correct.

Thus computing $(i,j) = OPT(i,j)$.

Also at each step no. of coins decreas by 2 so algo is bound to terminate.

Optimal Strategy which guarantees a win)

Consider coins $c_1, c_2, \ldots\ldots, c_A$ A = Even. Let

S_{Even} = sum of coins at Even pos.

S_{Odd} = sum of coins at odd pos.

If $(S_{Even} \geq S_{Odd})$

pick coins at Even pos. only.

Else

pick coins at odd pos. only.

Saransh - Agarwal 2019MT60763

Proof of correctness)
The above strategy is optimal to win and always possible to follow.
Since Total no of coins is Even. That means no of coins at Even pos = no of coins at odd pos.

WLOG) Let $S_{Even} \geq S_{Odd}$ so pick coin at Last pos which is Even, Now second player has choice B/w first pos [which is odd] and (A-1) pos which is also odd, so it picks any coin it doesn't matter, now again we have coins at Even and odd pos. pick coin at Even pos. now second player has to choose B/w odd pos. and this way games continues at end you have all coins at Even pos. and you win.

Similarly we can argue for $S_{Even} < S_{Odd}$.

Run-Time Analysis) : we have to iterate through array of coins which will take O(n) Time.

Ques 5) Input : Given an $p \times q$ chessboard with some squares removed.

Output : find max. no of rooks on boards such that none of them can attack each other.

Algo)

1) Construct graph with rows and columns as vertices.
2) Now we will have edges from rows to every columns if corresponding square is not removed, else no edge.
3) Run maximum - bipartite matching on this graph.
4) Output maximum matching.

Proof of correctness:)

→ If no square is removed then we can easily say than maximum no. of rooks $= \max(p, q)$. as we can place rook in each row such that they don't attack each other.

→ Also we can place at max. one rook at each block. i.e each row corresponds at max one column where rook can be placed.

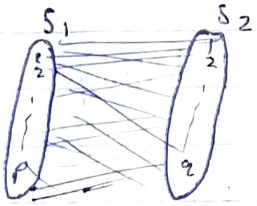
→ Now we have consider our given problem as graph.

→ The rows and col. are taken as vertexes.

→ There will be edges from Every row to Every columns if corresponding square is not removed. If square is removed then there will be no edge, b/w that row and col.

Now we can observe that we will get two sets.

S_1 = set of all rows.
S_2 = set of all columns.



Each row is having edge to every col. if square is not removed.

which is nothing but bipartite graph as S_1 and S_2 are disjoints also, and there are no Edges within S_1, S_2.

→ Now if we run maximum-bipartite matching on this graph we will get max. no of rooks placed.

→ As matching in max.-bipartite matching algo, gives set of Edges such that no vertex is touched by more than one Edge.

~~Lemma) The Edges~~

Lemma) The number of rooks that can be placed is atmost the no. Equal to maximum-matching in the above bipartite graph.

Saransh - Agarwal 2019mT60763

Proof) As max. - matching gives set of edges st no vertex is touched by more than one edge.

→ Now that edge will be disconnected from all other rows, cols thus rook can placed on that edge ⇒ rook can be placed at the block formed by intersection of that row, col.

→ And this edge is not connected with any other vertex so rook will not be in attacking pos at this edge.

→ No of rooks = atleast no of edges.

Now we will show ↯ No of rooks > no of edges they will attack each other.

→ Suppose give statement is true then there is rook placed at edge e st e ∈ graph and e ∉ max. - matching. This implies ~~this~~ that there exists a vertex(V_1) which appears in two edges

→ But then rook placed corresponding to vertex (V_1) and rook placed at edge e will attack each other. which is contradiction.

Thus we have max. no of rooks = no of edges = no. of max. - matching.

Running Time Analysis)

We know maximum - bipartite matching is done in $O(hC)$ Time

in our case $h = p$ $C = pq$

so Total Time = $\boxed{O(p^2q)}$ Ans

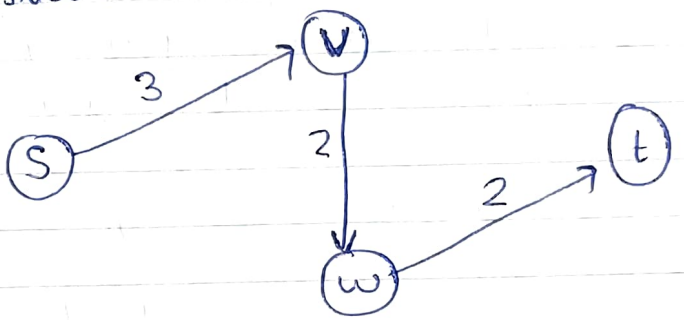
Ques 6)

a) Let $\vec{G} = (V, \vec{E})$ be a flow network with source s and sink t, and a +ve integer capacity c_e on every edge e. If f is maximum s-t flow in $\vec{G}$, then f saturates every edge out of s with flow [i.e for all edges e out of s, we have $f(e) = c_e$].

Ans) The given statement is false.

We will show by counterexample.

Consider

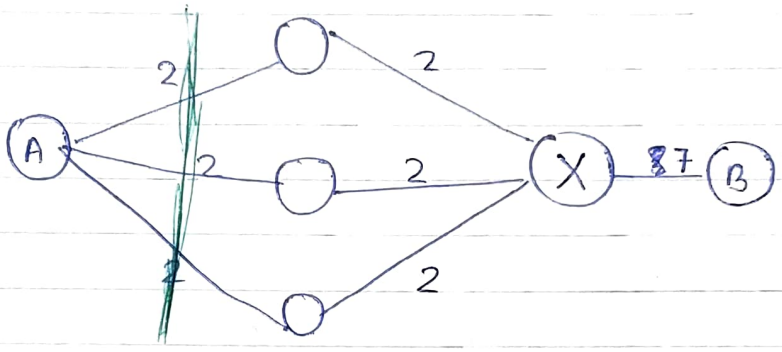


In the above example If we consider maximum-flow from (S-V-W-t) S-t path then it will be 2 due to 2 capacity in edge (v-w).
But Then (S-V) has 2 as max-capacity so f does not saturates every edge out of s with flow.
Hence give statement is false.

b) Let $\vec{G} = (V, \vec{E})$ be a flow network with source s and sink t, and a positive integer capacity $\{c_e \mid e \in \vec{E}\}$. Now suppose we add 1 to every capacity; then (A, B) is still a minimum s-t cut with respect to these new capacities $\{c_e + 1 \mid e \in \vec{E}\}$.

Ans) The given statement is false.

We will show by counterexample.



The min cut will be shown in green.
Now if we increase each capacity by 1
Then new capacity will be 3, 3, 3, 3, 3, 3, 8
Then Total capacity of green cut will becomes 9. while cut from X-B has capacity 8 so it will become new s-t cut.
Thus minimum s-t cut changes. Hence given statement is false.