Topology sort

#include <cstdio>

#include <algorithm>

#include <vector>

using namespace std;

#define PB push\_back

#define MAXN 105

#define WHITE 0

#define GRAY 1

#define BLACK 2

int Time;

int numV;

struct V{

int index;

int color;

int start;

int finish;

};

V v[MAXN];

vector<V\*> adj[MAXN];

void DFS\_visit(V &u){

u.color = GRAY;

u.start = ++Time;

for(V\* cur : adj[u.index])

if(!cur->color) // if(cur.color == WHITE)

DFS\_visit(\*cur);

u.color = BLACK;

u.finish = ++Time;

}

void DFS(){

Time = 0;

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

v[i].color = WHITE;

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

if(!v[i].color) // if v[i].color == WHITE

DFS\_visit(v[i]);

}

bool cmp(V a, V b){

return a.finish > b.finish;

}

void input(int n){

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

v[i].index = i+1;

while(n--){

int t;

int nxt;

cin >> t >> nxt;

adj[t].PB(&v[nxt-1]);

}

}

void topology\_sort(){

DFS();

sort(v, v+numV, cmp);

}

Maxflow

int FordFulkerson(int s, int t, int n){

int ret = 0;

while(1){

memset(v, 0, sizeof(v));

if(!DFS(s, t, n)) // source, end, totalV

break;

ret += FindFlow(s, t, n);

}

return ret;

}

bool DFS(int cur, int t, int n){

v[cur] = true;

if(cur == t)

return true;

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

if(v[i])

continue;

if(cap[cur][i] - flow[cur][i] > 0 || flow[i][cur] > 0){

pa[i] = cur;

if(DFS(i, t, n))

return true;

}

}

return false;

}

int FindFlow(int s, int t, int n){

int pre;

int f = INF;

for(int i = t; i != s; i = pa[i]){

pre = pa[i];

if(cap[pre][i]-flow[pre][i]>0)

f = min(f, cap[pre][i]-flow[pre][i]);

else

f = min(f, flow[i][pre]);

}

for (int i = t; i != s; i = pa[i]){

pre = pa[i];

if(cap[pre][i] - flow[pre][i] > 0)

flow[pre][i] += f;

else

flow[i][pre] -= f;

}

return f;

}

void Artic\_dfs(int cur, int parent){

int child = 0;

dfn[cur] = low[cur] = ++Index;

bool cutv = false; // 只要有一圈不能到ancestors就会是cutv

int size = Vp[cur].size();

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

int next = Vp[cur][i];

if(dfn[next] == 0){

child++;

Artic\_dfs(next, cur);

low[cur] = min(low[cur], low[next]);

if(dfn[cur] <= low[next]) // 若能到ancestors,low会较小 | < for bridge cur next

cutv = true; // 表示不能到ancestors， 是cut-vertex

}else if(next == parent)

continue;

else

low[cur] = min(low[cur], dfn[next]);

}

if((~parent && cutv )|| (!~parent && child >= 2)){// find bridge no need to use this

// printf("qq %d\n", cur);

++crit;

}

}

void BackTracking(){

if(solution generated){

process solution

return;

}

for (x = each value of current dimension)

if(condition){

solution[dimension] = x;

BackTracking(dimension+1);

}

}

void BFS(int root){

queue<int> Q;

Q.push(root);

visited[root]=true;

while(!Q.empty()){

int cur=Q.front();

Q.pop();

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

int next = adj[cur][i];

if(!visited[next]){

visited[next]=true;

Q.push(next);

}

}

}

}

int Bipartite(int nL, int nR){

memset(llink, 0, (nL+1)\*sizeof(int));

memset(rlink, 0, (nR+1)\*sizeof(int));

int ans = 0;

for(int i = 1; i <= nL; ++i){// must start from 1, coz 0 means unlinked

memset(used, false, (nR+1)\*sizeof(bool));

if(DFS(i))

++ans;

}

return ans;

}

//Bipartite DFS

bool DFS(int now){

for(int i = 0; i < (int)edg[now].size(); ++i){

int next = edg[now][i];

if(!used[next]){

used[next] = true;

if(!rlink[next] || DFS(rlink[next])){

llink[now] = next;

rlink[next] = now;

return true;

}

}

}

return false;

}

void DFS(int cur){

vis[cur] = true;

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

int next = adj[cur][i];

if(!vis[next])

DFS(next);

}

}

//Dinamic programming

int DP1(){//无限硬币求方法数

int v[] = {1, 5, 10, 25, 50};

int n;

long long int dp[7550] = {0};

dp[0] = 1;

for (int i = 0; i < 5; ++i){

for (int j = 0; j < 7490; ++j){

dp[j+v[i]] += dp[j];

}

}

while(scanf("%d", &n)!=EOF){

printf("%lld\n", dp[n]);

}

return 0;

}

int DP2(){//一个硬币求最高价值

int N, M;

int dp[12900] = {0};

scanf("%d %d", &N, &M);

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

int n, m;

scanf("%d %d", &n, &m);

for (int j = M-n; j >= 0; --j)

dp[j+n] = max(dp[j+n], dp[j] + m);

}

printf("%d\n", dp[M]);

return 0;

}

// disjoint

int Find(int x){

return x == p[x] ? x : p[x] = Find(p[x]);

}

void Union(int x, int y){

p[Find(x)] = Find(y);

}

int exGCD(int a, int b, int &X, int &Y){

if(b == 0){

X = 1;

Y = 0;

return a;

}else{

int gcd = exGCD(b, a%b, X, Y);

int tmp = X;

X = Y;

Y = tmp - (a/b)\*Y;

return gcd;

}

}

int gcd(int a, int b){

return a == 0 ? b : gcd(b % a, a);

}

// Hash

#define MAXN 100000

#define prime\_mod 1073676287

typedef long long T;

char str[MAXN+5];

T h[MAXN+5]; //hash 阵列

T h\_base[MAXN+5]; //h\_base[n] = (prime^n)%prime\_mod

inline void hash\_init(int len, T prime = 0xdefaced){

h\_base[0] = 1;

for (int i = 1; i <= len; ++i){

h[i] = (h[i-1]\*prime + str[i-1]) % prime\_mod;

h\_base[i] = (h\_base[i-1]\*prime) % prime\_mod;

}

}

inline T get\_hash(int l, int r){

return (h[r+1]-(h[l]\*h\_base[r-l+1])%prime\_mod+prime\_mod)%prime\_mod;

}

unsigned int hash (char \*str){

unsigned int seed = 131; // 31 131 1313 13131 131313 etc..

unsigned int key=0;

while (\*str)

key = key \* seed + (\*str++);

return (key%prime+prime)%prime;

// return (hash & 0x7FFFFFFF);

}

inline void KMP\_fail (char \*B, int \*pi){

int len = strlen(B);

pi[0] = -1;

for (int i = 1, cur\_pos = -1; i < len; ++i){

while(~cur\_pos && B[i] != B[cur\_pos+1])

cur\_pos = pi[cur\_pos];

if(B[i] == B[cur\_pos+1])

++cur\_pos;

pi[i] = cur\_pos;

}

// return cur\_pos to find period(the last of pi)

}

inline void KMP\_match(char \*A, char \*B, int \*pi){

int lenA = strlen(A);

int lenB = strlen(B);

for (int i = 1, cur\_pos = -1; i < lenA; ++i){

while(~cur\_pos && A[i]!=B[cur\_pos + 1])

cur\_pos = pi[cur\_pos];

if(A[i] == B[cur\_pos+1])

++cur\_pos;

if(cur\_pos + 1 == lenB){

// Match!!

cur\_pos = pi[cur\_pos];

}

}

}

int LCS(){

int n, m;

char a[102][31], b[102][31];

while(scanf("%s", a[1]) != EOF){

n = 2;

m = 1;

int pre[102][102] = {0};

int LCS[102][102] = {0};

while(scanf("%s", a[n]) && a[n][0] != '#')

++n;

while(scanf("%s", b[m]) && b[m][0] != '#')

++m;

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

for (int j = 1; j < m; ++j){

if(strcmp(a[i], b[j]) == 0){// if the zero index of the array is used, then cmp i-1 j-1

pre[i][j] = 0;

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

}else{

int up = LCS[i-1][j];

int ri = LCS[i][j-1];

if(up >= ri){

pre[i][j] = UP;

LCS[i][j] = up;

}else{

pre[i][j] = RI;

LCS[i][j] = ri;

}

}

}

}

stack<char\*>S;

--n;

--m;

while(n > 0 && m > 0){

switch(pre[n][m]){

case 0: S.push(a[n]);

--n;

--m;

break;

case UP: --n;

break;

case RI: --m;

break;

}

}

int i = 0;

while(!S.empty()){

if(i++)

printf(" ");

printf("%s", S.top());

S.pop();

}

puts("");

}

}

int LCS2(){

int N;

scanf("%d", &N);

int ans[N+2];

int arr[N+2];

int LCS[N+2][N+2];

int t;

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

scanf("%d", &t);

ans[t] = i;

}

while(scanf("%d", &t) != EOF){

arr[t] = 1;

for (int i = 2; i <= N; ++i){

scanf("%d", &t);

arr[t] = i;

}

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

LCS[0][i] = LCS[i][0] = 0;

}

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

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

if(ans[i] == arr[j]){// if the zero index of the array is used, then cmp i-1 j-1

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

}else{

if(LCS[i-1][j] >= LCS[i][j-1]){

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

}else{

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

}

}

}

}

printf("%d\n", LCS[N][N]);

}

}

int LIS(){

int arr[1005];

int LIS[1005];

int N;

scanf("%d", &N);

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

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

int Max = 1;

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

LIS[i] = 1;

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

if(arr[j] < arr[i] && LIS[j] >= LIS[i]){

LIS[i] = LIS[j] + 1;

Max = LIS[i] > Max ? LIS[i] : Max;

}

}

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

return 0;

}

int LIS\_2(){

int k, n;

while(scanf("%d%d", &k, &n) != EOF){

box B[35];

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

{

B[i].index = i;

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

scanf("%d", &B[i].dimen[j]);

sort(B[i].dimen, B[i].dimen + n);

}

sort(B + 1, B + 1 + k, cmp);

int LIS[k+1];

int pre[k+1]; // may be need to be initialized

int Max = 1;

int last = -1;

for (int i = 1; i <= k; ++i){// [first,last]

LIS[i] = 1;

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

if(smaller(B[j], B[i], n) && LIS[j] >= LIS[i]){ //CHECK : if same length which one to save (some case may use LIS[j]+1>=LIS[i])

LIS[i] = LIS[j] + 1;

pre[i] = j;

if((LIS[i] > Max) || (LIS[i] == Max && j < B[pre[i]].index)){ // CHECK2 : if same length

Max = LIS[i];

last = i;

}

}

}

}

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

stack<box> S;

while(Max--){

S.push(B[last]);

last = pre[last];

}

while(!S.empty()){

if(++Max) // !fisrt

putchar(' ');

box t = S.top();

printf("%d", t.index);

S.pop();

}

puts("");

}

}

int LIS\_3(){ // nlogn

vector<int> V;

int t;

while(scanf("%d", &t) != EOF)

V.push\_back(t);

int N = V.size();

int pre[N];

vector<int> VLIS; // can create a struct for VLIS and Vindex

vector<int> Vindex;

for (int i = 0; i < N; ++i){ // 感觉相同长度时要取先出现的好像就不能用这方法了……？//或许可以倒着实作

int j = lower\_bound(VLIS.begin(), VLIS.end(), V[i]) - VLIS.begin();

if(j == VLIS.size()){

VLIS.push\_back(V[i]);

Vindex.push\_back(i);

}else{

VLIS[j] = V[i];

Vindex[j] = i;

}

pre[i] = j > 0 ? Vindex[j-1] : j; // when j == 0 then pre is useless

}

int length = VLIS.size();

printf("%d\n-\n", length);

stack<int> S;

int last = Vindex[length-1];

while(length--){

S.push(V[last]);

last = pre[last];

}

while(!S.empty()){

int tmp = S.top();

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

S.pop();

}

return 0;

}

// merge sort

void Combine(int l,int mid,int r){

int i,j,cnt;

// Merge

i=l,j=mid+1,cnt=0;

while(i<=mid&&j<=r){

if(ary[j]<ary[i]) buf[cnt++]=ary[j++];

else buf[cnt++]=ary[i++];

}

// Remain

while(i<=mid) buf[cnt++]=ary[i++];

while(j<=r) buf[cnt++]=ary[j++];

// Copy back

for(i=l;i<=r;i++)

ary[i]=buf[i-l];

}

// Merge Sort

void MergeSort(int l,int r)

{

// Single Element

if(l==r) return;

// Divide

int mid=(l+r)/2;

MergeSort(l,mid);

MergeSort(mid+1,r);

Combine(l,mid,r);

}

// Prime

bool isPrime(long long int n){

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

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

return false;

return true;

}

void MakePrime(){

prime.push\_back(2);

prime.push\_back(3);

int primeNum = 2;

for (int i = 5, gap = 2; i < Max; i += gap, gap = 6-gap){

if(isPrime(i)){

++primeNum;

prime.push\_back(i);

}

}

}

int periodicStrings(){

char str[1000005];

while(gets(str) && str[0]!='.'){

const int length = strlen(str);

bool same = false;

int i; // i for longest period

int num; // num for # of repeat

for (i = 1; i <= length/2; ++i){

if(length % i != 0)

continue;

num = length / i;

bool sub\_same = true;

for (int j = 1; j < num; ++j){

if(sub\_same == false)

break;

int start = i\*j;

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

if(str[k] != str[start+k])

sub\_same = false;

}

if(sub\_same){

same = true;

break;

}

}

if(same)

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

else

printf("%d\n", 1);

}

return 0;

}

//

void Floyd\_init(int numV){

for (int i = 0; i < numV; ++i){ // index start from 0

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

dis[i][j] = INF;

dis[i][i] = 0;

}

}

void Floyd(int numV){

for (int k = 0; k < numV; ++k) // index start from 0

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

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

if(dis[i][k]+dis[k][j] < dis[i][j])

dis[i][j] = dis[i][k] + dis[k][j];

}

// n = numV, 有负环return true, 可知道负环经过sourse,觉得有没有经过都要知道可以一开始push全部进去

bool SPFA(int n, int source){

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

dis[i] = INF;

inqueue[i] = false;

count[i] = 0;

}

dis[source] = 0;

inqueue[source] = true;

queue<int> Q;

Q.push(source);

while(!Q.empty()){

int now = Q.front();

inqueue[now] = false;

Q.pop();

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

if(p[now][i] != INF && dis[now] + p[now][i] < dis[i]){

dis[i] = dis[now] + p[now][i];

if(!inqueue[i]){

Q.push(i);

inqueue[i] = true;

count[i]++;

if(count[i] >= n)

return true;

}

}

}

}

return false;

}

bool SPFA2(int n, int source){ // remember to clear Vp after each case

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

dis[i] = INF;

inqueue[i] = false;

count[i] = 0;

}

dis[source] = 0;

inqueue[source] = true;

queue<int> Q;

Q.push(source);

while(!Q.empty()){

int now = Q.front();

inqueue[now] = false;

Q.pop();

for (int i = 0; i < Vp[now].size(); ++i){

int r = Vp[now][i].r;// r means right end

int d = Vp[now][i].d;// d means distance

if(dis[now] + d < dis[r]){

dis[r] = dis[now] + d;

if(!inqueue[r]){

Q.push(r);

inqueue[r] = true;

count[r]++;

if(count[r] >= n)

return true;

}

}

}

}

return false;

}

struct edge{

int l, r, d;

}E[MAXN];

bool SSSP(int numV, int numE){ // false means no negative cycle, V index start from 0

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

dis[i] = INF;

dis[0] = 0;

for (int term = 0; term < numV; ++term){//numV-1+1

bool change = false;

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

const int left = E[i].l;

const int right = E[i].r;

const int length = E[i].d;

if(dis[right] > dis[left] + length){

dis[right] = dis[left] + length;

change = true;

}

}

if(!change)

return false;

}

return true;

}

struct Trie{

char ans[11];

int child[26];

}Tr[1000005];

void makeTrie(const char\* a, char\* b){

int i = 0;

while(\*b){

// printf("a\n");

const int b\_int = \*b - 'a';

if(Tr[i].child[b\_int] == 0)

Tr[i].child[b\_int] = ++num;

i = Tr[i].child[b\_int];

++b;

}

strcpy(Tr[i].ans, a);

}

// for i

void Tarjan(int cur){

int child = 0;

dfn[cur] = low[cur] = ++index;

instack[cur] = true;

S.push(cur);

int size = Vp[cur].size();

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

int next = Vp[cur][i];

if(dfn[next] == 0){

Tarjan(next);

if(low[next] < low[cur])

low[cur] = low[next];

}else if(instack[next] && dfn[next] < low[cur])

low[cur] = dfn[next];

}

if(dfn[cur] == low[cur]){

count++;

int next;

do{

next = S.top();

S.pop();

instack[next] = false;

// belong[next]=count; // or = cur

}while(next!=cur);

}

}