L14 : DP and BitMasking

1-Tut : Candy

Send Feedback

Gary is a teacher at XYZ school. To reward his N students he bought a packet of N candies all with different flavours. But the problem is some students like certain flavour while some doesn't. Now Gary wants to know the number of ways he can distribute these N candies to his N students such that every student gets exactly one candy he likes.

Input Format :

The first line of input will contain T(number of test cases), each test case follows as.
Line 1 : An integer N (1<= N <= 16) denoting number of students and candies.
Next N lines: N integers describing the preferences of one student. 1 at i'th (0 <= i < N) position denotes that this student likes i'th candy, 0 means he doesn't.
Assume input to be 0-indexed based.

Output Format :

Return the number of ways Gary can distribute these N candies to his N students such that every student gets exactly one candy he likes for each test case in a new line.

Sample Input:

```
1
3
1 1 1
1 1 1
1 1 1
```

Sample Output: 6

Explanation:

Since, all the students like all the candies, so, the candies can be distributed in the following 6 ways:

	First Student	Second Student	Third Student
Way 1	1st Candy	2nd Candy	3rd Candy
Way 2	1st Candy	3rd Candy	2nd Candy
Way 3	2nd Candy	1st Candy	3rd Candy
Way 4	2nd Candy	3rd Candy	1st Candy
Way 5	3rd Candy	1st Candy	2nd Candy
Way 6	3rd Candy	2nd Candy	1st Candy

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
int candies(int **like, int n, int person, int mask, int *dp)
{
    if (person >= n)
    {
        return 1;
    }

    if(dp[mask]!=-1)
    {
        return dp[mask];
    }
    int ans = 0;
    for (int i = 0; i < n; i++)
    {
        if (!(mask & (1 << i)) && like[person][i])
        {
            ans += candies(like, n, person + 1, mask | (1 << i), dp);
        }
    }
    dp[mask]=ans;
    return ans;
}


int solve(int **like,int n)
{
            int *dp = new int[1 << n];
    for (int i = 0; i < (1 << n); i++)
    {
        dp[i] = -1;
    }
    int ans= candies(like, n, 0, 0, dp);
    delete[]dp;
    return ans;
}

int32_t main(){

    // write your code here
    int t;
    cin>>t;
```

```
    while(t--){
        int n;
        cin>>n;
        int** like = new int*[n];

            for (int i = 0; i < n; i++) {


            like[i] = new int[n];
            }
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                cin>>like[i][j];
            }
        }

        cout<<solve(like,n)<<endl;
    }

    return 0;
}
```

2-Tut : Ghost Type

Send Feedback

Gengar has got an integer N. Now using his ghostly powers, he can create the permutation from 1 to N of this given number.

Since, he's a special kind of Poke'mon, so he thinks he deserves special permutations. He wants to find the total number of special permutations of length N, consisting of the integers from 1 to N.

A permutation is called special if it satisfies following condition:

If Ap & Aq == Ap, then p < q, where p and q are two distinct indices of permutation and A is the permutation itself. "&" denotes the bitwise and operation.
Help Gengar in finding the number of such permutations.

Input format:

The only line of input will consist of a single integer N denoting the length of the permutation.
Output format:

Output the total number of special permutations of length N.
Constraints:

1 ≤ N ≤ 20

SAMPLE INPUT **4**

SAMPLE OUTPUT **8**

Explanation

All the special permutations of length 4 are:

1 2 3 4
1 2 4 3
1 4 2 3
2 1 3 4
2 1 4 3
2 4 1 3
4 1 2 3

4 2 1 3

```
#include<bits/stdc++.h>
using namespace std;
#define rep(i, n) for (int i = 0; i < n; i++)
#define ll long long int
vector<int> submask[22];
bool vis[1 << 22];
ll dp[1 << 22];
ll rec(int mask, int n)
{
    if (mask == (1 << (n + 1)) - 2)
        return 1;
    if (vis[mask])
        return dp[mask];
    vis[mask] = 1;
    ll &ret = dp[mask];
    ret = 0;
    int x;
    for (int i = 1; i <= n; i++)
    {
        if (!(mask & (1 << i)))
        {
            bool ok = 1;
            rep(j, submask[i].size())
            {
                x = submask[i][j];
                if (!(mask & (1 << x)))
                    ok = 0;
            }
            if (ok)
            {
                ret += rec(mask | (1 << i), n);
```

```
            }
        }
    }
    return ret;
}
int main(){

    // write your code here
     int n;
    for (int i = 1; i <= 20; i++)
    {
        for (int j = i - 1; j >= 1; j--)
        {
            if ((i & j) == j)
                submask[i].push_back(j);
        }
    }
    cin >> n;
    cout << rec(0, n);
    return 0;
}
```

3-Tut : Dilemma

Send Feedback

Abhishek, a blind man recently bought N binary strings all of equal length .A binary string only contains '0's and '1's . The strings are numbered from 1 to N and all are distinct, but Abhishek can only differentiate between these strings by touching them. In one touch Abhishek can identify one character at a position of a string from the set. Find the minimum number of touches T Abhishek has to make so that he learn that all strings are different .

Input Format:

First line of input will contain T(number of test cases), each test case follows as.
Line1: contain an integer N (number of strings)
Next N line contain binary strings.

Output Format:

For each test case print the answer in newline.

Constraints:

1 <= T <= 50
1 <= N <= 10
1 <= |S| <= 100

Sample Input :

1
2
111010

100100

Sample Output :

2

Explanation

Abhishek touches 2nd bit from the start of both the binary string.
in the first touch (to string 1) he sees that the value is1.
in the second touch (to string 2)he sees that the value is 0.

so he concludes both strings are different in 2 touches.

```
#include<bits/stdc++.h>
using namespace std;
int dp[105][1 << 12];

int find_touches(int pos, int mask, vector<string> &v)
{
    if (!(mask & (mask - 1)) && mask)
    {
        return 0;
    }

    if (pos == -1 || mask == 0)
    {
        return 1000000;
    }

    if (dp[pos][mask])
    {
        return dp[pos][mask];
    }

    int newmask1 = 0, newmask2 = 0, touches = 0;

    for (int i = 0; i < v.size(); i++)
    {

        if ((mask >> i) & 1)
        {
            touches++;
            if (v[i][pos] == '0')
            {
                newmask1 |= (1 << i);
            }
            else
            {
```

```
36.                 newmask2 |= (1 << i);
37.             }
38.         }
39.     }
40.     return dp[pos][mask] = min(find_touches(pos - 1, newmask1, v) + find_touches(pos -
    1, newmask2, v) + touches, find_touches(pos - 1, mask, v));
41. }
42.
43. int solve(int n, vector<string> v)
44. {
45.     return find_touches(v[0].size() - 1, (1 << n) - 1, v);
46. }
47. int main(){
48.
49.     // write your code here
50.      int t;
51.     cin >> t;
52.     while (t--)
53.     {
54.         int n;
55.         cin >> n;
56.         vector<string> v;
57.         memset(dp, 0, sizeof(dp));
58.
59.         for (int i = 0; i < n; i++)
60.         {
61.             string s;
62.             cin >> s;
63.             v.push_back(s);
64.         }
65.         cout << solve(n, v) << endl;
66.     }
67.     return 0;
68. }
```

4-Ass : String Maker

Send Feedback

According to Ancient Ninjas , string making is an art form . There are various methods of string making , one of them uses previously generated strings to make the new one . Suppose you have two strings A and B , to generate a new string C , you can pick a subsequence of characters from A and a subsequence of characters from B and then merge these two subsequences to form the new string. Though while merging the two subsequences you can not change the relative order of individual subsequences. What this means is - suppose there two characters Ai and Aj in the subsequence chosen

from A , where i < j , then after merging if i acquires position k and j acquires p then k<p should be true and the same apply for subsequence from C.

Given string A , B , C can you count the number of ways to form string C from the two strings A and B by the method described above. Two ways are different if any of the chosen subsequence is different .

As the answer could be large so return it after modulo 10^9+7 .

Input Format :

First line will contain T(number of test cases), each test case consists of three lines.
Line 1 : String A
Line 2 : String B
Line 3 : String C

Output Format :

The number of ways to form string C for each test case in new line.

Constraints :

1 <= T <= 500
1 <= |A| , |B|, |C| <=50

Sample Input :

1
abc
abc
abc

Sample Output :

8

```
#include<bits/stdc++.h>
using namespace std;
#define m 1000000007
typedef long long int ll;
int helper(string a, string b, string c, ll ***dp)
{
    if(c.length()==0)
    {
        return 1;
    }
    if(a.length()<=0&&b.length()<=0)
    {
        return 0;
    }

    if(dp[a.length()][b.length()][c.length()]>-1)
    {
        return dp[a.length()][b.length()][c.length()]%m;
    }
    ll ans=0;
```

```
    for(ll i=0; i<a.length(); i++)
    {
        if(a[i]==c[0])
        {
            ans+=helper(a.substr(i+1), b, c.substr(1), dp)%m;
        }
    }
    for(ll i=0; i<b.length(); i++)
    {
        if(b[i]==c[0])
        {
            ans+=helper(a, b.substr(i+1), c.substr(1), dp)%m;
        }
    }
    dp[a.length()][b.length()][c.length()]=ans%m;
    return ans%m;
}
int solve(string a, string b, string c)
{
    ll ***dp=new ll **[51];
    for(ll i=0; i<51; i++)
    {
        dp[i]=new ll *[51];
        for(ll j=0; j<51; j++)
        {
            dp[i][j]=new ll [51];
            for(ll k=0; k<51; k++)
            {
                dp[i][j][k]=-1;
            }
        }
    }
    ll ans= helper(a, b, c, dp)%m;
    for(ll i=0; i<51; i++)
    {
        for(ll j=0; j<51; j++)
        {
            delete[]dp[i][j];
        }
    }
    return ans;
}

int main(){
```

```
    // write your code here
    int n;
    cin>>n;
    while(n--){
        string a,b,c;
        cin>>a>>b>>c;
        cout<<solve(a,b,c)<<endl;
    }
    return 0;
}
```

5-Ass : Counting Strings

Send Feedback

Given a string 's' consisting of upper case alphabets, i.e. from 'A' to 'Z'. Your task is to find how many strings 't' with length equal to that of 's', also consisting of upper case alphabets are there satisfying the following conditions:

-> String 't' is lexicographical larger than string 's'.
-> When you write both 's' and 't' in the reverse order, 't' is still lexicographical larger than 's'.
Find out number of such strings 't'. As the answer could be very large, take modulo 10^9 + 7.

Input Format:

First line will contain T(number of test cases).
Each test case consists of a single line containing the string s.

Output Format:

For each test case output the number of strings (t) %(10^9 + 7) in new line.

Constraints:

1 <= T <= 50
1 <= |S| <= 10^5

Sample Input:

```
2
A
XKS
```

Sample output:

```
25

523
```

```
#include<bits/stdc++.h>
using namespace std;
int mod = 1e9 + 7;

int countStrings(string &s){

                int n = s.size();
```

```
8.                      long long int req = 0, dp[n + 1], arr[n + 1], ans = 0;
9.
10.                     for (int i = 0; i < n; i++)
11.                     {
12.                             arr[i] = ('Z' - s[i]);
13.                     }
14.
15.                     dp[n - 1] = arr[n - 1];
16.
17.                     for (int i = n - 2; i >= 0; i--)
18.                     {
19.
20.                             req = (arr[i + 1] + (26ll * req) % mod) % mod;
21.
22.                             dp[i] = (arr[i] + (arr[i] * req) % mod) % mod;
23.                     }
24.
25.                     for (int i = 0; i < n; i++)
26.                     {
27.
28.                             ans = (ans + dp[i]) % mod;
29.                     }
30.
31.                     return ans;
32.             }
33.
34. int main(){
35.
36.     // write your code here
37.     int t;
38.                     cin >> t;
39.                     while (t--)
40.                     {
41.                             string s;
42.                             cin >> s;
43.                             cout << countStrings(s) << endl;
44.                     }
45.     return 0;
46. }
```

6-Ass : Number of APs

Send Feedback

Given an array of n positive integers. The task is to count the number of Arithmetic Progression subsequences in the array. As the answer could be very large, output it modulo 10^9 + 7.

Note: Empty sequence or single element sequence is Arithmetic Progression.

Input Format:

First Line: N (the size of the array)
Second Line: Elements of the array separated by spaces.

Output:

Print total number of subsequences

Input Constraints:

1 <= arr[i] <= 1000
1 <= sizeof(arr) <= 1000

Sample Input 1 :

3
1 2 3

Sample output:

8

Sample Output Explanation:

Total subsequence are: {}, { 1 }, { 2 }, { 3 }, { 1, 2 }, { 2, 3 }, { 1, 3 }, { 1, 2, 3 }

Sample Input 2:

7
1 2 3 4 5 9 10

Sample Output:

37

```
#include<bits/stdc++.h>
using namespace std;
int mod = 1e9 + 7;
int numofAP(int arr[], int n){

                int minElem = *min_element(arr, arr + n);
                int maxElem = *max_element(arr, arr + n);

                int totalAPs = n + 1;
                int sum[1001];
        //CODE BY KAMAL CHAUHAN
                for (int d = (minElem - maxElem); d <= (maxElem - minElem); d++)
                {
                        memset(sum, 0, sizeof(sum));

                        for (int i = 0; i < n; i++)
                        {
```

```
                                        int a = 1;
                                        if (arr[i] - d >= 1 && arr[i] - d <= 1000)

                                                a = (a + sum[arr[i] - d]) % mod;

                                        totalAPs = ((totalAPs + a - 1) % mod + mod) % mod;
                                        sum[arr[i]] = (sum[arr[i]] + a) % mod;
                                }
                        }
                        return totalAPs;
}
int main(){

    // write your code here
    int n;
                        cin >> n;
                        int arr[n];
                        for (int i = 0; i < n; i++)
                        {
                                cin >> arr[i];
                        }
                        cout << numofAP(arr, n);
    return 0;
}
```