TEMPLATE: pre define

#include <iostream>

#include <cstring>

#include <algorithm>

#include <cmath>

#include <cstdio>

#include <vector>

#include <queue>

#include <set>

#include <map>

#include <stack>

#define ll long long

#define local

using namespace std;

const int MOD = 1e9+7;

const int inf = 0x3f3f3f3f;

const int PI = acos(-1.0);

const int maxn = 1e4+10;

int main() {

#ifdef local

if(freopen("/Users/Andrew/Desktop/data.txt", "r", stdin) == NULL) printf("can't open this file!\n");

#endif

#ifdef local

fclose(stdin);

#endif

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEMPLATE: IDA\*

bool dfs(int step, int limit) {

    if(step == limit) {

        for(int i = 1; i <= 9; ++i) {

            if(num[i] != goal[i])

                return 0;

        }

        return 1;

    }

    if(!estimate()) return 0;

    else {

    }

    return 0;

}

for(j = step; ; j = nextd) {

    nextd = inf;

    if(dfs(0, j)) break;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEMPLATE: DP

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

for (int j = limit; j >= size[i]; j--)

dp[j] = max(dp[j], dp[j-size[i]]+value[i]); // 逆序

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*完全背包\*\*\*\*

https://www.cnblogs.com/Kalix/p/7622102.html

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

for(int j = w[i] ; j <= limit ; ++j) // 枚举重量

dp[j] = max(dp[j],dp[j-w[i]]+v[i]); // 正序

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*多重背包\*\*\*\*

void MultiP(int cost, int amount, int val) {

if (cost\*amount >= limit) { //CompletePack

for (int i = cost; i <= limit; ++i) // 枚举重量

dp[i] = max(dp[i], dp[i-cost]+val);

return;

}

int k = 1;

while (k <= amount) {

for (int i = limit; i >= cost\*k; --i) //ZeroandOnePack 枚举重量

dp[i] = max(dp[i], dp[i-cost\*k]+val\*k);

amount -= k;

k \*= 2;

}

for (int i = limit; i >= cost\*amount; --i) //ZeroandOnePack

dp[i] = max(dp[i], dp[i-cost\*amount]+val\*amount);

}

\*\*\*\*LIS\*\*\*\*

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

\*lower\_bound(dp, dp+i, a[i]) = a[i];

}

\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEMPLATE: NUMBER THEORY:

\*\*\*\*\*高次幂运算\*\*\*\*\*

int Pow\_Mod (long long a, long long n) {

int ret = 1;

int tmp = a%MOD;

while (n) {

if (n&1) ret = (ret\*tmp)%MOD;

tmp = (tmp\*tmp)%MOD;

n >>= 1;

}

return ret;

}

\*\*\*\*\*优化：高次幂运算\*\*\*\*\*

ll qsc(ll a,ll b,ll mod){

ll sum = 0;

while (b) {

if(b&1) sum = (sum + a) % mod;//乘法转换成了加法来进行！

a = (a + a) % mod;

b>>=1;

}

return sum % mod;

}

ll qsm(ll a,ll b,ll mod){

ll res = 1;

while (b) {

if(b&1) res = qsc(res, a, mod);

a = qsc(a, a, mod);

b>>=1;

}

return res % mod;

}

\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*素数筛法\*\*\*\*\*

void filter (int m) { //筛选1-m的素数

int n = sqrt(m+0.5);

memset(isprime, 0, sizeof(isprime));

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

if (!isprime[i])

for (int j = i\*i; j <= m; j+=i)

isprime[j] = 1;

isprime[1] = 1;

}

\*\*\*\*\*质因子分解\*\*\*\*\*

1.

for (int i = 1; i\*i <= val; ++i) {

if (val%i != 0) continue;

}

2.

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

            while (n%i==0) {

factor[c++]=i;

n/=i;

}

}

\*\*\*\*\*\*\*\*欧几里得算法 & 扩展欧几里得算法\*\*\*\*\*\*\*\*\*\*

int gcd(int a, int b) {

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

}

int exgcd(int a, int b, int& x, int& y) {

int d = a;

if (b != 0) {

d = exgcd(b, a%b, y, x);// gcd(a, b) == gcd(b, a%b)

y -= (a/b) \* x; //逆过程得到x, y的解。

} else {

x = 1; y = 0;

}

return d; // d = gcd(a, b)

}

\*\*\*\*\*\*\*\*\*\*求解同余方程\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void solve() {

// ax + by = c

// ax TONGYU c (mod b) ax和c关于b同余

int d = exgcd(a, b, x, y);

if (c%d != 0) {

// no answer

} else {

int ans;

ans = x \* (c/d);

d = b/d;

ans = (ans%d + d) % d;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEMPLATE： NETWORK FLOW

////////////EK：邻接矩阵实现

int path[maxn];

int vis[maxn];

int EK(int start, int end) {

int sum = 0;

queue<int> q;

while (true) {

memset(path, 0, sizeof(path));

memset(vis, 0, sizeof(vis));

while (q.size()) q.pop();

q.push(start);

vis[1] = 1;

while (q.size()) {

int u = q.front();

q.pop();

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

if (!vis[i] && flow[u][i]) {

path[i] = u;

vis[i] = 1;

if (i == end)

break;

q.push(i);

}

}

}

if (!vis[end])

break;

int tmp = inf;

for (int i = end; i != 1; i = path[i])

tmp = min(tmp, flow[path[i]][i]);

for (int i = end; i != 1; i = path[i]) { //减去tmp值，建立反向弧

flow[path[i]][i] -= tmp;

flow[i][path[i]] += tmp;

}

sum += tmp;

}

return sum;

}

////////////EK：邻接表实现

struct Edge {

int u, v, val;

int pre;

} edge[maxn];

struct Pre {

int EdgeNum, Vertex;

} pre[maxn];

bool vis[maxn];

int point[maxn];

int cnt;

int N, M;

int EK(int start, int end) {

int sum = 0;

queue<int> q;

while (true) {

memset(vis, 0, sizeof(vis));

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

pre[i].EdgeNum = -1;

pre[i].Vertex = 0;

}

while (q.size()) q.pop();

q.push(start);

vis[start] = 1;

while (q.size()) {

int u = q.front();

q.pop();

for (int i = point[u]; i != -1; i = edge[i].pre) {

if (!vis[edge[i].v] && edge[i].val) {

pre[edge[i].v].Vertex = edge[i].u;

pre[edge[i].v].EdgeNum = i;

vis[edge[i].v] = 1;

if (edge[i].v == end)

break;

q.push(edge[i].v);

}

}

}

if (!vis[end])

break;

int tmp = inf;

for (int i = end; i != start; i = pre[i].Vertex) // i!=start not i != 0

tmp = min(tmp, edge[pre[i].EdgeNum].val);

for (int i = end; i != start; i = pre[i].Vertex) { //before start there's no edge!

edge[pre[i].EdgeNum].val -= tmp;

edge[pre[i].EdgeNum ^ 1].val += tmp;

}

sum += tmp;

}

return sum;

}

///////////////////////

DINIC算法：

const int maxn = 505;

const int maxedge = maxn\*maxn;

class Graph {

private:

int cnt\_point;

int cnt\_edge;

int point[2\*maxn];

int pre[maxedge];

int W[maxedge];

int V[maxedge];

int Depth[2\*maxn];

int cur[2\*maxn];//cur就是记录当前点u循环到了哪一条边

queue<int> Q;

public:

int s,t;

void init() {

cnt\_edge = 0;

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

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

}

void AddEdge(int u, int v, int w) {

V[cnt\_edge] = v;

W[cnt\_edge] = w;

pre[cnt\_edge] = point[u];

point[u] = cnt\_edge++;

cnt\_point = max(cnt\_point, u);

}

bool dfs(int u, int flow) {

if (u == t)

return flow;

for (int &i = cur[u]; i != -1; i = pre[i]) { //注意这里的&符号，这样i增加的同时也能改变cur[u]的值，达到记录当前弧的目的

if ((Depth[V[i]]==Depth[u]+1) && (W[i]!=0)) {

int di = dfs(V[i], min(flow, W[i]));

if (di > 0) {

W[i] -= di;

W[i^1] += di;

return di;

}

}

}

return 0;

}

bool bfs() {

while (!Q.empty())

Q.pop();

memset(Depth, 0, sizeof(Depth));//init depth

Depth[s] = 1;

Q.push(s);

while (Q.size()) {

int u = Q.front();

Q.pop();

for (int i = point[u];i != -1;i = pre[i])

if ((Depth[V[i]]==0) && (W[i]>0)) {

Depth[V[i]] = Depth[u]+1;

Q.push(V[i]);

}

}

if (Depth[t] > 0)

return 1;

return 0;

}

int Dinic() {

int ans = 0;

while (bfs()) {

for (int i = 0; i <= cnt\_point; i++)// init cur!

cur[i] = point[i];

while (int d = dfs(s, inf))

ans += d;

}

return ans;

}

};

//////////匈牙利算法/////////

邻接表： O(n\*m)

邻接矩阵：O(n^3)

bool find(int x) {

int i,j;

for (j=1;j<=m;j++){ //扫描每个妹子

if (connect[x][j]==true && used[j]==false) {

used[j]=1;

if (match[j]==0 || find(match[j])) {

match[j]=x;

return true;

}

}

}

return false;

}

main():

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

memset(used, 0, sizeof(used)); //这个在每一步中清空

if (find(i)) sum += 1;

}

/////////HK算法：邻接矩阵////////

int nx, ny, dis;

int dx[maxn], dy[maxn], mx[maxn], my[maxn];

bool vis[maxn];

int g[maxn][maxn];

bool bfs() {

queue<int> q;

dis = inf;

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

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

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

if(mx[i] == -1) { // x fail to match

q.push(i);

dx[i] = 0;

}

}

while(!q.empty()) {

int u = q.front();

q.pop();

if(dx[u] > dis)

break;

for(int v = 0; v < ny; v++) { // pick the right node

if(g[u][v] && dy[v]==-1) {

dy[v] = dx[u]+1;

if(my[v] == -1)

dis = dy[v];

else {

dx[my[v]] = dy[v]+1;

q.push(my[v]);

}

}

}

}

return dis != inf;

}

bool dfs(int u) {

for(int v = 0; v < ny; v++) {

if(!vis[v] && g[u][v] && dy[v]==dx[u]+1) {

vis[v] = 1;

if(my[v]!=-1 && dy[v]==dis)

continue;

if(my[v]==-1 || dfs(my[v])) {

my[v] = u;

mx[u] = v;

return true;

}

}

}

return false;

}

int HK() {

int res = 0;

memset(mx,-1,sizeof(mx));//x match

memset(my,-1,sizeof(my));//y match

while(bfs()) {

memset(vis,0,sizeof(vis));

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

if(mx[i]==-1 && dfs(i))

res++;

}

}

return res;

}

/////////HK算法：邻接表////////

int nx, ny, dis, cnt;

int dx[maxn], dy[maxn], mx[maxn], my[maxn], point[maxn], pre[maxn\*maxn], vertex[maxn\*maxn];

bool vis[maxn];

void AddEdge(int u, int v) {

vertex[cnt] = v;

pre[cnt] = point[u];

point[u] = cnt++;

}

bool bfs() {

queue<int> q;

dis = inf;

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

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

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

if(mx[i] == -1) { // x fail to match

q.push(i);

dx[i] = 0;

}

}

while(!q.empty()) {

int u = q.front();

q.pop();

if(dx[u] > dis)

break;

for(int i = point[u]; i != -1; i = pre[i]) { // pick the right node

int v = vertex[i];

if(dy[v]==-1) {

dy[v] = dx[u]+1;

if(my[v] == -1)

dis = dy[v];

else {

dx[my[v]] = dy[v]+1;

q.push(my[v]);

}

}

}

}

return dis != inf;

}

bool dfs(int u) {

for(int i = point[u]; i != -1; i = pre[i]) { // pick the right node

int v = vertex[i];

if(!vis[v] && dy[v]==dx[u]+1) {

vis[v] = 1;

if(my[v]!=-1 && dy[v]==dis)

continue;

if(my[v]==-1 || dfs(my[v])) {

my[v] = u;

mx[u] = v;

return true;

}

}

}

return false;

}

int HK() {

int res = 0;

memset(mx,-1,sizeof(mx));//x match

memset(my,-1,sizeof(my));//y match

while(bfs()) {

memset(vis,0,sizeof(vis));

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

if(mx[i]==-1 && dfs(i))

res++;

}

}

return res;

}

///////////////

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEMPLATE: union\_find set

int find (int x) {

return x==pa[x] ? x:pa[x]=find(pa[x]);

}

void unite (int x, int y) {

x = find(x); y = find(y);

if (x == y) return;

pa[x] = y;

}

bool same (int x, int y) {

return find(x)==find(y);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEMPLATE: Dictionary tree

void insert(int id) {

int u = 0;

int k;

for(int i = 0; i < int(strlen(s)); ++i) {

k = ID(s[i]);

if(trie[u][k] == 0) {

memset(trie[num], 0, sizeof(trie[num])); //如果不存在结点，再开辟存储空间

alpha[num] = s[i]; //记录字符

trie[u][k] = num++; //结点标号为num

}

u = trie[u][k]; // 向下走一层

}

f[id] = val[u]; //邻接表存储枝干

val[u] = id;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEMPLATE: DIJKSTRA

Vector:

int n;

struct qnode {

int v;

int c;

qnode(int v=0,int c=0) : v(v), c(c) {}

bool operator <(const qnode &r)const {//priority\_queue 默认从大到小排列

return c > r.c;

}

};

struct Edge {

int v, cost;

Edge(int v=0,int cost=0) : v(v), cost(cost){}

};

vector<Edge>E[maxn]; // E[i].clear

bool vis[maxn];

int dist[maxn];

void Dijkstra(int n,int start) {

bool vis[maxn]; // already init in the dijkstra

ll dist[maxn]; // already init in the dijkstra, overflow! priority\_queue <qnode> que;

while(!que.empty()) que.pop();

dist[start] = 0;

que.push(qnode(start, 0));

qnode tmp;

while(!que.empty()) {

tmp = que.top(); que.pop();

int u = tmp.v;

if(vis[u]) continue;

vis[u] = true;

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

int v = E[u][i].v;

int cost = E[u][i].cost;

if(!vis[v] && dist[v]>dist[u]+cost) {

dist[v] = dist[u]+cost;

que.push(qnode(v, dist[v]));

}

}

}

}

void addedge(int u,int v,int w) { //有向边

E[u].push\_back(Edge(v,w));

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

邻接表：

const int maxn = (1e4+10) \* 2;

const int maxedge = 100\*100;

int n, m;

struct qnode {

int v;

int c;

qnode(int v=0,int c=0) : v(v), c(c) {}

bool operator <(const qnode &r)const {//priority\_queue 默认从大到小排列

return c > r.c;

}

};

struct Edge {

int v, w, pre;

} edge[maxedge];

int point[maxn]; //point[i] = -1

int cnt;

bool vis[maxn];

int dist[maxn];

void AddEdge(int u, int v, int w) {

edge[cnt].v = v;

edge[cnt].w = w;

edge[cnt].pre = point[u];

point[u] = cnt++;

}

void Dijkstra(int n,int start) {

bool vis[maxn]; // already init in the dijkstra

ll dist[maxn]; // already init in the dijkstra, overflow! priority\_queue <qnode> que;

while(!que.empty()) que.pop();

dist[start] = 0;

que.push(qnode(start, 0));

qnode tmp;

while(!que.empty()) {

tmp = que.top(); que.pop();

int u = tmp.v;

if(vis[u]) continue;

vis[u] = true;

for(int i = point[u]; i != -1; i = edge[i].pre) {

int v = edge[i].v;

int w = edge[i].w;

if(!vis[v] && dist[v]>dist[u]+w) {

dist[v] = dist[u]+w;

que.push(qnode(v, dist[v]));

}

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEMPLATE：SPFA -- 邻接表 SLF优化

const int maxn = (1e4+10) \* 2;

const int maxedge = 100\*100;

int n, m;

struct Edge {

int v, w;

int pre;

} edge[maxedge];

int dist[maxn];

int inque[maxn];

int point[maxn]; // point[i] = -1

int cnt; // cnt = 0

void AddEdge(int u, int v, int w) {

edge[cnt].v = v;

edge[cnt].w = w;

edge[cnt].pre = point[u];

point[u] = cnt++;

}

void spfa (int start) {

memset(inque, 0, sizeof(inque));

memset(dist, 0x3f, sizeof(dist));

dist[start] = 0;

deque<int> q;

while (!q.empty()) q.pop\_front();

q.push\_front(start);

while (q.size()) {

int u = q.front();

q.pop\_front();

inque[u] = 0;

for (int i = point[u]; i != -1; i = edge[i].pre) {

int v = edge[i].v;

if(dist[v] > dist[u]+edge[i].w) {

dist[v] = dist[u]+edge[i].w;

if(!inque[v]) {

inque[v] = 1;

if (!q.empty() && dist[v] > dist[q.front()])

q.push\_back(v);

else q.push\_front(v);

}

}

}

}

}

SPFA：邻接矩阵

int railway[maxn][maxn];

int inque[maxn];

int dist[maxn];

void spfa (int start) {

memset(dist, 0x3f, sizeof(dist));

dist[start] = 0;

queue<int> q;

q.push(start);

memset(inque, 0, sizeof(inque));

while (q.size()) {

int u = q.front();

q.pop();

inque[u] = 0;

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

if (u == i) continue;

if (railway[u][i] != inv) continue;

if(dist[i] > dist[u]+1) {

dist[i] = dist[u]+1;

if(!inque[i]) {

inque[i] = 1;

q.push(i);

}

}

}

}

}

Floyd：

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

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

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

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

segment tree:

void Build(int o, int l, int r) {

//inv[o] = 0; 区间修改需要懒惰标记！

if (l == r) {

sum[o] = a[l];

return;

}

int m = (l+r)>>1;

Build(o<<1, l, m); Build(o<<1|1, m+1, r);

sum[o] = sum[o<<1]+sum[o<<1|1];

}

void add(int q, int o, int c, int l, int r) { // 点修改

if (l == r) {

sum[o] += c;

return;

}

int m = (l+r)>>1;

if (q <= m) add(q, o<<1, c, l, m);

else add(q, o<<1|1, c, m+1, r);

sum[o] = sum[o<<1]+sum[o<<1|1];

}

int query(int o, int l, int r, int ql, int qr) { // 区间查找

if ((ql<=l) && (qr>=r))

return sum[o];

int m = (l+r)>>1;

int ans = 0;

if (ql <= m) ans += query(o<<1, l, m);

if (qr > m) ans += query(o<<1|1, m+1, r);

return ans;

}

-----区间整体修改-------

void add(int o, int l, int r, int ql, int qr, int val) {

if ((ql<=l) && (qr>=r)) {

sum[o] = (r-l+1)\*val;

inv[o] = val;

return;

}

int m = (l+r)>>1;

PushDown(o, m-l+1, r-m);////检查是否有之前遍历的懒惰标记

if (ql <= m) add(o<<1, l, m, ql, qr, val);

if (qr > m) add(o<<1|1, m+1, r, ql, qr, val);

sum[o] = sum[o<<1|1]+sum[o<<1];

}

void PushDown(int o, int ln, int rn) {

//由于用法，懒惰标记只可能向下遍历，并且是在下一次更新或者查询操作中

if (inv[o]) {

// 更新的是下一层的区间

sum[o<<1] = ln\*inv[o];//区间修改

sum[o<<1|1] = rn\*inv[o];//区间修改

inv[o<<1] = inv[o<<1|1] = inv[o];

inv[o] = 0;//消除标记

}

}

int query(int o, int l, int r, int ql, int qr) {

if ((ql<=l) && (qr>=r)) {

return sum[o];

}

int m = (l+r)>>1;

PushDown(o, m-l+1, r-m);

int ans = 0;

if (ql <= m) ans += query(o<<1, l, m, ql, qr);

if (qr > m) ans += query(o<<1|1, m+1, r, ql, qr);

return ans;

}

------整体添加------

void add(int o, int l, int r, int ql, int qr, int val) {

if ((ql<=l) && (qr>=r)) {

sum[o] += val\*(r-l+1);

inv[o] += val;

return;

}

int m = (l+r)>>1;

PushDown(o, m-l+1, r-m);

if (ql <= m) add(o<<1, l, m, ql, qr, val);

if (qr > m) add(o<<1|1, m+1, r, ql, qr, vla);

sum[o] = sum[o<<1]+sum[o<<1|1];

}

void PushDown(int o, int ln, int rn) {

if (inv[o]) {

inv[o<<1] += inv[o];

inv[o<<|1] += inv[o];

sum[o<<1] += inv[o]\*ln;

sum[o<<1|1] += inv[o]\*rn;

inv[o] = 0;

}

}

int query(int o, int l, int r, int ql, int qr) {

if ((ql<=l) && (qr>=r)) {

return sum[o];

}

int m = (l+r)>>1;

PushDown(o, m-l+1, r-m);

int ans = 0;

if (ql <= m) ans += query(o<<1, l, m, ql, qr);

if (qr > m) ans += query(o<<1|1, m+1, r, ql, qr);

return ans;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary\_Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

struct node {

int val;

node \*lch, \*rch;

};

node \*root = NULL;

root = insert(root, 1);

node \*insert(node \*p, int x) {

if (p == NULL) {

node \*q = new node;

q->val = x;

q->lch = q->rch = NULL;

return q;

} else {

if (x < p->val) p->lch = insert(p->lch, x);

else p->rch = insert(p->rch, x);

return p;

}

}

bool find(node \*p, int x) {

if (p == NULL) return false;

else if (p->val == x) return true;

else if (p->val < x) return find(p->lch, x);

else return fidn(p->rch, x);

}

node \*remove(node \*p, int x) {

if (p == NULL) return NULL;

else if (x < p->val) remove(p->lch, x);

else if (x > p->val) remove(p->rch, x);

else if (p->lch == NULL) { // 没有左子树， 将右子树提上去

node \*q = p->rch;

delete p;

return q;

}

else if (p->lch->rch == NULL) { // 左子树没有右子树，将左子树直接提上去

node \*q = p->lch;

q->rch = p->rch;

delete p;

return q;

}

else { //不满足上述情况， 将左子树的最大子孙提上去

node \*q;

for (q = p->lch; q->rch->rch != NULL; q = q->rch) ;

node \*r = q->rch; //r为最大子孙

q->rch = r->lch //将q连上r的左子树

r->lch = p->lch;//用r替代p

r->rch = p->rch;//用r替代p

delete p;

return r;

}

return p;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*KMP\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//解决连续匹配问题

void getFail(char \*p,int \*f){//f为失配函数

int m=strlen(p);

f[0]=0;f[1]=0;//f[i]表示到第i个元素时的最长公共前后缀

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

int j=f[i];//j=失配状态,最长公共前后缀

while (j&&p[i]!=p[j])//失配状态不为0,且模版链前后两元素不相同！

j=f[j];//不匹配，回到之前一个失配状态

f[i+1]=p[i]==p[j]?j+1:0;//i+1的失配状态+1(模版链前后两元素相同),或者令其=0

}

}

void find(char \*T,char \*p,int \*f){ // T为模版字符串，p为匹配串

int n=strlen(T),m=strlen(p);

getFail(p, f);

int j=0;//当前结点编号，初始为0号结点

for (int i=0; i<n; i++) {//模版串当前指针

while(j&&T[i]!=p[j]) j=f[j];//顺着失配边走，直到可以匹配，或者回至j=0状态0

if(p[j]==T[i]) j++;

if(j==m) printf("%d\n",i-m+1);//找到

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary\_Search\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//对二分的理解：如果写成l-r>1这样

//取r为答案的话，如果令l = 0（区间的最小值），无法取到0

//要令l = -1 为区间的最小值。

int l = -1; int r = n-1;

while (r-l > 1) {

int mid = (l+r)>>1;

if (ok(mid)) {

r = mid;

} else {

l = mid;

}

}

printf("%d\n", r); //r包括在ok里面，所以应该输出r

//高精度：

while (T--) {

double m = (l+r)>>1;

if (ok(m)) l = m;

else r = m;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

链表 List

//初始化nex, pre数组

//数据存在1-n中

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

nex[i] = i+1;

}

nex[n] = -1;

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

pre[i] = i-1;

//删除结点

nex[pre[i]] = nex[i];

pre[nex[i]] = pre[i];

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*全排列\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//交换a,b两数

void Swap(int &a,int &b)

{

int temp=a;

a=b;

b=temp;

}

//将数组区间转化为数字

int getNum(int list[], int f, int r)

{

int i = 0, num = 0;

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

num = list[i] + num \* 10; //进位

return num;

}

//进行全排列并对每种排列结果进行处理

// arange the k-th to m-th number

void Prim(int list[], int k, int m)

{

if(k==m) //前缀是最后一个位置,此时出现一种排列数. 不需要再递归

{

// perform your algorithm here

printf("%d\n", getNum(list, 0, n-1));

}

else

{

for(int i=k;i<=m;i++) //全排列数组list[]

{

//交换前缀,使之产生下一个前缀.

Swap(list[k],list[i]);

Prim(list,k+1,m);

//将前缀换回来,继续做上一个的前缀排列.

Swap(list[k],list[i]);

}

}

}