Lowest Common Ancestor using Binary Lifting:-

Given a tree. For a query (u, v) find lowest common ancestor of nodes u and v. If there are multiple nodes we pick the one that is farthest away from the root ~~node~~ node.

↳ lowest ancestor

Approach

Binary Lifting is a dynamic programming approach.

We precompute ~~no~~ ancestor [1, n] [1, log(n)] where ancestor [i] [j] contains 2^j j-th ancestor of node i.

1st Choice

Page No.
Date: 10/3/23

For computing ancestor[ ][ ]:-

ancestor Matrix: ancestor[i][j] = ith node's 2^jth ancestor in the path

Base initialization:

ancestor[i][j] = ancestor[i][0] (first parent (2^0) of each node is given)

Transition:

ancestor[i][j] = ancestor[ancestor[i][j-1]]

Meaning: $A(i, 2^j) = A(A(i, 2^{(j-1)}), 2^{(j-1)})$

To find the (2^j)-th ancestor of i, recursively find ith node's $2^{(j-1)}$th ancestor's $2^{(j-1)}$th ancestor. ($2^j = 2^{(j-1)} + 2^{(j-1)}$)

So ancestor[i][j] = parent[i] if j = 0 and
ancestor[i][j] = ancestor[ancestor[i][j-1]][j-1]
if j > 0

1st Choice

Page No.
Date

We first check whether a node is an ancestor of other or not and if one node is ancestor of other then it is the LCA of those two nodes.

Code for find path ancestor of nodes.

```
vvi adj ; (n+1)
;
Vector<bool> vis (n+1)
vvi par; (log2(n+1), (n+1))
int l = ceil(log2(n))
int n.
void dfs(int s) {
    vis[s] = true
    for (int i=1; i<=l; i++)
        par[s][i] = par[par[s][i-1]][i-1];
    for (x:adj[s])
        par[x][0] = s; dfs(x)
}
```

```
int solve(int u, int p){
    for(int i=l; i>=0; i--){
        if(p & 1<<i)
            u = par[u][i];
    }
    return u;
}
int main{
    take edges in adj // one based indexing
    l = ceil(log2(n))
    par[1][0] = 1
    dfs(1);
    while(q--){
        cin >> u >> p;
        cout << solve(u,p) << endl;
    }
```

Minimum Element in Stack

Design a stack that supports push, pop, top and retrieving min element in constant time.

Approach 1

Use two stacks, 1 normal, other to store min at each instant.

Approach 2 (Constant space) **

O(1) variables.

Use math.

Let suppose at an instant the current min be x. And now it is to be updated to y.

What we can do is, use a mathematical equation in 2 variables x and y with a constant k.

Push k to stack whenever we update min. Otherwise push the element.

We will store current min variable and put it just like the stack so that at the time when we need to backtrack the stack we can compute 'u' using 'c' and 'k'

Eg 4

$2 * c - p = k$

as $p > c$, $\Rightarrow 2 * p - c < c$

So whenever while popping we get an element less than min, then need to backtrack.

* To return top(), if stack.top is less than min, then we will return the min element itself.

1st Choice

Page No.

How to implement stack using priority queue or heap

Approach

Using count/time as key in map.

Max size square sub matrix with all 1s.

Ex.

0	1	1	0	1
1	1	0	1	0
0	1	1	1	0
1	1	1	1	0
1	1	1	1	1
0	0	0	0	0

→

0	1	1	0	1
1	1	0	1	0
0	1	1	1	0
1	1	2	2	0
1	2	2	3	1
0	0	0	0	0

1st Choice

Page No.
Date

Ques 4 Dynamic programming.

⇒ 2D matrix dp.

~~dp[i][j] will for row and column from the start.~~

dp[i][j] will store matrix that can be generated ending at the position.

- Copy first row and first column as it is

Now traverse matrix.

When we encounter a 1.

$dp[i][j] = \min(dp[i-1][j-1], dp[i-1][j], dp[i][j-1]) + 1$

else

$dp[i][j] = 0$

Take max from dp[i][j].

The Celebrity Problem

N people in a party. One celebrity may be there may be not.

Celebrity does not know anyone in the party.

Everyone know the celebrity.

Find if there's a celebrity in a(n) the given ! is making of who knows who.

Approach $O(n^2)$ is simple, try $O(N)$.

We will try to find potential celebrity.

By method of eliminating.

If we ask A and B whether A knows B

If A knows B ⇒ A can't be a celebrity

If B knows A

A don't know B ⇒ B can't be a celebrity.

This will give us n-1 questions to find a potential celebrity.

Check first two elements, eliminate one of them. Now check resultant with other elements in similar manner.
Finally check whether the potential celebrity is actually a celebrity or not.

For the above approach, there can be 3 implementation.

1. Recursion.

```
find potential(n)
{ if(n==0)
    return -1.
  int id = find potential(n-1)
  if(id==-1)
    return n-1;
  if(id know n-1)
    return n-1
  else if(n-1 know id) else return -1. }
    return id
```

2. Stack.

Push all elements in stack.

(1) — pop two elements check potential and push back to stack

Repeat (1) until there's only one element in stack.

3. Two pointer.

i → start
j → end
while(i != j){
    check i and j
    if(eliminate(i))
        i++
    else j--
}
when i == j, i will be potential celebrity.

Longest Valid Parenthesis

Given ~~just the~~ a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) ~~&~~ parenthesis substring.

Ex: s = "(()", output: 2.

Ex: ")()())", output: 4

Ex: "", output: 0

Sol^n Brute force is $O(n^3)$, $O(n)$ to check $O(n^2)$ for combinations.

Approach 1: DP $O(n), O(n)$

1-D dp, only substr ending with ')' could be Sol^n.

otherwise 0.

dp[i] → longest valid substr ending at i.

Case1:- $S[i-1] == '('$

$dp[i] = dp[i-2] + 2$.

Case2:- $S[i-1] == ')'$

if $(s[i - dp[i-1] - 1] == '(')$

$dp[i] = dp[i-1] + dp[i - 2 - dp[i-1]] + 2$.

Take max of all $dp[i]$.

Approach-2 Stack $(O(n), O(n))$

Stack keeps the track of the last index that is unmatched. (Nearest to left)

Whenever we encounter '(' we push its index to stack.

When we encounter ')' we will match it with top by popping the element.

top from
Now the stack will tell me the nearest to left flag index that's unmatched.

ans = max(ans, i - st.top).

Initially push -1 into the stack that the unmatched flag initially.

* Whenever our stack becomes empty that means our ')' got matched with the flag itself.
So we don't check answer & and simply push current index to stack as a flag for future.

Code:

```
st.push(-1)
loop(0 to n-1)
  if(s[i] == '(')  st.push(i);
  else {
    st.pop()
    if(st.empty()){
      st.push(i)
    } else {
      ans = max(ans, i - st.top())
    }
```

** Approach 3 O(n), O(1)

Keep two counters left and right.

( )

'(' '('

while left > right

there's hope to get matched in future

do nothing

when left < right

this is a breakpoint, there's not hope for this ')' to match onwards

reset left and right to 0.

when left == right

possible ans.

ans = max(ans, left);

* We will do two passes first will match lefts with right, second will match rights with lefts. In the similar way.

KMP Algorithm O(n)

txt[0....n-1], pat[0....n-1]

⇒ Create lps[] array of size of the pattern.
↳ longest prefix suffix

lps[i] stores length of the maximum matching proper prefix which is also a suffix of the sub pattern pat[0...i]

```
computeLPS Array()
    int len=0
    lps[0]=0
    int i=1;
    while(i<M){
        if(pat[i]==pat[len]){
            len++
            lps[i]=len
        } i++
```

```
else {                    // pat[i] != pat[len]
    if(len != 0) {
        len = lps[len-1]
                          // don't increment i here
    } else {
        lps[i] = 0;
        i++;
    }
}
end loop
```

1st Choice

Page No.
Date

Searching Algo:-

→ Keep matching txt[i] and pat[j] and keep incrementing i and j while they keep matching.

When we see a mismatch

- We know that characters pat[0...j-1] match with txt[i-j...i-1]
- lps[j-1] is count of character of pat[0...j-1] that are both proper prefix and suffix
- We donot need to match these lps[j-1] bcoz these will anyway match.

KMP Search () {

int M = pat.length
int N = txt.length

int lps[M]

computeLPSArray();

int i = 0
int j = 0

while (i < N) {
if (pat[j] == txt[i]) {
j++
i++
}

if (j == M) {
pattern found
j = lps[j-1]
}

```
else if (i<M && pat[j] != txt[i])
    if (j != 0)
        j = lps[j-1]
    else
        i++
}
}
```

String Stream

<< → add a string to stringstream object

>> → read something from ss object.

int countwords (string s)
{
    stringstream ss(s);
    string word
    while (ss >> word)
        count++;
}

Mainly used to break string into words.

1st Choice

Page No.
Date / /

Reverse the String :

Given a string, reverse the string word by word.

Eg: "The sky is blue"

output :- "blue is sky The"

Code:

```
String answer = "";

    String stream SS(A);
    String word;
    Stack (String) st;
    while( SS>> word){
        st. push(word);
    }
    while( !st.empty() ){
        String x = st. top(); st.pop()
        answer += (x + " ");
    }
return answer;
```

Longest Palindromic Substring

Use longest common substring by using an extra string that is reverse of the original string.

Twist

Edge case: dcabafe/abacd → matched

reverse dcabafe abacd

output = dcaba

correct ans = aba

Reason: The substring should match at the same position in the reverse string

Solution, answer is update iff

$$j == n - i + dp[i][j]$$

dcabafe/abacd

i / dp[i][j]

dcabafeabacd

j

n - i

Segment tree

- Sum of a given range type problems

- Representation of a segment tree

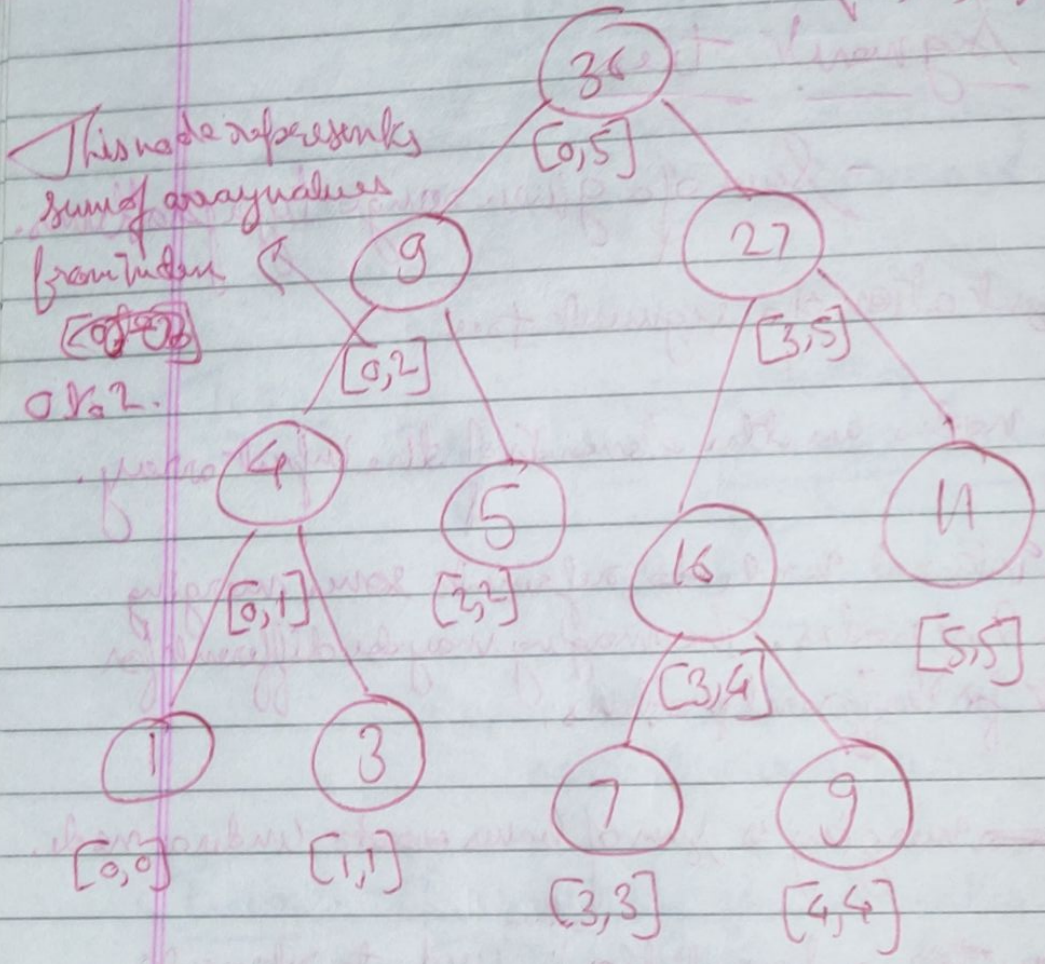
1. Leaf nodes are the elements of the input array.

2. Each internal node represents some merging of the leaf nodes. The merging may be different for different for different problems.

Here merging is sum of leaves under a node.

An array representation of tree is used to represent Segment trees. For each node at index i, the left child is $2i+1$, right child is $2i+2$. and the parent is at $\lfloor (i-1)/2 \rfloor$.

{1, 3, 5, 7, 9, 11}



Construction of segment tree

We start with a segment arr [0...n-1] and every time we divide the current segment into two halves. Then recur for both halves and for each segment store sum in the corresponding node.

All the levels of the constructed ST will be completely filled except the last level.

The tree will be a full binary tree. n leaves
n-1 internal nodes
2*n-1

Size of the tree will be?

There are n leaf nodes and n-1 internal nodes, hence 2n-1 when n is a power of 2. Else it is 2x-1 where x is next greater power of 2 than n.

Query for sum of given range

int getSum(node, l, r)

If the range of the node is within l and r

return value in the node

else if the range of the node is completely outside l and r

return 0

else

return getSum(node's left child, l, r)

+ getSum(node's right child, l, r).

Update a value

Also recursive, given an index which needs to be updated. Let diff be the value to be added.

Start from the root and add diff to all nodes which have given index in their range.

1st Choice

Page No.
Date

Code:-

int getMid(int s, int e){ return s+(e-s)/2; }

vector<int> constructST(int arr[], int n){

int x = (int)(ceil(log2(n))) // height for tree.

int max_size = 2*(int) pow(2,x)-1;

vector<int> st(max_size)

construct(arr, 0, n-1, st, 0);

return st

}

void construct(int arr[], int ss, se, vector<int>& st, si)

if(ss == se){

st[si] = arr[ss]

return arr[ss];

}

int mid = getMid(ss, se)

st[si] = construct(arr, ss, mid, st, si*2+1) +

construct(arr, mid+1, se, st, si*2+2)

return st[si] }

1st Choice

Page No.

```
void updateValue(int arr[], st, n, i, newVal)
{
    if(i<ss || i>n-1)
        return
    int diff = new_val - arr[i];
    arr[i] = new_val;
    update(st, 0, n-1, i, diff, 0);
}

void update(st, ss, se, i, diff, si){
    if(i<ss || i>se)
        return
    st[si] += diff;
    if(se != ss)
        int mid = mid(ss, se)
        update(st, ss, mid, i, diff, 2*si+1)
        update(st, mid+1, se, i, diff, 2*si+2)
}
```

```
int getSum(st, n, qs, qe)
    if(qs < 0 || qe > n-1 || qs > qe) {
    return -1; // Invalid input
    }
    return sum(st, 0, n-1, qs, qe, 0);
}

int sum(st, ss, se, qs, qe, si) {
    if (qs <= ss && qe >= se)
        return st[si];

    if (se < qs || ss > qe)
        return 0;

    int mid = getMid(ss, se)

    return sum(st, ss, mid, qs, qe, 2*si+1) +
           sum(st, mid+1, qs, qe, 2*si+2)
}
```