1 DP

1.1 KMP

```cpp
void ComputePrefix(string s, int next[])
{
    int n = s.length();
    int q, k;
    next[0] = 0;
    for (k = 0, q = 1; q < n; q++)
    {
        while (k > 0 && s[k] != s[q])
            k = next[k];
        if (s[k] == s[q])
            k++;
        next[q] = k;
    }
}
void KMPMatcher(string text, string pattern)
{
    int n = text.length();
    int m = pattern.length();
    int next[pattern.length()];
    ComputePrefix(pattern, next);

    for (int i = 0, q = 0; i < n; i++)
    {
        while (q > 0 && pattern[q] != text[i])
            q = next[q];
        if (pattern[q] == text[i])
            q++;
        if (q == m)
        {
            cout << "Pattern occurs with shift " << i - m + 1 << endl;
            q = 0;
        }
    }
}
string s = "abcdabcdebcd";
string p = "bcd";
KMPMatcher(s, p);
cout << endl;
return 0;
```

1.2 LCS

```cpp
int LCS(vector<int> Ans, vector<int> num) // Ans 跟 num 都要 index 從1開始放
{
    vector<vector<int>> LCS(N + 1, vector<int>(N + 1, 0));
    for (int i = 1; i <= N; ++i)
    {
        for (int j = 1; j <= N; ++j)
        {
            if (Ans[i] == num[j])
                LCS[i][j] = LCS[i - 1][j - 1] + 1;
            else
                LCS[i][j] = max(LCS[i - 1][j], LCS[i][j - 1]);
        }
    }
    // printf("%d\n", LCS[N][N]);
    return LCS[N][N];
    //列印 LCS
    vector<int> k;
    while (n && m)
    {
        if (LCS[n][m] != max(LCS[n - 1][m], LCS[n][m - 1]))
        {
            k.push_back(arr1[n]);
            n--;
            m--;
        }
        else if (LCS[n][m] == LCS[n - 1][m])
            n--;
        else if (LCS[n][m] == LCS[n][m - 1])
            m--;
    }

    reverse(k.begin(), k.end());
}
```

1.3 LIC

```cpp
void getMaxElementAndPos(vector<int> &LISTbl, vector<int> &LISLen, int tNum,
    int tlen, int tStart, int &num, int &pos)
{
    int max = numeric_limits<int>::min();
    int maxPos;
    for (int i = tStart; i >= 0; i--)
    {
        if (LISLen[i] == tlen && LISTbl[i] < tNum)
        {
            if (LISTbl[i] > max)
            {
                max = LISTbl[i];
                maxPos = i;
            }
        }
    }
    num = max;
    pos = maxPos;
}
int LIS(vector<int> &LISTbl)
{
    if (LISTbl.size() == 0)
        return 0;
    vector<int> LISLen(LISTbl.size(), 1);
    for (int i = 1; i < LISTbl.size(); i++)
    {
        for (int j = 0; j < i; j++)
        {
            if (LISTbl[j] < LISTbl[i])
                LISLen[i] = max(LISLen[i], LISLen[j] + 1);
        }
    }

    int maxlen = *max_element(LISLen.begin(), LISLen.end());
    int num, pos;
    vector<int> buf;
    getMaxElementAndPos(LISTbl, LISLen, numeric_limits<int>::max(),
                        maxlen, LISTbl.size() - 1, num, pos);
    buf.push_back(num);
    for (int len = maxlen - 1; len >= 1; len--)
    {
        int tnum = num;
        int tpos = pos;
        getMaxElementAndPos(LISTbl, LISLen,
                            tnum, len, tpos - 1, num, pos);
        buf.push_back(num);
    }
    reverse(buf.begin(), buf.end());
    for (int k = 0; k < buf.size(); k++) // 列印
    {
        if (k == buf.size() - 1)
            cout << buf[k] << endl;
        else
            cout << buf[k] << ",";
    }
    return maxlen;
}
```

1.4 Max_subarray

```cpp
/*Kadane's algorithm*/
int maxSubArray(vector<int>& nums) {
    int local_max = nums[0], global_max = nums[0];
    for(int i = 1; i < nums.size(); i++){
        local_max = max(nums[i],nums[i]+local_max);
        global_max = max(local_max, global_max);
    }
    return global_max;
}
```

1.5 MFlow

```cpp
typedef long long ll;
struct MF
{
    static const int N = 5000 + 5;
    static const int M = 60000 + 5;
    static const ll oo = 10000000000000LL;

    int n, m, s, t, tot, tim;
    int first[N], next[M];
    int u[M], v[M], cur[N], vi[N];
    ll cap[M], flow[M], dis[N];
    int que[N + N];

    void Clear()
    {
        tot = 0;
        tim = 0;
        for (int i = 1; i <= n; ++i)
            first[i] = -1;
    }
    void Add(int from, int to, ll cp, ll flw)
    {
        u[tot] = from;
        v[tot] = to;
        cap[tot] = cp;
        flow[tot] = flw;
        next[tot] = first[u[tot]];
        first[u[tot]] = tot;
        ++tot;
    }
    bool bfs()
    {
        ++tim;
        dis[s] = 0;
        vi[s] = tim;

        int head, tail;
        head = tail = 1;
        que[head] = s;
        while (head <= tail)
        {
            for (int i = first[que[head]]; i != -1; i = next[i])
            {
                if (vi[v[i]] != tim && cap[i] > flow[i])
                {
                    vi[v[i]] = tim;
                    dis[v[i]] = dis[que[head]] + 1;
                    que[++tail] = v[i];
                }
            }
            ++head;
        }
        return vi[t] == tim;
    }
    ll dfs(int x, ll a)
    {
        if (x == t || a == 0)
            return a;
        ll flw = 0, f;
        int &i = cur[x];
        for (i = first[x]; i != -1; i = next[i])
        {
            if (dis[x] + 1 == dis[v[i]] && (f = dfs(v[i], min(a, cap[i]
                 - flow[i]))) > 0)
            {
                flow[i] += f;
                flow[i ^ 1] -= f;
                a -= f;
                flw += f;
                if (a == 0)
                    break;
            }
        }
        return flw;
    }
    ll MaxFlow(int s, int t)
    {
        this->s = s;
        this->t = t;
        ll flw = 0;
        while (bfs())
        {
            for (int i = 1; i <= n; ++i)
                cur[i] = 0;
            flw += dfs(s, oo);
        }
        return flw;
    }
};
// MF Net;
// Net.n = n;
// Net.Clear();
// a 到 b (注意從1開始!!!!)
// Net.Add(a, b, w, 0);
// Net.MaxFlow(s, d)
// s 到 d 的 MF
```

2 Geometry

2.1 Convexhull 2D

```
bool same(double a, double b) { return abs(a - b) < 0; }
struct P // 台大
{
    double x, y;
    P() : x(0), y(0) {}
    P(double x, double y) : x(x), y(y) {}
    P operator+(P b) { return P(x + b.x, y + b.y); }
    P operator-(P b) { return P(x - b.x, y - b.y); }
    P operator*(double b) { return P(x * b, y * b); }
    P operator/(double b) { return P(x / b, y / b); }
    double operator*(P b) { return x * b.x + y * b.y; }
    double operator^(P b) { return x * b.y - y * b.x; }
    double abs() { return hypot(x, y); }
    P unit() { return *this / abs(); }
    P spin(double o)
    {
        double c = cos(o), s = sin(o);
        return P(c * x - s * y, s * x + c * y);
    }
    double angle() { return atan2(y, x); }
};
bool operator<(const P &a, const P &b) { return same(a.x, b.x) ? a.y < b.y : a.x < b.x; }
bool operator>(const P &a, const P &b) { return same(a.x, b.x) ? a.y > b.y : a.x > b.x; }
#define crx(a, b, c) ((b - a) ^ (c - a)) // (向量OA叉積向量OB。) > 0 表示從OA到OB為逆時針旋轉。
vector<P> convex(vector<P> ps) // Andrew's Monotone Chain
{
    vector<P> p;

    sort(ps.begin(), ps.end(), [](P &a, P &b)
         { return a.y < b.y || (a.y == b.y && a.x < b.x); });
    for (int i = 0; i < ps.size(); ++i)
    {
        while (p.size() >= 2 && crx(p[p.size() - 2], ps[i], p[p.size() - 1]) >= 0)
            p.pop_back();
        p.push_back(ps[i]);
    }
    int t = p.size();
    for (int i = (int)ps.size() - 2; i >= 0; --i)
    {
        while (p.size() > t && crx(p[p.size() - 2], ps[i], p[p.size() - 1]) >= 0)
            p.pop_back();
        p.push_back(ps[i]);
    }
    // p.pop_back(); //起點依照題目
    return p;
}
```

3 Graph

3.1 Bellman-Ford

```
/*SPA - Bellman-Ford*/
#include<bits/stdc++.h>
#define inf 99999 //define by you maximum edges weight
using namespace std;
vector<vector<int> > edges;
vector<int> dist;
vector<int> ancestor;
void BellmanFord(int start,int node){
    dist[start] = 0;
    for(int it = 0; it < node-1; it++){
        for(int i = 0; i < node; i++){
            for(int j = 0; j < node; j++){
                if(edges[i][j] != -1){
                    if(dist[i] + edges[i][j] < dist[j]){
                        dist[j] = dist[i] + edges[i][j];
                        ancestor[j] = i;
                    }
                }
            }
        }
    }

    for(int i = 0; i < node; i++)  // negative cycle detection
        for(int j = 0; j < node; j++)
            if(dist[i] + edges[i][j] < dist[j])
            {
                cout<<"Negative cycle!"<<endl;
                return;
            }
}
int main(){
    int node;
    cin>>node;
    edges.resize(node,vector<int>(node,inf));
    dist.resize(node,inf);
    ancestor.resize(node,-1);
    int a,b,d;
    while(cin>>a>>b>>d){
        /*input: source destination weight*/
        if(a == -1 && b == -1 && d == -1)
            break;
        edges[a][b] = d;
    }
    int start;
    cin>>start;
    BellmanFord(start,node);
    return 0;
}
```

3.2 BFS-queue

```
/*BFS - queue version*/
#include<bits/stdc++.h>
using namespace std;
void BFS(vector<int> &result,vector<pair<int,int> > edges,int node,int start){
    vector<int> pass(node, 0);
    queue<int> q;
    queue<int> p;
    q.push(start);
    int count = 1;
    vector<pair<int, int>> newedges;
    while(!q.empty()){
        pass[q.front()] = 1;
        for (int i = 0; i < edges.size(); i++){
            if(edges[i].first == q.front() && pass[edges[i].second] == 0){
                p.push(edges[i].second);
                result[edges[i].second] = count;
            }
            else if(edges[i].second == q.front() && pass[edges[i].first] == 0){
                p.push(edges[i].first);
                result[edges[i].first] = count;
            }
            else
                newedges.push_back(edges[i]);
        }
        edges = newedges;
        newedges.clear();
        q.pop();
        if(q.empty() == true){
            q = p;
            queue<int> tmp;
            p = tmp;
            count++;
        }
    }
}
int main(){
    int node;
    cin >> node;
    vector<pair<int, int>> edges;
    int a, b;
    while(cin>>a>>b){
        /*a = b = -1 means input edges ended*/
        if(a == -1 && b == -1)
            break;
        edges.push_back(pair<int, int>(a, b));
    }
    vector<int> result(node, -1);
    BFS(result, edges, node, 0);

    return 0;
}
```

3.3 DFS-rec

```
/*DFS - Recursive version*/
#include<bits/stdc++.h>
using namespace std;
map<pair<int,int>,int> edges;
vector<int> pass;
vector<int> route;
void DFS(int start){
    pass[start] = 1;
    map<pair<int,int>,int>::iterator iter;
    for(iter = edges.begin(); iter != edges.end(); iter++){
```

```cpp
        if((*iter).first.first == start && (*iter).second == 0 && pass[(*iter).first.second] == 0){
            route.push_back((*iter).first.second);
            DFS((*iter).first.second);
        }
        else if((*iter).first.second == start && (*iter).second == 0 && pass[(*iter).first.first] == 0){
            route.push_back((*iter).first.first);
            DFS((*iter).first.first);
        }
    }
}
int main(){
    int node;
    cin>>node;
    pass.resize(node,0);
    int a,b;
    while(cin>>a>>b){
        if(a == -1 && b == -1)
            break;
        edges.insert(pair<pair<int,int>,int>(pair<int,int>(a,b),0));
    }
    int start;
    cin>>start;
    route.push_back(start);
    DFS(start);
    return 0;
}
```

3.4 Dijkstra

```cpp
/*SPA - Dijkstra*/
#include<bits/stdc++.h>
#define inf INT_MAX
using namespace std;
vector<vector<int> > weight;
vector<int> ancestor;
vector<int> dist;
void dijkstra(int start){
    priority_queue<pair<int,int> ,vector<pair<int,int> > ,greater<pair<int,int> > > pq;
    pq.push(make_pair(0,start));
    while(!pq.empty()){
        int cur = pq.top().second;
        pq.pop();
        for(int i = 0; i < weight[cur].size(); i++){
            if(dist[i] > dist[cur] + weight[cur][i] && weight[cur][i] != -1){
                dist[i] = dist[cur] + weight[cur][i];
                ancestor[i] = cur;
                pq.push(make_pair(dist[i],i));
            }
        }
    }
}
int main(){
    int node;
    cin>>node;
    int a,b,d;
    weight.resize(node,vector<int>(node,-1));
    while(cin>>a>>b>>d){
        /*input: source destination weight*/
        if(a == -1 && b == -1 && d == -1)
            break;
        weight[a][b] = d;
    }
    ancestor.resize(node,-1);
    dist.resize(node,inf);
    int start;
    cin>>start;
    dist[start] = 0;
    dijkstra(start);
    return 0;
}
```

3.5 union_find

```cpp
int find(int x,vector<int> &union_set){
    if(union_set[x] != x)
        union_set[x] = find(union_set[x], union_set); //compress path
    return union_set[x];
}
void merge(int x,int y,vector<int> &union_set,vector<int> &rank){
    int rx, ry;
    rx = find(x,union_set);
    ry = find(y,union_set);
    if(rx == ry)
        return;
    /*merge by rank -> always merge small tree to big tree*/
    if(rank[rx] > rank[ry])
        union_set[ry] = rx;
    else
    {
        union_set[rx] = ry;
        if(rank[rx] == rank[ry])
            ++rank[ry];
    }
}
int main(){
    int node;
    cin >> node; //Input Node number
    vector<int> union_set(node, 0);
    vector<int> rank(node, 0);
    for (int i = 0; i < node; i++)
        union_set[i] = i;
    int edge;
    cin >> edge; //Input Edge number
    for(int i = 0; i < edge; i++)
    {
        int a, b;
        cin >> a >> b;
        merge(a, b, union_set,rank);
    }
    /*build party*/
    vector<vector<int> > party(node, vector<int>(0));
    for (int i = 0; i < node; i++)
        party[find(i, union_set)].push_back(i);
}
```

4 Mathematics

4.1 Extended Euclidean

```cpp
// ax + by = gcd(a,b)
pair<long long, long long> extgcd(long long a, long long b)
{
    if (b == 0)
        return {1, 0};
    long long k = a / b;
    pair<long long, long long> p = extgcd(b, a - k * b);
    //cout << p.first << " " << p.second << endl;
    //cout << "商數(k)=  " << k << endl << endl;
    return {p.second, p.first - k * p.second};
}

int main()
{
    int a, b;
    cin >> a >> b;
    pair<long long, long long> xy = extgcd(a, b); //(x0,y0)
    cout << xy.first << " " << xy.second << endl;
    cout << xy.first << " * " << a << " + " << xy.second << " * " << b << endl;
    return 0;
}
// ax + by = gcd(a,b) * r
/*find |x|+|y| -> min*/
int main()
{
    long long r, p, q; /*px+qy = r*/
    int cases;
    cin >> cases;
    while (cases--)
    {
        cin >> r >> p >> q;
        pair<long long, long long> xy = extgcd(q, p); //(x0,y0)
        long long ans = 0, tmp = 0;
        double k, k1;
        long long s, s1;
        k = 1 - (double)(r * xy.first) / p;
        s = round(k);
        ans = llabs(r * xy.first + s * p) + llabs(r * xy.second - s * q);
        k1 = -(double)(r * xy.first) / p;
        s1 = round(k1);
        /*cout << k << endl << k1 << endl;
            cout << s << endl << s1 << endl;*/
        tmp = llabs(r * xy.first + s1 * p) + llabs(r * xy.second - s1 * q);
        ans = min(ans, tmp);

        cout << ans << endl;
    }
    return 0;
}
```

4.2 Hex to Dec

```cpp
int HextoDec(string num) //16 to 10
{
    int base = 1;
    int temp = 0;
    for (int i = num.length() - 1; i = 0; i--)
    {
        if (num[i] = '0' && num[i] = '9')
        {
            temp += (num[i] - 48) base;
            base = base 16;
        }
        else if (num[i] = 'A' && num[i] = 'F')
        {
            temp += (num[i] - 55) base;
            base = base 16;
        }
    }
    return temp;
}
void DecToHex(int p_intValue) //10 to 16
{
    char l_pCharRes = new (char);
    sprintf(l_pCharRes, % X, p_intValue);
    int l_intResult = stoi(l_pCharRes);
    cout l_pCharRes n;
    return l_intResult;
}
```

4.3 PI

```cpp
#define PI acos(-1)
#define PI M_PI
```

4.4 Prime table

```cpp
// 埃拉托斯特尼篩法
const int maxn = 10000000;
bitset<maxn> prime;
```

```cpp
void sieve()
{
    for (int i = 2; i * i < maxn; ++i)
    {
        if (prime[i] == 0)
        {
            for (int j = i * i; j < maxn; j += i)
                prime[j] = 1;
        }
    }
}
/* 0跟1要寫if過濾掉 */
// if(!prime[數字])
//      我是質數
```

4.5 二分逼近法

```cpp
#define eps 1e-14
void half_interval()
{
    double L = 0, R = /*區間*/, M;
    while (R - L >= eps)
    {
        M = (R + L) / 2;
        if (/*函數*/ > /*方程式目標*/)
            L = M;
        else
            R = M;
    }
    printf("%.3lf\n", R);
}
```

4.6 四則運算

```cpp
string s = ""; //開頭是負號要補0
long long int DFS(int le, int ri) // (0, string final index)
{
    int c = 0;
    for (int i = ri; i >= le; i--)
    {
        if (s[i] == ')')
            c++;
        if (s[i] == '(')
            c--;
        if (s[i] == '+' && c == 0)
            return DFS(le, i - 1) + DFS(i + 1, ri);
        if (s[i] == '-' && c == 0)
            return DFS(le, i - 1) - DFS(i + 1, ri);
    }
    for (int i = ri; i >= le; i--)
    {
        if (s[i] == ')')
            c++;
        if (s[i] == '(')
            c--;
        if (s[i] == '*' && c == 0)
            return DFS(le, i - 1) * DFS(i + 1, ri);
        if (s[i] == '/' && c == 0)
            return DFS(le, i - 1) / DFS(i + 1, ri);
        if (s[i] == '%' && c == 0)
            return DFS(le, i - 1) % DFS(i + 1, ri);
    }
    if ((s[le] == '(') && (s[ri] == ')'))
        return DFS(le + 1, ri - 1); //去除刮號
    if (s[le] == ' ' && s[ri] == ' ')
        return DFS(le + 1, ri - 1); //去除左右兩邊空格
    if (s[le] == ' ')
        return DFS(le + 1, ri); //去除左邊空格
    if (s[ri] == ' ')
        return DFS(le, ri - 1); //去除右邊空格
    long long int num = 0;
    for (int i = le; i <= ri; i++)
        num = num * 10 + s[i] - '0';
    return num;
}
```

4.7 數字乘法組合

```cpp
void toans(vector<vector<int>> &ans, vector<int> com)
{
    // sort(com.begin(), com.end());
    ans.push_back(com);
    // for (auto i : com)
    //     cout << i << ' ';
    // cout << endl;
}
void finds(int j, int old, int num, vector<int> com, vector<vector<int>> &ans)
{
    for (int i = j; i <= sqrt(num); i++)
    {
        if (old == num)
            com.clear();
        if (num % i == 0)
        {
            vector<int> a;
            a = com;
            a.push_back(i);
            finds(i, old, num / i, a, ans);
            a.push_back(num / i);
            toans(ans, a);
        }
    }
}
int main()
{
    vector<vector<int>> ans;
    vector<int> zero;
    finds(2, num, num, zero, ans);
    // num 為 input 數字
    for (int i = 0; i < ans.size(); i++)
    {
        for (int j = 0; j < ans[i].size() - 1; j++)
            cout << ans[i][j] << " ";
        cout << ans[i][ans[i].size() - 1] << endl;
    }
}
```

4.8 數字加法組合

```cpp
void printCombination(vector<int> const &out, int m, vector<vector<int>> &ans)
{
    for (int i : out)
        if (i > m)
            return;
    ans.push_back(out);
}

void recur(int i, int n, int m, vector<int> &out, vector<vector<int>> &ans)
{
    if (n == 0)
        printCombination(out, m, ans);
    for (int j = i; j <= n; j++)
    {
        out.push_back(j);
        recur(j, n - j, m, out, ans);
        out.pop_back();
    }
}
int main()
{
    vector<vector<int>> ans;
    vector<int> zero;
    recur(1, num, num, zero, ans);
    // num 為 input 數字
    for (int i = 0; i < ans.size(); i++)
    {
        for (int j = 0; j < ans[i].size() - 1; j++)
            cout << ans[i][j] << " ";
        cout << ans[i][ans[i].size() - 1] << endl;
    }
}
```

4.9 羅馬數字

```cpp
int romanToInt(string s)
{
    unordered_map<char, int> T;
    T['I'] = 1;
    T['V'] = 5;
    T['X'] = 10;
    T['L'] = 50;
    T['C'] = 100;
    T['D'] = 500;
    T['M'] = 1000;

    int sum = T[s.back()];
    for (int i = s.length() - 2; i >= 0; --i)
    {
        if (T[s[i]] < T[s[i + 1]])
            sum -= T[s[i]];
        else
            sum += T[s[i]];
    }
    return sum;
}
```

4.10 質因數分解

```cpp
void cal(int in)
{
    for (long long x = 2; x <= in; x++)
    {
        while (in % x == 0)
        {
            cout << x << "*";
            in /= x;
        }
    }
}
```

5 Other

5.1 Weighted Job Scheduling

```cpp
struct Job
{
    int start, finish, profit;
};
bool jobComparataor(Job s1, Job s2)
{
    return (s1.finish < s2.finish);
}
int latestNonConflict(Job arr[], int i)
{
    for (int j = i - 1; j >= 0; j--)
    {
        if (arr[j].finish <= arr[i].start)
            return j;
    }
    return -1;
}
int findMaxProfit(Job arr[], int n)
{
    sort(arr, arr + n, jobComparataor);
    int *table = new int[n];
    table[0] = arr[0].profit;
    for (int i = 1; i < n; i++)
    {
        int inclProf = arr[i].profit;
        int l = latestNonConflict(arr, i);
        if (l != -1)
            inclProf += table[l];
        table[i] = max(inclProf, table[i - 1]);
    }
    int result = table[n - 1];
    delete[] table;

    return result;
}
```

5.2 數獨解法

```
int getSquareIndex(int row, int column, int n)
{
    return row / n * n + column / n;
}

bool backtracking(vector<vector<int>> &board, vector<vector<bool>> &rows, vector<vector<bool>> &cols,
                  vector<vector<bool>> &boxs, int index, int n)
{
    int n2 = n * n;
    int rowNum = index / n2, colNum = index % n2;
    if (index >= n2 * n2)
        return true;

    if (board[rowNum][colNum] != 0)
        return backtracking(board, rows, cols, boxs, index + 1, n);

    for (int i = 1; i <= n2; i++)
    {
        if (!rows[rowNum][i] && !cols[colNum][i] && !boxs[getSquareIndex(rowNum, colNum, n)][i])
        {
            rows[rowNum][i] = true;
            cols[colNum][i] = true;
            boxs[getSquareIndex(rowNum, colNum, n)][i] = true;
            board[rowNum][colNum] = i;
            if (backtracking(board, rows, cols, boxs, index + 1, n))
                return true;
            board[rowNum][colNum] = 0;
            rows[rowNum][i] = false;
            cols[colNum][i] = false;
            boxs[getSquareIndex(rowNum, colNum, n)][i] = false;
        }
    }
    return false;
}
/*用法 main*/
int n = sqrt(數獨邊長大小) /*e.g. 9*9 n=3*/
vector<vector<int>> board(n * n + 1, vector<int>(n * n + 1, 0));
vector<vector<bool>> isRow(n * n + 1, vector<bool>(n * n + 1, false));
vector<vector<bool>> isColumn(n * n + 1, vector<bool>(n * n + 1, false));
vector<vector<bool>> isSquare(n * n + 1, vector<bool>(n * n + 1, false));

for (int i = 0; i < n * n; ++i)
{
    for (int j = 0; j < n * n; ++j)
    {
        int number;
        cin >> number;
        board[i][j] = number;
        if (number == 0)
            continue;
        isRow[i][number] = true;
        isColumn[j][number] = true;
        isSquare[getSquareIndex(i, j, n)][number] = true;
    }
}
if (backtracking(board, isRow, isColumn, isSquare, 0, n))
    /*有解答*/
else
    /*解答*/
```

6 String

6.1 sliding window

```
string minWindow(string s, string t) {
    unordered_map<char, int> letterCnt;
    for (int i = 0; i < t.length(); i++)
        letterCnt[t[i]]++;
    int minLength = INT_MAX, minStart = -1;
    int left = 0, matchCnt = 0;
    for (int i = 0; i < s.length(); i++)
    {
        if (--letterCnt[s[i]] >= 0)
            matchCnt++;
        while (matchCnt == t.length())
        {
            if (i - left + 1 < minLength)
            {
                minLength = i - left + 1;
                minStart = left;
            }
            if (++letterCnt[s[left]] > 0)
                matchCnt--;
            left++;
        }
    }
    return minLength == INT_MAX ? "" : s.substr(minStart, minLength);
}
```

6.2 split

```
vector<string> mysplit(const string& str, const string& delim)
{
    vector<string> res;
    if ("" == str)
        return res;

    char *strs = new char[str.length() + 1];
    strcpy(strs, str.c_str());

    char *d = new char[delim.length() + 1];
    strcpy(d, delim.c_str());

    char *p = strtok(strs, d);
    while (p)
    {
        string s = p;
        res.push_back(s);
        p = strtok(NULL, d);
    }
    return res;
}
```

7 data structure

7.1 Bigint

```
//台大
struct Bigint{
    static const int LEN = 60;
    static const int BIGMOD = 10000;
    int s;
    int vl, v[LEN];
    //  vector<int> v;
    Bigint() : s(1) { vl = 0; }
    Bigint(long long a) {
        s = 1; vl = 0;
        if (a < 0) { s = -1; a = -a; }
        while (a) {
            push_back(a % BIGMOD);
            a /= BIGMOD;
        }
    }
    Bigint(string str) {
        s = 1; vl = 0;
        int stPos = 0, num = 0;
        if (!str.empty() && str[0] == '-') {
            stPos = 1;
            s = -1;
        }
        for (int i=SZ(str)-1, q=1; i>=stPos; i--) {
            num += (str[i] - '0') * q;
            if ((q *= 10) >= BIGMOD) {
                push_back(num);
                num = 0; q = 1;
            }
        }
        if (num) push_back(num);
        n();
    }
    int len() const {
        return vl;//return SZ(v);
    }
    bool empty() const { return len() == 0; }
    void push_back(int x) {
        v[vl++] = x; //v.PB(x);
    }
    void pop_back() {
        vl--; //v.pop_back();
    }
    int back() const {
        return v[vl-1]; //return v.back();
    }
    void n() {
        while (!empty() && !back()) pop_back();
    }
    void resize(int nl) {
        vl = nl; //v.resize(nl);
        fill(v, v+vl, 0); //fill(ALL(v), 0);
    }
    void print() const {
        if (empty()) { putchar('0'); return; }
        if (s == -1) putchar('-');
        printf("%d", back());
        for (int i=len()-2; i>=0; i--)
            printf("%.4d",v[i]);
    }
    friend std::ostream& operator << (std::ostream& out, const Bigint &a) {
        if (a.empty()) { out << "0"; return out; }
        if (a.s == -1) out << "-";
        out << a.back();
        for (int i=a.len()-2; i>=0; i--) {
            char str[10];
            snprintf(str, 5, "%.4d", a.v[i]);
            out << str;
        }
        return out;
    }
    int cp3(const Bigint &b)const {
        if (s != b.s) return s - b.s;
        if (s == -1) return -(-*this).cp3(-b);
        if (len() != b.len()) return len()-b.len();//int
        for (int i=len()-1; i>=0; i--)
            if (v[i]!=b.v[i]) return v[i]-b.v[i];
        return 0;
    }
    bool operator<(const Bigint &b)const
    { return cp3(b)<0; }
    bool operator<=(const Bigint &b)const
    { return cp3(b)<=0; }
    bool operator==(const Bigint &b)const
    { return cp3(b)==0; }
    bool operator!=(const Bigint &b)const
    { return cp3(b)!=0; }
```

```cpp
    bool operator>(const Bigint &b)const
    { return cp3(b)>0; }
    bool operator>=(const Bigint &b)const
    { return cp3(b)>=0; }
    Bigint operator - () const {
        Bigint r = (*this);
        r.s = -r.s;
        return r;
    }
    Bigint operator + (const Bigint &b) const {
        if (s == -1) return -(-(*this)+(-b));
        if (b.s == -1) return (*this)-(-b);
        Bigint r;
        int nl = max(len(), b.len());
        r.resize(nl + 1);
        for (int i=0; i<nl; i++) {
            if (i < len()) r.v[i] += v[i];
            if (i < b.len()) r.v[i] += b.v[i];
            if(r.v[i] >= BIGMOD) {
                r.v[i+1] += r.v[i] / BIGMOD;
                r.v[i] %= BIGMOD;
            }
        }
        r.n();
        return r;
    }
    Bigint operator - (const Bigint &b) const {
        if (s == -1) return -(-(*this)-(-b));
        if (b.s == -1) return (*this)+(-b);
        if ((*this) < b) return -(b-(*this));
        Bigint r;
        r.resize(len());
        for (int i=0; i<len(); i++) {
            r.v[i] += v[i];
            if (i < b.len()) r.v[i] -= b.v[i];
            if (r.v[i] < 0) {
                r.v[i] += BIGMOD;
                r.v[i+1]--;
            }
        }
        r.n();
        return r;
    }
    Bigint operator * (const Bigint &b) {
        Bigint r;
        r.resize(len() + b.len() + 1);
        r.s = s * b.s;
        for (int i=0; i<len(); i++) {
            for (int j=0; j<b.len(); j++) {
                r.v[i+j] += v[i] * b.v[j];
                if(r.v[i+j] >= BIGMOD) {
                    r.v[i+j+1] += r.v[i+j] / BIGMOD;
                    r.v[i+j] %= BIGMOD;
                }
            }
        }
        r.n();
        return r;
    }
    Bigint operator / (const Bigint &b) {
        Bigint r;
        r.resize(max(1, len()-b.len()+1));
        int oriS = s;
        Bigint b2 = b; // b2 = abs(b)
        s = b2.s = r.s = 1;
        for (int i=r.len()-1; i>=0; i--) {
            int d=0, u=BIGMOD-1;
            while(d<u) {
                int m = (d+u+1)>>1;
                r.v[i] = m;
                if((r*b2) > (*this)) u = m-1;
                else d = m;
            }
            r.v[i] = d;
        }
        s = oriS;
        r.s = s * b.s;
        r.n();
        return r;
    }
    Bigint operator % (const Bigint &b) {
        return (*this)-(*this)/b*b;
    }
};
```

7.2 分數

```cpp
class Rational
{
  friend istream &operator>>(istream &, Rational & );
  friend ostream &operator<<(ostream &, const Rational & );
public:
  Rational()  //constructor one
  {
    m_numeitor = 0;
    m_denominator = 1;
  }
  Rational(int a, int b)  //constructor two
  {
    if (b < 0 || b == 0)  //avoids negative denominators. && prevents a 0 denominator
    {
      cout << "This Rational number can't be used.\n\n";
      m_numeitor = 0;
      m_denominator = 0;
    }
    else
    {
      cout << "This Rational number can be used.\n\n";
      m_numeitor = a;
      m_denominator = b;
    }
  }
  Rational operator+(const Rational& a);  //加
  Rational operator-(const Rational& a);  //減
  Rational operator*(const Rational& a);  //乘
  Rational operator/(const Rational& a);  //除
  bool operator==(const Rational& a);   //相等
  void reduce();  //化簡
private:
  int m_numeitor;
  int m_denominator;
};
istream &operator>>(istream &input, Rational &test )
{
  char temp;

  input >> test.m_numeitor;
  input >> temp;
  input >> test.m_denominator;
  Rational final(test.m_numeitor, test.m_denominator);  //final用來告訴使用者這數字符不符合!
  if (test.m_denominator < 0 || test.m_denominator == 0)   //不符合(再輸入一次)
  {
    while (test.m_denominator < 0 || test.m_denominator == 0)   //有可能輸入的東西還是不符合,所以用迴圈
    {
      cout << "Enter another Rational number (n/d): ";
      input >> test.m_numeitor;
      input >> temp;
      input >> test.m_denominator;
      Rational final(test.m_numeitor, test.m_denominator);  //final用來告訴使用者這數字符不符合!
    }
    return input;
  }
  else
    return input;
}
ostream &operator<<(ostream &output, const Rational &test )
{
  output << test.m_numeitor;
  if(test.m_numeitor == 0)
    return output;
  if (test.m_denominator == 1)
    return output;
  else
  {
    output << "/";
    output << test.m_denominator;
  }
  return output;
}
Rational Rational::operator+(const Rational& a)
{
  Rational c;
  c.m_denominator = this->m_denominator * a.m_denominator;  //通分(同乘)
  c.m_numeitor = (this->m_numeitor * a.m_denominator) + (a.m_numeitor * this->m_denominator);
  c.reduce();
  return c;
}
Rational Rational::operator-(const Rational& a)
{
  Rational c;
  c.m_denominator = this->m_denominator * a.m_denominator;
  c.m_numeitor = (this->m_numeitor * a.m_denominator) - (a.m_numeitor * this->m_denominator);
  c.reduce();
  return c;
}
Rational Rational::operator*(const Rational& a)
{
  Rational c;
  c.m_denominator = this->m_denominator * a.m_denominator;
  c.m_numeitor = this->m_numeitor * a.m_numeitor;
  c.reduce();
  return c;
}
Rational Rational::operator/(const Rational& a)
{
  Rational c;
  c.m_denominator = this->m_denominator * a.m_numeitor;
  c.m_numeitor = this->m_numeitor * a.m_denominator;
  c.reduce();
  return c;
}
bool Rational::operator==(const Rational& a)
{
  if (m_numeitor == a.m_numeitor)
  {
    if (m_denominator == a.m_denominator)
      return true;
    else
      return false;
  }
  else
    return false;
}
void Rational::reduce()
{
  int i;
  int max;
  if(m_numeitor> m_denominator)
    max = m_numeitor;
  else
    max = m_denominator;
  for (i = 2; i <= max; i++)
  {
    if (m_denominator % i == 0 && m_numeitor % i == 0)
    {
      m_denominator /= i;
      m_numeitor /= i;
      i = 1;
      max = m_denominator;
      continue;
    }
  }
}
```

ACM ICPC Team Reference - Angry Crow Takes Flight!

Contents