[Week 7 Assignment]

Date.....

Ques. Last occurrence of a char.

Method1, // using rfind() method of STL

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    string str = "abcidideifg";
    char x = 'i';
    int lastIndex = str.rfind(x);
    if(lastIndex == -1){
        cout << "character not
                        present";
    }
    else{
        cout << "last occurrence of "<<
        x << "in" << str << " is at "
        << lastIndex << "index";
    }
    return 0;
}
```

Date.....

Method 2, // using user defined functi
-on iteratively

```
#include<bits/stdc++.h>
using namespace std;
int findLast(string str, char x)
{
    for(int i = str.size()-1; i >= 0;
                                i--){
        if(str[i] == x){
            return i;
        }
    }
    return -1;
}
int main()
{
    string str = "abcidideifg";
    char x = 'i';
    int lastIndex = findLast(str, x);
    if(lastIndex == -1){
        cout << "not found";
    }
    else{
        cout <<"Found at " << lastInde
            -x " << "index";
    }
    return 0;
}
```

Date.....

Method 3, // using recursion left to right search.

```
#include<bits/stdc++.h>
using namespace std;
void findLast(string str, char x,
                int i, int &ans)
{
    //base case
    if(i >= str.length()){
        return;
    }
    // I case solution
    if(str[i] == x){
        ans = i;
    }
    // recursive call
    findLast(str, x, i+1, ans);
}
int main()
{
    string str = "abcdighimpric";
    char x = 'i';
    int i = 0;
    int ans = -1;
    findLast(str, x, i, ans);
    cout << "Found at" << ans;
    return 0;
}
```

Date.....

Method 4, // using recursion right to left search.

```
#include <bits/stdc++.h>
using namespace std;
void findLast(string str, char x,
              int i, int &ans)
{
    // base case
    if(i < 0){
        return;
    }
    // 1 case solution
    if(str[i] == x){
        ans = i;
        return;
    }
    // recursive call
    findLast(str, x, i-1, ans);
}
int main()
{
    string str = "abcdgihimpric";
    char x = 'h';
    int i = str.size()-1;
    int ans = -1;
    findLast(str, x, i, ans);
    cout << "Found at" << ans;
    return 0;
}
```

Date.....

Ques. Reverse a string recursively.

Method 1, // iteration → while loop.

```
#include<bits/stdc++.h>
using namespace std;
void reverseIter(string &str){
    int s = 0;
    int e = str.length()-1;
    while(s <= e){
        swap(str[s], str[e]);
        s++;
        e--;
    }
}
int main(){
    string str = "abcdef";
    cout << "Before: " << str;
    reverseIter(str);
    cout << "After: " << str;
    return 0;
}
```

Date.....

Method 2, // recursion solution.

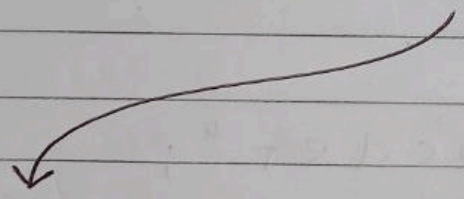
```
#include<bits/stdc++.h>
using namespace std;
void reverseRecur(string &str,
                  int s, int e){
    // base case
    if(s > e){
        return;
    }
    // 1 case solve
    swap(str[s], str[e]);
    // recursive call
    reverseRecur(str, s+1, e-1);
}
int main(){
    string str = "abcdef";
    int s = 0;
    int e = str.length() - 1;
    cout << "Before: " << str;
    reverseRecur(str, s, e);
    cout << "After: " << str;
    return 0;
}
```

Date.....

Ques. Add Strings.

Given two non-negative integers, represented as strings, return the sum of both the numbers as a string.
Solve the problem without any built in library for handling large integers.
Can't transfer / convert the input to integers directly.
Example:-

num1 = "11"
num2 = "123"
output = "134"



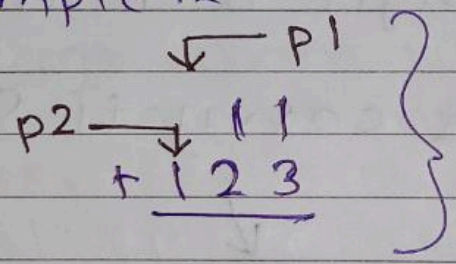
```
  11
+ 123
-----
 134
```

} we start addition from the right generally, first we add 1 & 3 & the result is stored in ans. If there is any carry then we move the carry to the left side.

same appraach we follow here also.

Date.....

Approach :- we declare 2 pointers p1 & p2, p1 will point to last character of num1, p2 will point to last chara -cter of num2. Then we declare one carray variable which is initialized with 0 initially, because we don't have any carry in starting.
Now, first step is to extract the last digits from both of the strings, i.e., extract 1 form 11, i.e, num1, and extract 3 from 123, i.e, num2, & convert them to integer. Along with that we have to check that if p1 or p2 has any valid numbers or not. if no valid number then we add '0'.
For example :-



p1 is empty, but p2 has still 1 number left. So, we add p2 with 0 in place of p1

for this we use ternary operator,

```
int n1 = (p1 >= 0 ? num1[p1] : '0')
                                - '0';
int n2 = (p2 >= 0 ? num2[p2] : '0')
                                - '0';
```

→ agr p1 >= 0 hai, means string bchi hui hai to n1 me num1[p1] daal do else agr string nhi bchi to 0 daal do & - '0' means string ko integer me convert krdo.

Spiral

Teacher's Sign.....

Date.....

Now, we have both the characters, i.e., 1 & 3.
Now we add 1 & 3 & also add the carry, i.e., 0

```
int csum = n1 + n2 + carry;
```

Now, we extract the digit from the sum & the carry from the sum. For example,

csum = 1 + 3 + 0
csum = 4

Digit is 4 after extracting & carry is 0.

```
int digit = csum % 10;
```

```
carray = csum / 10;
```

After that will push the digit in the output string,

```
ans.push_back(digit + '0');
```

Then we call recursively for the next two characters, i.e.,

```
addRE(num1, p1-1, num2, p2-1, carry);
```

code:-

```
#include<bits/stdc++.h>
using namespace std;
string addRE(string num1, int p1,
  string num2, int p2, int carry){
  //base case
  if(p1<0 and p2<0){
     if(carry != 0){
        // agr carry bcha hua hai to
        1 size ki string bnayi &
        bche hue carry ko string
        me convert kiya, and return
        krwa diya
        return string(1, carry + '0');
     }
     // agr carry nhi bcha to empty
     string return krdi
     return "";
  }
  // 1 case solution
  int n1 = (p1>=0 ? num1[p1] : '0')
                    - '0';
  int n2 = (p2>=0 ? num2[p2] : '0')
                    - '0';
   int csum = n1 + n2 + carry;
   int digit = csum % 10;
   carry = csum / 10;
   string ans = "";
   ans.push_back(digit + '0');
   ans += addRE(num1, p1-1, num2,
                    p2-1, carry);
} return ans;
```

Date.....

```
string addStrings(string num1,
                  string num2){
    int p1 = num1.size()-1;
    int p2 = num2.size()-1;
    int carry = 0;
    string ans = addRE(num1, p1,
                       num2, p2, carry);
    reverse(ans.begin(), ans.end());
    return ans;
}
int main(){
    string num1 = "7894";
    string num2 = "22";
    string ans = addStrings(num1,
                            num2);
    cout << ans;
    return 0;
}
```

we can optimize the above code by sending the ans string by reference. So, the recursive function will not send or create the copy again & again, it will save memory space as well as the time.

Date.....

Ques: check palindrome using recursion.

Method 1, // using iteration.

```
#include<bits/stdc++.h>
using namespace std;
bool checkPalindrome(string str){
    int i=0;
    int j=str.length()-1;
    while(i<=j){
        if(str[i]!=str[j]){
            return false;
        }
        i++;
        j--;
    }
    return true;
}
int main(){
    string str="naman";
    if(checkPalindrome(str)){
        cout<<"Palindrome";
    }
    else{
        cout<<"Not Palindrome";
    }
    return 0;
}
```

Date.....

Method 2, // using recursion.

```
#include<bits/stdc++.h>
using namespace std;
bool checkPalindrome(string str, int i,
                     int j){
    //base case
    if(i > j){
        return true;
    }
    // 1 case solve
    if(str[i] != str[j]){
        return false;
    }
    return checkPalindrome(str, i+1,
                           j-1);
}
int main(){
    string str = "naman";
    int i = 0;
    int j = str.length() - 1;
    if(checkPalindrome(str, i, j)){
        cout << "Palindrome";
    }
    else{
        cout << "Not Palindrome";
    }
    return 0;
}
```

Date.....

Que: Remove all occurrences of a substring from a string.

Method 1, // using iteration.

```
#include<bits/stdc++.h>
using namespace std;
string removeOccurrence(string str,
                        string part)
{
    int pos = str.find(part);
    while(pos != string::npos){
        str.erase(pos, part.length());
        pos = str.find(part);
    }
    return str;
}
int main()
{
    string str = "daabcbaabcbc";
    string part = "bc";
    string ans = removeOccurrence
                        (str, part);
    cout << "final string after removv
     -ing" << part << "from" << str <<
     "is:" << ans;
    return 0;
}
```

Method 2, // recursive solution.

```
#include<bits/stdc++.h>
using namespace std;
void removeOccurrence(string &str,
                      string &part){
    int pos = str.find(part);
    if(pos != string::npos){
        //means part string has been
        located
        //now we remove this part
        string from str
        string leftPart = str.substr(0,
                                     pos);
        string rightPart = str.substr
                (pos + part.length(),
                    str.length());
        str = leftPart + rightPart;
        removeOccurrence(str, part);
    }
    else{
        //base case
        return;
    }
}
int main(){
    string str = "daabcbaabcbc";
    string part = "bc";
    removeOccurrence(str, part);
    cout << "after: " << str << endl;
    return 0;
}
```

Date.....

Ques: Print all subarrays of an array

Method 1, // using iteration - $O(n^3)$

```
#include <bits/stdc++.h>
using namespace std;
void printSubArray(int arr[], int n)
{
    // pick starting point
    for(int i = 0; i < n; i++){
        // pick ending point
        for(int j = i; j < n; j++){
            // print subarray between
            starting & ending points
            for(int k = i; k <= j; k++){
                cout << arr[k] << " ";
            }
            cout << endl;
        }
    }
}
int main()
{
    int arr[] = {45, 78, 96, 14, 69, 22};
    int n = 6;
    printSubArray(arr, n);
    return 0;
}
```

Method 2, // using recursion.

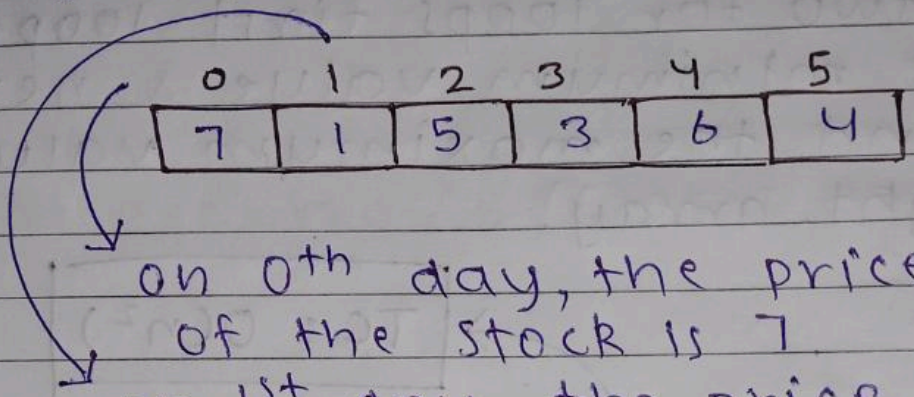
```
#include<bits/stdc++.h>
using namespace std;
void printSubArray(int arr[], int n,
                   int start, int end)
{
    // base case
    if(end == n){
        return;
    }
    // 1 case solution
    for(int i=start; i<=end; i++){
        cout << arr[i] << " ";
    }
    cout << endl;
    // recursive call
    printSubArray(arr, n, start, end+1);
}
void printSubArrayUtil(int arr[], int n)
{
    for(int start=0; start<n; start++)
    {
        int end = start;
        printSubArray(arr, n, start, end);
    }
}
int main()
{
    int arr[]={45, 78, 69, 14, 44, 50};
    int n=6;
    printSubArrayUtil(arr, n);
    return 0;
}
```

Teacher's Sign .....

Date.....

Ques Buy & sell stocks.

In this problem, we've given an array 'prices' where 'prices[i]' is the price of a given stock on i^{th} day.



on 0^{th} day, the price of the stock is 7
on 1^{st} day, the price of the stock is 1, and so on.

we want to maximize the profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock. we have to return the maximum profit we can achieve from the transaction. If we can not achieve any profit, then we return 0.

Approach:- //iterative approach.

we declare two variables mini and maxi, which will store the minimum & maximum values, and at last we return maxi - mini.
By running two for loops first loop will find the minimum value & next loop will find the maximum value from the right array.

code:-

TC : $O(n^2)$

```
#include<bits/stdc++.h>
using namespace std;
int maxProfit(vector<int> &prices){
    int mini = INT_MAX;
    int maxi = INT_MIN;
    int ans;
    for(int i=0; i<prices.size(); i++)
    {
        mini = min(mini, prices[i]);
        for(int j=i; j<prices.size(); j++)
        {
            if(prices[j] > prices[i])
            {
                maxi = max(maxi, prices[j]);
            }
        }
    }
    ans = maxi - mini;
    return ans;
}
```


Teacher's Sign .....

Date.....

```
int main(){
    vector<int> prices{7,6,5,3,8,40};
    int ans = maxProfit(prices);
    cout << ans;
    return 0;
}
```

TC: O(n)

```
int maxProfit(vector<int> &num)
{
    int mini = INT_MAX;
    int maxi = INT_MINI;
    for(int i=0; i<num.size(); i++)
    {
        if(num[i] < mini){
            mini = num[i];
        }
        if((num[i] - mini) > maxi)
        {
            maxi = num[i] - mini;
        }
    }
    return maxi;
}
```

Teacher's Sign

Date.....

Method 2, // using recursion.

```
#include<bits/stdc++.h>
using namespace std;
void maxProfitFinder(vector<int>
        &prices, int i, int &mini,
                        int &maxi)
{
    // base case
    if(i == prices.size()){
        return;
    }
    // 1 case solution
    if(prices[i] < mini){
        mini = prices[i];
    }
    if((prices[i] - mini) > maxi){
        maxi = prices[i] - mini;
    }
    // recursion call
    maxProfitFinder(prices, i+1, mini,
                        maxi);
}
int maxProfit(vector<int> &prices)
{
    int mini = INT_MAX;
    int maxi = INT_MIN;
    int i = 0;
    maxProfitFinder(prices, i, mini, maxi);
    return maxi;
}
```

Teacher's Sign .....

Date.....

```
int main()
{
    vector<int> prices{7, 6, 5, 1, 3, 8, 40};
    int ans = maxProfit(prices);
    cout << ans;
    return 0;
}
```

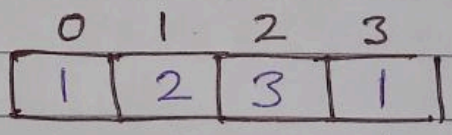
Date.....

Ques: House Robber Problem.

You are a professional robber, planning to rob house along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent house have security systems connected & it will automatically contact the police if two adjacent houses were broken into on the same night.

Given an integer "arr" representing the amount of money of each house, return the maximum amount of money you can rob without alerting the police.

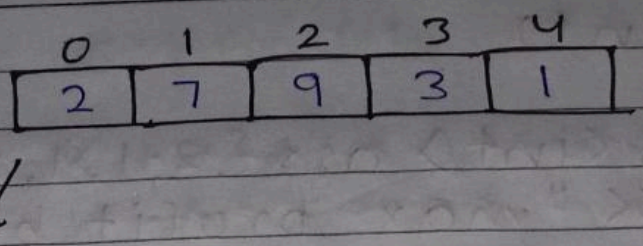
Example :-



1+3 = 4 2+1 = 3

max. profit is 4 while rob 0th & 2nd house.

Example:-



output: 12

Explanation:- Rob house 0th, (amount = 2), Rob house 2nd, (amount = 9), Rob house 4th, (amount = 1).

Total amount = 2 + 9 + 1 = 12

Approach:- we take 1st house amount & call recursively for i+2 house, & we take no first house amount & call for i+1 house. Then return maximum of both the amount of houses.

code:-

```
#include <bits/stdc++.h>
using namespace std;
int rob(vector<int> &arr, int i){
    // base case
    if(i >= arr.size()){
        return 0;
    }
    // 1 case solution.
    int robAmt1 = arr[i] + rob(arr, i+2);
    int robAmt2 = 0 + rob(arr, i+1);
    return max(robAmt1, robAmt2);
}
```

Date.....

```
int main()
{
    vector<int> arr{2,1,1,2};
    cout << "max profit by rob
        adjacent houses are:" <<
                    rob(arr, 0);
    return 0;
}
```

Date.....

Ques Integer to English words.

convert a non-negative integer num to its English words representation.

Example :-
Input :- 123
Output :- "One Hundred Twenty Three".

Example :-
Input :- 12345
Output :- Twelve Thousand Three Hundred Forty Five.

Example :-
Input :- 1234567
Output :- One Million Two Hund-red Thirty Four Thousand Five Hundred Sixty Seven.

Approach :- First we create a vector of Pairs which will track the record of numbers to english words, like :- {50, "Fifty"}, {20, "Twenty"}, etc. Then we run a loop on that vector & check / find the number from which our given number is just >=.

Date.....

code:-

```
#include<bits/stdc++.h>
using namespace std;
vector<pair<int, string>> mp = {
        {1000000000, "Billion"},
        {1000000, "Million"}, {1000,
         "Thousand"}, {100, "Hundred"},
        {90, "Ninety"}, {80, "Eighty"},
         {70, "Seventy"}, {60, "Sixty"},
        {50, "Fifty"}, {40, "Forty"},
         {30, "Thirty"} {20, "Twenty"},
        {19, "Nineteen"}, {18, "Eighteen"},
        {17, "seventeen"}, {16, "Sixteen"},
        {15, "Fifteen"}, {14, "FOurteen"},
        {13, "Thirteen"}, {12, "Twelve"},
        {11, "Eleven"}, {10, "Ten"} {9, "Ni
         -ne"}, {8, "Eight"}, {7, "Seven"},
        {6, "Six"} {5, "Five"} {4, "four"},
         {3, "Three"}, {2, "Two"}, {1, 'One'}
    };
string numberTowords(int num){
    //base case
    if(num == 0){
        return "Zero";
    }
    //1 case solution
    for(auto it: mp){
        //find that no. from which the
        given number is just >=, like
        123 is just >= 100
        if(num >= it.first){
```

```
        string a = " ";
        // because >= 100 Ki values me hum
        aisa bol skte hai that one Hundr
         -ed, Two Thousand, etc.
        if (num >= 100){
            // now we calculate how many
            hundreds are there, means
             1 hundred or 2 hundred
            a = numberToWords(num /
                              it.first) + " ";
        }
        string b = it.second;
        string c = "";
        if(num % it.first != 0){
            c = " " + numberToWords(num
                              % it.first);
        }
        return a + b + c;
    }
}
 return "";
}
int main(){
    int num;
    cout << "enter num: ";
    cin >> num;
    string ans = numberToWords(num);
    cout << num << " --> " << ans;
    return 0;
}
```

Date.....

Ques wild card Matching

Given an input string 's' and pattern 'p', implement wildcard pattern matching with support for '?' and '*' where:-

i) '?' matches any single character
ii) '*' matches any sequence of characters (including the empty sequences).

The matching should cover the entire input string (not partial).

Example 1:-

string s = | c | b |

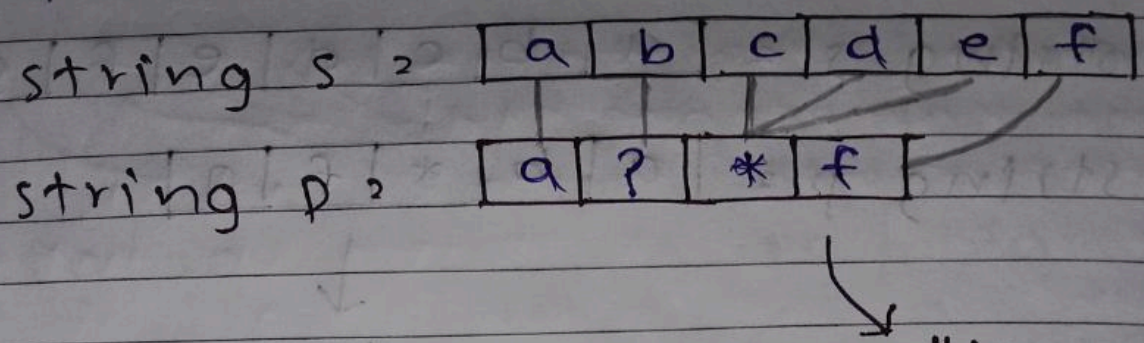
string p = | * |

return "true".

because '*' can consume any sequence of characters. so '*' will consume "cb" & return true.

Date.....

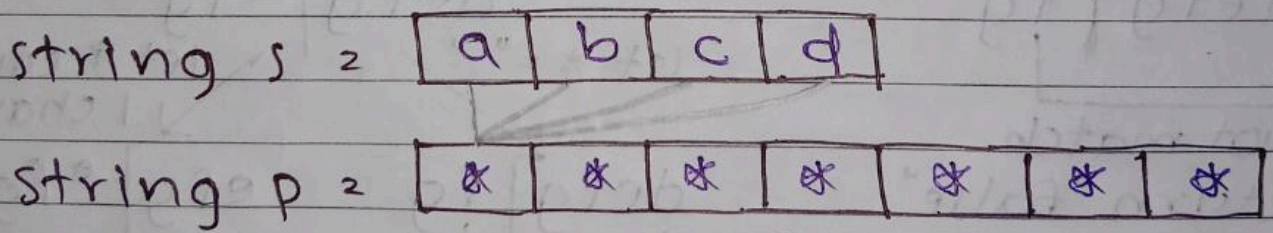
Example 2:-



"true"

because 'a' is matched with 'a', then '?' is matched with 'b', now '*' will consume 'c', 'd', 'e', then 'f' is matched with 'f'. So, we return true.

Example 3:-

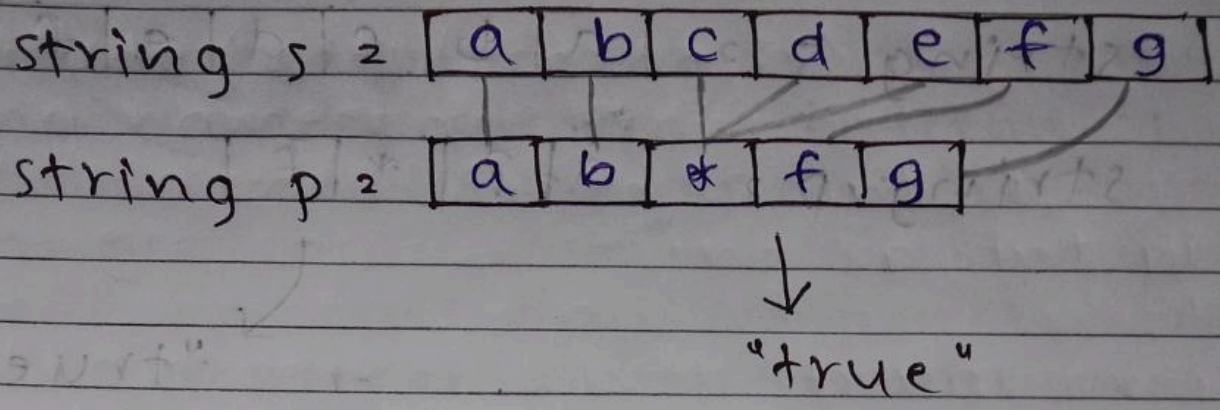


"true"

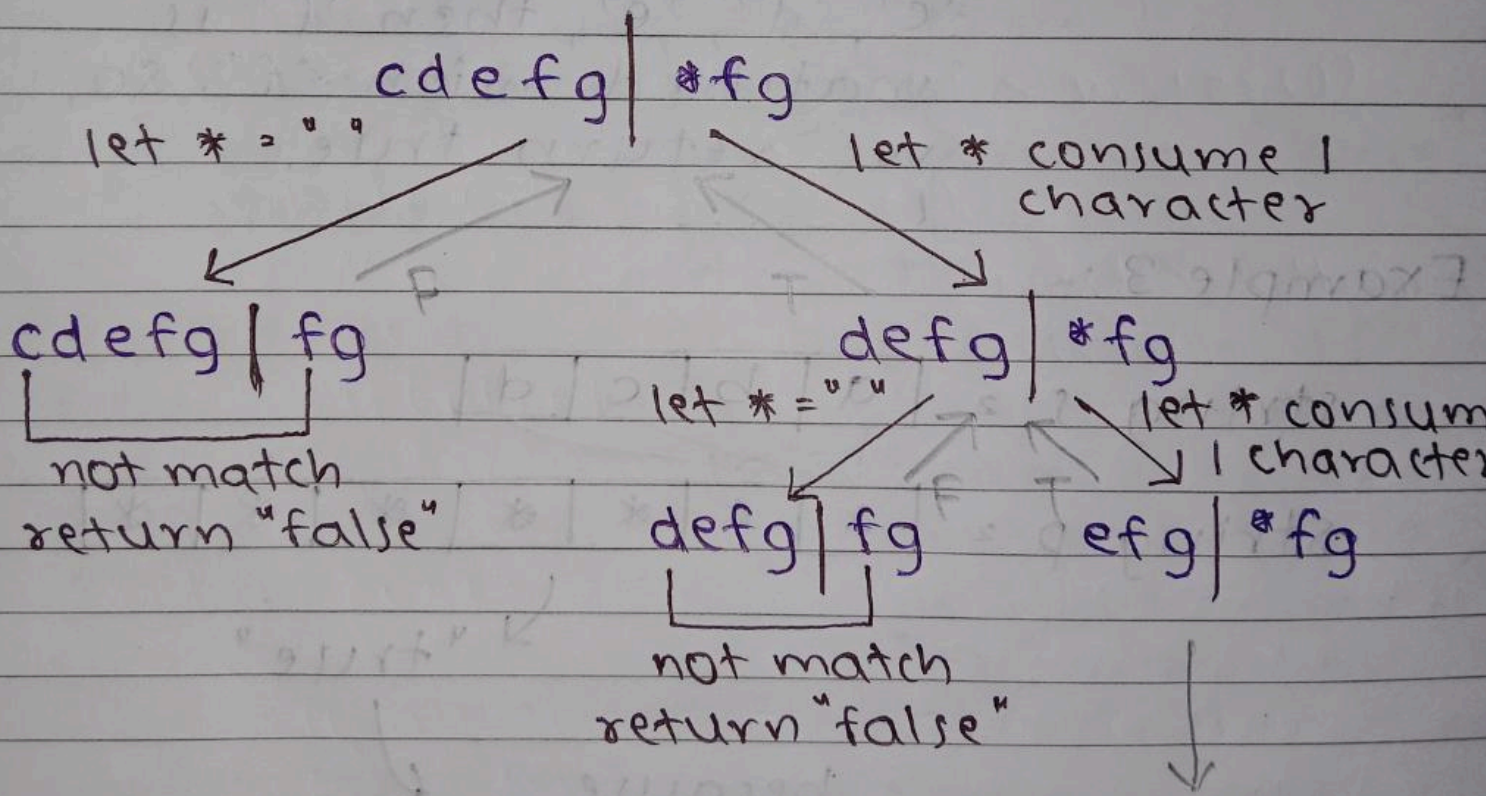
because first '*' will consume all characters "abcd", rest of the '*' will treated as null strings. so, return true.

Date.....

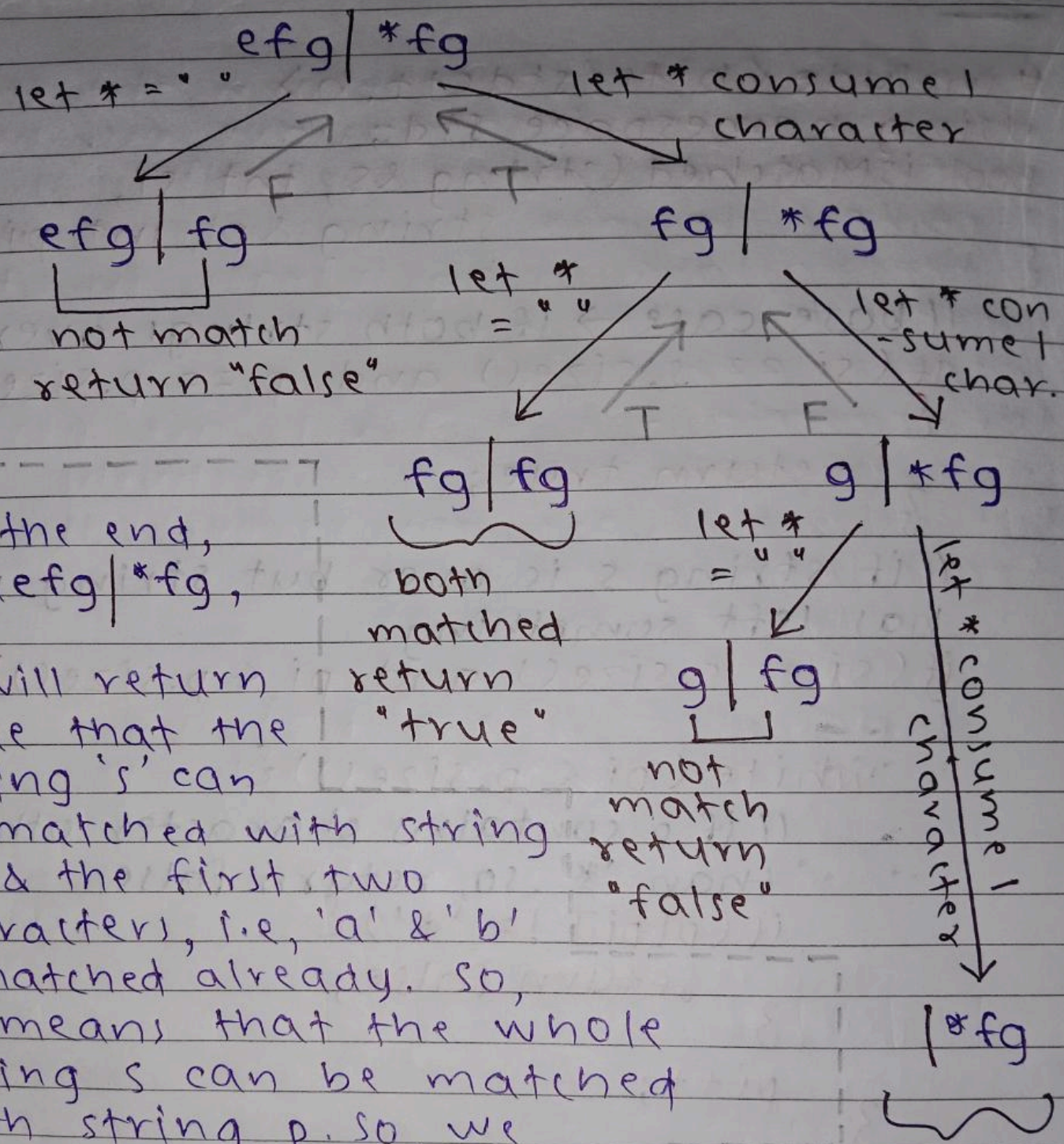
Example 4,



Explanation :- 'a' is matched with 'a', 'b' is matched with 'b'.
How '*' know that how many chara -cted it should consume ?



continued on next page.



At the end, cdefg|*fg,

it will return true that the string 's' can be matched with string 'p', & the first two characters, i.e, 'a' & 'b' is matched already. So, it means that the whole string s can be matched with string p. So, we return "true".

String s is empty & string p has more character other than '*'. So return "false"

Teacher's Sign .....

code :-

```
#include<bits/stdc++.h>
using namespace std;
bool isMatched(string &s, int si,
               string &p, int pi)
{
    // base case → if both strings over
    if(si == s.size() and p == p.size())
    {
        return true;
    }
    // if string s is over but string p
    has left something.
    if(si == s.size() and pi < p.size())
    {
        while(pi < p.size()){
            // if p contains character other
            than '*'. so, return false.
            if(p[pi] != '*'){
                return false;
            }
            pi++;
        }
        // if all characters are '*' then
        return true.
        return true;
    }
    // 1 case solution
    // if si character of s & pi character
    of p are matched or pi of p
    contains '?' so we check further
```

```
if(s[si] == p[pi] or p[pi] == '?'){
    return isMatched(s, si+1, p,
                     pi+1);
}
// if p[pi] is '*' so we have to
check how many characters of
string s can '*' consume
if(p[pi] == '*'){
    // treat '*' as empty
    bool caseLeft = isMatched(s, si,
                              p, pi+1);
    // let '*' consume 1 character
    bool caseRight = isMatched(s,
                     si+1, p, pi);

    return caseLeft || caseRight;
}
// character doesn't matched
return false;
}
int main()
{
    string s = "abcdef";
    string p = "a?*f";
    int si = 0; // pointer for s
    int pi = 0; // pointer for p
    if(isMatched(s, si, p, pi)){
        cout << "matched";
    }
    else{
        cout << "not matched";
    }
return 0;
}
```

Date.....

Ques Perfect Squares.

Given an integer n, return the least number of perfect square numbers that sum to n.
A perfect square is an integer that is the square of an integer, in other words, it is the product of some integer with itself.
For example :- 1, 4, 9 & 16 are perfect squares of 1, 2, 3 & 4 respectively.
But 3 and 11 are not perfect squares.
Example :-

Input, n = 12
Output, 3
Explanation, $2^2 + 2^2 + 2^2$, i.e.,
$4 + 4 + 4 = 12$.

3 times.

Input, n = 13
output, 2
Explanation, $2^2 + 3^2$, i.e.,
$4 + 9 = 13$.

2 times.

Teacher's Sign.....

Date.....

Approach:- we start checking to get the n with the square of 1, as we know $1^2 = 1$, if $n = 12$, so $1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2$, i.e., $1+1+1+1+1+1+1+1+1+1+1+1 = 12$

Here we need 12 times 1, to make 12.

Then we start checking the n with the square of 2, as we know $2^2 = 4$, if $n = 12$, so $2^2 + 2^2 + 2^2$, i.e., $4+4+4 = 12$.

Here we need 3 times 4, to make 12

Now, we start checking the n with the square of 3, as we know $3^2 = 9$, if $n = 12$, if we add 3^2 times, i.e., $9+9 = 18 > 12$, means we cant add 3^2 times, then it means add 2^2, 3 times

Date.....

code:-

```
#include<bits/stdc++.h>
using namespace std;
int numSquareHelper(int n){
    //base case
    if(n==0){
        return 1;
    }
    if(n<0){
        return 0;
    }
    //1 case solution & recursive call
    int ans = INT_MAX;
    int i=1;
    int end = sqrt(n);
    while(i<=end){
        int perfectSquare = i*i;
        int noPerfectSquares = 1 +
        numSquareHelper(n-perfectSqua
                                -re);
        if(noPerfectSquares<ans)
        {
            ans = noPerfectSquares;
        }
        i++;
    }
    return ans;
}
```

Date.....

```
int numSquares(int n){
    int finalAns = numSquareHelper
                        (n) - 1;
    return finalAns;
}
int main(){
    int n = 12;
    cout << numSquares(n);
    return 0;
}
```

Date.....

Ques: Minimum cost for tickets.

We have planned some train travelling in advance. The days of the year in which you will travel are given as an integer array "days". Each day is an integer from 1 to 365

The train tickets are sold in 3 different ways :-

i) 1-day pass is sold for costs[0] dollars.

ii) 7-day pass is sold for costs[1] dollars.

iii) 30-day pass is sold for costs[2] dollars.

The passes allow that many days of consecutive travel.

For example, if we get a 7-day pass on day 2, then we can travel for 7 days: 2, 3, 4, 5, 6, 7 & 8.

we have to return the minimum no. of dollars we need to travel everyday in the days given.

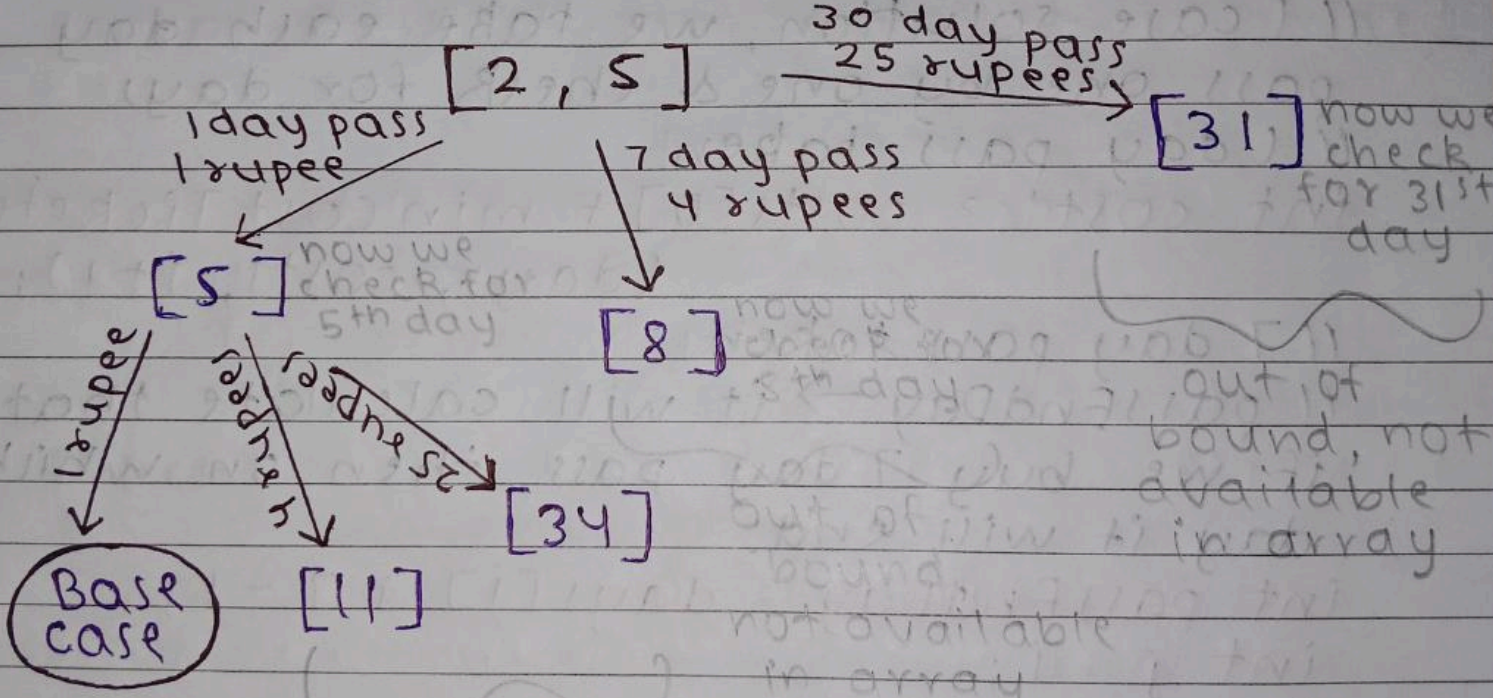
Teacher's Sign.....

Date.....

Approach:- we will take a pointer 'i' which will iterate on days & we try taking each pass on that day, i.e., 1 day pass, 7 day pass & 30 day pass. when we take 7 day or 30 day pass we have to also calculate that when are the pass valid day.

For example,

days = [2, 5]
costs = [1, 4, 25]



this means we take 2 passes of 1-1 day

Spiral

Teacher's Sign.....

Date.....

code :-

```
#include<bits/stdc++.h>
using namespace std;
int mincostTickets(vector<int> &days,
                   vector<int> &costs, int i)
{
    // base case
    if(i >= days.size()){
        return 0;
    }
    // 1 case solution, we take each day
    pass one by one & check for days
    // 1 day pass taken
    int cost1 = costs[0] + mincostTickets
                          (days, costs, i+1);
    // 7 day pass taken
    // passEndDay → it will calculate that
    if we buy 7 day pass, then on which
    day it will end.
    int passEndDay = days[i] + 7 - 1;
    int j = i;
    // through this loop we iterate over
    the days array & go to that day on
    which our 7 day pass ended.
    while(j < days.size() and days[j]
                          <= passEndDay)
    {
        j++;
    }
    // check from next day of pass end.
    int cost7 = costs[1] + mincostTickets
                          (days, costs, j);
```

```
// 30 day pass taken
passEndDay = days[i] + 30 - 1;
j = i;
while (j < days.size() and days[j]
                    <= passEndDay)
{
    j++;
}
int cost30 = costs[2] + minCostTick
        -ets(days, costs, j);

return min(cost1, min(cost7,
                    cost30));
}
int main()
{
    vector<int> days{1,2,3,4,5,6,7,8,9,
                    10,30,31};
    vector<int> costs{2,7,15};
    // i represent 0th day
    int i = 0;
    cout << minCostTickets(days, costs,
                    i);
    return 0;
}
```

Date.....

Ques Number of dice roll with target sum.

You have 'n' dices, & each die has 'K' faces numbered from 1 to K.

Given three integers n, K and target, return the number of possible ways (out of the K^n total ways) to roll the dice, so the sum of the face-up numbers equal target. Since the answer may be too large, return it modulo $10^9 + 7$.

Example 1 :- Input : n = 1, K = 6, target = 3

output = 1

Explanation = You throw one die with 6 faces. There is only way to get a sum of 3.

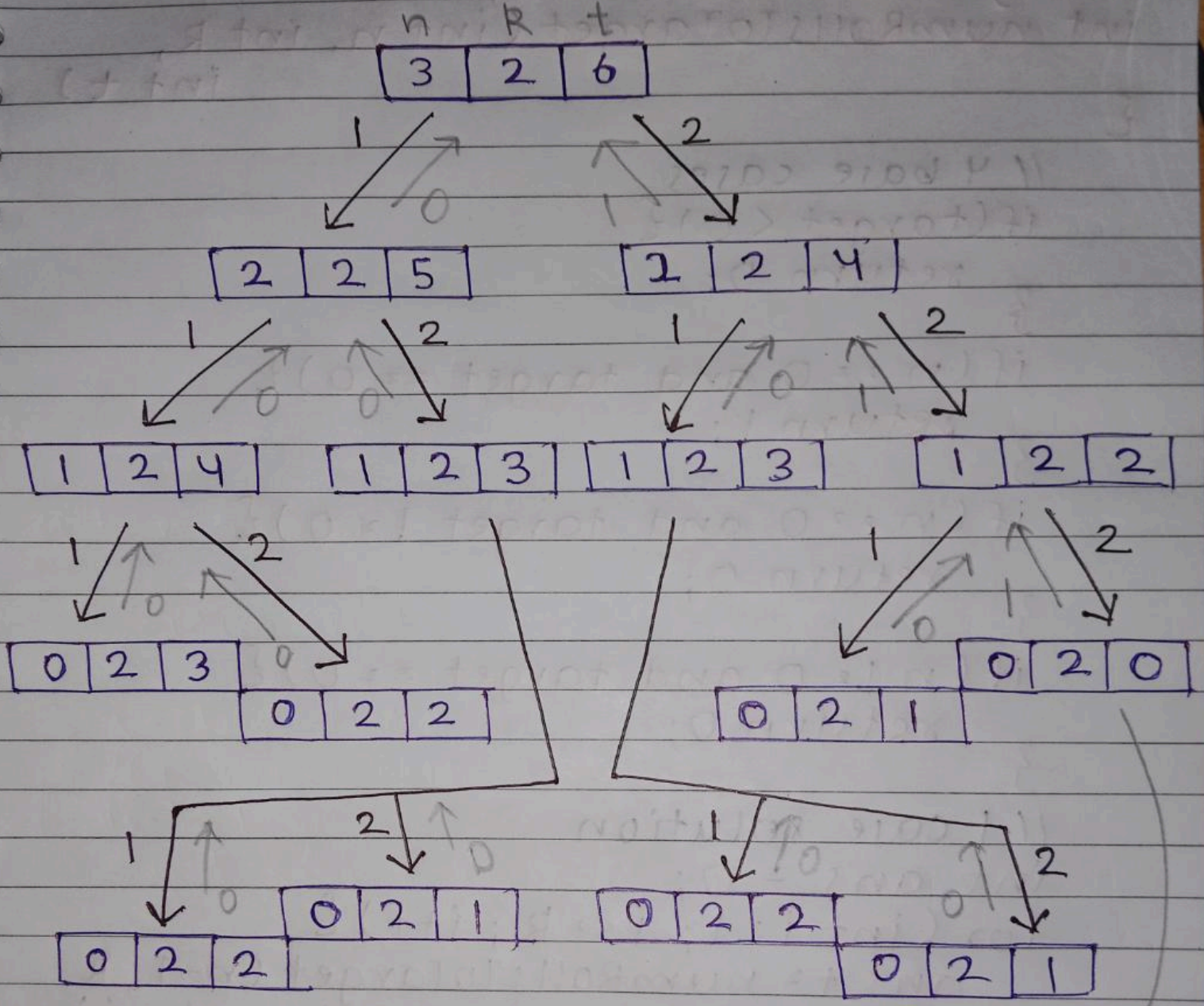
Example 2 :- Input : n = 2, K = 6, target = 7

output : 6

Explanation = we throw two dice, each with 6 faces. There are 6 ways to get a sum of 7 : 1+6, 2+5, 3+4, 4+3, 5+2, 6+1.

Teacher's Sign.....

For example, suppose we have 3 dices, faces are 2 and target are 6, i.e., n = 3, R = 2, t = 6.



For rest of the cases, n = 0, means no dices are left but target is still greater than 0. Means, value not found. So, we return 0.

when n = 0, and t = 0, means we've thrown all dices & target is also 0. So we return 1.

Spiral

Teacher's Sign .....

code:-

```
#include<bits/stdc++.h>
using namespace std;
int numRollsToTarget(int n, int R,
                     int target)
{
    // 4 base cases
    if(target < 0){
        return 0;
    }
    if(n == 0 and target == 0){
        return 1;
    }
    if(n == 0 and target != 0){
        return 0;
    }
    if(n != 0 and target == 0){
        return 0;
    }
    // 1 case solution
    int ans = 0;
    for(int i=1; i<=R; i++){
        ans += numRollsToTarget(n-1, R,
                                target-i);
    }
    return ans;
}
int main()
{
    int n = 3;
    int R = 6;
```

this loop is because each time we have to check for each face of dice from 1 to R.

```
int target = 15;
cout << " total ways to find the
  target " << target << " with " << n
  << " dices of " << K << " faces are: "
  << numRollsToTarget(n, K, target);
return 0;
}
```