PAGE NO.
DATE: / /20

DP

Q s = (( ) (? ?) ?) ?
Find no of assignments to get balanced parenthesis at end.

```
int rec(int level, int bal)
{
if (bal<0) return 0;
if (i == level == s.length()) { if (bal==0) return 1;
                                else return 0; }
if (s[i]=='(') return rec(level+1, bal+1);
if (s[i]==')') return rec(level+1, bal-1);
if (s[i]=='?') return rec(level+1, bal+1) + rec(level+1, bal-1);
}
```

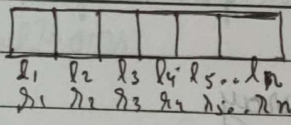
Q Find number of distinct subsequences of string.

abab

1→ (a) (b) a b
2→ (ab) (ba) ab
3→ (aba) (bab)
4→ (abab)

""	a	b	a	b
	1	2	4	8-1	14-2
				=7	=12

dp[i] = 2 * dp[i-1] - dp[lastpos-1]

Q N sized array 

$l_i \le a_i \le r_i$ $N, K \le 500$

Sum of array = X. Find no of possible ways.

if $l_i, R_i \le 500$ rec(level, sum) → rec(level+1, sum + l_i) … rec(level+1, sum + r_i)

Q N shops N days h[i][j] → i^{th} shop on j^{th} day, Happiness += h[i][j]
2 consecutive days you cannot go to the same shop. Maximise the happiness.

Teacher's Signature

$dp[i][j] = \max_{k \neq j} (dp[i-1][k] + h[i][j])$

we can keep track of topmost 2 values instead of $dp[N][M]$ we can do $dp[2][M]$

```
for(int i=1; i<=n; i++)
{
for (int j=0; j<m; j++) cin >> h[i];
pair<int,int> mx1={0,-1}, mx2={0,-1};

for (int j=0; j<m; j++) //to get topmost 2 of prev day
{ if (dp[i-1][j] > mx1.ff) { mx2=mx1;
                             mx1={dp[i-1][j],j};
  else if (dp[i-1][j] > mx2.ff) mx2={dp[i-1][j],j};
}

for (int j=0; j<m; j++) //to fill todays dp
{
 if (mx1.ss==j) dp[i][j]=mx2.ff+h[i];
 else           dp[i][j]=mx1.ff+h[i];
}
}
```

if want to save space $dp[i] \to dp[i\%2]$

Q Maximum Sum Subarray

$dp[i] = \max(0, dp[i-1]) + a[i]$

take max across all positions

Q Knapsack Problem

rec(level, wt) → rec(level+1, wt)

→ rec(level+1, wt − w(level)) + p(level)

PAGE NO.
DATE: / / 20

```
int rec(int level, int currcw)
{
 if (level == n) return 0;
 if (dp[level][currcw] != -1) return dp[level][currcw];
 int ans = rec(level+1, currcw);
 if (currcw - w[level] >= 0) ans = max(ans, rec(level+1, currcw - w[level] + profit[level]));
 return dp[level][currcw] = ans;
}
```

Remove this +1 to get ∞ supply

The LIS Problem

$rec(i) = max(1 + rec(j))$ for $0 \le j < i$ & $a[j] > a[i]$

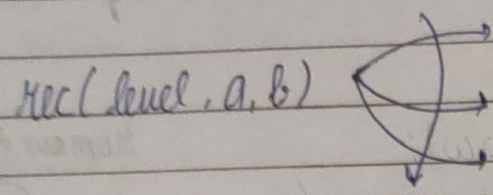
$O(N^2)$

```
int rec(int level)
{
 if (level < 0) return 0;
 if (dp[level] != -1) return dp[level];
 int ans = 1; // By default
 for (int prev = 0; prev < level; prev++)
 if (a[prev] < a[level]) ans = max(ans, 1 + rec(prev));
 return dp[level] = ans;
}

int LIS (vector<int> a)
{
 vector<int> sols;
 for (auto x : a) { if (sols.empty() || sols.back() < x) sols.pb(x);
                    else { auto it = lower_bound (all(sols), x);
                           *it = x; }
                  }
 return sols.size();
}
```

Teacher's Signature

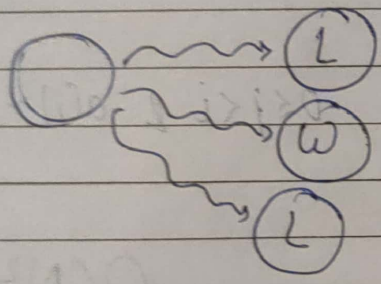
Q There are N players, each has X_i, Y_i
for each player, we can ignore or take X_i or Y_i
↳ we can take exactly A X_i & B Y_i
↳ score += $(X_i, Y_i, 0)$ find max score



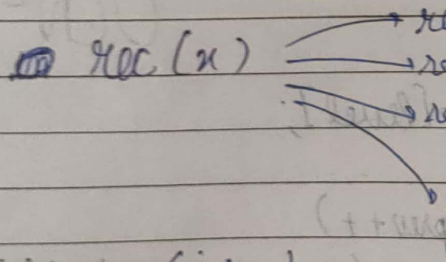
GAME DP

There are N chips.

Q You can take 2^m chips $[m \in [0, \infty)]$ from board.
Then $N-2^m$ chips are left on board.



If I can push the ~~set~~ situation to a loosing state, my current state will be a winning state




```
int rec(int x)
{
  if (x==0) return 0;
  if (dp[x]!=-1) return dp[x];
  int ans=0; //By default I am in losing state
  for(int m=0; (1<<m)<=x; m++)
  if (rec(x-(1<<m))==0) ans=1;
  return dp[x]=ans;
}
```

0 1 1 0 1 1

its periodic

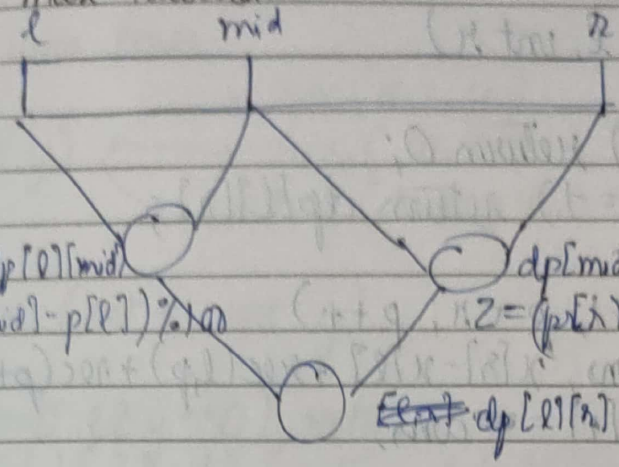
PAGE NO.
DATE : / /20

Q $N \le 500$ $a_1 \; [a_2 \; a_3] \; \ldots \; a_n$

↓

Replace by ~~$a_2 + a_3$~~ $(a_2+a_3)\%100$ and add $a_2 + a_3$ to reward.

Find max reward.




```
int rec(int l, int r)
{
if (l == r) return 0;
if (dp[l][r] == -1)
{
int ans = 0, tot = 0;
for (int i = l; i <= r; i++) tot += a[i];
int sum = 0;
for (int mid = l; mid < r; mid++)
{
sum += arr[mid];
ans = max(ans, rec(l, mid) + rec(mid+1, r) + (sum%100 +
                                  (tot - sum)%100));
}
dp[l][r] = ans;
}
return dp[l][r] = ans;
}
```

Q Rod cutting problem

There is a rod with length x_n and some weak points at $x_1, x_2 \ldots x_n$. You can break the rod, the length of current segments gets added to the final cost. Break with minimal cost.

Teacher's Signature

cost = ~~2+8=10~~ 10 + 8 OR 10 + 5

$$DP[l][r] = \begin{cases} \min_{\text{over all } p\text{'s}} (x_r - x_l) + DP(l,p) + DP(p,r) \end{cases}$$

```
int rec(int l, int r)
{
if(l+1==r) return 0;
if(dp[l][r]!=-1) return dp[l][r];
int ans = 1e9;
for(int p=l+1; p<r; p++)
ans = min(ans, x[r]-x[l] + rec(l,p) + rec(p+1,r));
return dp[l][r] = ans;
}
```

Substring / Subsequence Ideas

Q Find number of Binary strings of N length that don't contain "0100" as subsequence OR substring.

We will put in state how much prefix we have seen in "0100"

rec(level, x) → T[x] match → rec(level+1, x+1)

rec(level, x) → !T[x] not match → rec(level+1, x)

```
int rec(int level, int x)
{
if(x==4) return 0;
if(level==n) return 1;
if(dp[level][x]!=-1) return dp[level][match];
int ans = rec(level+1, x) + rec(level+1, x+1);
return dp[level][match] = ans;
}
```

we don't match — we match

Q. 2 piles (A, B) stones in them
In one turn → take any no. of stones from one pile
→ take equal no. of stones from both
Player who can't move loses.
Find who loses.

rec(x,y) ⟷ rec(z,y) 0 ≤ z ≤ x-1
rec(x,z) 0 ≤ z ≤ y-1
rec(x-k, y-k) 1 ≤ k ≤ min(x,y)

```
int rec(int x, int y)
{
if(x==0 && y==0) return 0;          // losing state
if(dp[x][y] != -1) return dp[x][y];

int ans=0;
for(int z=0; z<x; z++) if(rec(z,y)==0) ans=1;
for(int z=0; z<y; z++) if(rec(x,z)==0) ans=1;
for(int z=1; z<=min(x,y); z++) if(rec(x-z,y-z)==0) ans=1;
return dp[x][y]=ans;
}
```

If current state is winning

DP with caching across Queries

Q. You have n coins with certain values. Find all money sums you can create using these coins.

Changing state from how much we've taken to how much we've left to ~~take~~ make.
In query problems, taking how much is left to make is a better state.

rec(level, left) → take: rec(level+1, left - a[level])
→ not take: rec(level+1, left)
merge with OR operations.

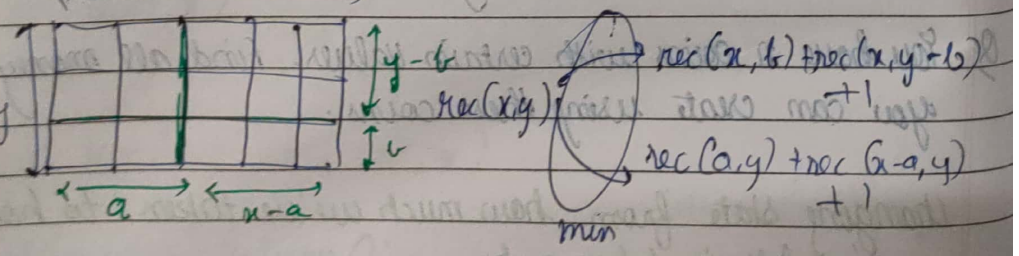
Teacher's Signature

```
int q; cin>>q; while(q--)
{
  int t; cin>>t;
  if(rec(0,t)) cout<<"Yes"<<endl;
}

int rec(int level, int left)
{
if(left<0) return 0;
if(left==0) return 1;
if(level==n+1) return 0;
if(dp[level][left]!=-1) return dp[level][left];

int ans=0;
if(rec(level+1, left) || (rec(level+1, left-a[level]))
ans=1;
return dp[level][left]=ans;
}
```

Q Given an axb rectangle, you have to cut it into squares. In 1 move you can make 1 cut. Find min. number of moves to make all squares.




```
int rec(int x, int y)
{
if(x==y) return 0;
if(dp[x][y]!=-1) return dp[x][y];
  int ans=INT_MAX;
for(int a=1; a<x; a++) ans=min(ans, rec(a,y)+rec(x-a,y)+1);
~ Same for b
return dp[x][y]=ans; }
```

Teacher's Signature

There can be only 15 prime factors

2 Watson has N elements. He wants the largest subsequence such that all pairs are relatively coprimes.
$1 \le N \le 50$
$1 \le A_i \le 50$

pairs

All numbers taken must not share a pf.

```
vector<int> sieve(51,0); sieve[1]=1; sieve[0]=0;
for(int i=2; i<=50; i++)
{
  if(sieve[i]==0) { sieve[i]=i;
                    for(int j=i+i; j<=50; j+=i) ~~sieve[j]=~~
                    if(sieve[j]==0) sieve[j]=i; }
}

map<int ~~set~~ set<int>> mp; mp[1].insert(1);
for(int i=2; i<=50; i++)
{
  int num=i;
  while(num!=1) { mp[i].insert(~~sieve~~ sieve[num]);
                  num/=sieve[num]; }
}
pos[2]=0; pos[3]=1; ... pos[47]=14;

int rec(int level, int mask)
{
if(level==n || mask==((1<<n)-1)) return 0;
if(dp[level][mask]!=-1) return dp[level][mask];
if(a==1) return dp[level][mask]=1+rec(level+1,mask);
bool flag=true; ~~int~~ int newmask=mask;
int ans=rec(level+1,mask);
for(auto pf: mp[a[level]]) { if(mask&(1<<pos[pf])) flag=false;
                             newmask|=(1<<pos[pf]); }
if(flag) ans=max(ans, 1+rec(level+1, newmask));
return dp[level][mask]=ans;
}
```

Teacher's Signature

PAGE NO.
DATE: / /20

Q. Fredo has two strings S & T. He asks if he removes S[i...j] from S, can T still be a subsequence of S.
Yes → if T is still subsequence of S
No → if T cannot be subsequence of S
length ≤ 10^5

Preprocessing → prefix[i] = largest prefix of T in S[0...i]
suffix[i] = largest suffix of T in S[i...n]

string s, t; cin >> s >> t;
int j=0;
for(int i=1; i<=s.size(); i++)
{
if(j<t.size() && s[i-1]==t[j]) j++;
prefix[i] = j;
}
j=0;
for(int i=s.size(); i>=1; i--)
{
if(j<t.size() && s[i-1]==t[t.size()-j-1]) j++;
suffix[i]=j;
}

cin >> i >> j; if(prefix[i-1]+suffix[j+1] >= t.size()) yes;
else no;

Q. N children, each can get [0,a[i]] candies. No candies should be left over. Find number of different ways.

dp[0][0 ≤ x ≤ a[i]] = 1 for base case
dp[x][y] = dp[x-1][y] + dp[x-1][y-1] ... + dp[x-1][y-a[x]] transition

[build prefix sums on dp[x-1]
dp[x][y] = prefix[y] - (y-a[x]==0 ? 0 : prefix[y-a[x]-1]
dp[n-1][k] gives ans

Teacher's Signature

```
cin >> n >> k; read(a, n);
vvi dp(n, vi(k+1, 0));

for (int i=0; i<=a[0]; i++) dp[0][i] = 1;

for (int i=1; i<n; i++)
{
  vector<int> prefix(k+1);
  prefix[0] = dp[i-1][0];
  for (int j=1; j<=k; j++) prefix[j] = prefix[j-1] + dp[i-1][j];

  for (int j=0; j<=k; j++)
  {
    dp[i][j] = prefix[j] - (j - a[i] == 0 ? 0 : prefix[j - a[i] - 1]);
  }
}

cout << dp[n-1][k];
```

KNAPSACK ⇒ 2

Taro has decided to choose some N items and carry them home in a ~~bo~~ knapsack of capacity W. Find maximum profit he can get.

$1 \le N \le 100$

$1 \le W \le 10^9$ → we can't memorise this.

$1 \le w_i \le W$

$1 \le v_i \le 10^3$

max value can be $10^3 \times 10^2 = 10^5$ if you take all items. However you have weight limit W. So we will query from 10^5 to 0 and do rec(level, i) → ~~can we make~~

rec(0, i) will give min weight to make i profit if it is less than W, break and print.

Teacher's Signature

```
for (int i = 100000; i >= 0; i--)
{
if (rec(0, i) <= W) { cout << i << endl; break; }
}

int rec(int level, int left)
{
  if (left == 0) return 0;
  if (level == n) return 1e17;
  if (dp[level][left] != -1) return dp[level][left];

  int ans = rec(level+1, left);
  if (left - profit[level] >= 0)
  ans = min(ans, weight[level] + rec(level+1, left - profit[level]));
  return dp[level][left] = ans;
}
```

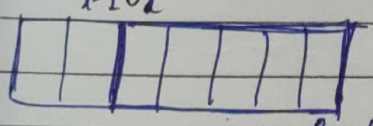
use left and not taken as state to cache across queries

Q Palindrome Partitioning 3

You are given string s and int k.

① Change some characters of s

② Partition s into k non-empty disjoint palindrome strings. Return minimal characters you need to change



$dp(i, j) \xrightarrow{\min} dp(i-1, j-1) + cost(i, level)$

choose some $i <=$ level and make i to level palindrome then call for i-1 & j-1 partitions

```
int rec(int level, int k)
{
  if (k == 0 && level != -1) return 1e9;
  if (level == -1) return k == 0 ? 0 : 1e9;  if (dp[?][?] != -1)...
  int ans = 1e9;
  for (int i = level; i >= 0; i--)
  ans = min(ans, rec(i-1, k-1) + cost(i, level));
  return dp[level][k] = ans;
}
```

Teacher's Signature

PAGE NO.
DATE: / / 20

Q MCM DP

Given an array, if you burst i^{th} balloon, you get score nums[i-1] * nums[i] * nums[i+1], if i-1 / i+1 goes out of bound consider 1.

Find max happiness = Σ after bursting all balloons

example [3,1,5,8] → change → [1,3,1,5,8,1] (we won't burst these)

last → [1,8,1] score += 1*8*1

2nd last → [1,3,8,1] score += 3*8*1

3rd last → [1,3,5,8,1] score += 3*5*8

4th last → [1,3,1,5,8] score += 3*1*5

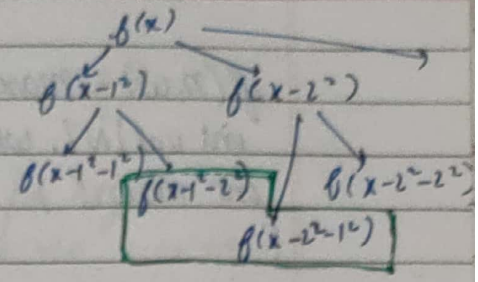
∴ for l & r choose one which gives maximum score

```
int rec(int l=1, int r=n)
{
if (l>r) return 0;
if (dp[l][r] != -1) return dp[l][r];
int ans=0;
for (int p=l; p<=r; p++)
ans= max(ans, nums[l-1]*nums[p]*nums[r+1]+
          rec(l,p-1)+rec(p+1,r));
return dp[l][r]=ans;
}
```

Teacher's Signature

Q. Minimum no. of squares

$f(x) = \min(f(x - i*i) + 1)$




```
int MinSquare (int n)
{
  if (n==1 || n==0 || n==2 || n==3) return n;
  int ans = MOD;
  for (int i=1; i*i<=n; i++)
  {
    ans = min(ans, MinSquare(n-i*i) + 1);
  }
  return ans; }
```

↓ NOW MEMOISE

```
int main()
{
  cin>>n;
  vector<int> dp(n+1, MOD);
  cout<<dp[n-1];
}

int MinSquare(int n, vector<int> dp)
{
if (n==1 || 2 || 3 || 4) return n;
int ans = INT_MAX;  ——→ if (dp[n] != MOD) return dp[n];
for (int i=1; i*i<=n; i++)
{
  dp[n] = min(dp[n], 1 + MinSquare(n-i*i, dp));
}
return dp[n];
}
```

TABULATION

```
int n; cin>>n; vector<int> dp(n+1, MOD);
dp[0]=0; dp[1]=1; dp[2]=2; dp[3]=3;
for (int i=1; i*i<=n; i++)
{ for (int j=0; j
```

Teacher's Signature

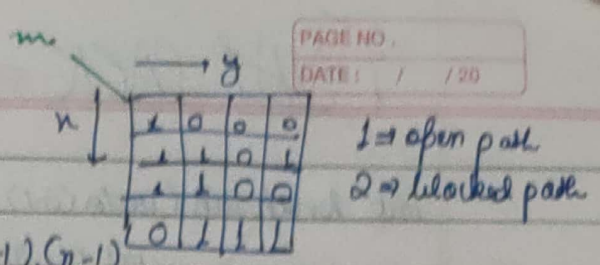
KNAP SACK 2D - dp array

```
int knapsack(vector<int> weight, vector<int> value,
             int n, maxWeight)
{
vector<vector<int>> dp(n, vector<vector<int>> maxWeight);
return solveMem(weight, value, n-1, maxWeight, dp);
}

int solveMem(vector<int>& weight, vector<int>& value, int index,
             int capacity)
{
 if (index==0) { if (weight[0] <= capacity)
                   return value[0];
                 else
                   return 0; }
 if (dp[index][capacity] != -1) return dp[ ][ ];
int include = 0; exclude = 0;
if (weight[index] <= capacity)
  include = value[index] + solveMem(weight, value, index-1, capacity, dp);
  exclude = 0 + solveMem(weight, value, index-1, capacity, dp);
int dp[index][capacity]
int ans = max(include, exclude);
return ans; dp[index, capacity]
}
```

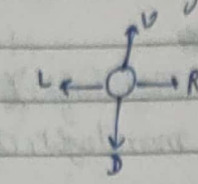
→ TABULATION

```
int solveTab(vector<int>& weight, value, capacity)
{
vector<vector<int>> dp(n, vector<int>(maxWeight+1), 0);
for (int w=weight[0] ; w <= capacity ; w++)
{ if (weight[0] <= capacity)
    return value[0]; dp[0][w] = value;
  else
    return 0 dp[0][w] = 0;
}
for (int index = 1; index < n ; index++)
{ for (int w=0; w <= capacity ; w++)
{ int include = 0; if (weight[index] < w)
  include = value[index] + dp[index-1][w-weight[index]];
  include = 0 + dp[index-1][w];
  dp[index][w] = max(exclude, include);
return dp[n-1][capacity];
```



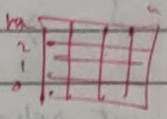
Rat in a maze problem

Rat has to go from (0,0) to (n-1),(n-1)




```
vector<string> findPath(vector<vector<int>> &m, int n)
{
  vector<string> ans;          // if (m[0][0] == 0) return ans;
  int srcx=0, srcy=0;
  vector<vector<int>> visited = m;
  for(i=0;i<n;i++) for(j=0;j<n;j++) visited[i][j]=0;
  string path="";
  solve(m, n, srcx, srcy, visited, path);
  sort(ans.begin(), ans.end());
  return ans;

void solve(vector<vector<int>> &m, int n, vector<string> &ans,
           int x, int y, visited, path)
  {
//base case if (x==n-1 && y==n-1)
    { ans.push_back(path); return; }
//mark visited[x][y] == 1;
    int newx, newy;
//down  newx = x+1; newy = y;
    if(isSafe(newx, newy, n, visited, m))
      { path.push_back('D');
        solve(m, n, ans, newx, newy, visited, path);
        path.pop_back(); }
//up //left //right
// unmark visited[x][y] = 0;
  }
```



Teacher's Signature

n item	1	2	3	4	... n
weight	w_1	w_2	w_3	...	w_n
value	v_1	v_2	v_3	...	v_n

Maximum W

PAGE NO.
DATE : / /

Knapsack Problem

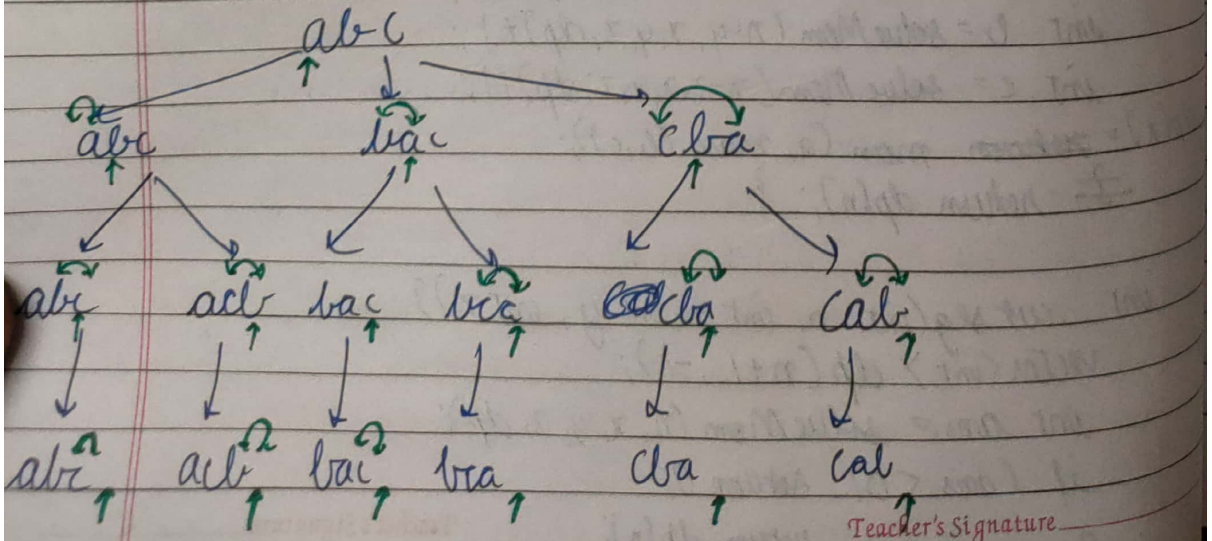
Permutations of a string

```
void solve(vector<int> nums, int index, vector<vector<int>>& ans)
{
    if(index >= nums.size())
    {
        ans.push_back(nums);
        return;
    }

    for(int i=index; i<nums.size(); i++)
    {
        swap(nums[i], nums[index]);
        solve(nums, index+1, ans);
        swap(nums[i], nums[index]);
    }
}

vector<vector<int>> permute(vector<int>&nums)
{
    vector<vector<int>> ans;
    int index = 0;
    solve(nums, index, ans);
    return ans;
}
```

Base case



Teacher's Signature

Cut the road into X, Y, Z segment lengths, find max no. of segment

Using recursion

```
int solve(int n, int x, int y, int z)
{
 if(n==0) return 0;
 if(n<0) return INT_MIN;

 int a = solve(n-x, x,y,z)+1;
 int b = solve(n-y, x,y,z)+1;
 int c = solve(n-z, x,y,z)+1;
 return max(a, max(b,c));
 }
int cutseg(int n, int x, int y, int z){
 int ans = solve(n,x,y,z);
 if(ans<0) return 0;
          return ans; }
```

Using recursion + MEMOIZATION

```
int solveMem(int n, int x, int y, int z, vector<int> &dp)
 {
  if(n==0) return 0;
  if(n<0) return INT_MIN;
  if(dp[n]!=-1) return dp[n];
  int a
  int a = solveMem(n-x, x,y,z,dp) +1;
  int b = solveMem(n-y, x,y,z,dp)+1;
  int c = solveMem(n-z,x,y,z,dp)+1;
dp[n] = return max(a, max(b,c));
  return dp[n]; }

int cutseg(int n, int x, int y, int z){
  vector<int> dp(n+1,-1);
  int ans = solveMem(n,x,y,z,dp);
  if(ans<0) return 0;
  }          return dp[n];
```

Teacher's Signature