SCC

#define maxnode 1000

vector<int>gr[maxnode],disc(maxnode),low(maxnode),scc(maxnode),stk(maxnode);///all nodes in an scc will be marked similar value;

int t=0,sccno=0;

stack<int>s;

void dfs\_scc(int p,int u)

{

disc[u]=low[u]=++t;

s.push(u);

stk[u]=1;

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

{

int v=gr[u][i];

if(disc[v]==-1)

{

dfs\_scc(u,v);

low[u]=min(low[u],low[v]);

}

else if(stk[v]==1) low[u]=min(low[u],disc[v]);///if back-edge

}

if(low[u]==disc[u])

{

++sccno;

while(1)

{

int x=s.top();

s.pop();

scc[x]=sccno;

stk[u]=2;///cross-edge now

if(x==u) break;

}

}

}

void SCC(int n)

{

/\*

Do necessary init and clear

\*/

for(int i=0;i<n+3;i++)

{

scc[i]=stk[i]=low[i]=0;

disc[i]=-1;

}

sccno=0;

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

{

t=0;

if(disc[i]==-1) dfs\_scc(-1,i);

}

}

int main()

{

int n,m;

cin>>n>>m;

while(m--)

{

int x,y;

cin>>x>>y;

gr[x].pb(y);

}

SCC(n);

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

{

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

{

if(scc[j]==i) cout<<" "<<j;

}

cout<<endl;

}

}

Articulation point

#define maxnode 10000+7

vector<int>gr[maxnode];

vector<int>arti(maxnode),low(maxnode),disc(maxnode,-1);///true if is an articulation point

int t=0;

void dfs(int p,int u)

{

disc[u]=low[u]=++t;

int child=0;

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

{

int v=gr[u][i];

if(v==p) continue;

if(disc[v]==-1)

{

dfs(u,v);

low[u]=min(low[u],low[v]);

if(p!=-1 && disc[u]<=low[v]) arti[u]=1;

child++;

}

else

{

low[u]=min(low[u],disc[v]);

}

}

if(p==-1 && child>1) arti[u]=1;

}

void ArticulationPoint(int n)

{

/\*

Do necessary init and clear

\*/

for(int i=0;i<n+3;i++)

{

arti[i]=low[i]=0;

disc[i]=-1;

}

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

{

t=0;

if(disc[i]==-1) dfs(-1,i);

}

}

int main()

{

int n,m;

cin>>n>>m;

/\*

Clear Graph

\*/

while(m--)

{

int x,y;

cin>>x>>y;

gr[x].pb(y);

gr[y].pb(x);

}

ArticulationPoint(n);

for(int i=0;i<n;i++) if(arti[i]) cout<<" "<<i;

return 0;

}

Articulation Bridge

#define maxnode 1000

vector<int>gr[maxnode];

vector<int>disc(maxnode,-1),low(maxnode);

vector<pii>bridge;

int t=0;

void dfs(int p,int u)

{

disc[u]=low[u]=++t;

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

{

int v=gr[u][i];

if(v==p) continue;

else if(disc[v]==-1)

{

dfs(u,v);

low[u]=min(low[u],low[v]);

if(disc[u]<low[v]) bridge.pb(pii(u,v));

}

else low[u]=min(low[u],disc[v]);

}

}

void ArticulationBridge(int n)

{

/\*

Do necessary init and clear

\*/

for(int i=0; i<n+3; i++)

{

low[i]=0;

disc[i]=-1;

}

bridge.clear();

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

{

t=0;

if(disc[i]==-1) dfs(-1,i);

}

}

int main()

{

int n,m;

cin>>n>>m;

/\*

Clear Graph

\*/

while(m--)

{

int x,y;

cin>>x>>y;

gr[x].pb(y);

gr[y].pb(x);

}

ArticulationBridge(n);

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

{

cout<<bridge[i].fs<<" "<<bridge[i].sc<<endl;

}

return 0;

}

Table Knapsack

int main()

{

int n;

cin>>n;

vector<int>cost(n+1),weight(n+1);

for(int i=1;i<=n;i++) cin>>cost[i];

for(int i=1;i<=n;i++) cin>>weight[i];

int cap;

cin>>cap;

int tbl[n+1][cap+1];

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

{

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

{

if(i==0 || j==0) tbl[i][j]=0;

else

{

if(j-weight[i]>-1) tbl[i][j]=max(tbl[i-1][j],cost[i]+tbl[i-1][j-weight[i]]);

else tbl[i][j]=tbl[i-1][j];

}

}

}

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

{

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

{

cout<<" "<<tbl[i][j];

}

cout<<endl;

}

cout<<tbl[n][cap]<<endl;

return 0;

}

Sum of all subarray

arr[] = [1, 2, 3], n = 3

All subarrays : [1], [1, 2], [1, 2, 3],

[2], [2, 3], [3]

here first element 'arr[0]' appears 3 times

second element 'arr[1]' appears 4 times

third element 'arr[2]' appears 3 times

Every element arr[i] appears in two types of subsets:

i) In sybarrays beginning with arr[i]. There are

(n-i) such subsets. For example [2] appears

in [2] and [2, 3].

ii) In (n-i)\*i subarrays where this element is not

first element. For example [2] appears in

[1, 2] and [1, 2, 3].

Total of above (i) and (ii) = (n-i) + (n-i)\*i

= (n-i)(i+1)

For arr[] = {1, 2, 3}, sum of subarrays is:

arr[0] \* ( 0 + 1 ) \* ( 3 - 0 ) +

arr[1] \* ( 1 + 1 ) \* ( 3 - 1 ) +

arr[2] \* ( 2 + 1 ) \* ( 3 - 2 )

= 1\*3 + 2\*4 + 3\*3

= 20

///O(n)

Stirling Number Of Second Kind

// Returns count of different partitions of n

// elements in k subsets

int countP(int n, int k)

{

// Table to store results of subproblems

int dp[n+1][k+1];

// Base cases

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

dp[i][0] = 0;

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

dp[0][k] = 0;

// Fill rest of the entries in dp[][]

// in bottom up manner

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

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

if (j == 1 || i == j)

dp[i][j] = 1;

else

dp[i][j] = j\*dp[i-1][j] + dp[i-1][j-1];

return dp[n][k];

}

// Driver program

int main()

{

cout << countP(5, 2);

return 0;

}

Second Shortest Path-1

///loj 1099

vector<pair<int,int> >gr[5002];

int d[2][5002];

int flag[2][5002];

void dijkstra(int s)

{

priority\_queue<pair<int,pair<int,int> > ,vector< pair<int,pair<int,int> > > ,greater< pair<int,pair<int,int> > > >q; ///dis,t=0(shortest),1(second shortest),u

d[0][s]=0;

q.push({0,{0,s}});

while(!q.empty())

{

int t=q.top().sc.fs;

int u=q.top().sc.sc;

q.pop();

if(flag[t][u]) continue;

flag[t][u]=1;

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

{

int v=gr[u][i].fs;

int dd=gr[u][i].sc;

if(d[0][v]>d[t][u]+dd)

{

d[1][v]=d[0][v];

d[0][v]=d[t][u]+dd;

q.push({d[1][v],{1,v}});

q.push({d[0][v],{0,v}});

}

else if(d[0][v]<d[t][u]+dd && d[1][v]>d[t][u]+dd)///need to give d[0][v]<d[t][u]+dd becasue the second shortest cant be equal to shortest,if can be equal dont give this

{

d[1][v]=d[t][u]+dd;

q.push({d[1][v],{1,v}});

}

}

}

}

int main()

{

int t,cas=1;

cin>>t;

while(t--)

{

int n,r;

cin>>n>>r;

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

{

gr[i].clear();

d[0][i]=d[1][i]=1<<30;

flag[0][i]=flag[1][i]=0;

}

while(r--)

{

int x,y,w;

cin>>x>>y>>w;

gr[x].pb({y,w});

gr[y].pb({x,w});

}

dijkstra(1);

cout<<"Case "<<cas++<<": "<<d[1][n]<<endl;

}

return 0;

}

Second Shortest Path-2

vector<pii>gr[5002];

int d1[5002],d2[5002];

void dijkstra(int s)

{

d1[s]=0;

priority\_queue<pii,vector<pii>,greater<pii> >q;

q.push(pii(0,s));

while(!q.empty())

{

int u=q.top().sc;

q.pop();

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

{

int v=gr[u][i].fs;

int d=gr[u][i].sc;

if(d1[u]+d<d1[v])

{

int temp=d1[v];

d1[v]=d1[u]+d;

d2[v]=min(d2[v],min(temp,min(d2[u]+d,d1[u]+3\*d)));

q.push(pii(d1[v]+d2[v],v));

}

else if(d1[u]+d>d1[v] && d1[u]+d<d2[v]) d2[v]=d1[u]+d;

else d2[v]=min(d2[v],min(d1[u]+3\*d,d2[u]+d));

}

}

}

int main()

{

int t,cas=1;

cin>>t;

while(t--)

{

int n,m;

cin>>n>>m;

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

{

gr[i].clear();

d1[i]=d2[i]=30000000; ///not giving 1<<30 cuz we'll be adding stuffs to it

}

while(m--)

{

int x,y,w;

cin>>x>>y>>w;

gr[x].pb(pii(y,w));

gr[y].pb(pii(x,w));

}

dijkstra(1);

cout<<"Case "<<cas++<<": "<<min(d2[n],d2[1]+d1[n])<<endl;

///d2[1]+d1[n] case is checked again because when working with

///node 1, its d2[1] is not yet found and remains inf

}

return 0;

}

Trie

#define mx 1000 ///no. of nodes=number of strings\*max string size

struct trietree

{

int tr[mx][26];

int leaf[mx];

int nodenumber;

int root=0;

trietree()

{

memset(tr,-1,sizeof tr);

memset(leaf,0,sizeof leaf);

nodenumber=0;

}

void insert\_string(string s)

{

int node=root;

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

{

if(tr[node][s[i]-'a']==-1)

{

tr[node][s[i]-'a']=++nodenumber;

}

node=tr[node][s[i]-'a'];

}

leaf[node]++;

}

int look(string s)

{

int node=root;

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

{

if(tr[node][s[i]-'a']==-1) return 0;

node=tr[node][s[i]-'a'];

}

if(leaf[node]>0) return 1;

}

};

QTREE-HLD

///

#define node 10003

#define ln 14

using namespace std;

vector<pair<pii,int> >gr[node]; ///v,w,indexxOfEdge

int table[node][ln],level[node],siz[node],indexx[node];

int chainNo[node],chainHead[node],chain,posBase[node],ptr,base[node];

int tree[4\*node];

void dfs(int u)

{

siz[u]=1;

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

{

int v=gr[u][i].fs.fs;

int w=gr[u][i].fs.sc;

int in=gr[u][i].sc;

if(level[v]==-1)

{

table[v][0]=u;

level[v]=level[u]+1;

indexx[in]=v;

dfs(v);

siz[u]+=siz[v];

}

}

}

int lca\_query(int x,int y)

{

if(level[x]<level[y]) swap(x,y);

int lg;

for(lg=1; (1<<lg)<=level[x]; lg++);

lg--;

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

{

if(level[x]-(1<<i)>=level[y]) x=table[x][i];

}

if(x==y) return x;

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

{

if(table[x][i]!=table[y][i])

{

x=table[x][i];

y=table[y][i];

}

}

return table[x][0];

}

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

{

if(chainHead[chain]==-1)

{

chainHead[chain]=u;

}

ptr++;

chainNo[u]=chain;

posBase[u]=ptr;

base[ptr]=cost;

int bc=-1,mx=-1,cs=-1;

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

{

int v=gr[u][i].fs.fs;

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

if(v!=prev && siz[v]>mx) mx=siz[v],bc=v,cs=gr[u][i].fs.sc;

}

if(bc!=-1) hld(bc,u,cs);

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

{

int v=gr[u][i].fs.fs;

if(v!=prev && v!=bc)

{

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

chain++;

hld(v,u,gr[u][i].fs.sc);

}

}

return;

}

void build(int nodes,int b,int e)

{

if(b==e) tree[nodes]=base[b];

else

{

int m=(b+e)/2;

build(nodes\*2,b,m);

build(nodes\*2+1,m+1,e);

tree[nodes]=max(tree[nodes\*2],tree[nodes\*2+1]);

}

}

int seg\_query(int nodes,int b,int e,int i,int j)

{

if(b>=i && j>=e) return tree[nodes];

else if(i>e || j<b) return -1;

int m=(b+e)/2;

return max( seg\_query(nodes\*2,b,m,i,j), seg\_query(nodes\*2+1,m+1,e,i,j) );

}

void update(int nodes,int b,int e,int i,int val)

{

if(b==e && e==i) {tree[nodes]=val;return;}

int m=(b+e)/2;

if(i<=m) update(nodes\*2,b,m,i,val);

else update(nodes\*2+1,m+1,e,i,val);

tree[nodes]=max(tree[nodes\*2],tree[nodes\*2+1]);

return;

}

int queryUp(int x,int v)///v=lca

{

if(x==v) return 0;

int mx=-1;

while(1)

{

if(chainNo[x]==chainNo[v])

{

if(x==v) return mx;

return max(mx,seg\_query(1,1,ptr,posBase[v]+1,posBase[x]));

}

mx=max(mx,seg\_query(1,1,ptr,posBase[ chainHead[ chainNo[x] ] ],posBase[x]));

x=table[chainHead[ chainNo[x] ] ][0];

}

return mx;

}

int query(int x,int y)

{

int v= lca\_query(x,y);

return max(queryUp(x,v),queryUp(y,v));

}

int main()

{

int t;

scanf("%d",&t);

while(t--)

{

int n;

scanf("%d",&n);

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

{

gr[i].clear();

chainHead[i]=-1;

level[i]=-1;

for(int j=0; j<ln; j++) table[i][j]=-1;

}

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

{

int x,y,w;

scanf("%d%d%d",&x,&y,&w);

gr[x].pb({{y,w},i});

gr[y].pb({{x,w},i});

}

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

{

cout<<i<<":";

for(int j=0;j<gr[i].size();j++)

{

cout<<" "<<gr[i][j].fs.fs;

}

cout<<endl;

}\*/

level[1]=0;

dfs(1);

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

{

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

{

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

{

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

}

}

}

ptr=-1;

chain=1;

hld(1,0,0);

//ptr--;

//cout<<"ptr = "<<ptr<<endl;

//for(int i=0;i<6;i++) cout<<" "<<base[i]; cout<<endl;

//for(int i=1;i<=n;i++) cout<<" "<<posBase[i]; cout<<endl;

//for(int i=1;i<=n;i++) cout<<" "<<chainNo[i]; cout<<endl;

//cout<<"no of chains "<<chain<<endl;

build(1,1,ptr);

char s[10];

while(scanf("%s",s))

{

if(strcmp(s,"DONE")==0) break;

if(strcmp(s,"QUERY")==0)

{

int x,y;

scanf("%d%d",&x,&y);

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

}

else

{

int x,val;

scanf("%d%d",&x,&val);

x=posBase[ indexx[x] ];

update(1,1,ptr,x,val);

}

}

}

return 0;

}

Number of palindrome from a to b

ll dp[20][20][20];

vector<int>v;

int siz,hsiz;

ll pal(int ind,int leftmost\_lo,int leftmost\_hi)

{

if(ind<=hsiz)

{

if(leftmost\_lo>=leftmost\_hi) return 1;

else return 0;

}

if(dp[ind][leftmost\_lo][leftmost\_hi]!=-1) return dp[ind][leftmost\_lo][leftmost\_hi];

ll ret=0;

int start;

if(ind==siz) start=1;

else start=0;

int n=siz-ind+1;

for(int i=start; i<=9; i++)

{

int newleftmost\_lo=leftmost\_lo;

int newleftmost\_hi=leftmost\_hi;

if(i<v[ind] && ind>newleftmost\_lo) newleftmost\_lo=ind;

if(i<v[n] && n>newleftmost\_lo) newleftmost\_lo=n;

if(i>v[ind] && ind>newleftmost\_hi) newleftmost\_hi=ind;

if(i>v[n] && n>newleftmost\_hi) newleftmost\_hi=n;

ret+=pal(ind-1,newleftmost\_lo,newleftmost\_hi);

}

return dp[ind][leftmost\_lo][leftmost\_hi]=ret;

}

ll ara[20];

int main()

{

ara[0]=1;

ll add=9;

for(int i=1; i<=17; i+=2)

{

ara[i]=ara[i-1]+add;

ara[i+1]=ara[i]+add;

add\*=10;

}

int t,cas=1;

cin>>t;

while(t--)

{

ll a,b;

cin>>a>>b;

v.clear();

if(a>b) swap(a,b);

a--;

ll ans1=0;

if(a>=0)

{

v.pb(0);

if(a==0) v.pb(0);

else

{

while(a)

{

v.pb(a%10);

a/=10;

}

}

memset(dp,-1,sizeof dp);

siz=v.size()-1;

hsiz=siz/2;

ans1=pal(siz,0,0);

v.clear();

ans1+=ara[siz-1];

}

v.pb(0);

if(b==0) v.pb(0);

else

{

while(b)

{

v.pb(b%10);

b/=10;

}

}

siz=v.size()-1;

hsiz=siz/2;

memset(dp,-1,sizeof dp);

ll ans2=pal(siz,0,0);

ans2+=ara[siz-1];

cout<<"Case "<<cas++<<": "<<ans2-ans1<<"\n";

}

return 0;

}

MOs

// Program to compute sum of ranges for different range

// queries

///O( (m+n)\*sqrt(n) )

//It cannot work for problems where we have update operations also mixed with sum queries.

#include <bits/stdc++.h>

using namespace std;

// Variable to represent block size. This is made global

// so compare() of sort can use it.

int block;

// Structure to represent a query range

struct Query

{

int L, R;

};

// Function used to sort all queries so that all queries

// of same block are arranged together and within a block,

// queries are sorted in increasing order of R values.

bool compare(Query x, Query y)

{

// Different blocks, sort by block.

if (x.L/block != y.L/block)

return x.L/block < y.L/block;

// Same block, sort by R value

return x.R < y.R;

}

// Prints sum of all query ranges. m is number of queries

// n is size of array a[].

void queryResults(int a[], int n, Query q[], int m)

{

// Find block size

block = (int)sqrt(n);

// Sort all queries so that queries of same blocks

// are arranged together.

sort(q, q + m, compare);

// Initialize current L, current R and current sum

int currL = 0, currR = 0;

int currSum = 0;

// Traverse through all queries

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

{

// L and R values of current range

int L = q[i].L, R = q[i].R;

// Remove extra elements of previous range. For

// example if previous range is [0, 3] and current

// range is [2, 5], then a[0] and a[1] are subtracted

while (currL < L)

{

currSum -= a[currL];

currL++;

}

// Add Elements of current Range

while (currL > L)

{

currSum += a[currL-1];

currL--;

}

while (currR <= R)

{

currSum += a[currR];

currR++;

}

// Remove elements of previous range. For example

// when previous range is [0, 10] and current range

// is [3, 8], then a[9] and a[10] are subtracted

while (currR > R+1)

{

currSum -= a[currR-1];

currR--;

}

// Print sum of current range

cout << "Sum of [" << L << ", " << R

<< "] is " << currSum << endl;

}

}

// Driver program

int main()

{

int a[] = {1, 1, 2, 1, 3, 4, 5, 2, 8};

int n = sizeof(a)/sizeof(a[0]);

Query q[] = {{0, 4}, {1, 3}, {2, 4}};

int m = sizeof(q)/sizeof(q[0]);

queryResults(a, n, q, m);

return 0;

}

Matrix Expo

///Did not Deal with doing mod; for that put mod in the multiply function;all replace all int with long long

struct Matrix

{

vector<vector<int> >mat;

int row,col;

Matrix() {}

Matrix (const Matrix &a)

{

mat=a.mat;

row=a.row;

col=a.col;

}

Matrix(int row,int col):row(row),col(col)

{

mat.resize(row);

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

{

mat[i].resize(col);

}

}

Matrix(vector<vector<int> >val): mat(val),row(val.size()),col(val[0].size()) {}

void takeIn(int i,int j,int val)

{

mat[i][j]=val;

}

Matrix operator\*(const Matrix &a) const

{

int m=a.col;

vector<vector<int> >res(row,vector<int>(m));

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

{

for(int k=0; k<m; k++)

{

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

{

res[i][k]+=mat[i][j]\*a.mat[j][k];

}

}

}

return Matrix(res);

}

Matrix identity(int n)

{

Matrix a;

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

{

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

{

if(i==j) a.mat[i][j]=1;

}

}

return a;

}

Matrix expo(int power)

{

if(power==1) return mat;

if(power==0) return identity(row);

Matrix x=expo(power/2);

x=x\*x;

if(power&1) x=x\*mat;

return x;

}

void printMat()

{

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

{

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

{

cout<<mat[i][j]<<" ";

}

cout<<endl;

}

}

void WholeInput()

{

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

{

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

{

cin>>mat[i][j];

}

}

}

};

int main()

{

/\*

///MULTIPLYING TWO MATRICES

int n,m,k;

cin>>n>>m>>k;

Matrix a(n,m),b(m,k);

a.WholeInput();

b.WholeInput();

Matrix res=a\*b;

res.printMat();

///END MULTIPLY

\*/

///FINDING THE N-TH FIBONACCCI

Matrix M(2,2);

M.takeIn(0,0,1);

M.takeIn(0,1,1);

M.takeIn(1,0,1);

M.takeIn(1,1,0);

Matrix F(2,1);

F.takeIn(0,0,1);

F.takeIn(1,0,1);

int nth;

while(cin>>nth)

{

Matrix r(M.expo(nth-1)\*F);

cout<<r.mat[0][0]<<endl;

}

return 0;

}

Minimum Vertex Cover

/\*

এটি একটি NP-hard প্রবলেম,অর্থাৎ এই প্রবলেমের কোনো পলিনমিয়াল টাইম সলিউশন নেই।

তবে গ্রাফটি যদি Tree হয় অর্থাত n-1 টা edge থাকে আর কোনো সাইকেল না থাকে তাহলে

ডাইনামিক প্রোগ্রামিং বা ম্যাক্স ফ্লো/বাইপারটাইট ম্যাচিং এর সাহায্যে প্রবলেমটি সলভ করা সম্ভব।

\*/

vector<int>g[100005];

int dp[100005][2];

int minvert(int p,int u,int taken)

{

if(dp[u][taken]!=-1) return dp[u][taken];

int mx=0;

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

{

int v=g[u][i];

if(v==p) continue;

if(taken==0)

{

mx+=minvert(u,v,1);

}

else

{

mx+=min(minvert(u,v,1),minvert(u,v,0));

}

}

return dp[u][taken]=mx+taken;

}

int main()

{

int n,x,y;

cin>>n;

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

for(int i=0;i<n-1;i++)

{

cin>>x>>y;

g[x].pb(y);

g[y].pb(x);

}

cout<<min(minvert(-1,1,0),minvert(-1,1,1))<<endl;

return 0;

}

BPM

// M is number of applicants and N is number of jobs

#define M 6

#define N 6

// A DFS based recursive function that returns true if a

// matching for vertex u is possible

bool bpm(bool bpGraph[M][N], int u, bool seen[], int matchR[])

{

// Try every job one by one

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

{

// If applicant u is interested in job v and v is

// not visited

if (bpGraph[u][v] && !seen[v])

{

seen[v] = true; // Mark v as visited

// If job 'v' is not assigned to an applicant OR

// previously assigned applicant for job v (which is matchR[v])

// has an alternate job available.

// Since v is marked as visited in the above line, matchR[v]

// in the following recursive call will not get job 'v' again

if (matchR[v] < 0 || bpm(bpGraph, matchR[v], seen, matchR))

{

matchR[v] = u;

return true;

}

}

}

return false;

}

// Returns maximum number of matching from M to N

int maxBPM(bool bpGraph[M][N])

{

// An array to keep track of the applicants assigned to

// jobs. The value of matchR[i] is the applicant number

// assigned to job i, the value -1 indicates nobody is

// assigned.

int matchR[N];

// Initially all jobs are available

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

int result = 0; // Count of jobs assigned to applicants

for (int u = 0; u < M; u++)

{

// Mark all jobs as not seen for next applicant.

bool seen[N];

memset(seen, 0, sizeof(seen));

// Find if the applicant 'u' can get a job

if (bpm(bpGraph, u, seen, matchR))

result++;

}

return result;

}

int main()

{// Let us create a bpGraph shown in the above example

bool bpGraph[M][N] = { {0, 1, 1, 0, 0, 0},

{1, 0, 0, 1, 0, 0},

{0, 0, 1, 0, 0, 0},

{0, 0, 1, 1, 0, 0},

{0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 1} };

cout << "Maximum number of applicants that can get job is "<< maxBPM(bpGraph);

return 0;

}

2D kadane

int kadane(vector<int>v, int\* start, int\* finish)///address of start and finish

{

int sum=0,maxsum=-1<<30,s;

\*finish=-1;

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

{

sum+=v[i];

if(sum<0)

{

sum=0;

s=i+1;

}

else if(sum>maxsum)

{

\*start=s;

\*finish=i;

maxsum=sum;

}

}

if(\*finish!=-1) return maxsum;

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

{

if(maxsum<v[i])

{

maxsum=v[i];

\*start=\*finish=i;

}

}

return maxsum;

}

int kadane\_2D(vector< vector<int> >v , int\* top,int\* left,int\* right, int\* bottom)

{

int MX=-1<<30;

///select two columns by n^2

///for each row in the range of the selected column find cummulative sum

///run kadane in that cummulative sum array to find the rows of either sides

int n=v.size();///row

int m=v[0].size();///column

///start and end column

for(int stcol=0;stcol<m;stcol++)

{

vector<int>temp(n);

for(int encol=stcol;encol<m;encol++)

{

for(int i=0;i<n;i++) temp[i]+=v[i][encol];

int start,finish;

int mx=kadane(temp,&start,&finish);

if(mx>MX)

{

MX=mx;

\*left=stcol;

\*right=encol;

\*top=start;

\*bottom=finish;

}

}

}

return MX;

}

int main()

{

/\*int M[4][5] = {{1, 2, -1, -4, -20},

{-8, -3, 4, 2, 1},

{3, 8, 10, 1, 3},

{-4, -1, 1, 7, -6}

};\*/

//int M[4][5]={ -10,-20,-3,-4,-5,-10,-20,-38,-74,-85,-107,-20,-37,-48,-500,-100,-200,-30,-40,-50};

vector< vector<int> >v(4,vector<int>(5));

for(int i=0;i<4;i++)

{

for(int j=0;j<5;j++)

{

v[i][j]=M[i][j];

}

}

int t,l,r,b;

cout<<kadane\_2D(v,&t,&l,&r,&b)<<endl;

cout<<"top "<<t<<endl;

cout<<"left "<<l<<endl;

cout<<"right "<<r<<endl;

cout<<"bottom "<<b<<endl;

return 0;

}

LCS

int a,b;

string aa,bb;

int dp[1007][1007];

int lcs(int i,int j)

{

///worst-case comlexity O(2^n)

if(i==a || j==b) return 0;

if(dp[i][j]!=-1) return dp[i][j];

if(aa[i]==bb[j]) return dp[i][j]=1+lcs(i+1,j+1);

else return dp[i][j]=max(lcs(i+1,j),lcs(i,j+1));

}

void print(int i,int j)

{

if(i==a || j==b) return;

if(aa[i]==bb[j])

{

cout<<aa[i]<<" "<<i+1<<" "<<j+1<<endl;

print(i+1,j+1);

}

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

{

print(i+1,j);

}

else

{

print(i,j+1);

}

}

string s;

void printAll(int i,int j)

{

if(i==a || j==b)

{

cout<<s<<endl;

return;

}

if(aa[i]==bb[j])

{

s+=aa[i];

printAll(i+1,j+1);

s.pop\_back();

}

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

{

printAll(i+1,j);

printAll(i,j+1);

}

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

{

printAll(i+1,j);

}

else

{

printAll(i,j+1);

}

}

int table[1007][1007];

int tabledp(int n,int m)

{

///worst-case complexity O(n\*m)

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

{

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

{

if(i==0 || j==0) table[i][j]=0;

else if(aa[i-1]==bb[j-1]) table[i][j]=1+table[i-1][j-1];

else table[i][j]=max(table[i-1][j],table[i][j-1]);

}

}

return table[n][m];

}

void printtable()

{

int index=table[a][b];

char lcsprn[index+1 ];

lcsprn[index]=0;

int i=a,j=b;

while(i>0 && j>0)

{

if(aa[i-1]==bb[j-1])

{

lcsprn[index-1]=aa[i-1];

index--;

i--;

j--;

}

else if(table[i-1][j]>table[i][j-1])///to print all:

{

///if(table[i-1][j]==table[i][j-1])

///then we have to go both ways; how? I donno :3

i--;

}

else

{

j--;

}

}

puts(lcsprn);

}

int space\_optimised\_table\_lcs()

{

int n=a,m=b;

int optable[2][m+1];

int in;

memset(optable,0,sizeof optable);

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

{

in=i&1;///odd hoile 1

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

{

if(i==0 || j==0) optable[in][j]=0;

else if(aa[i-1]==bb[j-1])

{

optable[in][j]=1+optable[1-in][j-1];

}

else

{

optable[in][j]=max(optable[1-in][j],optable[in][j-1]);

}

}

}

return optable[in][m];

}

int main()

{

int n;

int cas=1;

cin>>n;

while(n--)

{

cin>>a>>aa>>b>>bb;///aa is of length a

///recursion

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

cout<<"LCS = "<<lcs(0,0)<<endl;

//print(0,0);

//printAll(0,0);

///table dp

/\*Note that table[i][j] contains length of LCS of

aa[0..i-1] and bb[0..j-1] \*/

cout<<"LCS = "<<tabledp(a,b)<<endl;

printtable();

cout<<"space optimised = "<<space\_optimised\_table\_lcs()<<endl;

}

return 0;

}

LIS

int lis(vector<int>v)

{

vector<int>temp;

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

{

int x=v[i];

///strictly increasing er khetre LOWER naile UPPER bound

int in=upper\_bound(temp.begin(),temp.end(),x)-temp.begin();

if(in==temp.size()) temp.pb(x);

else temp[in]=min(x,temp[in]);

}

return temp.size();

}

Bell Number

int bellNumber(int n)

{

int bell[n+1][n+1];

bell[0][0] = 1;

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

{

// Explicitly fill for j = 0

bell[i][0] = bell[i-1][i-1];

// Fill for remaining values of j

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

bell[i][j] = bell[i-1][j-1] + bell[i][j-1];

}

return bell[n][0];

}

All Possible Increasing Subsequence

using namespace std;

ll tree[400000];

ll query (int node,int b, int e,int i,int j)

{

if(i<=b and e<=j) return tree[node];

else if(i>e || j<b) return 0;

else

{

int m=(b+e)/2;

int lc=node\*2;

int rc=lc+1;

return query(lc,b,m,i,j)+query(rc,m+1,e,i,j);

}

}

void update (int node,int b, int e,int val,int i)

{

if(b==e && b==i) tree[node]=val;

else if(i>=b and i<=e)

{

int m=(b+e)/2;

int lc=node\*2;

int rc=lc+1;

update(lc,b,m,val,i);

update(rc,m+1,e,val,i);

tree[node]=tree[lc]+tree[rc];

}

}

bool cmp(pii a,pii b)

{

if(a.fs==b.fs) return a.sc>b.sc; ///(for strictly increasing)

else return a.fs<b.fs;

}

int main()

{

int t;

scanf("%d",&t);

int cas=1;

while(t--)

{

int n;

scanf("%d",&n);

memset(tree,0,sizeof tree);

vector<pii>ara(n+1);

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

{

scanf("%d",&ara[i].fs);

ara[i].sc=i;

}

ll ans=0;

ll a=0;

sort(ara.begin(),ara.end(),cmp);

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

{

a=query(1,1,n,1,ara[i].sc);

update(1,1,n,(a+1)%md,ara[i].sc);

}

printf("Case %d: %lld\n",cas++,tree[1]%md);

}

return 0;

}

BCC

#define maxnode 1000+7

vector<int>gr[maxnode],disc(maxnode,-1),low(maxnode);

stack<pii>s;

int t=0;

void dfs\_bcc(int p,int u)

{

disc[u]=low[u]=++t;

int child=0;

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

{

int v=gr[u][i];

if(v==p) continue;

else if(disc[v]==-1)

{

s.push(pii(u,v));

child++;

dfs\_bcc(u,v);

low[u]=min(low[u],low[v]);

if((p==-1 && child>1) || disc[u]<=low[v] )

{

cout<<"BCC : \n";

while(1)

{

int x=s.top().fs;

int y=s.top().sc;

cout<<x<< " "<<y<<endl;

s.pop();

if(x==u && y==v) break;

}

}

}

else if(low[u]>disc[v]) ///edge in stack

{

s.push(pii(u,v));

low[u]=min(low[u],disc[v]);

}

}

}

void BCC(int n)

{

/\*

Do required clearing and init

\*/

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

{

if(disc[i]==-1)

{

dfs\_bcc(-1,i);

if(!s.empty()) cout<<"BCC : "<<endl;

while(!s.empty())

{

cout<<s.top().fs<<" "<<s.top().sc<<endl;

s.pop();

}

}

}

}

int main()

{

int n,m;

cin>>n>>m;

while(m--)

{

int x,y;

cin>>x>>y;

gr[x].pb(y);

gr[y].pb(x);

}

BCC(n);

return 0;

}

BIT

int getParent(int ind)

{

return ind=ind - (ind & -ind);

}

int getNext(int ind)

{

return ind=ind + (ind & -ind);

}

#define maxn 1000+5

int n;

vector<int>ara(maxn),tree(maxn);

void update(int ind,int val)

{

ind++;

while(ind<n+1)

{

tree[ind]+=val;

ind=getNext(ind);

}

}

int getSum(int ind) ///cummulative sum from 0 to ind

{

ind++;

int sum=0;

while(ind>0)

{

sum+=tree[ind];

ind=getParent(ind);

}

return sum;

}

void buildTree()

{

///first tree is initialised to 0;

///do update opration for all nodes

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

{

update(i,ara[i]);

}

}

int main()

{

cin>>n;

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

{

cin>>ara[i];

}

buildTree();

cout<<getSum(8)<<" "<<getSum(4)<<endl;

update(3,7);

cout<<getSum(8)<<" "<<getSum(4)<<endl;

update(2,8);

cout<<getSum(8)<<" "<<getSum(4)<<endl;

return 0;

}

3 string LCS

int do\_lcs(string a,string b,string c)

{

int la=a.length();

int lb=b.length();

int lc=c.length();

int ara[la+1][lb+1][lb+1];

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

{

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

{

for(int k=0;k<=lc;k++)

{

if(i==0 || j==0 || k==0) ara[i][j][k]=0;

else if(a[i-1]==b[j-1]&& b[j-1]==c[k-1]) ara[i][j][k]=1+ara[i-1][j-1][k-1];

else ara[i][j][k]=max(ara[i-1][j][k],max(ara[i][j-1][k],ara[i][j][k-1]));

}

}

}

return ara[la][lb][lc];

}

Aho Corasik

#include<bits/stdc++.h>

#define pb push\_back

using namespace std;

///Max number of states = sum of length of patterns

#define MAXS 500

///Max characters

#define MAXC 26

///list of output strings in each node

///use int if u wanna keep indices of patterns in out

vector<string>out[MAXS];

///failure nodes

int f[MAXS];

int trie[MAXS][MAXC];

int buildTrie(vector<string>pat, int n)

{

/\*\* Building Trie \*/

for(int i=0;i<MAXS;i++) out[i].clear();

memset(trie,-1,sizeof trie);

///ROOT is 0

int states=1;

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

{

string str=pat[i];

int node=0;

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

{

int ch=str[j]-'a';

if(trie[node][ch]==-1) trie[node][ch]=states++;

node=trie[node][ch];

}

out[node].pb(str);

}

/\*\* Failure Function Building \*/

///adding a track back to root, for those children of root that dont exist

for(int i=0;i<MAXC;i++) if(trie[0][i]==-1) trie[0][i]=0;

memset(f, -1, sizeof f);

queue<int>q;

///All children of root fail to root.

for(int i=0;i<MAXC;i++) if(trie[0][i]!=0) f[trie[0][i]]=0,q.push(trie[0][i]);

while(!q.empty())

{

int node=q.front();

q.pop();

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

{

if(trie[node][i]!=-1)

{

int failure=f[node];

while(trie[failure][i]==-1)

{

failure=f[failure];

}

failure=trie[failure][i];

f[trie[node][i]]=failure;

for(int k=0;k<out[failure].size();k++)

{

out[trie[node][i]].pb(out[failure][k]);

}

q.push(trie[node][i]);

}

}

}

return states;

///no. of states in trie

}

void AhoCorasick(vector<string>pat, int n, string text)

{

///builds trie, failure and stores outputs

buildTrie(pat,n);

int curState=0;

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

{

int ch=text[i]-'a';

while(trie[curState][ch]==-1)

{

curState=f[curState];

}

curState=trie[curState][ch];

for(int j=0;j<out[curState].size();j++)

{

//if same word is in dictionary multiple times, will be printed multiple times

//also if word in text multiple times, also printed multiple times

//these are to be handled according to the problem

cout<<out[curState][j]<< " appears in text from "<<i-out[curState][j].size()+1<<"\n";

}

}

}

int main()

{

int n;

cin>>n;

vector<string>pat(n);

for(int i=0;i<n;i++) cin>>pat[i];

string text;

cin>>text;

AhoCorasick(pat,n,text);

return 0;

}

/\*

4

he she hers his

ahishers

\*/

/\*\*

I stored in out all the strings that end at a specific node

I could also: use bitset to mark those positions (index of strings) which are output at that position //bitset<MAXS>out[MAXS];

\*/

DSU on TREE

vector<int>gr[nd],col(nd),cnt(nd),siz(nd),st(nd),fn(nd),ver(nd);

vector<LL>ans(nd);

int t,n;

int mex=0;

LL ANS=0;

void dfssiz(int u,int p)

{

t++;

ver[t]=u;

st[u]=t;

siz[u]=1;

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

{

if(gr[u][i]!=p)

{

dfssiz(gr[u][i],u);

siz[u]+=siz[gr[u][i]];

}

}

fn[u]=t;

}

void dfs(int u,int p,int keep)

{

int mx=-1,bigchild=-1;

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

{

int v=gr[u][i];

if(v==p) continue;

if(mx<siz[v])

{

mx=siz[v];

bigchild=v;

}

}

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

{

int v=gr[u][i];

if(v==p || v==bigchild) continue;

dfs(v,u,0);

}

if(bigchild!=-1)dfs(bigchild,u,1);

int temp=mex;

cnt[col[u]]++;

mex=max(mex,cnt[col[u]]);

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

{

int v=gr[u][i];

if(v==p || v==bigchild) continue;

for(int j=st[v]; j<=fn[v]; j++)

{

cnt[ col[ ver[j] ] ]++;

mex=max(mex,cnt[ col[ ver[j] ] ]);

}

}

if(mex!=temp) ANS=0;

set<int>stk;

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

{

int v=gr[u][i];

if(v==p || v==bigchild) continue;

for(int j=st[v]; j<=fn[v]; j++)

{

if(cnt[ col[ ver[j] ] ]==mex) stk.insert(col[ ver[j] ]);

}

}

if(cnt[col[u]]==mex) stk.insert(col[u]);

for(auto i=stk.begin();i!=stk.end();i++)

{

ANS+=0ll+(\*i);

}

ans[u]=ANS;

if(keep==0) {mex=0;ANS=0;}

if(keep==0)

for(int i=st[u]; i<=fn[u]; i++)

{

cnt[ col[ ver[i] ] ]--;

}

return ;

}

int main()

{

ios\_base::sync\_with\_stdio(false);

cin>>n;

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

{

cin>>col[i];

}

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

{

int x,y;

cin>>x>>y;

gr[x].pb(y);

gr[y].pb(x);

}

t=0;

dfssiz(1,-1);

dfs(1,-1,0);

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

{

if(i>1) cout<<" ";

cout<<ans[i];

}

return 0;

}

Edit Distance

#include<bits/stdc++.h>

using namespace std;

string aa,bb;

int dp[1000][1000];

int rec\_edit(int i,int j)

{

///(i,j) means (0 to i),(0 to j) will be calculated and stored in (i,j)

if(dp[i][j]!=-1) return dp[i][j];

if(i==0) return j;

if(j==0) return i;

if(aa[i-1]==bb[j-1]) return dp[i][j]=rec\_edit(i-1,j-1);

return dp[i][j]=1+min(rec\_edit(i-1,j),min(rec\_edit(i,j-1),rec\_edit(i-1,j-1)));

}

void print\_edit(int n,int m, int table[100][100])

{

string a=aa;///for printing the string while editing

///all opertaions done on first string

int i=n;

int j=m;

while(i>-1 && j>-1)

{

if(i==0 && j==0) break;

else if(i==0)

{

cout<<j<<"I";///j characters inserted

a.insert(0,bb.substr(0,j));

i=j=-2;

}

else if(j==0)

{

cout<<i<<"D";///i characters deleted

a.erase(0,i);

i=j=-2;

}

else if(aa[i-1]==bb[j-1])

{

cout<<"M";///match found

i--;

j--;

}

else

{

if(table[i][j]==1+table[i-1][j-1])

{

cout<<"S";///character aa[i] substituted by charater bb[j]

a[i-1]=bb[j-1];

i--;

j--;

}

else if(table[i][j]==1+table[i-1][j])

{

cout<<"D";/// character aa[i] deleted

a.erase(a.begin()+i-1);

i--;

}

else if(table[i][j]==1+table[i][j-1])

{

cout<<"I";///character bb[j] inserted at i

a.insert(a.begin()+i,bb[j-1]);

j--;

}

}

cout<<" string: "<<a<<endl;

if(i==-2 || j==-2) break;

}

}

int table\_edit(int n,int m)

{

//int table[n+2][m+2];

int table[100][100];

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

{

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

{

if(i==0) table[i][j]=j;

else if(j==0) table[i][j]=i;

else if(aa[i-1]==bb[j-1]) table[i][j]=table[i-1][j-1];

else table[i][j]=1+min( table[i-1][j-1], min( table[i-1][j],table[i][j-1] ) );

}

}

/\*for(int i=0;i<=n;i++)

{

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

{

cout<<table[i][j];

}

cout<<endl;

}\*/

print\_edit(n,m,table);

return table[n][m];

}

int space\_optimised(int n,int m)

{

int table[2][m+1];

int bi;

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

{

bi=i&1;

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

{

if(i==0) table[bi][j]=j;

else if(j==0) table[bi][j]=i;

else if(aa[i-1]==bb[j-1]) table[bi][j]=table[1-bi][j-1];

else table[bi][j]=1+min(table[1-bi][j],min(table[bi][j-1],table[1-bi][j-1]));

}

}

return table[bi][m];

}

int main()

{

cin>>aa>>bb;

memset(dp,-1,sizeof dp);

cout<<rec\_edit(aa.length(),bb.length())<<endl;

cout<<endl<<table\_edit(aa.length(),bb.length())<<endl;

cout<<space\_optimised(aa.length(),bb.length())<<endl;

return 0;

}

RMQ(sparse table)

int n,lg;

vector<int>v(1000);

int table[1000][1000];///[n][lg];

void prn()

{

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

{

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

{

cout<<table[i][j];

}

cout<<endl;

}

}

void pre\_process()

{

///O(nlogn)

///table stores index of minimum array element from i to i+2^j -1

for(int j=0; (1<<j)<=n ;j++ ) /// small to big ranges

{

for(int i=0; i+(1<<j)-1<n; i++)

{

if(j==0) table[i][j]=i;

else

{

if(v[table[i][j-1]]<v[table[i+(1<<(j-1))][j-1]])

{

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

}

else table[i][j]=table[i+(1<<(j-1))][j-1];

}

}

}

//prn();

}

int query(int l,int r)

{

///O(1)

int Lg=log2(r-l+1);

///nearsest power of 2 to the range l to r;

return min(v[table[l][Lg]],v[table[r-(1<<Lg)+1][Lg]]);

}

int main()

{

cin>>n;

lg=log2(n);

for(int i=0;i<n;i++) cin>>v[i];

pre\_process();

int q;

cin>>q;

while(q--)

{

int x,y;

cin>>x>>y;

cout<<query(x,y)<<endl;

}

return 0;

}

Huge Mod

ll hugemod(string a,string b,int m)

{

int la=a.length();

int lb=b.length();

ll A=0;

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

{

A=(A\*10+a[i]-'0')%m;

}

ll x=1;

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

{

x=bigmod(x,10,m);

x=(x\*bigmod(A,b[i]-'0',m))%m;

}

return x;

}

Counting On bit

int normal\_count(int n)

{

int cnt=0;

while(n)

{

cnt+=n&1;

n=n>>1;

}

return cnt;

}

int Brian\_Kernighan(int n)

{

int cnt=0;

while(n)

{

n=n&(n-1);

cnt++;

}

return cnt;

}

///http://www.geeksforgeeks.org/next-higher-number-with-same-number-of-set-bits/

/\*\*

Going upwards:

Find the rightmost occurrence of "01" in the number and make it "10".

Justify all following 1-bits as far to the right as possible.

Going downwards:

Find the rightmost occurrence of "10" in the number and make it "01".

Left-justify all following 1-bits (i.e. don't do anything if the bit you just set is already followed by a 1).

\*/

int next\_greater(int n)

{

int a=(~n)+1; //-n

int t=a&n;

int x=n+t;

int y=x^n;

y=y/t;

y=y>>2;

return x|y;

}

int next\_small(int y)

{

int t = y + 1;

int u = t ^ y;

int v = t & y;

return v - (v & -v) / (u + 1);

}

int main()

{

///built\_in

//cout<< \_\_builtin\_popcount(7) << endl;

/\*

\_\_builtin\_popcount = int

\_\_builtin\_popcountl = long int

\_\_builtin\_popcountll = long long

\*/

cout<<next\_greater(156)<<endl;

return 0;

}

Digit Dp

// Given two integers a and b. The task is to print

// sum of all the digits appearing in the

// integers between a and b

#include "bits/stdc++.h"

using namespace std;

// Memoization for the state results

long long dp[20][180][2];

// Stores the digits in x in a vector digit

long long getDigits(long long x, vector <int> &digit)

{

while (x)

{

digit.push\_back(x%10);

x /= 10;

}

}

// Return sum of digits from 1 to integer in

// digit vector

long long digitSum(int idx, int sum, int tight,

vector <int> &digit)

{

// base case

if (idx == -1)

return sum;

// checking if already calculated this state

if (dp[idx][sum][tight] != -1 and tight != 1)

return dp[idx][sum][tight];

//else if(tight==1) cout<<"idx="<<idx<<" sum="<<sum<<"\n";

long long ret = 0;

// calculating range value

int k = (tight)? digit[idx] : 9;

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

{

// caclulating newTight value for next state

int newTight = (digit[idx] == i)? tight : 0;

// fetching answer from next state

ret += digitSum(idx-1, sum+i, newTight, digit);

}

if (!tight)

dp[idx][sum][tight] = ret;

return ret;

}

// Returns sum of digits in numbers from a to b.

int rangeDigitSum(int a, int b)

{

// initializing dp with -1

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

// storing digits of a-1 in digit vector

vector<int> digitA;

getDigits(a-1, digitA);

// Finding sum of digits from 1 to "a-1" which is passed

// as digitA.

long long ans1 = digitSum(digitA.size()-1, 0, 1, digitA);

cout<<"shesh"<<endl;

// Storing digits of b in digit vector

vector<int> digitB;

getDigits(b, digitB);

// Finding sum of digits from 1 to "b" which is passed

// as digitB.

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

long long ans2 = digitSum(digitB.size()-1, 0, 1, digitB);

cout<<ans1<<endl;

cout<<ans2<<endl;

return (ans2 - ans1);

}

// driver function to call above function

int main()

{

long long a = 123, b = 1024;

cout << "digit sum for given range : "

<< rangeDigitSum(a, b) << endl;

return 0;

}

Digit Dp -2

///https://stackoverflow.com/questions/22394257/how-to-count-integers-between-large-a-and-b-with-a-certain-property/22394258#22394258

Indeed, there is an approach to this pattern which turns out to work quite often. It can also be used to enumerate all the X with the given property, provided that their number is reasonably small. You can even use it to aggregate some associative operator over all the X with the given property, for example to find their sum.

To understand the general idea, let us try to formulate the condition X ≤ Y in terms of the decimal representations of X and Y.

Say we have X = x1 x2 ... xn - 1 xn and Y = y1 y2 ... yn - 1 yn, where xi and yi are the decimal digits of X and Y. If the numbers have a different length, we can always add zero digits to the front of the shorter one.

Let us define leftmost\_lo as the smallest i with xi < yi. We define leftmost\_lo as n + 1 if there is no such i. Analogously, we define leftmost\_hi as the smallest i with xi > yi, or n + 1 otherwise.

Now X ≤ Y is true if and exactly if leftmost\_lo <= leftmost\_hi. With that observation it becomes possible to apply a dynamic programming approach to the problem, that "sets" the digits of X one after another. I will demonstrate this with your example problems:

Compute the number f(Y) of integers X with the property X ≤ Y and X has the digit sum 60

Let n be the number of Y's digits and y[i] be the i-th decimal digit of Y according to the definition above. The following recursive algorithm solves the problem:

count(i, sum\_so\_far, leftmost\_lo, leftmost\_hi):

if i == n + 1:

# base case of the recursion, we have recursed beyond the last digit

# now we check whether the number X we built is a valid solution

if sum\_so\_far == 60 and leftmost\_lo <= leftmost\_hi:

return 1

else:

return 0

result = 0

# we need to decide which digit to use for x[i]

for d := 0 to 9

leftmost\_lo' = leftmost\_lo

leftmost\_hi' = leftmost\_hi

if d < y[i] and i < leftmost\_lo': leftmost\_lo' = i

if d > y[i] and i < leftmost\_hi': leftmost\_hi' = i

result += count(i + 1, sum\_so\_far + d, leftmost\_lo', leftmost\_hi')

return result

///Now we have f(Y) = count(1, 0, n + 1, n + 1) and we have solved the problem. We can add memoization to the function to make it fast. The runtime is O(n4) for this particular implementation. In fact we can cleverly optimize the idea to make it O(n). This is left as an exercise to the reader (Hint: You can compress the information stored in leftmost\_lo and leftmost\_hi into a single bit and you can prune if sum\_so\_far > 60). The solution can be found at the end of this post.

Compute the number f(Y) of integers X with the property X ≤ Y and X is palindromic

This one is slightly tougher. We need to be careful with leading zeroes: The mirror point of a palindromic number depends on how many leading zeroes we have, so we would need to keep track of the number of leading zeroes.

There is a trick to simplify it a bit though: If we can count the f(Y) with the additional restriction that all numbers X must have the same digit count as Y, then we can solve the original problem as well, by iterating over all possible digit counts and adding up the results.

So we can just assume that we don't have leading zeroes at all:

count(i, leftmost\_lo, leftmost\_hi):

if i == ceil(n/2) + 1: # we stop after we have placed one half of the number

if leftmost\_lo <= leftmost\_hi:

return 1

else:

return 0

result = 0

start = (i == 1) ? 1 : 0 # no leading zero, remember?

for d := start to 9

leftmost\_lo' = leftmost\_lo

leftmost\_hi' = leftmost\_hi

# digit n - i + 1 is the mirrored place of index i, so we place both at

# the same time here

if d < y[i] and i < leftmost\_lo': leftmost\_lo' = i

if d < y[n-i+1] and n-i+1 < leftmost\_lo': leftmost\_lo' = n-i+1

if d > y[i] and i < leftmost\_hi': leftmost\_hi' = i

if d > y[n-i+1] and n-i+1 < leftmost\_hi': leftmost\_hi' = n-i+1

result += count(i + 1, leftmost\_lo', leftmost\_hi')

return result

The result will again be f(Y) = count(1, n + 1, n + 1).

UPDATE: If we don't only want to count the numbers, but maybe enumerate them or compute some aggregate function from them which does not expose group structure, we need to enforce the lower bound on X as well during the recursion. This adds a few more parameters.

UPDATE 2: O(n) Solution for the "digit sum 60" example:

In this application we place the digits from left to right. Since we are only interested in whether leftmost\_lo < leftmost\_hi holds true, let us add a new parameter lo. lo is true iif leftmost\_lo < i and false otherwise. If lo is true, we can use any digit for the position i. If it is false, we can only use the digits 0 to Y[i], since any larger digit would cause leftmost\_hi = i < leftmost\_lo and can thus not lead to a solution. Code:

def f(i, sum\_so\_far, lo):

if i == n + 1: return sum\_so\_far == 60

if sum\_so\_far > 60: return 0

res = 0

for d := 0 to (lo ? 9 : y[i]):

res += f(i + 1, sum + d, lo || d < y[i])

return res

Arguably, this way of looking at it is somewhat simpler, but also a bit less explicit than the leftmost\_lo/leftmost\_hi approach. It also doesn't work immediately for somewhat more complicated scenarios like the palindrome problem (although it can be used there as well).

Find Subarray with given Sum

// C++ program to print subarray with sum as given sum

#include<bits/stdc++.h>

using namespace std;

// Function to print subarray with sum as given sum

void subArraySum(int arr[], int n, int sum)

{

// create an empty map

unordered\_map<int, int> map;

// Maintains sum of elements so far

int curr\_sum = 0;

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

{

// add current element to curr\_sum

curr\_sum = curr\_sum + arr[i];

// if curr\_sum is equal to target sum

// we found a subarray starting from index 0

// and ending at index i

if (curr\_sum == sum)

{

cout << "Sum found between indexes "

<< 0 << " to " << i << endl;

return;

}

// If curr\_sum - sum already exists in map

// we have found a subarray with target sum

if (map.find(curr\_sum - sum) != map.end())

{

cout << "Sum found between indexes "

<< map[curr\_sum - sum] + 1

<< " to " << i << endl;

return;

}

map[curr\_sum] = i;

}

// If we reach here, then no subarray exists

cout << "No subarray with given sum exists";

}

// Driver program to test above function

int main()

{

int arr[] = {10, 2, -2, -20, 10};

int n = sizeof(arr)/sizeof(arr[0]);

int sum = -12;

subArraySum(arr, n, sum);

return 0;

}

Closest Pair Of points

struct ClosestPairOfPoint

{

///points taken as double

///considered all distances ans the square of distances within double

double inf=numeric\_limits<double>::max();

vector< pair<double,double> > pointsx,pointsy;

ClosestPairOfPoint() {}

void addpoint(double x,double y)

{

pointsx.push\_back({x,y});;

}

void Clr()

{

pointsx.clear();

pointsy.clear();

}

double dist(pair<double,double>a, pair<double,double>b)

{

double d=(a.first-b.first)\*(a.first-b.first)+(a.second-b.second)\*(a.second-b.second);

return sqrt(d);

}

static bool compx(pair<double,double>a , pair<double,double>b)

{

return a.first<b.first;

}

static bool compy(pair<double,double>a , pair<double,double>b)

{

return a.second<b.second;

}

double bruteforce(int b,int e)

{

double mn=inf;

for(int i=b;i<=e;i++)

{

for(int j=i+1;j<=e;j++)

{

mn=min(mn,dist(pointsx[i],pointsx[j]));

}

}

return mn;

}

double stripClosest(vector<pair<double,double> >strip , double d)

{

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

{

for(int j=i+1;j<strip.size() && strip[j].second-strip[i].second<=d ; j++)

{

d=min(d,dist(strip[i],strip[j]));

}

}

return d;

}

double ClosestPoint( int b,int e, vector< pair<double,double> >& py )

{

if(e-b+1<=3) return bruteforce(b,e);

int mid=(b+e)/2;

int midx=pointsx[mid].first;

vector< pair<double,double> >yl,yr;

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

{

if(py[i].first<=midx) yl.push\_back(py[i]);

else yr.push\_back(py[i]);

}

double dl=ClosestPoint(b,mid,yl);

double dr=ClosestPoint(mid+1,e,yr);

double d=min(dl,dr);

vector< pair<double,double> > strip;

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

{

if(abs(midx-py[i].first)<=d) strip.push\_back(py[i]);

}

d=min(d,stripClosest(strip,d));

return d;

}

double run()

{

pointsy=pointsx;

sort(pointsx.begin(),pointsx.end(),compx);

sort(pointsy.begin(),pointsy.end(),compy);

int n=pointsx.size()-1;

return ClosestPoint(0, n, pointsy );

}

};

ConvexHull(Grahams)

int n,fx=1<<30,fy=1<<30,in;///first points

vector<pii>p,Hull;

int cw\_ccw(pii o,pii a,pii b)

{

///is OB cw or ccw wrt OA?

///OA x OB >0 ccw , <0 cw , =0 collinear

int x=(a.fs-o.fs)\*(b.sc-o.sc)-(b.fs-o.fs)\*(a.sc-o.sc);

return x;

}

ll dist(pii a,pii b)

{

return (a.fs-b.fs)\*(a.fs-b.fs)+(a.sc-b.sc)\*(a.sc-b.sc);

}

bool comp (pii a, pii b)

{

int x=cw\_ccw(pii(fx,fy),a,b);

if(x<0) return false;

else if(x>0) return true;

else if(dist(pii(fx,fy),a)>dist(pii(fx,fy),b)) return false;

else return true;

}

void find\_first\_point()

{

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

{

if(p[i].sc<fy)

{

in=i;

fy=p[i].sc;

fx=p[i].fs;

}

else if(p[i].sc==fy && p[i].fs<fx)

{

in=i;

fx=p[i].fs;

}

}

}

void MakeHull()

{

find\_first\_point();

p.erase(p.begin()+in);

sort(p.begin(),p.end(),comp);

p.insert(p.begin(),pii(fx,fy));

for(int i=1; i<p.size()-1; i++)

{

if(cw\_ccw(pii(fx,fy),p[i],p[i+1])==0)

{

p.erase(p.begin()+i);

i--;

}

}

if(p.size()<3) return;

Hull.pb(p[0]);

Hull.pb(p[1]);

Hull.pb(p[2]);

int hullsize=2;

for(int i=3; i<p.size(); i++)

{

while(cw\_ccw(Hull[hullsize-1],Hull[hullsize],p[i])<=0)

{

Hull.pop\_back();

hullsize--;

}

Hull.pb(p[i]);

hullsize++;

}

}

void prntHull()

{

cout<<"Vertices of Convex Hull : "<<endl;

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

{

cout<<Hull[i].fs<<" "<<Hull[i].sc<<endl;

}

}

ConvexHull (Monotone Chain)

int ccw(pii a,pii b,pii c)

{

///ab X ac

return (b.fs-a.fs)\*(c.sc-a.sc)-(b.sc-a.sc)\*(c.fs-a.fs);

}

///probably works either way, first wrt to x coordinate or y coordinate

bool cmp(pii a,pii b)

{

if(a.sc==b.sc) return a.fs<b.fs;

else return a.sc<b.sc;

}

vector<pii> monotone (vector<pii>p)

{

int n=p.size();

vector<pii>hull;

if(n<3) return hull;

sort(p.begin(),p.end(),cmp);

hull.pb(p[0]);

hull.pb(p[1]);

int hullsize=2;

///lower hull

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

{

while(hullsize>=2 && ccw(hull[hullsize-2],hull[hullsize-1],p[i])<=0)

{

hull.pop\_back();

hullsize--;

}

hull.pb(p[i]);

hullsize++;

}

int t=hullsize;

///upper hull

for(int i=n-2;i>-1;i--)

{

///at least 2 points in upper hull

while(hullsize>t && ccw(hull[hullsize-2],hull[hullsize-1],p[i])<=0)

{

hull.pop\_back();

hullsize--;

}

hull.pb(p[i]);

hullsize++;

}

hull.pop\_back();

hullsize--;

return hull;

}

int main()

{

int n;

cin>>n;

vector<pii>p(n);

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

{

cin>>p[i].fs>>p[i].sc;

}

vector<pii>hull;

hull=monotone(p);

cout<<"HUll :"<<endl;

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

{

cout<<hull[i].fs<<" "<<hull[i].sc<<endl;

}

return 0;

}