Please try yourself 

A little collections:

1. DP
2. Coin change
3. LCS
4. LIS
5. Edit Distance
6. Digit DP
7. Knapsack
8. Bitmask
9. Searching
10. Binary Search
11. Trinary Search
12. Number Theory
13. Big mod
14. Inverse modular
15. Prime Sieve
16. Euler phi
17. Extended GCD
18. Prime factorization
19. Number of Divisor
20. Sum of Divisor
21. Inverse phi
22. Floyds Cycle Finding
23. Probability
24. Combinatorics
25. Sterling Number 1
26. Sterling Number 2
27. Derangement ( with recur without recur)
28. Graph
29. BFS
30. DFS
31. Topological Sorting
32. Articulation points
33. Bridge
34. SCC
35. Floyd War shall
36. Heavy Light Decomposition (1348- Aladin and the return journey && QTREE – Query on a tree).
37. Greedy
38. Maximum Sum 1D O(n)
39. Maximum Sum 2D O(n^3)
40. Maximum Rectangle O(n^2)
41. Geometry
42. Picks Theorem
43. Segment Segment Intersection
44. Union of rectangle
45. Closest Pair
46. String
47. LC Substring
48. KMP
49. Miscellaneous
50. Meet in the Middle
51. Tower of Hanoi
52. TSP
53. Sliding Window
54. Solving Linear Recurrence With Matrix Exponentiation
55. Magic Square construction
56. Gaussian Elimination

\*\*\*\*\*\*\*\*\*\*\*Searching codes\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Binary Search:

int LoC(int A[], int left, int right, int item) {

int mid;

while (right - left > 1) {

mid = left + (right - left) / 2;

if (A[mid] <= item)

left = mid;

else

right = mid;

}

return left;

}

int FoC(int A[], int left, int right, int item) {

int mid;

while (right - left > 1) {

mid = left + (right - left) / 2;

if (A[mid] >= item)

right = mid;

else

left = mid;

}

return right;

}

int NoCs(int A[], int size, int item) {

// Note the boundary parameters

int left = FoC(A, -1, size - 1, item);

int right = LoC(A, 0, size, item);

if (A[left] == item && A[right] == item) // If the item is in the array A

return (right - left + 1);

else // If the item is not in the array A

return 0;

}

Trinary Search:

#include<bits/stdc++.h>

using namespace std;

#define eps 1e-12

int TrSr(double left , double right)

{

while(fabs(right - left)> eps)

{

double mid1 = left + (right - left)/3.0;

double mid2 = right - (right - left)/3.0;

if(mid1>mid2)

left = mid1;

if(mid2>mid1)

right = mid2;

}

}

\*\*\*\*\*\*\*\*\*\*Number Theory\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Big mod:

#include<bits/stdc++.h>

using namespace std;

int mod = 1000000007;

int bigmod( int b , int p )

{

if(b==0)

return 1;

int x = bigmod(b, p/2 );

x %=mod;

x = (x\*x)%mod;

if(b&1)

x = (x\*b)%mod;

return x;

}

Sieves:

#include<stdio.h>

#include<math.h>

int prime[100000],a[10000000];

int prime\_sieve()

{

int i,j,k=0,lm,n;

n=1000000;

a[0]=a[1]=1;

for(i=4; i<=n; i+=2)

a[i]=1;

//prime[k++]=2;

lm=sqrt(n);

for(i=3; i<=lm; i+=2)

{

if(!a[i])

{

//prime[k++]=i;

for(j=i\*i; j<=n; j+=2\*i)

{

a[j]=1;

}

}

}

prime[k++]=2;

for(i=3; i<=n; i+=2)

{

if(!a[i])

{

prime[k++]=i;

printf("%d ",prime[k-1]);

}

}

}

int phi[99999999];

int euler\_sieve()

{

int i,j,c,n=99999998;

phi[0]=99999999;

phi[1]=0;

for(i=2;i<=n;i++)

{

if(!phi[i])

{

for(j=i;j<=n;j+=i)

{

if(!phi[j])

phi[j]=j;

phi[j]/=i;

phi[j]\*=(i-1);

}

/\*if(phi[0]%i==0)

{

phi[0]/=i;

phi[0]\*=(i-1);

}\*/

}

}

}

#include<stdio.h>

#include<math.h>

unsigned long long int phi[5000010];

int phi\_seieve()

{

long long int i,j,k=5000005;

phi[0]=1;

phi[1]=1;

for(i=2;i<k;i++)

{

if(!phi[i])

{

for(j=i;j<k;j+=i)

{

if(!phi[j])

phi[j]=j;

phi[j]\*=(i-1);

phi[j]/=i;

}

}

phi[i]=phi[i-1]+phi[i]\*phi[i];

}

}

int main()

{

//prime\_sieve();

euler\_sieve();

printf("%d\n",phi[99999998]);

}

Extended GCD:

#include<stdio.h>

int d, x, y;

void extendedEuclid(int A, int B)

{

if(B == 0)

{

d = A;

x = 1;

y = 0;

//printf("1\n");

}

else

{

//printf("2\n");

//printf("%d %d\n",A,B);

extendedEuclid(B, A%B);

int temp = x;

x = y;

//printf("%d %d\n",A,B);

y = temp - (A/B)\*y;

//printf("%d %d\n",x,y);

//printf("3\n");

}

}

int main( )

{

int a , b ,c;

extendedEuclid(16, 10);

// actual result

int x1 = x\*(c/d) +(b/d)\*n;

int y1 = y\*(c/d) - (a/d)\*n;

printf("%d %d %d\n",d,x,y);

return 0;

}

Prime Factorization:

#include<stdio.h>

#include<math.h>

prime\_factor(int n)

{

int a;

if (n%2==0)

return 2;

for (a=3; a<=sqrt(n); a++)

{

if (n%a==0)

return a;

}

return n;

}

int main()

{

int r,n,a[1000],k=1,j,i,c,tem;

scanf("%d",&n);

while (n>1)

{

r = prime\_factor(n);

a[k]=r;

n /= r;

k++;

}

for(j=1; j<k;)

{

c=0;

if(a[j]!=0)

{

for(i=j; i<k; i++)

{

if(a[j]==a[i])

{

c++;

}

else

{

break;

}

}

printf("prime %d present %d times\n",a[j],c);

j=i;

}

}

return 0;

}

Number of Divisor:

#include<bits/stdc++.h>

using namespace std;

int dns[1000000];

int NODs(int n)

{

for(int i = 1 ; i<= n;i++)

{

for(int j = i ; j<=n ; j+=i)

dns[i]++;

}

}

int prime[10000],dnp[10000];

int NODp(int n)

{

int cnt = 0,dn=1;

/\*while(n%2==0)

{

n/=2;

cnt++;

}\*/

for(int i = 0 ; prime[i]<= sqrt(n) ;i++)

{

cnt =0;

if(n%prime[i]==0)

{

while(n%prime[i]==0)

{

n/=prime[i];

cnt++;

}

dn\*=(cnt+1);

}

}

if(n!=1)

dn\*=2;

return dn;

}

int main()

{

}

Sum of Divisor:

#include<bits/stdc++.h>

using namespace std;

int sd[100000];

int SODs(int n)

{

for(int i = 1 ; i<= n ;i++)

{

for(int j = i; j<= n ; j+=i)

{

sd[i] +=i;

}

}

}

int main()

{

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*Combinatorics\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Sterling Number:

#include<bits/stdc++.h>

using namespace std;

int ar2[1000][1000];

int stl2(int n ,int k)

{

if(k==1 || n==k)

return 1;

int &v = ar2[n][k];

if(v!=-1)

return v;

else

{

return v = stl2(k-1 , n-1) + k\*stl2(n-1 , k);

}

}

int fact(int n)

{

if(n==0)

return 1;

return n\*fact(n-1);

}

int ar1[1000][1000];

int stl1( int n , int k)

{

if(n==k)

return 1;

if(k==1)

return fact(n-1);

int &v = ar1[n][k];

if(v!=-1)

return v;

else

{

return v = stl1(n-1 , k-1) + (n-1)\*stl1(n-1 , k);

}

}

int main()

{

return 0;

}

\*\*\*Derangement:

#include<bits/stdc++.h>

using namespace std;

typedef long long int ll;

ll d\_ar[100];

ll dearrangement(ll n)

{

if(n==1)

return 0;

if(n==2)

return 1;

if(d\_ar[n]!=-1)

return d\_ar[n];

return d\_ar[n]=(n-1)\*(dearrangement(n-1)+dearrangement(n-2));

}

int main()

{

ll n,i;

memset(d\_ar,-1,sizeof(d\_ar));

scanf("%lld",&n);

for(i=1;i<=20;i++)

printf("%lld\n",dearrangement(i));

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*Graph Theory\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BFS:

#include<bits/stdc++.h>

#include<vector>

#include<queue>

using namespace std;

vector<int>adj[100];

int visited[100];

void bfs(int s,int n)

{

memset(visited,0,sizeof(visited));

queue<int>Q;

Q.push(s);

visited[s]=1;

while(!Q.empty())

{

int u= Q.front();

Q.pop();

int i;

for(i=0;i<adj[u].size();i++)

{

if(visited[adj[u][i]]==0)

{

int v=adj[u][i];

visited[v]=1;

Q.push(v);

}

}

}

}

int main()

{

}

DFS:

#include<bits/stdc++.h>

#include<stack>

#include<vector>

using namespace std;

vector<int>adj[100];

int edge\_id[100];

int visited[100];

void dfs(int s,int n)

{

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

visited[i]=edge\_id[i]=0;

stack<int>S;

S.push(s);

while(!S.empty())

{

int u = S.top();

S.pop();

while(edge\_id[u]<adj[u].size())

{

int v=adj[u][edge\_id[u]];

edge\_id[u]++;

if(visited[v]==0)

{

visited[v]=1;

S.push(u);

S.push(v);

break;

}

}

}

}

int main()

{

}

TS:

#include<bits/stdc++.h>

#include<stack>

#include<vector>

using namespace std;

vector<int>adj[100];

int edge\_id[100];

int visited[100];

typedef vector< int > vi;

typedef pair< int , int> ii;

typedef vector < ii > vii;

vi ts;

int unvisited = 0, visitedd = 1;

void dfs2(int u)

{

visited[u] = visitedd;

for( int i = 0 ; i < adj[u].size() ; i++)

{

if( visited[adj[u][i]]==unvisited)

dsf2( adj[u][i]);

}

ts .push\_back( u);

}

int main()

{

ts.clear();

memset( visited, unvisited , sizeof(visited));

for( int i = 0 ; i < v ; i++)

{

if(visited[i]== unvisited)

{

dfs2(i);

}

}

for( int i = ts.size()-1 ; i>=0 ;i--)

cout<<ts[i]<<" ";

cout<<endl;

}

FLOODFILL:

#include<bits/stdc++.h>

using namespace std;

int dr[]={ 1 , 1 , 1, 0 , 0 ,-1 ,-1 ,-1 };

int dc[]={-1 , 0 , 1, -1, 1 ,-1 , 0 , 1 };

int grid[100][100];

int floodfill(int i, int j , char c1 , char c2)

{

if( i <0 || i >=r || j<0 || j>=c )

return 0;

if(grid[i][j] != c1)

return 0;

int ans = 1;

grid[i][j] = c2;

for(int l = 0; l<8 ;l++)

{

ans += floodfill( i + dr[l] , j + dc[l] , c1 , c2);

}

return ans;

}

int main()

{

return 0;

}

EDGES CHECKING:

#include<bits/stdc++.h>

using namespace std;

int explored = 0, unexplored = -1, visitedd = 1 ;

int dfs\_num[1000] , dfs\_parent[1000];

void graphcheck(s)

{

dfs\_num[u] = explored;

for( int i = 0 ;i< adj[u].size();i++)

{

int v = adj[u][i];

if(dfs\_num[v]==unexplored)

{

dfs\_parent[v] = u;

graphcheck(v);

}

else if( dfs\_num[v]==explored)

{

if(v == dfs\_parent[u])

printf("Two ways exist to go child node from parent node");

else

{

printf("BAck edge exits");

}

}

else if( dfs\_num[v] == visited)

printf("forward edge exits");

}

}

int main()

{

memset( dfs\_num , unexplored , sizeof(unexplored));

graphcheck(s);

return 0;

}

Articulation points and Bridge:

#include<bits/stdc++.h>

using namespace std;

int dfs\_num[100], dfs\_low[100], dfs\_parent[100];

typedef pair< int , int > ii;

typedef vector< ii> vii;

typedef vector< int > vi;

vii bridge;

vi AP;

int dfsNumbercounter = 0;

void ArticulationPointAndBridge( int u)

{

dfs\_num[u] = dfs\_low[u] = dfsNumbercounter ++;

for( int i = 0; i < adj[u].size(); i++)

{

int v = adj[u][i];

if( dfs\_num[v] == unvisited)

{

dfs\_parent[v] = u;

if( u == dfsroot)

rootchildren++;

ArticulationPoinAndBridge(v);

if(dfs\_low[v] >= dfs\_num[u])

Ap.push\_back(u);

if(dfs\_low[v] > dfs\_num[u])

{

ii p;

p.first = u ;

p.second = v;

bridge.push\_back(p);

}

dfs\_low[u] = min ( dfs\_low[u] , dfs\_low[v]);

}

else if( v != dfs\_parent[u])

{

dfs\_low[u] = min( dfs\_low[u] , dfs\_num[v]);

}

}

}

int main()

{

memset()

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

{

cout<<AP[i] << " ";

}

cout<<endl;

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

{

cout<< bridge[i].first << "->"<<bridge[i].second<<endl;

}

return 0;

}

SCC:

Tarjan’s Algo:

#include<bits/stdc++.h>

using namespace std;

typedef vector< int > vi;

vi dfs\_num , dfs\_low , S ,visited ;

int unvisited = -1;

vi adj[100];

int dfsNumbercounter = 0 , numSCC =0;

void tarjanSCC( int u)

{

dfs\_num[u] = dfs\_low[u] = dfsNumbercounter++;

S.push\_back(u);

visited[u] = 1;

for( int i = 0 ; i < adj[u].size() ; i++)

{

int v = adj[u][i] ;

if(dfs\_num[v] == unvisited)

{

tarjanSCC(v);

}

if(visited[v])

{

dfs\_low[u] = min( dfs\_low[u] , dfs\_low[v]);

}

if(dfs\_low[u] == dfs\_num[u])

{

printf("SCC %d: ",++numSCC);

while(!S.empty())

{

int p = S.back();

S.pop\_back();

printf(" %d",p);

if(p==u)

break;

visited[p] = 0;

}

cout<<endl;

S.push\_back(u);

}

}

}

int main()

{

dfs\_num.assign( 10 , unvisited);

dfs\_low.assign(10 ,0);

visited.assign(10 ,0);

adj[0].push\_back(1);

adj[1].push\_back(3);

adj[1].push\_back(4);

//adj[1].push\_back(9);

adj[3].push\_back(2);

adj[2].push\_back(1);

adj[4].push\_back(5);

adj[5].push\_back(6);

adj[6].push\_back(1);

// adj[4].push\_back(8);

// adj[8].push\_back(1);

// adj[9].push\_back(10);

for( int i = 0 ; i < 7 ; i++)

{

if(dfs\_num[i] == unvisited)

tarjanSCC(i);

}

return 0;

}

SCC by Kosaraju :

#include<bits/stdc++.h>

using namespace std;

typedef vector < int > vi;

vi adj[100], adjT[100];

vi dfs\_num, S;

int DFS\_WHITE = -1;

void Kosaraju( int u , int pass)

{

dfs\_num[u] = 1;

vi neighbor ;

if( pass == 1)

neighbor = adj[u];

else

neighbor = adjT[u];

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

{

int v = neighbor[i];

if(dfs\_num[v] == DFS\_WHITE)

{

Kosaraju(v, pass);

}

}

S.push\_back(u);

}

int main()

{

dfs\_num.assign(v , DFS\_WHITE);

S.clear();

for( int i = 0 ; i < v ; i++)

if(dfs\_num[i] == DFS\_WHITE)

Kosaraju(i, 1);

int numSCC =0;

dfs\_num.assign(v ,DFS\_WHITE);

for(int i = v - 1 ; i >= 0 ; i-- )

{

if(dfs\_num[S[i]] == DFS\_WHITE)

{

++numSCC;

Kosaraju(S[i] , 2);

}

}

return 0;

}

\*\*\*\*\*Heavy Light Decomposition\*\*\*\*\*\*\*\*\*

\*1348-Aladin and the return journey

#include<bits/stdc++.h>

using namespace std;

#define MXN 100005

vector< int > adj[MXN];

int ar[MXN] , N;

int parent[MXN] , level[MXN] , subSize[MXN] , chainHead[MXN] , chainInd[MXN] , baseArray[MXN] , posInBaseArray[MXN];

int baseCount = -1,chainNo = 0;

int sparse[MXN][22];

int tree[6\*MXN];

void dfs(int cur , int prev , int depth)

{

parent[cur] = prev;

level[cur] = depth;

subSize[cur] = 1;

int sz = adj[cur].size();

for(int i= 0; i < sz ; i++)

{

if(adj[cur][i] != prev )

{

dfs(adj[cur][i] , cur , depth+1);

subSize[cur] += subSize[adj[cur][i]];

}

}

}

void hld(int cur , int prev)

{

if(chainHead[chainNo] == -1)

chainHead[chainNo] = cur;

chainInd[cur] = chainNo;

baseArray[++baseCount] = ar[cur];

posInBaseArray[cur] = baseCount;

int sz = adj[cur].size();

int ind = -1 , maxi = -1;

for(int i = 0; i < sz ; i++)

{

if(subSize[adj[cur][i]] > maxi && prev != adj[cur][i])

{

ind = i;

maxi = subSize[adj[cur][i]];

}

}

if(ind >= 0)

{

hld( adj[cur][ind] , cur);

}

for(int i = 0; i < sz; i++)

{

if( i!= ind && adj[cur][i] != prev)

{

chainNo++;

hld(adj[cur][i] , cur);

}

}

}

void lca\_init()

{

memset(sparse , -1 , sizeof(sparse));

for(int i = 0; i < N ; i++)

sparse[i][0] = parent[i];

for(int j = 1; (1 << j) < N ;j++ )

{

for(int i = 0; i < N ; i++)

{

if(sparse[i][j-1]!=-1)

sparse[i][j] = sparse[sparse[i][j-1]][j-1];

}

}

}

int lca\_query(int p , int q)

{

if(level[p] < level[q])

swap(p , q);

int log = 1;

while(1)

{

if( (1 << log) > level[p])

{

break;

}

log++;

}

log -=1;

for(int i = log ; i >= 0 ; i--)

{

if( (level[p] - (1 << i)) >= level[q])

{

p = sparse[p][i];

}

}

if( p == q)

return q;

for(int i = log; i >= 0; i--)

{

if(sparse[p][i] !=-1 && sparse[p][i]!=sparse[q][i])

{

p = sparse[p][i];

q = sparse[q][i];

}

}

return parent[p];

}

void make\_tree(int node , int beg , int endd)

{

if(beg==endd)

{

tree[node] = baseArray[beg];

return;

}

int left = node << 1;

int right = left + 1;

int mid = (beg + endd) >> 1;

make\_tree(left , beg , mid);

make\_tree(right , mid+1 , endd);

tree[node] = tree[left] + tree[right];

return;

}

void update\_tree(int node , int beg , int endd , int pos , int val)

{

if(beg == pos && endd == pos)

{

baseArray[beg] = val;

tree[node] = val;

return;

}

int left = node << 1;

int right = left + 1;

int mid = (beg + endd ) >> 1;

if( mid >= pos)

{

update\_tree(left , beg , mid , pos , val);

}

else

{

update\_tree(right , mid+1 , endd , pos , val);

}

tree[node] = tree[left] + tree[right];

return;

}

int query\_tree(int node , int beg , int endd , int i , int j)

{

if(i > j)

return 0;

if(beg == i && endd == j)

{

return tree[node];

}

int left = node << 1;

int right = left + 1;

int mid = (beg + endd) >> 1;

int l = query\_tree(left , beg , mid , i ,min( mid , j));

int r = query\_tree(right , mid+1 , endd , max(i , mid+1 ) , j);

return l+r;

}

int query\_up(int u , int v)

{

int ui = chainInd[u];

int vi = chainInd[v];

if(v == u)

return baseArray[posInBaseArray[v]];

if(ui == vi)

{

int up = posInBaseArray[u];

int vp = posInBaseArray[v];

if(up > vp)

swap(up , vp);

return query\_tree(1 , 0 , baseCount , up , vp);

}

else

{

int vh = chainHead[vi];

int res1 = query\_up(u , parent[vh]);

return res1 + query\_tree(1 , 0 , baseCount , posInBaseArray[vh] , posInBaseArray[v]);

}

}

int query(int u , int v)

{

int LCA = lca\_query(u , v);

int res1 = query\_up(LCA , u);

int res2 = query\_up( LCA , v);

return res1 + res2 - baseArray[posInBaseArray[LCA]];

}

void change(int pos , int val)

{

int pi = posInBaseArray[pos];

update\_tree(1 , 0 , baseCount , pi , val);

return;

}

int main()

{

int tc;

scanf("%d",&tc);

int cs = 0;

while(tc--)

{

memset(chainHead , -1, sizeof(chainHead));

for(int i = 0; i < MXN ; i++)

adj[i].clear();

baseCount = -1;

cs++;

scanf("%d",&N);

for(int i = 0; i < N; i++)

scanf("%d",&ar[i]);

int u,v;

for(int i = 1; i < N; i++)

{

scanf("%d %d",&u, &v);

adj[u].push\_back(v);

adj[v].push\_back(u);

}

dfs(0 , -1 , 0);

hld(0 , -1);

lca\_init();

make\_tree(1 , 0 , baseCount);

int q, cmd , a , b;

scanf("%d",&q);

printf("Case %d:\n", cs);

for(int i = 0; i < q; i++)

{

scanf("%d %d %d",&cmd , &a, &b);

if(cmd)

{

change(a , b);

}

else

{

int res = query(a , b);

printf("%d\n", res);

}

}

}

return 0;

}

\*QTREE- Query on a tree

#include<bits/stdc++.h>

using namespace std;

#define MXN 100005

int chainNo = 0, chainInd[MXN] , chainHead[MXN] , chainPos[MXN],chainSize[MXN] , subSize[MXN];

vector < int > adj[MXN] , costs[MXN];

int baseArray[MXN] , cnt = -1 , posInBaseArray[MXN];

int parent[MXN] , level[MXN] , sparse[MXN][22];

vector< pair<int , int > > vii;

int tree[MXN\*6];

int N;

void dfs(int cur , int prev , int depth)

{

//cout<<"node "<<cur<<" parent "<<prev<<" level "<<depth<<endl;

parent[cur] = prev;

level[cur] = depth;

subSize[cur] = 1;

int sz = adj[cur].size();

for(int i = 0; i < sz; i++)

{

if(adj[cur][i]!=prev)

{

dfs(adj[cur][i] , cur , depth+1);

subSize[cur] +=subSize[adj[cur][i]];

}

}

}

void hld(int cur , int cost , int prev)

{

// cout<<" Edge "<<prev<<"->"<<cur<<" stored "<<cur<<endl;

if(chainHead[chainNo] == -1)

chainHead[chainNo] = cur;

chainInd[cur] = chainNo;

// chainPos[cur] = chainSize[chainNo];

// chainSize[chainNo]++;

posInBaseArray[cur] = ++cnt;

baseArray[cnt] = cost;

int ind = -1 , maxi = -1 , mxc = -1;

int sz = adj[cur].size();

for(int i = 0 ; i < sz; i++)

{

if( subSize[adj[cur][i]] > maxi && prev != adj[cur][i])

{

maxi = subSize[adj[cur][i]] ;

ind = i;

mxc = costs[cur][ind];

}

}

if(ind >= 0)

{

hld( adj[cur][ind] , mxc , cur);

}

for(int i = 0; i < sz; i++)

{

if(ind != i && adj[cur][i] != prev)

{

chainNo++;

hld( adj[cur][i] , costs[cur][i] , cur);

}

}

return;

}

void lca\_init()

{

memset(sparse , -1 , sizeof(sparse));

for(int i = 1; i <= N; i++)

sparse[i][0] = parent[i];

for(int j = 1; 1<<j <= N ; j++)

{

for(int i = 1 ; i <= N; i++)

{

if(sparse[i][j-1]!=-1)

{

sparse[i][j] = sparse[sparse[i][j-1]][j-1];

}

}

}

}

int lca\_query( int p , int q)

{

if(level[p]<level[q])

swap(p,q);

int log = 1;

while(1)

{

if(1 << log > level[p])

break;

log++;

}

log -=1;

for(int i = log; i>=0 ; i--)

{

if((level[p] - (1 << i)) >= level[q] )

{

p = sparse[p][i];

}

}

if(p==q)

return p;

for(int i = log ; i>= 0; i--)

{

if(sparse[p][i] != -1 && sparse[p][i] != sparse[q][i])

{

p = sparse[p][i];

q = sparse[q][i];

}

}

return parent[p];

}

void make\_tree(int node , int beg , int endd)

{

if(beg == endd)

{

tree[node] = baseArray[beg];

return;

}

int left = node<<1;

int right = left+1;

int mid = (beg + endd) >> 1;

make\_tree(left , beg , mid);

make\_tree(right , mid+1 , endd);

tree[node] = max( tree[left] , tree[right]);

return;

}

void update(int node , int beg , int endd , int pos , int val)

{

if(beg==pos && endd == pos)

{

//cout<<node<<" "<<beg<<" "<<endd<<" "<<pos<<endl;

baseArray[beg] = val;

tree[node] = val;

return;

}

int left = node << 1;

int right = left + 1;

int mid = (beg + endd ) >> 1;

if(mid >= pos)

update(left , beg , mid , pos , val);

else

update(right , mid+1 , endd , pos , val);

tree[node] = max( tree[left] , tree[right]);

return;

}

int query\_tree(int node , int beg , int endd , int i , int j)

{

if(i > j)

return -1;

if(beg == i && endd == j)

{

//cout<<node<<" "<<beg<<" "<<endd<<" "<<tree[node]<<endl;

return tree[node];

}

int left = node << 1;

int right = left + 1;

int mid = (beg + endd) >>1 ;

return max(query\_tree(left , beg , mid , i , min( j , mid)) , query\_tree(right , mid+1 , endd , max( i , mid+1 ) , j));

}

int query\_up(int u , int v)

{

//cout<<u<<" "<<v<<endl;

int ui = chainInd[u];

int vi = chainInd[v];

if(u == v)

return 0;

if(ui == vi)

{

int up = posInBaseArray[u];

int vp = posInBaseArray[v];

if(up > vp)

swap( up , vp);

return query\_tree(1 , 0 , cnt , up+1 , vp );

}

else

{

int uh = chainHead[ui];

int vh = chainHead[vi];

// cout<<vh<<endl;

int res1 = query\_up(u , parent[vh]);

//cout<<baseArray[posInBaseArray[vh]]<<endl;

res1 = max(res1 , baseArray[posInBaseArray[vh]]);

//cout<<res1<<endl;

return max( query\_tree(1 , 0 , cnt , posInBaseArray[vh]+1 , posInBaseArray[v]) , res1);

}

}

int query(int u , int v)

{

int LCA = lca\_query(u , v);

//cout<<"LCA "<<LCA<<endl;

//cout<<baseArray[posInBaseArray[u]]<<" array value "<<baseArray[posInBaseArray[v]]<<endl;

int res1 = query\_up(LCA , u);

int res2 = query\_up(LCA , v);

return max( res1 , res2);

}

void change(int edge , int val)

{

int u = vii[edge].first;

int v = vii[edge].second;

int ui = posInBaseArray[u];

int vi = posInBaseArray[v];

//cout<<ui<<" EDGE "<<vi<<endl;

int en;

if(ui > vi)

{

update(1 , 0 , cnt , ui , val);

}

else

{

update( 1, 0 , cnt , vi , val);

}

}

int main()

{

//freopen( "output.txt", "w" , stdout);

int tc;

scanf("%d",&tc);

while(tc--)

{

cnt = -1;

memset(chainHead , -1 , sizeof(chainHead));

memset(chainInd , 0 , sizeof(chainInd));

memset(subSize , 0 , sizeof(subSize));

memset(parent , 0 , sizeof(parent));

memset(level , 0 , sizeof(level));

scanf("%d",&N);

int u , v, c;

for(int i = 1; i < N ; i++)

{

cin>>u>>v>>c;

adj[u].push\_back(v);

costs[u].push\_back(c);

adj[v].push\_back(u);

costs[v].push\_back(c);

vii.push\_back(make\_pair(u, v));

}

char str[100];

dfs(1 , 0 , 0);

// for(int i = 1 ; i <= N; i++)

// cout<<parent[i]<<" parent of "<<i<<endl;

// cout<<endl;

hld(1 , -1 , -1);

make\_tree(1 , 0 , cnt);

lca\_init();

// for(int i = 0; i < N ; i++)

// cout<<baseArray[i]<<" ";

// cout<<endl;

while(1)

{

scanf(" %s", str);//>>u>>v;

//cout<<str<<" "<<u<<" "<<v<<endl;

if(strcmp(str , "DONE")==0)

break;

scanf(" %d %d",&u , &v);

if(strcmp(str , "CHANGE")==0)

{

change(u - 1 , v);

// for(int i = 0; i < N; i++)

// cout<<i<<" "

}

else if(strcmp(str , "QUERY")==0)

{

int res = query(u , v);

printf("%d\n",res);

}

}

for(int i = 0 ; i < MXN ; i++)

{

adj[i].clear();

costs[i].clear();

}

vii.clear();

}

return 0;}

\*\*\*\*\*\*\*\*\*\*\*\*D YNAMIC PROGRAMMING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LIS:

#include<bits/stdc++.h>

using namespace std;

const int inf = 9999999999;

int n;

int sequence[100];

int L[100];

int I[100];

void takeInput()

{

scanf("%d",&n);

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

{

scanf("%d",&sequence[i]);

}

}

int lis()

{

int i;

I[0] = - inf;

for( i = 1; i<= n;i++)

{

I[i] = inf;

}

int lislength = 0;

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

{

int low , high , mid;

low = 0;

high = lislength;

while(low<=high)

{

mid = (high + low)/ 2;

if(I[mid]<sequence[i])

{

low = mid + 1;

}

else

{

high = mid - 1;

}

}

I[low] = sequence[i];

L[i] = low;

if(lislength< low)

lislength = low;

}

return lislength;

}

int lisarry[100];

void findsequence( int maxlength )

{

int i ,j ;

i = 0;

/\*for( j = 1; i< n;i++ )

{

if(L[j] > L[i])

{

i = j;

}

}

int top = L[i] - 1;\*/

int top;

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

{

if(L[i] == maxlength)

{

top = L[i] -1;

break;

}

}

cout<<top<<endl;

lisarry[top] = sequence[i];

top--;

for( j = i-1; j>=0 ;j-- )

{

if(sequence[j]<sequence[i]&&L[j]==L[i] - 1 )

{

i = j;

lisarry[top] = sequence[j];

top--;

}

}

for(i = 0 ;i<maxlength ; i++)

{

if(i)

printf(" ");

printf("%d",lisarry[i]);

}

cout<<endl;

}

int main()

{

takeInput();

int mx= lis();

cout<<mx<<endl;

findsequence(mx);

return 0;

}

LCS:

#include<bits/stdc++.h>

using namespace std;

string str1, str2, pre[101][101];

string lexi( string a , string b)

{

cout<<a<<" "<<b<<endl;

if(a.size() > b.size())

return a;

else if(b.size() > a.size())

return b;

else

{

if(a.compare(b)<= 0)

return a;

else

return b;

}

}

int dp[101][101];

string lcs(int i, int j)

{

if(i==str1.size()||j==str2.size())

{

return "";

}

if(dp[i][j]== 1)

return pre[i][j];

else

{

if(str1[i] == str2[j])

{

pre[i][j].push\_back(str2[j]);

pre[i][j] += lcs(i+1 , j+1);

dp[i][j] = 1;

return pre[i][j];

}

else

{

string x = lcs(i+1 , j);

string y = lcs(i , j+1);

if(x.size()>y.size())

pre[i][j] = x;

else if( y.size() > x.size())

pre[i][j] = y;

else

{

if(x.compare(y)<0)

pre[i][j] = x;

else

{

pre[i][j] = y;

}

}

dp[i][j] = 1;

return pre[i][j];

}

}

}

/\*string ans, smallest , large= "zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz";

void print(int i , int j)

{

if(i== str1.size()||j==str2.size())

{

if(ans<smallest)

smallest = ans;

return;

}

if(str1[i]==str2[j])

{

ans+=str1[i];

print(i+1 , j+1);

ans.erase(ans.end() - 1);

}

else

{

if(dp[i+1][j]<dp[i][j+1]) print(i, j+1);

else if(dp[i][j+1]<dp[i+1][j])

print(i+1 ,j);

else

{

print(i+1 , j);

print(i, j+1);

}

}

}

\*/

int main()

{

int tc, cs= 0;

cin>>tc;

getchar();

while(tc--)

{

cs++;

cin>>str1>>str2;

//cout<<str1<<" "<<str2<<endl;

memset(dp, -1 , sizeof(dp));

// ans.clear();

//smallest = large;

//cout<<

for(int i = 0 ; i < 101; i++)

{

for(int j = 0 ; j < 101 ; j++)

{

pre[i][j].clear();

}

}

string res = lcs(0,0);

//<<endl;

//print(0,0);

//cout<<smallest<<endl;

//cout<<pre[0][0].size() <<endl;

if(pre[0][0].size())

cout<<"Case "<<cs<<": "<<res<<endl;

else

cout<<"Case "<<cs<<": :("<<endl;

}

return 0;

}

EDIT DISTANCE:

#include<bits/stdc++.h>

using namespace std;

string A , B;

int dparry[2010][2010];

int dp( int i , int j)

{

if(i== A.size())

return B.size() - j;

if( j == B.size())

return A.size() - i;

if(dparry[i][j]!=-1)

return dparry[i][j];

else

{

if(A[i]==B[j])

return dparry[i][j] = dp(i+1, j+1);

else

{

return dparry[i][j] = min ((1+ dp(i+1, j)), min((1+ dp(i, j+1)), (1+dp(i+1, j+1))));

}

}

}

int main()

{

int tc;

cin>>tc;

while(tc--)

{

memset( dparry , -1, sizeof(dparry));

cin>> A>> B;

int res = dp(0,0);

cout<<res<<endl;

}

return 0;

}

DIGIT DP:

#include<bits/stdc++.h>

using namespace std;

string ToString(int n)

{

string s;

while(n!=0)

{

s+=(n%10)+'0';

n/=10;

}

reverse(s.begin(),s.end());

//cout<<s<<endl;

return s;

}

bool primeChecking( int n )

{

//cout<<n<<endl;

bool flag = true;

int lm = sqrt(n);

if(n<2)

return false;

for( int i = 2; i<= lm ; i++)

{

if(n%i==0)

{

flag= false;

break;

}

}

return flag;

}

int k;

string uplimit, dwlimit,str;

int dp[12][2][85][85];

int digitDp(string &s ,int odd , int even, int index , bool smaller )

{

//cout<<str<<endl;

// cout<<index<<endl;

if(index == s.length())

{

/\*int odd = 0, even = 0;

for(int i = 0 ; i < str.length() ;i++)

{

if(i%2)

even+=(str[i] - '0');

else

odd += ( str[i] - '0');

}\*/

//cout<<odd<<" "<<even<<endl;

if(primeChecking(even - odd))

{

//cout<<abs(odd-even)<<endl;

return 1;

}

else

return 0;

}

if(dp[index][smaller][odd][even]!=-1)

{

return dp[index][smaller][odd][even];

}

else

{

int limit = 9;

if(smaller)

{

limit = s[index] - '0';

}

int counter =0;

for(int i = 0 ; i <= limit ; i++)

{

// cout<<limit<<" "<<s[index]<<endl;

bool ns;

if(i<s[index]-'0')

{

ns = 0;

}

else

{

ns = smaller;

}

//str[index] = (i+'0');

if(s.length()%2==1 && index%2==0 || s.length()%2==0 &&index%2==1 )

counter += digitDp(s, odd + i , even, index+1 , ns );

else

counter += digitDp(s, odd , even + i, index+1 , ns );

}

//cout<<counter<<endl;

return dp[index][smaller][odd][even] = counter;

}

}

int main()

{

int tc;

cin>>tc;

while(tc--)

{

int a ,b;

cin>>a>>b;

uplimit = ToString(b);

dwlimit = ToString(a-1);

// str.assign(uplimit.size());

memset(dp, -1 , sizeof(dp));

//cout<<uplimit.length()<<" "<<dwlimit.length()<<endl;

int res = digitDp(uplimit ,0,0 ,0 , 1);

//cout<<res<<endl;

memset(dp , -1, sizeof(dp));

res -= digitDp(dwlimit,0,0, 0 ,1);

cout<<res<<endl;

}

return 0;

}

BITMASK:

KNAPSACK:

#include<bits/stdc++.h>

using namespace std;

int s , n;

struct knpsk{

int sz, value;

} ar[2001];

int sack[2001][2001];

int knapsack( int amount , int pos)

{

if(amount < 0)

return -99999999;

if(amount==0|| pos >= n)

return 0;

if(sack[amount][pos]==-1)

{

return sack[amount][pos] = max(knapsack(amount , pos+1 ) , ar[pos].value + knapsack(amount - ar[pos].sz , pos+1));

}

else

return sack[amount][pos];

}

int main()

{

//int s, n ;

cin>>s>>n;

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

cin>> ar[i].sz >> ar[i].value;

memset(sack , -1 , sizeof(sack));

int res = knapsack(s, 0);

cout<<res<<endl;

return 0;

}

EXACT CHANGE:

#include<bits/stdc++.h>

using namespace std;

int pymnt,coin,coin\_value[10010];

#define inf 99999999

struct change{

int amount;

int count;

};

change mini( change a , change b)

{

if(a.amount ==b.amount)

{

if(a.count<b.count)

return a;

else

return b;

}

else if( a.amount < b.amount)

return a;

else

return b;

}

change add( change a, change b)

{

a.amount = a.amount + b.amount;

a.count = a.count + b.count;

return a;

}

change bill[110][10010];

change dp(int cnt, int val )

{

if(val >= pymnt )

{

change q;

q.amount = 0;

q.count = 0;

return q;

}

if(cnt>=coin)

{

change r;

r.amount = inf;

r.count = inf;

return r;

}

if(bill[cnt][val].amount == -1 && bill[cnt][val].count == -1)

{change p ;

p.amount = coin\_value[cnt];

p.count = 1;

return bill[cnt][val] = mini ( add(p, dp(cnt+1, val + coin\_value[cnt])), dp(cnt+1, val));

}

else

return bill[cnt][val];

}

int main()

{

int tc;

scanf("%d",&tc);

while(tc--)

{

memset(bill, -1, sizeof(bill));

scanf("%d",&pymnt);

scanf("%d",&coin);

int j;

for(j=0;j<coin;j++)

scanf("%d",&coin\_value[j]);

change res = dp(0,0);

cout<<res.amount<<" "<<res.count<<endl;

}

return 0;

}

**String Matching Algorithm:**

**KVM:**

const int mxn = 100000;

int pi[mxn];

string pattern , givenString;

int patternLength ,givenStringLength , patternCount;

void buildPatternArray()

{

int j = 0;

pi[0] = 0;

for(int i = 1; i < patternLength; i++)

{

while(j>0 && pattern[j]!=pattern[i])

{

j = pi[j-1];

}

if(pattern[j]==pattern[i]){

pi[i] = j+1;

j++;

}

else{

pi[i]=0;

j = 0;

}

}

}

void patternMatching()

{

int j = 0;

for(int i = 0; i < givenStringLength; i++)

{

while( j > 0 && pattern[j]!=givenString[i])

{

j = pi[j-1];

}

if(j+1 == patternLength)

{

patternCount ++;

j = pi[j];

}

else

{

if(pattern[j]==givenString[i])

j++;

}

cout<<j<<" "<<i<<endl;

}

}