CODEFORCES β
Sponsored by Telegram

Enter | Register

HOME CONTESTS GYM PROBLEMSET GROUPS RATING API VK CUP

USAXENA95 BLOG TEAMS SUBMISSIONS GROUPS CONTESTS

usaxena95's blog

SOS Dynamic Programming [Tutorial]

By **usaxena95**, 7 weeks ago,

Introduction

In this post, I am going to share my little knowledge on how to solve some problems involving calculation of **Sum over Subsets(SOS)** using dynamic programming. Thus the name **SOS DP**. I have chosen this topic because it appears frequently in contests as medium-hard and above problems but has very few blogs/editorials explaining the interesting DP behind it. I also have a predilection for this since I came across it for the first time in ICPC Amritapuri Regionals 2014. Since then I have created many questions based on this concept on various platforms but the number of accepted solutions always seems to be disproportionate to the lucidity of the concept. Following is a small attempt to bridge this gap ☺

Problem

I will be addressing the following problem: Given a fixed array **A** of 2^N integers, we need to calculate ∀ x function **F(x)** = Sum of all **A[i]** such that **x&i = i**, i.e., **i** is a subset of **x**.

$$F[mask] = \sum_{i \subseteq mask} A[i]$$

Prerequisite

- Basic Dynamic Programming
- Bitmasks

In no way this should be considered an introduction to the above topics.

Solutions

Bruteforce

```
for(int mask = 0;mask < (1<<N); ++mask){
        for(int i = 0;i < (1<<N); ++i){
                if((mask&i) == i){
                        F[mask] += A[i];
                }
        }
}
```

This solution is quite straightforward and inefficient with time complexity of $O(4^N)$

Suboptimal Solution

```
// iterate over all the masks
for (int mask = 0; mask < (1<<n); mask++){
```

→ Pay attention

Before contest
Codeforces Round #364 (Div. 1)
24:35:57

Before contest
Codeforces Round #364 (Div. 2)
24:35:57

Like 102 people like this. Be the first of your friends.

→ Top rated

#	User	Rating
1	Petr	3597
2	tourist	3580
3	TooDifficult	3416
4	Egor	3103
5	rng_58	3102
6	jcvb	2999
7	vepifanov	2992
8	matthew99	2986
9	subscriber	2968
10	Zlobober	2964

Countries | Cities | Organizations View all →

→ Top contributors

#	User	Contrib.
1	Errichto	178
2	gKseni	171
3	Petr	168
4	Zlobober	163
5	Edvard	159
6	Swistakk	157
7	chrome	149
8	rng_58	148
9	Xellos	145
10	amd	143

View all →

→ Find user

Handle:

Find

```
        F[mask] = A[0];
    // iterate over all the subsets of the mask
    for(int i = mask; i > 0; i = (i-1) & mask){
        F[mask] += A[i];
    }
}
```

Not as trivial, this solution is more efficient with time complexity of $O(3^N)$. To calculate the time complexity of this algorithm, notice that for each mask we iterate only over its subsets. Therefore if a mask has *K* on bits, we do 2^K iterations. Also total number of masks with *K on* bits is $\binom{N}{K}$. Therefore total iterations = $\sum_{k=0}^{N} \binom{N}{K} 2^k = (1+2)^N = 3^N$.

SoS Dynamic Programming solution

In this approach we will try to iterate over all subsets of mask in a smarter way. A noticeable flaw in our previous approach is that an index **A[x]** with **x** having **K** *off* bits is visited by 2^K **masks**. Thus there is repeated recalculation.
A reason for this overhead is that we are not establishing any relation between the **A[x]'s** that are being used by different **F[mask]'s**. We must somehow add another state to these masks and make semantic groups to avoid recalculation of the group.

Denote $S(mask) = \{x | x \subseteq mask\}$. Now we will partition this set into non intersecting groups. $S(mask, i) = \{x | x \subseteq mask \ \&\& \ mask \oplus x \leq 2^i\}$, that is set of only those subsets of **mask** which *differ* from **mask** only in the first **i** bits (zero based).
For example $S(\mathbf{1011010}, 3) = \{\mathbf{1011010}, \mathbf{1010010}, \mathbf{1011000}, \mathbf{1010000}\}$. Using this we can denote any set as a union of some non intersecting sets.

Lets try to relate these sets of numbers. S(mask, i) contains all subsets of *mask* which differ from it only in the first **i** bits.
Consider that **i**th bit of mask is **0**. In this case no subset can differ from mask in the **i**th bit as it would mean that the numbers will have a **1** at **i**th bit where **mask** has a **0** which would mean that it is not a subset of **mask**. Thus the numbers in this set can now only differ in the first **i-1** bits. $\Longrightarrow$ S(mask,i) = S(mask, i-1).
Consider that **i**th bit of mask is **1**. Now the numbers belonging to S(mask, i) can be divided into two non intersecting sets. One containing numbers with **i**th bit as **1** and differing from *mask* in the next **i-1** bits. Second containing numbers with **i**th bit as **0** and differing from *mask*$\oplus 2^i$ in next **i-1** bits. $\Longrightarrow$ S(mask, i) = S(mask, i-1) $\cup$ S(mask$\oplus 2^i$, i-1).

$$S(mask, i) = \begin{cases} S(mask, i-1) & i^{th} \ bit \ OFF \\ S(mask, i-1) \bigcup S(mask \oplus 2^i, i-1) & i^{th} \ bit \ ON \end{cases}$$

The following diagram depicts how we can relate the **S(mask,i)** sets on each other. Elements of any set **S(mask,i)** are the **leaves** in its subtree. The red prefixes depicts that this part of mask will be common to all its members/children while the black part of mask is allowed to differ.

→ Recent actions

mlv68 → Help needed in a SPOJ problem!!

theGalang → Problem on Spoj problem (INGRED

Tornad0 → A great idea for reading codes

reality → Codeforces Round #361 (Div. 2) Editorial

AboAlmanal → Really annoying problem with my code

shas19 → Closest point for all points in a plane

Mohamed_Ayman → Simple Solution for 699-C Div2 & 698-A Div1 (Vacations)

GlebsHP → Codeforces Round #363 problems analysis

usaxena95 → SOS Dynamic Programming [Tutorial]

GlebsHP → Codeforces Round #363

gskhirtladze → Central European Olympiad in Informatics 2016

Edvard → Разбор задач Educational Codeforces Round 14

dans → Codeforces Round #323 Editorial

cgy4ever → Codeforces Round #270 Editorial

halin.george → Codeforces Round #358 (Div. 2) Editorial

likecs → Bug in implementation

LelouchRiBritannia → Should I use zero-based index or one-based index

SarvagyaAgarwal → Need help with Game Theory !

arun_prasad → How to model this problem as a Nim problem?

EdenHazard → Good contest with T-shirts

Ishtiaq11 → Problem Solving using Python

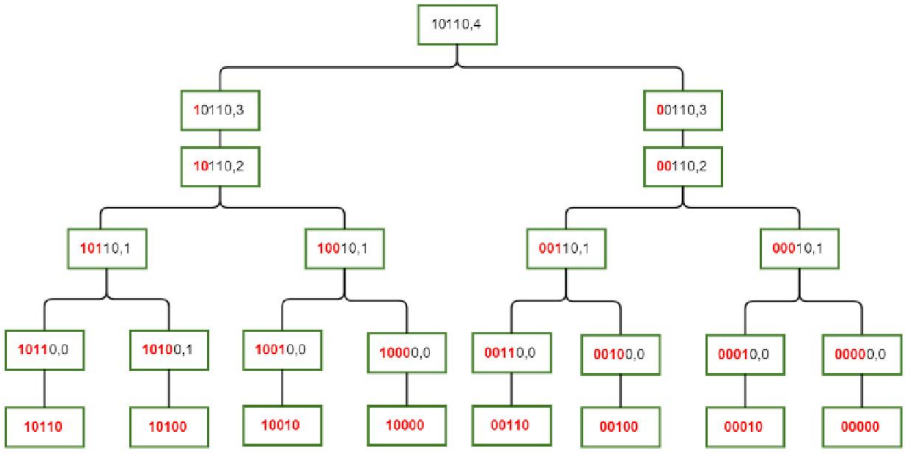
Superty → Small bug in Codeforces interface

wfe2016 → How to Prepare for the IOI given a good mathematical background

rng_58 → New Programming Contest Website: AtCoder

Tornad0 → understand problem LRU

Detailed →



Kindly note that these relations form a directed acyclic graph and not necessarily a rooted tree (think about different values of **mask** and same value of **i**)
After realization of these relations we can easily come up with the corresponding dynamic programming.

```
//iterative version
for(int mask = 0; mask < (1<<N); ++mask){
        dp[mask][-1] = A[mask]; //handle base case separately (leaf states)
        for(int i = 0;i < N; ++i){
                if(mask & (1<<i))
                        dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
                else
                        dp[mask][i] = dp[mask][i-1];
        }
        F[mask] = dp[mask][N-1];
}

//memory optimized, super easy to code.
for(int i = 0; i<(1<<N); ++i)
        F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
        if(mask & (1<<i))
                F[mask] += F[mask^(1<<i)];
}
```

The above algorithm runs in $O(N\,2^N)$ time.

Discussion Problem

Now you know how to calculate Sum over Subsets for a **fixed** array **A**. What would happen if **A** and **F** are SOS functions of each other ☺ . Consider following modification to the problem. Assume H1, H2 to be 32 bit integer valued hash functions (just to avoid any combinatoric approach to circumvent this problem) and can be evaluated at any point in constant time.:

$$F[mask] = H1\left(\sum_{i \subset mask} G[i]\right); \quad G[mask] = H2\left(\sum_{i \subseteq mask} F[i]\right)$$

I enjoyed solving this with **_shil**. Lets discuss the approaches in comments :)

Practice Problems

I hope you enjoyed it. Following are some problems built on SOS.
Vim War

Jersey Number
Covering Sets
Beautiful Sandwich
Jzzhu and Numbers
Vowels
Special Pairs
Pepsi Cola
Strange Functions
Uchiha and Two Products
Subset
COCI 2011/2012 Problem KOSARE
Compatible Numbers

Read more »

dynamic programming, bitmask, subset

+54 usaxena95 7 weeks ago 20

Divide by Zero 4.0 Summary

By **usaxena95**, history, 6 months ago,

Statistics

Total Users/Teams who have made a submission: **754**

Total Submissions: **4212**

Number of distinct users/teams with correct submissions: **498**

Contest Link

Feedback form

Winners

- Rank 1: **Sumeet.Varma**
- Rank 2: **amankedia1994**
- Rank 3: **PrashantM**

The Editorials of the following problems are prepared by me, **aditya1495** and **_shil**.

Shil and RasenShuriken

(setter: **_shil**)
Total number of pairs such that their product is even. Count total number of odd numbers = x.

Product of two numbers is odd if the both numbers are odd.

Ans = **N*(N-1)/2 — x*(x-1)/2**

Shil Loves Exclusive Or

(setter: **_shil**)
Use the fact that Xor of 2x and 2x+1 is 1.
Therefore xor of 1 to 4n+3 is 0.

You can derive the appropriate answer from the following results
F(4n+3) = 0
F(4n) = 4n
F(4n+1) = 4n ^ (4n+1) = 1

F(4n+2) = (4n+2)^1

Utkarsh and Random Trees

(setter: **usaxena95**)
Dp[i] = expected distance of node i from root.
Dp[i] = 1 + 0.5*(Dp[i-1] + Dp[i-2])
Also an observation is that the height of the tree is equal to the distance of node N from 1.
Therefore ans = **Dp[N]**.
Complexity **O(N + T)**

This is invalid if problem is to randomly take one of last 3 nodes as parent of i. So **Dp[i] = 1 + (Dp[i-1] + Dp[i-2] + Dp[i-3]) / 3** remains analogous. But expected height is not same as **Dp[N]**.

Same Old Matrix Problem

(setter: **aditya1495**)

Solving this problem required considering some cases.

First we will generate 2 DP Tables:

1.) dp[i][j] = Maximum money I can collect, moving only right or down and travelling from (1, 1) to (i, j).
2.) rdp[i][j] = Maximum money I can collect, moving from (i, j) to (N, M).

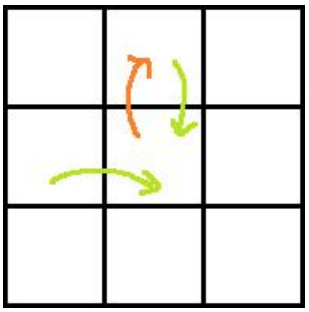
We can solve the problem using above two DP tables and splitting the possible scenarios in 6 different cases.

Let variable RES store your answer. Initialise RES with dp[N][M], as you are sure to make this profit without any change in direction.

Consider the following images for clarity.

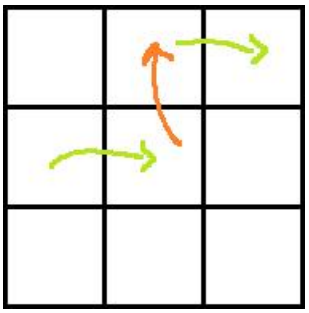
Red arrow is the move obtained by paying P dollars. Two green arrows indicate moves before and after the move with red arrow. Which one is taken before and which one is after can easily be understood.

Case 1:



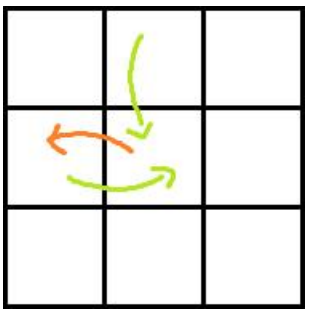
Ans: dp[i][j] + A[i − 1][j + 1] + rdp[i][j + 1] − P

Case 2:



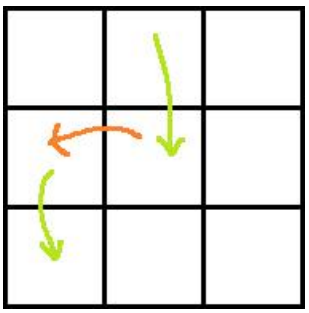
Ans: rdp[i − 1][j + 2] + dp[i][j] + A[i][j + 1] + A[i − 1][j + 1] − P

Case 3:



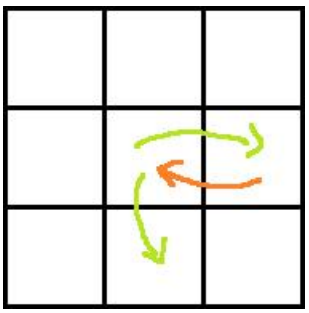
Ans: dp[i][j] + A[i + 1][j − 1] + rdp[i + 1][j] − P

Case 4:



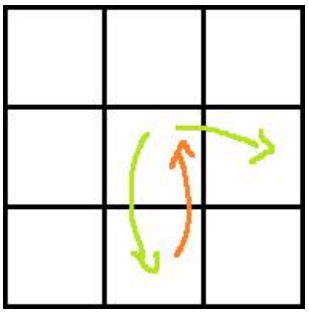
Ans: rdp[i + 2][j − 1] + dp[i][j] + A[i + 1][j − 1] + A[i + 1][j] − P

Case 5:



Ans: dp[i][j] + rdp[i + 1][j] + A[i][j + 1] − P

Case 6:



Ans: dp[i][j] + rdp[i][j + 1] + A[i + 1][j] − P

Take maximum of RES with Ans for all valid cells (i, j) for all valid cases.

Note: Don't forget that you cannot move out of grid. So make sure that **A**, **dp**, and **rdp** values exist for required positions before processing a case. Also since money in cell can be negative, make sure that you initialize **dp** and **rdp** with −INF. Many code made this mistake and got a penalty.

Utkarsh and Sub Array Sum

(setter: **usaxena95**) How to calculate F(A) efficiently ? Lets consider the i^{th} bit. Let the number of elements in **A** who has i^{th} bit **ON** be **x**. Number of elements whose i^{th} bit is **OFF** be **y** = N-x. Xor of a sequence of bits is 1 if the total number of 1's are odd. Number of sub sequences whose xor sum has i^{th} bit **ON** = number of sub sequences having odd number of **1**.

This can calculated by combinatorics = (Take any number of 0's)*(take odd number of 1's) = $(2^x)*({}^yC_1 + {}^yC_3 + {}^yC_5 + ..+) = (2^x) * (2^{y-1}) = 2^{x+y-1} = 2^{N-1}$. We add this to answer if **y ≥ 1**. Which means that add 2^{N-1} to ans if OR of i^{th} bits of A is 1 (i.e. **y ≥ 1**)

Result: F(A) depends only on the length of A and **Bitwise Or** of all the elements of A.

F(A) = OR * 2^{N-1} where N = size of A. So the problem was reduced to efficiently calculating the value of OR of a subarray and also deal with updates. This can be done easily with segment trees.

Complexity = **O(N + Q * LogN)** (Solution by **karanaggarwal**)

If you consider each bit individually, solution will be **O(30 * Q * logN)** BIT will get TLE.

Utkarsh and Tree Labelling

(setter: **usaxena95**) This was supposed to be a cumbersome hurdle which had a subtle solution.

One way to solve it is

- Do a DFS order
- Travel in reverse order
- You need to know the sum of all the unique elements in the range L..R
- This can be done as in DQUERY using BIT and map storing the last occurrence of a all values.

Another way to solve it storing for each node **v** all the unique values of F in the subtree of **v** in a set **S[v]** similar to problem Knapsack on Tree which is discussed later.

A very simple observation to make is that value of **F[v]** just depends on the distance of the node **v** from the farthest leaf present in the subtree of **v**. Quite a few participants were able

to notice this and were able to finish subtly in 40-60 LOC.

How can we prove this formally? Proof by induction:
Base case: The value of F[v] is 0 for all leaves. If Height H = 0 then F[v] is fixed = 0.

Assume for any node **u** whose height is ≤ **H** has **F[u]** dependent on just **H** (i.e. F[u] = someFunction(Height(u)))

Induction step: Consider node **v** with height = **H + 1**. Let **v** have children u_1, u_2,..u_n. It means that some children u_x's must have height = H. Other children will have height < **H**. As **subtree of u_x contains nodes of all the heights from 1 to H**, all those children of **v** whose height is &lt **H** are useless as their F[u] value must have been present in value of some node in subtree of u_x. (using induction step that **F[u]** is **independent of connectivity** of graph and just dependent on height of u if height ≤ **H**).
As the other children do not contribute to the calculation of v, we can just focus on child $\mathbf{u_x}$.
As F[u_x] is a function of only H and only this child will contribute to calculation of **F[v]**, therefore F[v] = is a function of only H+1.

We can precalc the answer for any height H as Pre[H].
if(Pre[i-1] is repeated in prefix i) then Pre[i] = Pre[i-1] + (Pre[H-1]+1) * $2^{Pre[H-1]}$ mod M
else Pre[i] = Pre[i-1]

Complexity **O(N + H*LogH)** (Solution by **gvaibhav21**) O(N) from dfs. H = height of root.

Shil and Special XorSum

(setter: **_shil**)
In this problem we were required to find Σ **RMQ(i,j) ^A[i]^A[j]** where i<j. **RMQ(i,j)** gives max of all the elements from i to j.

We will use *divide and conquer* approach for this problem. Suppose we are currently at state (l,r) , lets find **m** such that **A[m]** = max(A[l],A[l+1]..A[r]) and **l ≤ m ≤ r**. If there are many such **m**, pick anyone. This can be found easily using segment trees.

Now we will process for every bit position from 0 to 19. Suppose we are considering jth bit. We will find total number of pairs **(p,q)** such that **l ≤ p ≤ m ≤ q ≤ r** and jth bit is on in **A[m]^A[p]^A[q]**.

Let $\mathbf{l_1}$ be total number of positions in **[l,m]** having j^{th} bit on and $\mathbf{l_0}$ be total number of positions in **[l,m]** having j^{th} bit off. Similarly let $\mathbf{r_1}$ be total number of positions in **[m,r]** having j^{th} bit on and $\mathbf{r_0}$ be total number of positions in **[m,r]** having j^{th} bit off.

If j^{th} bit is on in **A[m]** , then total number of such **(p,q)** pairs is $\mathbf{(l_1*r_1 + l_0*r_0 -1)}$. If j^{th} bit is off in **A[m]** , then total number of such **(p,q)** pairs is $\mathbf{(l_1*r_0 + l_0*r_1 -1)}$. Hence we will add in answer $\mathbf{2^j*(l_1*r_1 + l_0*r_0 -1)}$ in first case and $\mathbf{2^j*(l_1*r_0 + l_0*r_1 -1)}$ in second case.
Then we will move forward to state **(l,m-1)** and **(m+1,r)**. Solution complexity is **O(N*logN +N*20)**.

See the solution for more understanding and implementation details:solution by **PrashantM**
The recursive solution I explained is implemented in iterative way in above solution.

Knapsack on Tree

(Setters: **architkarandikar**, **usaxena95** and **aditya1495**)

We are given a bound that no subset can have move than A[v] nodes from subtree rooted at node v. Hence if A[v] <= Size of subtree at v, then we have to choose the A[v] nodes have largest B values from subtree at v.

Lets maintain a Set for each node which stores valid nodes from it's subtree. If the Size of Set[v] exceeds A[v], then we'll remove smallest (Size of Set[v] — A[v]) nodes from Set[v].

Now to generate the Set[v], we'll use Set[x] for all x such that v is a parent of x. Note that if a node in subtree of x is not present in Set[x], then it will also be absent from Set[v] too and so on for all the ancestors. Hence we have to generate Set[v] by merging all the Set[x] and take A[v] largest elements in Set[v]. Then our answer would be elements of Set[1]. We can iterate on Set[1] and take summation of B values of its elements and that would be the answer.

Doing this plainly would be **O(n^2 log n)** which won't fit in in time limit. We need some smarter way of doing this.

To generate Set[v] of any node v, we'll first recognize the heavy child of this node. Heavy child would be the one which has largest size of it's Set (We'll refer as Heavy Set). We can then iterate on all the Light children and insert their elements in this Heavy Set. This Heavy set would now contain the nodes of subtree at v, the largest A[v] of which are required for Set[v]. We can resize Heavy set to contain the largest A[v] elements and now, instead of wasting time in copying those elements in Set[v], we can simply make a note that "If you want to see Set[v], see this Heavy set instead". This can be done by maintaining a 1-1 mapping using an array or using pointers and making the pointer of Set[v] point to this Heavy set.

Hence, we have now reduced **O(n^2 log n)** to **O(n log^2 n)** which fits perfectly in time limits! Ta-da!

Better Purchases

(Setter **aditya1495**)

This question is solvable by modelling it as a Minimum Cost Flow Network. Since it was a bipartite graph, min-cost matching can be used. Constraints are such that we need O(n^3) solution. We can use, for example Hungarian Algorithm which fits in time.

M = Number of shelves
N = Number of Companies
C[i] = Capacity of ith shelf
X[i] = Free goods that company i provides
Y[i] = Charged goods that company i provides
P[i] = Price per unit of charged good of Company i

We need some basic greedy things to clear out before we proceed. First step would be to sort the shelves in decreasing order of their capacities. Then if M > N, we'll take largest N shelves else we'll have to use all M. Let M' = min(M, N). We'll take largest M' shelves. Also, we'll take free goods from a company first and then only charged goods.

Assume that total number of goods you will order is sum of capacities all the M' shelves i.e. fill all shelves completely. Lets name it MaxGoods. We'll reduce it by counting the empty slots remaining at the end. Now, connect each company with the each shelf with flow capacity 1 and cost being as follows:

Cost[i][j] => if (X[i] + Y[i] <= C[j]) then Y[i] * P[i] + (C[j] − X[i] − Y[i]) * INF
if (X[i] < C[j]) then (C[j] − X[i]) * P[i]
if (X[i] >= C[j]) then 0

You can choose INF value greater than Sum (P[i] * Y[i]). Constraints permitted INF = 10^9. Here (C[j] − X[i] − Y[j]) indicates the amount of free slots that will remain if we match company i with shelf j. We "charge" such a match INF for each empty slot it creates, since we want to utilize maximum space (and hence maximum goods!).

Finding the min-cost match will give Flow = M' (obvious!). Useful part here would be the value of Cost. Since our matching will try to minimize total cost, it will try matching Companies-Shelves such that INF is used as less as possible and hence maximize goods in shelves. So the number of free slots that will be there in the end will be number of INF that

you can reduce from Cost i.e Cost / INF. All the remaining cost will be contributed by charged goods. So actual cost will be Cost % INF. (% is modulo operation).

So finally, the number of goods that we could purchase is:

PurchasedGoods = MaxGoods − EmptySlots = MaxGoods − Cost / INF.
TotalCost = Cost % INF.

Note that we are doing integer division in Cost / INF.

Many coders tried some greedy approaches which failed system test cases. Proof of failure of greedy approach is out of the scope of this write-up (unfortunately). Hence it is left as an exercise for reader. But a brute-force solution and a test generator can do the trick in case you need such a case.

Shil and Shortest Path

(setter **_shil**)

In this problem , we need to find lexicographical K^{th} shortest path from 1 to N. Let d[i] be the length of shortest path from any node i to N and dp[i] be total number of shortest path from any node i to N. Both of these can be found easily using BFS. Now we will build the lexicographical K^{th} string step wise. On each step, we will decide which character to add at the end of our current string. Final string formed after processing all such steps will be our lexicographical K^{th} shortest path string. Hence we will make d[1] such steps.

Let STATE denotes array of all the nodes we are processing in our current step. Initially, STATE will contain only node 1. On j^{th} step, STATE will contain all the nodes having their shortest path distance from N equal to d[1]-j+1 and that could lie on lexicographical K^{th} shortest path.While moving for next step, we will build new STATE array and forgot our previous STATE array.At last step, STATE will contain only last node N.

In each step, we will iterate from 'a' to 'z' until we have found the character that should be included in our string at j^{th} step. When we find that any particular character c should be included in our string, we will add it to the string and move to $j+1^{th}$ step. Now lets see how to decide that character c should be included in our string or not at certain j^{th} step.

In any particular step , we will iterate on all the nodes U present in our STATE array.For each node U , consider all nodes V adjacent to U such that d[V]=d[U]-1 and label of edge (U,V) is c. Let S denotes the total number of all the shortest path containing the edge (U,V) and their previous edges having the labels we have included in the string till now. If S>=K , we will include character c in our string , else we will subtract S from K and move to c+1 character.After we have decided that we should include certain character c in our string , we have to build new STATE array before moving to next step.This new STATE array can be build by inserting all the nodes V such that there exist edge (U,V) where U is present in our previous state , d[V]=d[U]-1 and label of edge (U,V) is c. For calculating S and dp , we have to take special care to check for overflow since total number of shortest path in any graph could be very high.See the solution for seeing how to calculate S and implementation details.

solution

Feel free to discuss your own strategies of solving the problems in the comments. :D

Read more »

dibz2016, editorial

+52 usaxena95 6 months ago 9

Invitation to Divide by Zero 4.0

By **usaxena95**, 6 months ago,

FLUXUS — IIT Indore & **Programming Club, IIT Indore** are proud to announce the fourth edition of **Divide By Zero!**

It will be an ACM-ICPC style contest with all problems visible from the start and no hacks. Its a 5 hour contest with 10 problems.

Details :

Contest Date : 30th January 2016
Time : 2100 to 0200 IST
Programming Partner : Codechef
Contest Link

Finally, don't forget to register.

The problems in the contest are prepared and tested by **usaxena95**, **aditya1495** and **_shil**.
Special thanks to **architkarandikar** for his problem ideas.

Prizes

There are cash prizes for top 3 Indians as follows:

- Rank 1: **₹ 10,000**
- Rank 2: **₹ 8,000**
- Rank 3: **₹ 7,000**

Top **20** Indians will receive T-shirts.

For more details / contact details, visit:

- Our Facebook Event Page
- Or contact us at: Programming Club IIT Indore

Do share it with your friends who are interested :D
Good Luck! Hope to see you participating!

Read more »

dibz2016, **iit indore**, **fluxus**

+41 usaxena95 6 months ago 10

Invitation: Divide By Zero 3.0

By **usaxena95**, 16 months ago,

FLUXUS — IIT Indore & **Programming Club, IIT Indore** are proud to announce the third edition of **Divide By Zero!**

It will be an ACM-ICPC style contest with all problems visible from the start and no hacks.

Details :

Date : 29th March 2015

Time : 2130 to 0130 IST Official Time Announcement

Programming Partner : Codechef

Contest link

Prizes

This time we are giving away prizes of worth ₹ 10,000/-

Indian Winners

Top 25 Indian winners will get special Limited Edition Fluxus T-Shirts

In addition to T-Shirts, Top 3 Winners will also be getting Cash Prizes & Goodies

Non-Indian Winners

CodeChef will be sponsoring 3 T-Shirts to the Top 3 winners

Finally, don't forget to register at FLUXUS as Prizes are only applicable to users with a Fluxus-ID

Also, we would like to thank our problem setters and testers:**usaxena95**, **aditya1495**, [user:harshil,2015-03-27], **gaurav708**

Do share it with your friends who are interested :D

For more details / contact details, visit:

Our Facebook Event Page

Our Fluxus Event Page

Or mail us at: sainathbatthala@gmail.com

EDIT: Due to huge response, our chefs have decided to add a few more delicious dishes and to extend duration of the feast to 4 hrs! :)

So please note our new timings:

Start: Mar 29 at 9:30pm (Remains same)

End: Mar 30 at 1:30am (Extended)

Read more »

iit indore, **dbyz15**, **coding contest**

+24 usaxena95 16 months ago 8

Codeforces (c) Copyright 2010-2016 Mike Mirzayanov
The only programming contests Web 2.0 platform
Server time: Jul/21/2016 21:26:41 UTC+5.5 (p1).
Desktop version, switch to mobile version.