

# 1 Basic

## 1.1 data range

```
1 int (-2147483648 to 2147483647)
2 unsigned int(0 to 4294967295)
3 long(-2147483648 to 2147483647)
4 unsigned long(0 to 4294967295)
5 long long(-9223372036854775808 to
6 9223372036854775807)
7 unsigned long long (0 to
8 18446744073709551615)
```

## 1.2 IO\_fast

```
1 void io()
2 {
3     ios::sync_with_stdio(false);
4     cin.tie(nullptr);
5     cout.tie(nullptr);
6 }
```

## 1.3 常忘記

```
1 round(double f); // 四捨五入
2 ceil(double f); // 無條件捨去
3 floor(double f); // 無條件進入
4
5 /*queue*/
6 queue<datatype> q;
7 front(); /*取出最前面的值(沒有移除掉!!)*/
8 back(); /*取出最後面的值(沒有移除掉!!)*/
9 pop(); /*移掉最前面的值*/
10 push(); /*新增值到最後面*/
11 empty(); /*回傳bool,檢查是不是空的queue*/
12 size(); /*queue 的大小*/
13
14 /*stack*/
15 stack<datatype> s;
16 top(); /*取出最上面的值(沒有移除掉!!)*/
17 pop(); /*移掉最上面的值*/
18 push(); /*新增值到最上面*/
19 empty(); /*回傳bool,檢查是不是空的stack*/
20 size(); /*stack 的大小*/
```

# 2 DP

## 2.1 3 維 DP 思路

```
1 解題思路: dp[i][j][k]
2 i 跟 j 代表 range i ~ j 的 value
3 k 在我的理解裡是視題目的要求而定的
4 像是 Remove Boxes 當中 k 代表的是在 i 之前還
5 有多少個連續的箱子
6 所以每次區間消去的值就是(k+1) * (k+1)
7 換言之,我認為可以理解成 k 的意義就是題目今
8 天所關注的重點,就是老師說的題目所規定的
9 "運算"
```

## 2.2 Knapsack Bounded

```
1 const int N = 100, W = 100000;
2 int cost[N], weight[N], number[N];
3 int c[W + 1];
4 void knapsack(int n, int w)
5 {
6     for (int i = 0; i < n; ++i)
7     {
8         int num = min(number[i], w / weight[i]);
9         for (int k = 1; num > 0; k *= 2)
10         {
11             if (k > num)
12                 k = num;
13             num -= k;
14             for (int j = w; j >= weight[i] * k; --j)
15                 c[j] = max(c[j], c[j - weight[i] * k] + cost[i] * k);
16         }
17     }
18     cout << "Max Prince" << c[w];
19 }
```

## 2.3 Knapsack sample

```
1 int Knapsack(vector<int> weight, vector<int>
2 value, int bag_Weight)
3 {
4     // vector<int> weight = {1, 3, 4};
5     // vector<int> value = {15, 20, 30};
6     // int bagWeight = 4;
7     vector<vector<int>>> dp(weight.size(),
8 vector<int>(bagWeight + 1, 0));
9     for (int j = weight[0]; j <= bagWeight;
10 ++j)
11         dp[0][j] = value[0];
12     // weight數組的大小就是物品個數
13     for (int i = 1; i < weight.size(); i++)
14     { // 遍歷物品
15         for (int j = 0; j <= bagWeight; j++)
16         { // 遍歷背包容量
17             if (j < weight[i]) dp[i][j] = dp
18 [i - 1][j];
```

```
15         else dp[i][j] = max(dp[i - 1][j]
16 , dp[i - 1][j - weight[i]]
17 + value[i]);
18     }
19     cout << dp[weight.size() - 1][bagWeight]
20 << endl;
21 }
```

## 2.4 Knapsack Unbounded

```
1 const int N = 100, W = 100000;
2 int cost[N], weight[N];
3 int c[W + 1];
4 void knapsack(int n, int w)
5 {
6     memset(c, 0, sizeof(c));
7     for (int i = 0; i < n; ++i)
8         for (int j = weight[i]; j <= w; ++j)
9             c[j] = max(c[j], c[j - weight[i]] + cost[i]);
10     cout << "最高的價值為" << c[w];
11 }
```

## 2.5 LCIS

```
1 int LCIS_len(vector<int> arr1, vector<int>
2 arr2)
3 {
4     int n = arr1.size(), m = arr2.size();
5     vector<int> table(m, 0);
6     for (int j = 0; j < m; j++)
7         table[j] = 0;
8     for (int i = 0; i < n; i++)
9     {
10         int current = 0;
11         for (int j = 0; j < m; j++)
12         {
13             if (arr1[i] == arr2[j])
14                 if (current + 1 > table[j])
15                     table[j] = current + 1;
16
17             if (arr1[i] > arr2[j])
18                 if (table[j] > current)
19                     current = table[j];
20         }
21     }
22     int result = 0;
23     for (int i = 0; i < m; i++)
24         if (table[i] > result)
25             result = table[i];
26     return result;
27 }
```

## 2.6 LCS

```
1 int LCS(vector<string> Ans, vector<string>
2 num)
3 {
4     int N = Ans.size(), M = num.size();
5     vector<vector<int>>> LCS(N + 1, vector<
6 int>(M + 1, 0));
7     for (int i = 1; i <= N; ++i)
8     {
9         for (int j = 1; j <= M; ++j)
10         {
11             if (Ans[i - 1] == num[j - 1])
12                 LCS[i][j] = LCS[i - 1][j - 1] + 1;
13             else
14                 LCS[i][j] = max(LCS[i - 1][j]
15 , LCS[i][j - 1]);
16         }
17     }
18     cout << LCS[N][M] << '\n';
19     //列印 LCS
20     int n = N, m = M;
21     vector<string> k;
22     while (n && m)
23     {
24         if (LCS[n][m] != max(LCS[n - 1][m],
25 LCS[n][m - 1]))
26         {
27             k.push_back(Ans[n - 1]);
28             n--;
29             m--;
30         }
31         else if (LCS[n][m] == LCS[n - 1][m])
32             n--;
33         else if (LCS[n][m] == LCS[n][m - 1])
34             m--;
35     }
36     reverse(k.begin(), k.end());
37     for (auto i : k)
38         cout << i << " ";
39     cout << endl;
40     return LCS[N][M];
41 }
```

## 2.7 LIS

```
1 void getMaxElementAndPos(vector<int> &LISTb1
2 , vector<int> &LISlen, int tNum, int
3 tlen, int tStart, int &num, int &pos)
4 {
5     int max = numeric_limits<int>::min();
6     int maxPos;
7     for (int i = tStart; i >= 0; i--)
8     {
9         if (LISlen[i] == tlen && LISTb1[i] <
10 tNum)
11         {
12             if (LISTb1[i] > max)
13             {
14                 max = LISTb1[i];
15                 maxPos = i;
16             }
17         }
18     }
19 }
```

```

16 num = max;
17 pos = maxPos;
18 }
19 int LIS(vector<int> &LISTbl)
20 {
21     if (LISTbl.size() == 0)
22         return 0;
23     vector<int> LISlen(LISTbl.size(), 1);
24     for (int i = 1; i < LISTbl.size(); i++)
25         for (int j = 0; j < i; j++)
26             if (LISTbl[j] < LISTbl[i])
27                 LISlen[i] = max(LISlen[i],
28                                 LISlen[j] + 1);
29     int maxlen = *max_element(LISlen.begin(),
30                               LISlen.end());
31     int num, pos;
32     vector<int> buf;
33     getMaxElementAndPos(LISTbl, LISlen,
34                          numeric_limits<int>::max(), maxlen,
35                          LISTbl.size() - 1, num, pos);
36     buf.push_back(num);
37     for (int len = maxlen - 1; len >= 1; len--)
38     {
39         int tnum = num;
40         int tpos = pos;
41         getMaxElementAndPos(LISTbl, LISlen,
42                              tnum, len, tpos - 1, num, pos);
43         buf.push_back(num);
44     }
45     reverse(buf.begin(), buf.end());
46     for (int k = 0; k < buf.size(); k++) // 列印
47         cout << buf[k] << endl;
48     return maxlen;
49 }

```

## 2.8 LPS

```

1 void LPS(string s)
2 {
3     int maxlen = 0, l, r;
4     int n = s.size();
5     for (int i = 0; i < n; i++)
6     {
7         int x = 0;
8         while ((s[i - x] == s[i + x]) && (i - x >= 0) && (i + x < n)) //odd length
9             x++;
10        x--;
11        if (2 * x + 1 > maxlen)
12        {
13            maxlen = 2 * x + 1;
14            l = i - x;
15            r = i + x;
16        }
17        x = 0;

```

```

18        while ((s[i - x] == s[i + 1 + x]) &&
19               (i - x >= 0) && (i + 1 + x < n)) //even length
20            x++;
21        if (2 * x > maxlen)
22        {
23            maxlen = 2 * x;
24            l = i - x + 1;
25            r = i + x;
26        }
27        cout << maxlen << '\n'; // 最後長度
28        cout << l + 1 << ' ' << r + 1 << '\n'; // 頭到尾
29    }

```

## 2.9 Max\_subarray

```

1 /*Kadane's algorithm*/
2 int maxSubArray(vector<int>& nums) {
3     int local_max = nums[0], global_max =
4         nums[0];
5     for (int i = 1; i < nums.size(); i++){
6         local_max = max(nums[i], local_max +
7                           nums[i]);
8         global_max = max(local_max,
9                           global_max);
10    }
11    return global_max;

```

## 2.10 Money problem

```

1 //能否湊得某個價位
2 void change(vector<int> price, int limit)
3 {
4     vector<bool> c(limit + 1, 0);
5     c[0] = true;
6     for (int i = 0; i < price.size(); ++i)
7         // 依序加入各種面額
8         for (int j = price[i]; j <= limit; ++j)
9             // 由低價位逐步到高價位
10            c[j] = c[j] | c[j - price[i]];
11        // 湊、湊、湊
12        if (c[limit]) cout << "YES\n";
13        else cout << "NO\n";
14    }
15    // 湊得某個價位的湊法總共幾種
16    void change(vector<int> price, int limit)
17    {
18        vector<int> c(limit + 1, 0);
19        c[0] = true;
20        for (int i = 0; i < price.size(); ++i)
21            for (int j = price[i]; j <= limit; ++j)
22                c[j] += c[j - price[i]];
23        cout << c[limit] << '\n';
24    }

```

```

25 // 湊得某個價位的最少錢幣用量
26 void change(vector<int> price, int limit)
27 {
28     vector<int> c(limit + 1, 0);
29     c[0] = true;
30     for (int i = 0; i < price.size(); ++i)
31         for (int j = price[i]; j <= limit; ++j)
32             c[j] = min(c[j], c[j - price[i]] + 1);
33     cout << c[limit] << '\n';
34 }
35 //湊得某個價位的錢幣用量，有哪幾種可能性
36 void change(vector<int> price, int limit)
37 {
38     vector<int> c(limit + 1, 0);
39     c[0] = true;
40     for (int i = 0; i < price.size(); ++i)
41         for (int j = price[i]; j <= limit; ++j)
42             c[j] |= c[j - price[i]] << 1; // 錢幣數量加一，每一種可能性都加一
43     for (int i = 1; i <= 63; ++i)
44         if (c[i] & (1 << i))
45             cout << "用 " << i << " 個錢幣可湊得價位 " << i;

```

## 3 Flow & matching

### 3.1 Edmonds\_karp

```

1 /*Flow - Edmonds-karp*/
2 /*Based on UVa820*/
3 #include<bits/stdc++.h>
4 #define inf 1000000;
5 using namespace std;
6
7 int getMaxFlow(vector<vector<int>> &capacity,
8               int s, int t, int n){
9     int ans = 0;
10    vector<vector<int>> residual(n+1, vector<int>(n+1, 0)); //residual network
11    while(true){
12        vector<int> bottleneck(n+1, 0);
13        bottleneck[s] = inf;
14        queue<int> q;
15        q.push(s);
16        vector<int> pre(n+1, 0);
17        while(!q.empty() && bottleneck[t] == 0){
18            int cur = q.front();
19            q.pop();
20            for (int i = 1; i <= n; i++){
21                if (bottleneck[i] == 0 && capacity[cur][i] > residual[cur][i]){
22                    q.push(i);
23                    pre[i] = cur;

```

```

24                bottleneck[i] = min(bottleneck[cur], capacity[cur][i] - residual[cur][i]);
25            }
26        }
27        if (bottleneck[t] == 0) break;
28        for (int cur = t; cur != s; cur = pre[cur]){
29            residual[pre[cur]][cur] += bottleneck[t];
30            residual[cur][pre[cur]] -= bottleneck[t];
31        }
32        ans += bottleneck[t];
33    }
34    return ans;
35 }
36 int main(){
37     int testcase = 1;
38     int n;
39     while (cin >> n){
40         if (n == 0) break;
41         vector<vector<int>> capacity(n+1, vector<int>(n+1, 0));
42         int s, t, c;
43         cin >> s >> t >> c;
44         int a, b, bandwidth;
45         for (int i = 0; i < c; ++i){
46             cin >> a >> b >> bandwidth;
47             capacity[a][b] += bandwidth;
48             capacity[b][a] += bandwidth;
49         }
50         cout << "Network " << testcase++ << endl;
51         ;
52         cout << "The bandwidth is " <<
53             getMaxFlow(capacity, s, t, n) << ".\n";
54         cout << endl;
55     }
56     return 0;

```

### 3.2 maximum\_matching

```

1 /*bipartite - maximum matching*/
2 #include<bits/stdc++.h>
3 using namespace std;
4 bool dfs(vector<vector<bool>> &res, int node,
5         vector<int> &x, vector<int> &y, vector<bool> &pass){
6     for (int i = 0; i < res[0].size(); i++){
7         if (res[node][i] && !pass[i]){
8             pass[i] = true;
9             if (y[i] == -1 || dfs(res, y[i], x, y, pass)){
10                 x[node] = i;
11                 y[i] = node;
12                 return true;
13             }
14         }
15     }
16     return false;

```

```

15     return false;
16 }
17 int main(){
18     int n,m,l;
19     while(cin>>n>>m>>l){
20         vector<vector<bool>> res(n, vector<
21             bool>(m, false));
22         for (int i = 0; i < l; i++){
23             int a, b;
24             cin >> a >> b;
25             res[a][b] = true;
26         }
27         int ans = 0;
28         vector<int> x(n, -1);
29         vector<int> y(n, -1);
30         for (int i = 0; i < n; i++){
31             vector<bool> pass(n, false);
32             if(dfs(res,i,x,y,pass))
33                 ans += 1;
34         }
35         cout << ans << endl;
36     }
37     return 0;
38 }
39 /*
40 4 3 5 //n matching m, l links
41 0 0
42 0 2
43 1 0
44 2 1
45 3 1
46 answer is 3
47 */

```

### 3.3 MFlow Model

```

1 typedef long long ll;
2 struct MF
3 {
4     static const int N = 5000 + 5;
5     static const int M = 60000 + 5;
6     static const ll oo = 1000000000000LL;
7
8     int n, m, s, t, tot, tim;
9     int first[N], next[M];
10    int u[M], v[M], cur[N], vi[N];
11    ll cap[M], flow[M], dis[N];
12    int que[N + N];
13
14    void Clear()
15    {
16        tot = 0;
17        tim = 0;
18        for (int i = 1; i <= n; ++i)
19            first[i] = -1;
20    }
21    void Add(int from, int to, ll cp, ll flw)
22    {
23        u[tot] = from;
24        v[tot] = to;
25        cap[tot] = cp;

```

```

26        flow[tot] = flw;
27        next[tot] = first[u[tot]];
28        first[u[tot]] = tot;
29        ++tot;
30    }
31    bool bfs()
32    {
33        ++tim;
34        dis[s] = 0;
35        vi[s] = tim;
36
37        int head, tail;
38        head = tail = 1;
39        que[head] = s;
40        while (head <= tail)
41        {
42            for (int i = first[que[head]]; i
43                != -1; i = next[i])
44            {
45                if (vi[v[i]] != tim && cap[i]
46                    > flow[i])
47                {
48                    vi[v[i]] = tim;
49                    dis[v[i]] = dis[que[head]]
50                        + 1;
51                    que[++tail] = v[i];
52                }
53            }
54            ++head;
55            return vi[t] == tim;
56        }
57        ll dfs(int x, ll a)
58        {
59            if (x == t || a == 0)
60                return a;
61            ll flw = 0, f;
62            int &i = cur[x];
63            for (i = first[x]; i != -1; i = next
64                [i])
65            {
66                if (dis[x] + 1 == dis[v[i]] && (
67                    f = dfs(v[i], min(a, cap[i]
68                        - flow[i]))) > 0)
69                {
70                    flow[i] += f;
71                    flow[i ^ 1] -= f;
72                    a -= f;
73                    flw += f;
74                    if (a == 0)
75                        break;
76                }
77            }
78            return flw;
79        }
80        ll MaxFlow(int s, int t)
81        {
82            this->s = s;
83            this->t = t;
84            ll flw = 0;
85            while (bfs())
86            {
87                for (int i = 1; i <= n; ++i)
88                    cur[i] = 0;
89                flw += dfs(s, oo);
90            }
91        }

```

```

86         return flw;
87     }
88 };
89 // MF Net;
90 // Net.n = n;
91 // Net.Clear();
92 // a 到 b (注意從1開始!!!!)
93 // Net.Add(a, b, w, 0);
94 // Net.MaxFlow(s, d)
95 // s 到 d 的 MF

```

## 4 Geometry

### 4.1 Line

```

1 template <typename T>
2 struct line
3 {
4     line() {}
5     point<T> p1, p2;
6     T a, b, c; //ax+by+c=0
7     line(const point<T> &x, const point<T> &
8         y) : p1(x), p2(y) {}
9     void pton()
10    { // 轉成一般式
11        a = p1.y - p2.y;
12        b = p2.x - p1.x;
13        c = -a * p1.x - b * p1.y;
14    }
15    T ori(const point<T> &p) const
16    { // 點和有向直線的關係 · >0左邊、=0在線上
17        <0右邊
18        return (p2 - p1).cross(p - p1);
19    }
20    T btw(const point<T> &p) const
21    { // 點投影落在線段上<=0
22        return (p1 - p).dot(p2 - p);
23    }
24    bool point_on_segment(const point<T> &p)
25    const
26    { // 點是否在線段上
27        return ori(p) == 0 && btw(p) <= 0;
28    }
29    T dis2(const point<T> &p, bool
30        is_segment = 0) const
31    { // 點跟直線/線段的距離平方
32        point<T> v = p2 - p1, v1 = p - p1;
33        if (is_segment)
34        {
35            point<T> v2 = p - p2;
36            if (v.dot(v1) <= 0)
37                return v1.abs2();
38            if (v.dot(v2) >= 0)
39                return v2.abs2();
40        }
41        T tmp = v.cross(v1);
42        return tmp * tmp / v.abs2();
43    }
44    T seg_dis2(const line<T> &l) const

```

```

41 { //兩線段距離平方
42     return min({dis2(l.p1, 1), dis2(l.p2
43         , 1), l.dis2(p1, 1), l.dis2(p2,
44             1)});
45 }
46 point<T> projection(const point<T> &p)
47 const
48 { // 點對直線的投影
49     point<T> n = (p2 - p1).normal();
50     return p - n * (p - p1).dot(n) / n.
51         abs2();
52 }
53 point<T> mirror(const point<T> &p) const
54 {
55     // 點對直線的鏡射 · 要先呼叫 pton 轉成一般式
56     point<T> R;
57     T d = a * a + b * b;
58     R.x = (b * b * p.x - a * a * p.x - 2
59         * a * b * p.y - 2 * a * c) / d;
60     R.y = (a * a * p.y - b * b * p.y - 2
61         * a * b * p.x - 2 * b * c) / d;
62     return R;
63 }
64 bool equal(const line &l) const
65 { // 直線相等
66     return ori(l.p1) == 0 && ori(l.p2)
67         == 0;
68 }
69 bool parallel(const line &l) const
70 {
71     return (p1 - p2).cross(l.p1 - l.p2)
72         == 0;
73 }
74 bool cross_seg(const line &l) const
75 {
76     return (p2 - p1).cross(l.p1 - p1) *
77         (p2 - p1).cross(l.p2 - p1) <= 0;
78     // 直線是否交線段
79 }
80 int line_intersect(const line &l) const
81 { // 直線相交情況 · -1無限多點、1交於一
82     點、0不相交
83     return parallel(l) ? (ori(l.p1) == 0
84         ? -1 : 0) : 1;
85 }
86 int seg_intersect(const line &l) const
87 {
88     T c1 = ori(l.p1), c2 = ori(l.p2);
89     T c3 = l.ori(p1), c4 = l.ori(p2);
90     if (c1 == 0 && c2 == 0)
91     { // 共線
92         bool b1 = btw(l.p1) >= 0, b2 =
93             btw(l.p2) >= 0;
94         T a3 = l.btw(p1), a4 = l.btw(p2);
95         if (b1 && b2 && a3 == 0 && a4 >=
96             0)
97             return 2;
98         if (b1 && b2 && a3 >= 0 && a4 ==
99             0)
100             return 3;
101         if (b1 && b2 && a3 >= 0 && a4 >=
102             0)

```

```

87         return 0;
88         return -1; //無限交點
89     }
90     else if (c1 * c2 <= 0 && c3 * c4 <= 0)
91         return 1;
92     return 0; //不相交
93 }
94 point<T> line_intersection(const line &l
95 ) const
96 { /*直線交點*/
97     point<T> a = p2 - p1, b = l.p2 - l.
98     p1, s = l.p1 - p1;
99     //if(a.cross(b)==0)return INF;
100     return p1 + a * (s.cross(b) / a.
101         cross(b));
102 }
103 point<T> seg_intersection(const line &l
104 ) const
105 { //線段交點
106     int res = seg_intersect(l);
107     if (res <= 0)
108         assert(0);
109     if (res == 2)
110         return p1;
111     if (res == 3)
112         return p2;
113     return line_intersection(l);
114 }
115 };

```

```

29         return x * b.x + y * b.y;
30     }
31     T cross(const point &b) const
32     {
33         return x * b.y - y * b.x;
34     }
35     point normal() const
36     { //求法向量
37         return point(-y, x);
38     }
39     T abs2() const
40     { //向量長度的平方
41         return dot(*this);
42     }
43     T rad(const point &b) const
44     { //兩向量的弧度
45         return fabs(atan2(fabs(cross(b)),
46             dot(b)));
47     }
48     T getA() const
49     { //對x軸的弧度
50         T A = atan2(y, x); //超過180度會變負
51         if (A <= -PI / 2)
52             A += PI * 2;
53         return A;
54     }
55 };

```

### 4.3 Polygon

### 4.2 Point

```

1 template <typename T>
2 struct point
3 {
4     T x, y;
5     point() {}
6     point(const T &x, const T &y) : x(x), y(y) {}
7     point operator+(const point &b) const
8     {
9         return point(x + b.x, y + b.y);
10    }
11    point operator-(const point &b) const
12    {
13        return point(x - b.x, y - b.y);
14    }
15    point operator*(const T &b) const
16    {
17        return point(x * b, y * b);
18    }
19    point operator/(const T &b) const
20    {
21        return point(x / b, y / b);
22    }
23    bool operator==(const point &b) const
24    {
25        return x == b.x && y == b.y;
26    }
27    T dot(const point &b) const
28    {

```

```

1 template <typename T>
2 struct polygon
3 {
4     polygon() {}
5     vector<point<T>> p; //逆時針順序
6     T area() const
7     { //面積
8         T ans = 0;
9         for (int i = p.size() - 1, j = 0; j
10             < (int)p.size(); i = j++)
11             ans += p[i].cross(p[j]);
12         return ans / 2;
13    }
14    point<T> center_of_mass() const
15    { //重心
16        T cx = 0, cy = 0, w = 0;
17        for (int i = p.size() - 1, j = 0; j
18            < (int)p.size(); i = j++)
19        {
20            T a = p[i].cross(p[j]);
21            cx += (p[i].x + p[j].x) * a;
22            cy += (p[i].y + p[j].y) * a;
23            w += a;
24        }
25        return point<T>(cx / 3 / w, cy / 3 /
26            w);
27    }
28    char ahas(const point<T> &t) const
29    { //點是否在簡單多邊形內，是的話回傳1、
30        在邊上回傳-1、否則回傳0

```

```

27 bool c = 0;
28 for (int i = 0, j = p.size() - 1; i
29     < p.size(); j = i++)
30     if (line<T>(p[i], p[j]).
31         point_on_segment(t))
32         return -1;
33     else if ((p[i].y > t.y) != (p[j]
34         .y > t.y) &&
35         t.x < (p[j].x - p[i].x)
36             * (t.y - p[i].y) /
37             (p[j].y - p[i].y)
38             + p[i].x)
39         c = !c;
40     return c;
41 }
42 char point_in_convex(const point<T> &x)
43 const
44 {
45     int l = 1, r = (int)p.size() - 2;
46     while (l <= r)
47     { //點是否在凸多邊形內，是的話回傳1
48         、在邊上回傳-1、否則回傳0
49         int mid = (l + r) / 2;
50         T a1 = (p[mid] - p[0]).cross(x -
51             p[0]);
52         T a2 = (p[mid + 1] - p[0]).cross
53             (x - p[0]);
54         if (a1 >= 0 && a2 <= 0)
55         {
56             T res = (p[mid + 1] - p[mid]
57                 ).cross(x - p[mid]);
58             return res > 0 ? 1 : (res >=
59                 0 ? -1 : 0);
60         }
61         else if (a1 < 0)
62             r = mid - 1;
63         else
64             l = mid + 1;
65     }
66     return 0;
67 }
68 vector<T> getA() const
69 { //凸包邊對x軸的夾角
70     vector<T> res; //一定是遞增的
71     for (size_t i = 0; i < p.size(); ++i
72         )
73         res.push_back((p[(i + 1) % p.
74             size()] - p[i]).getA());
75     return res;
76 }
77 bool line_intersect(const vector<T> &A,
78     const line<T> &l) const
79 { //O(logN)
80     int f1 = upper_bound(A.begin(), A.
81         end(), (l.p1 - l.p2).getA()) - A.
82         begin();
83     int f2 = upper_bound(A.begin(), A.
84         end(), (l.p2 - l.p1).getA()) - A.
85         begin();
86     return l.cross_seg(line<T>(p[f1], p[
87         f2]));
88 }
89 polygon cut(const line<T> &l) const
90 { //凸包對直線切割，得到直線l左側的凸包
91     polygon ans;

```

```

for (int n = p.size(), i = n - 1, j
    = 0; j < n; i = j++)
{
    if (l.ori(p[i]) >= 0)
    {
        ans.p.push_back(p[i]);
        if (l.ori(p[j]) < 0)
            ans.p.push_back(l.
                line_intersection(
                    line<T>(p[i], p[j])));
    }
    else if (l.ori(p[j]) > 0)
        ans.p.push_back(l.
            line_intersection(line<T>
                (p[i], p[j])));
}
return ans;
}
static bool graham_cmp(const point<T> &a
    , const point<T> &b)
{ //凸包排序函數 // 起點點不同
// return (a.x < b.x) || (a.x == b.x
    && a.y < b.y); //最左下角開始
return (a.y < b.y) || (a.y == b.y &&
    a.x < b.x); //Y最小開始
}
void graham(vector<point<T>> &s)
{ //凸包 Convexhull 2D
    sort(s.begin(), s.end(), graham_cmp);
    ;
    p.resize(s.size() + 1);
    int m = 0;
    // cross >= 0 順時針 * cross <= 0 逆
    時針旋轉
    for (size_t i = 0; i < s.size(); ++i
        )
    {
        while (m >= 2 && (p[m - 1] - p[m
            - 2]).cross(s[i] - p[m -
            2]) <= 0)
            --m;
        p[m++] = s[i];
    }
    for (int i = s.size() - 2, t = m +
        1; i >= 0; --i)
    {
        while (m >= t && (p[m - 1] - p[m
            - 2]).cross(s[i] - p[m -
            2]) <= 0)
            --m;
        p[m++] = s[i];
    }
    if (s.size() > 1) // 重複頭一次需扣
        掉
        --m;
    p.resize(m);
}
T diam()
{ //直徑
    int n = p.size(), t = 1;
    T ans = 0;
    p.push_back(p[0]);
    for (int i = 0; i < n; i++)

```

```

118 {
119     point<T> now = p[i + 1] - p[i];
120     while (now.cross(p[t + 1] - p[i]) > now.cross(p[t] - p[i]))
121         t = (t + 1) % n;
122     ans = max(ans, (p[i] - p[t]).abs2());
123 }
124 return p.pop_back(), ans;
125 }
126 T min_cover_rectangle()
127 { //最小覆蓋矩形
128     int n = p.size(), t = 1, r = 1, l;
129     if (n < 3)
130         return 0; //也可以做最小周長矩形
131     T ans = 1e99;
132     p.push_back(p[0]);
133     for (int i = 0; i < n; i++)
134     {
135         point<T> now = p[i + 1] - p[i];
136         while (now.cross(p[t + 1] - p[i]) > now.cross(p[t] - p[i]))
137             t = (t + 1) % n;
138         while (now.dot(p[r + 1] - p[i]) > now.dot(p[r] - p[i]))
139             r = (r + 1) % n;
140         if (!i)
141             l = r;
142         while (now.dot(p[l + 1] - p[i]) <= now.dot(p[l] - p[i]))
143             l = (l + 1) % n;
144         T d = now.abs2();
145         T tmp = now.cross(p[t] - p[i]) * (now.dot(p[r] - p[i]) - now.dot(p[l] - p[i])) / d;
146         ans = min(ans, tmp);
147     }
148     return p.pop_back(), ans;
149 }
150 T dis2(polygon &p1)
151 { //凸包最近距離平方
152     vector<point<T>> &P = p, &Q = p1.p;
153     int n = P.size(), m = Q.size(), l = 0, r = 0;
154     for (int i = 0; i < n; ++i)
155         if (P[i].y < P[l].y)
156             l = i;
157     for (int i = 0; i < m; ++i)
158         if (Q[i].y < Q[r].y)
159             r = i;
160     P.push_back(P[0]), Q.push_back(Q[0]);
161     T ans = 1e99;
162     for (int i = 0; i < n; ++i)
163     {
164         while ((P[l] - P[l + 1]).cross(Q[r + 1] - Q[r]) < 0)
165             r = (r + 1) % m;
166         ans = min(ans, line<T>(P[l], P[l + 1]).seg_dis2(line<T>(Q[r], Q[r + 1])));
167         l = (l + 1) % n;
168     }
169     return P.pop_back(), Q.pop_back(), ans;
170 }
171 static char sign(const point<T> &t)
172 {
173     return (t.y == 0 ? t.x : t.y) < 0;
174 }
175 static bool angle_cmp(const line<T> &A, const line<T> &B)
176 {
177     point<T> a = A.p2 - A.p1, b = B.p2 - B.p1;
178     return sign(a) < sign(b) || (sign(a) == sign(b) && a.cross(b) > 0);
179 }
180 int halfplane_intersection(vector<line<T>> &s)
181 { //半平面交
182     sort(s.begin(), s.end(), angle_cmp);
183     //線段左側為該線段半平面
184     int L, R, n = s.size();
185     vector<point<T>> px(n);
186     vector<line<T>> q(n);
187     q[L = R = 0] = s[0];
188     for (int i = 1; i < n; ++i)
189     {
190         while (L < R && s[i].ori(px[R - 1]) <= 0)
191             --R;
192         while (L < R && s[i].ori(px[L]) <= 0)
193             ++L;
194         q[++R] = s[i];
195         if (q[R].parallel(q[R - 1]))
196         {
197             --R;
198             if (q[R].ori(s[i].p1) > 0)
199                 q[R] = s[i];
200         }
201         if (L < R)
202             px[R - 1] = q[R - 1].line_intersection(q[R]);
203     }
204     while (L < R && q[L].ori(px[R - 1]) <= 0)
205         --L;
206     p.clear();
207     if (R - L <= 1)
208         return 0;
209     px[R] = q[R].line_intersection(q[L]);
210     for (int i = L; i <= R; ++i)
211         p.push_back(px[i]);
212     return R - L + 1;
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

#### 4.4 Triangle

```

1 template <typename T>
2 struct triangle
3 {
4     point<T> a, b, c;
5     triangle() {}

```

```

15 triangle(const point<T> &a, const point<
16 T> &b, const point<T> &c) : a(a), b(b),
17 c(c) {}
18 T area() const
19 {
20     T t = (b - a).cross(c - a) / 2;
21     return t > 0 ? t : -t;
22 }
23 point<T> barycenter() const
24 { //重心
25     return (a + b + c) / 3;
26 }
27 point<T> circumcenter() const
28 { //外心
29     static line<T> u, v;
30     u.p1 = (a + b) / 2;
31     u.p2 = point<T>(u.p1.x - a.y + b.y,
32 u.p1.y + a.x - b.x);
33 v.p1 = (a + c) / 2;
34 v.p2 = point<T>(v.p1.x - a.y + c.y,
35 v.p1.y + a.x - c.x);
36 return u.line_intersection(v);
37 }
38 point<T> incenter() const
39 { //內心
40     T A = sqrt((b - c).abs2()), B = sqrt(
41 ((a - c).abs2()), C = sqrt((a - b).abs2());
42 return point<T>(A * a.x + B * b.x +
43 C * c.x, A * a.y + B * b.y + C *
44 c.y) / (A + B + C);
45 }
46 point<T> perpcenter() const
47 { //垂心
48     return barycenter() * 3 -
49 circumcenter() * 2;
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }
115 }
116 }
117 }
118 }
119 }
120 }
121 }
122 }
123 }
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

## 5 Graph

### 5.1 Bellman-Ford

```

1 /*SPA - Bellman-Ford*/
2 #define inf 99999 //define by you maximum
3 edges weight
4 vector<vector<int>> > edges;
5 vector<int> dist;
6 vector<int> ancestor;
7 void BellmanFord(int start, int node){
8     dist[start] = 0;
9     for(int i = 0; i < node-1; i++){
10         for(int j = 0; j < node; j++){
11             for(int k = 0; k < node; k++){
12                 if(edges[j][k] != -1){
13                     if(dist[j] + edges[j][k] < dist[k]){
14                         dist[k] = dist[j] +
15 edges[j][k];
16                         ancestor[k] = j;
17                     }
18                 }
19             }
20         }
21     }
22 }
23 }
24 }
25 }
26 }
27 }
28 }
29 }
30 }
31 }
32 }
33 }
34 }
35 }
36 }
37 }
38 }
39 }
40 }
41 }
42 }
43 }
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }
115 }
116 }
117 }
118 }
119 }
120 }
121 }
122 }
123 }
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
```



```

20     else if (edges[i].second == q.
21         front() && pass[edges[i].
22         first] == 0)
23     {
24         p.push(edges[i].first);
25         result[edges[i].first] =
26             count;
27     }
28     else
29         newedges.push_back(edges[i])
30         ;
31     edges = newedges;
32     newedges.clear();
33     q.pop();
34     if (q.empty() == true)
35     {
36         q = p;
37         queue<int> tmp;
38         p = tmp;
39         count++;
40     }
41 }
42 int main()
43 {
44     int node;
45     cin >> node;
46     vector<pair<int, int>> edges;
47     int a, b;
48     while (cin >> a >> b)
49     {
50         /*a = b = -1 means input edges ended
51         */
52         if (a == -1 && b == -1)
53             break;
54         edges.push_back(pair<int, int>(a, b)
55         );
56     }
57     vector<int> result(node, -1);
58     BFS(result, edges, node, 0);
59     return 0;

```

### 5.3 DFS-rec

```

1  /*DFS - Recursive version*/
2  map<pair<int,int>,int> edges;
3  vector<int> pass;
4  vector<int> route;
5  void DFS(int start){
6      pass[start] = 1;
7      map<pair<int,int>,int>::iterator iter;
8      for(iter = edges.begin(); iter != edges.
9          end(); iter++){
10         if((*iter).first.first == start &&
11             (*iter).second == 0 && pass[(*)
12             iter).first.second] == 0){
13             route.push_back((*iter).first.
14                 second);
15             DFS((*iter).first.second);
16         }
17     }

```

```

13     else if((*iter).first.second ==
14         start && (*iter).second == 0 &&
15         pass[(*)iter).first.first] == 0){
16         route.push_back((*iter).first.
17             first);
18         DFS((*iter).first.first);
19     }
20 }
21 int main(){
22     int node;
23     cin>>node;
24     pass.resize(node,0);
25     int a,b;
26     while(cin>>a>>b){
27         if(a == -1 && b == -1)
28             break;
29         edges.insert(pair<pair<int,int>,int>
30             >(pair<int,int>(a,b),0));
31     }
32     int start;
33     cin>>start;
34     route.push_back(start);
35     DFS(start);
36     return 0;

```

### 5.4 Dijkstra

```

1  /*SPA - Dijkstra*/
2  #define inf INT_MAX
3  vector<vector<int>> > weight;
4  vector<int> ancestor;
5  vector<int> dist;
6  void dijkstra(int start){
7      priority_queue<pair<int,int>,vector<
8          pair<int,int>,greater<pair<int,
9              int>>> pq;
10     pq.push(make_pair(0,start));
11     while(!pq.empty()){
12         int cur = pq.top().second;
13         pq.pop();
14         for(int i = 0; i < weight[cur].size
15             (); i++){
16             if(dist[i] > dist[cur] + weight[
17                 cur][i] && weight[cur][i] !=
18                 -1){
19                 dist[i] = dist[cur] + weight
20                     [cur][i];
21                 ancestor[i] = cur;
22                 pq.push(make_pair(dist[i],i)
23                 );
24             }
25         }
26     }
27 }
28 int main(){
29     int node;
30     cin>>node;
31     int a,b,d;
32     weight.resize(node,vector<int>(node,-1))
33     ;
34     while(cin>>a>>b>>d){

```

```

27     /*input: source destination weight*/
28     if(a == -1 && b == -1 && d == -1)
29         break;
30     weight[a][b] = d;
31 }
32 ancestor.resize(node,-1);
33 dist.resize(node,inf);
34 int start;
35 cin>>start;
36 dist[start] = 0;
37 dijkstra(start);
38 return 0;
39 }

```

### 5.5 Floyd-warshall

```

1  /*SPA - Floyd-Warshall*/
2  #define inf 99999
3  void floyd_warshall(vector<vector<int>>&
4      distance, vector<vector<int>>& ancestor,
5      int n){
6      for (int k = 0; k < n; k++){
7          for (int i = 0; i < n; i++){
8              for (int j = 0; j < n; j++){
9                  if(distance[i][k] + distance
10                     [k][j] < distance[i][j])
11                  {
12                      distance[i][j] =
13                          distance[i][k] +
14                          distance[k][j];
15                      ancestor[i][j] =
16                          ancestor[k][j];
17                  }
18              }
19          }
20      }
21 }
22 int main(){
23     int n;
24     cin >> n;
25     int a, b, d;
26     vector<vector<int>> distance(n, vector<
27         int>(n,99999));
28     vector<vector<int>> ancestor(n, vector<
29         int>(n,-1));
30     while(cin>>a>>b>>d){
31         if(a == -1 && b == -1 && d == -1)
32             break;
33         distance[a][b] = d;
34         ancestor[a][b] = a;
35     }
36     for (int i = 0; i < n; i++)
37         distance[i][i] = 0;
38     floyd_warshall(distance, ancestor, n);
39     /*Negative cycle detection*/
40     for (int i = 0; i < n; i++){
41         if(distance[i][i] < 0){
42             cout << "Negative cycle!" <<
43                 endl;
44             break;
45         }
46     }
47     return 0;

```

### 5.6 Kruskal

```

1  /*mst - Kruskal*/
2  struct edges{
3      int from;
4      int to;
5      int weight;
6      friend bool operator < (edges a, edges b)
7          {
8              return a.weight > b.weight;
9          }
10 };
11 int find(int x,vector<int>& union_set){
12     if(x != union_set[x])
13         union_set[x] = find(union_set[x],
14             union_set);
15     return union_set[x];
16 }
17 void merge(int a,int b,vector<int>&
18     union_set){
19     int pa = find(a, union_set);
20     int pb = find(b, union_set);
21     if(pa != pb)
22         union_set[pa] = pb;
23 }
24 void kruskal(priority_queue<edges> pq,int n)
25 {
26     vector<int> union_set(n, 0);
27     for (int i = 0; i < n; i++)
28         union_set[i] = i;
29     int edge = 0;
30     int cost = 0; //evaluate cost of mst
31     while(!pq.empty() && edge < n - 1){
32         edges cur = pq.top();
33         int from = find(cur.from, union_set)
34         ;
35         int to = find(cur.to, union_set);
36         if(from != to){
37             merge(from, to, union_set);
38             edge += 1;
39             cost += cur.weight;
40         }
41         pq.pop();
42     }
43     if(edge < n-1)
44         cout << "No mst" << endl;
45     else
46         cout << cost << endl;
47 }
48 int main(){
49     int n;
50     cin >> n;
51     int a, b, d;
52     priority_queue<edges> pq;
53     while(cin>>a>>b>>d){
54         if(a == -1 && b == -1 && d == -1)
55             break;
56         edges tmp;
57         tmp.from = a;
58         tmp.to = b;
59         tmp.weight = d;

```

```

55     pq.push(tmp);
56 }
57 kruskal(pq, n);
58 return 0;
59 }

```

## 5.7 Prim

```

1  /*mst - Prim*/
2  #define inf 99999
3  struct edges{
4      int from;
5      int to;
6      int weight;
7      friend bool operator < (edges a, edges b
8      ){
9          return a.weight > b.weight;
10     };
11 void Prim(vector<vector<int>> gp,int n,int
12 start){
13     vector<bool> pass(n,false);
14     int edge = 0;
15     int cost = 0; //evaluate cost of mst
16     priority_queue<edges> pq;
17     for (int i = 0; i < n; i++){
18         if(gp[start][i] != inf){
19             edges tmp;
20             tmp.from = start;
21             tmp.to = i;
22             tmp.weight = gp[start][i];
23             pq.push(tmp);
24         }
25     }
26     pass[start] = true;
27     while(!pq.empty() && edge < n-1){
28         edges cur = pq.top();
29         pq.pop();
30         if(!pass[cur.to]){
31             for (int i = 0; i < n; i++){
32                 if(gp[cur.to][i] != inf){
33                     edges tmp;
34                     tmp.from = cur.to;
35                     tmp.to = i;
36                     tmp.weight = gp[cur.to][i];
37                     pq.push(tmp);
38                 }
39             }
40             pass[cur.to] = true;
41             edge += 1;
42             cost += cur.weight;
43         }
44     }
45     if(edge < n-1)
46         cout << "No mst" << endl;
47     else
48         cout << cost << endl;
49 }
50 int main(){
51     int n;
52     cin >> n;
53     int a, b, d;

```

```

53     vector<vector<int>> gp(n,vector<int>(n,
54         inf));
55     while(cin>>a>>b>>d){
56         if(a == -1 && b == -1 && d == -1)
57             break;
58         if(gp[a][b] > d)
59             gp[a][b] = d;
60     }
61     Prim(gp,n,0);
62     return 0;
63 }

```

## 5.8 Union\_find

```

1  int find(int x, vector<int> &union_set)
2  {
3      if (union_set[x] != x)
4          union_set[x] = find(union_set[x],
5          union_set); //compress path
6      return union_set[x];
7  }
8  void merge(int x, int y, vector<int> &
9  union_set, vector<int> &rank)
10 {
11     int rx, ry;
12     rx = find(x, union_set);
13     ry = find(y, union_set);
14     if (rx == ry)
15         return;
16     /*merge by rank -> always merge small
17     tree to big tree*/
18     if (rank[rx] > rank[ry])
19         union_set[ry] = rx;
20     else
21     {
22         union_set[rx] = ry;
23         if (rank[rx] == rank[ry])
24             ++rank[ry];
25     }
26 }
27 int main()
28 {
29     int node;
30     cin >> node; //Input Node number
31     vector<int> union_set(node, 0);
32     vector<int> rank(node, 0);
33     for (int i = 0; i < node; i++)
34         union_set[i] = i;
35     int edge;
36     cin >> edge; //Input Edge number
37     for (int i = 0; i < edge; i++)
38     {
39         int a, b;
40         cin >> a >> b;
41         merge(a, b, union_set, rank);
42     }
43     /*build party*/
44     vector<vector<int>> party(node, vector<
45     int>(0));
46     for (int i = 0; i < node; i++)
47         party[find(i, union_set)].push_back(
48         i);
49 }

```

## 6 Mathematics

### 6.1 Combination

```

1  /*input type string or vector*/
2  for (int i = 0; i < (1 << input.size()); ++i
3      )
4  {
5      string testCase = "";
6      for (int j = 0; j < input.size(); ++j)
7          if (i & (1 << j))
8              testCase += input[j];
9  }

```

### 6.2 Extended Euclidean

```

1  // ax + by = gcd(a,b)
2  pair<long long, long long> extgcd(long long
3  a, long long b)
4  {
5      if (b == 0)
6          return {1, 0};
7      long long k = a / b;
8      pair<long long, long long> p = extgcd(b,
9      a - k * b);
10     //cout << p.first << " " << p.second <<
11     endl;
12     //cout << "商數(k)= " << k << endl <<
13     endl;
14     return {p.second, p.first - k * p.second
15     };
16 }
17 int main()
18 {
19     int a, b;
20     cin >> a >> b;
21     pair<long long, long long> xy = extgcd(a,
22     b); //(x0,y0)
23     cout << xy.first << " " << xy.second <<
24     endl;
25     cout << xy.first << " * " << a << " + "
26     << xy.second << " * " << b << endl;
27     return 0;
28 }
29 // ax + by = gcd(a,b) * r
30 /*find |x|+|y| -> min*/
31 int main()
32 {
33     long long r, p, q; /*px+qy = r*/
34     int cases;
35     cin >> cases;
36     while (cases--)
37     {
38         cin >> r >> p >> q;
39         pair<long long, long long> xy =
40         extgcd(q, p); //(x0,y0)
41         long long ans = 0, tmp = 0;
42         double k, k1;
43         long long s, s1;

```

```

36     k = 1 - (double)(r * xy.first) / p;
37     s = round(k);
38     ans = llabs(r * xy.first + s * p) +
39     llabs(r * xy.second - s * q);
40     k1 = -(double)(r * xy.first) / p;
41     s1 = round(k1);
42     /*cout << k << endl << k1 << endl;
43     cout << s << endl << s1 << endl;
44     */
45     tmp = llabs(r * xy.first + s1 * p) +
46     llabs(r * xy.second - s1 * q);
47     ans = min(ans, tmp);
48     cout << ans << endl;
49 }
50 return 0;
51 }

```

### 6.3 Hex to Dec

```

1  int HextoDec(string num) //16 to 10
2  {
3      int base = 1;
4      int temp = 0;
5      for (int i = num.length() - 1; i = 0; i
6      --)
7      {
8          if (num[i] == '0' && num[i] == '9')
9          {
10             temp += (num[i] - 48) * base;
11             base = base * 16;
12         }
13         else if (num[i] == 'A' && num[i] == 'F'
14         || num[i] == 'a' && num[i] == 'f')
15         {
16             temp += (num[i] - 55) * base;
17             base = base * 16;
18         }
19     }
20     return temp;
21 }
22 void DecToHex(int p_intValue) //10 to 16
23 {
24     char l_pCharRes = new char;
25     sprintf(l_pCharRes, "%X", p_intValue);
26     int l_intResult = stoi(l_pCharRes);
27     cout << l_pCharRes << endl;
28     return l_intResult;
29 }

```

### 6.4 log

```

1  double mylog(double a, double base)
2  {
3      //a 的對數底數 b = 自然對數 (a) / 自然對
4      數 (b)。
5      return log(a) / log(base);
6  }

```

## 6.5 Mod

```

1 int pow_mod(int a, int n, int m) // a ^ n
  mod m;
2 { // a, n, m < 10 ^ 9
3   if (n == 0)
4     return 1;
5   int x = pow_mid(a, n / 2, m);
6   long long ans = (long long)x * x % m;
7   if (n % 2 == 1)
8     ans = ans * a % m;
9   return (int)ans;
10 }
11
12 // 加法: (a + b) % p = (a % p + b % p) % p;
13 // 減法: (a - b) % p = (a % p - b % p + p) % p;
14 // 乘法: (a * b) % p = (a % p * b % p) % p;
15 // 次方: (a ^ b) % p = ((a % p) ^ b) % p;
16 // 加法結合律: ((a + b) % p + c) % p = (a +
17   (b + c)) % p;
18 // 乘法結合律: ((a * b) % p * c) % p = (a *
19   (b * c)) % p;
20 // 加法交換律: (a + b) % p = (b + a) % p;
21 // 乘法交換律: (a * b) % p = (b * a) % p;
22 // 結合律: ((a + b) % p * c) = ((a * c) % p
23   + (b * c) % p) % p;
24
25 // 如果 a ≡ b(mod m) · 我們會說 a, b 在模 m
26   下同餘。
27 // 整除性: a ≡ b(mod m) ⇔ c ⋅ m = a - b, c
28   ⋅ Z ⇔ a ≡ b (mod m) ⇔ m|a-b
29 // 遞移性: 若 a ≡ b (mod c), b ≡ d(mod c) 則
30   a ≡ d (mod c)
31
32 ****基本運算****
33 // a ≡ b (mod m) ⇔ { a ± c ≡ b ± d (mod m) }
34 // c ≡ d (mod m) ⇔ { a * c ≡ b * d (mod m) }
35 // 放大縮小模數: k⋅Z+, a ≡ b (mod m) ⇔ k ⋅ a
36   ≡ k ⋅ b (mod k⋅m)

```

## 6.6 Permutation

```

1 // 全排列要先 sort !!!
2 // num -> vector or string
3 next_permutation(num.begin(), num.end());
4 prev_permutation(num.begin(), num.end());

```

## 6.7 PI

```

1 #define PI acos(-1)
2 #define PI_M_PI
3 const double PI = atan2(0.0, -1.0);

```

## 6.8 Prime table

```

1 const int maxn = sqrt(INT_MAX);
2 vector<int> p;
3 bitset<maxn> is_notp;
4 void PrimeTable()
5 {
6   is_notp.reset();
7   is_notp[0] = is_notp[1] = 1;
8   for (int i = 2; i <= maxn; ++i)
9   {
10    if (!is_notp[i])
11      p.push_back(i);
12    for (int j = 0; j < (int)p.size();
13        ++j)
14    {
15      if (i * p[j] > maxn)
16        break;
17      is_notp[i * p[j]] = 1;
18      if (i % p[j] == 0)
19        break;
20    }
21  }

```

## 6.9 primeBOOL

```

1 // n < 4759123141   chk = [2, 7, 61]
2 // n < 1122004669633   chk = [2, 13, 23,
3   1662803]
4 // n < 2^64   chk = [2, 325, 9375,
5   28178, 450775, 9780504, 1795265022]
6 vector<long long> chk = {};
7 long long fmul(long long a, long long n,
8   long long mod)
9 {
10   long long ret = 0;
11   for (; n >= 1)
12   {
13     if (n & 1)
14       (ret += a) %= mod;
15     (a += a) %= mod;
16   }
17   return ret;
18 }
19 long long fpow(long long a, long long n,
20   long long mod)
21 {
22   long long ret = 1LL;
23   for (; n >= 1)
24   {
25     if (n & 1)
26       ret = fmul(ret, a, mod);
27     a = fmul(a, a, mod);
28   }
29   return ret;
30 }
31 bool check(long long a, long long u, long
32   long n, int t)
33 {
34   a = fpow(a, u, n);

```

```

31 if (a == 0)
32   return true;
33 if (a == 1 || a == n - 1)
34   return true;
35 for (int i = 0; i < t; ++i)
36 {
37   a = fmul(a, a, n);
38   if (a == 1)
39     return false;
40   if (a == n - 1)
41     return true;
42 }
43 return false;
44 }
45 bool is_prime(long long n)
46 {
47   if (n < 2)
48     return false;
49   if (n % 2 == 0)
50     return n == 2;
51   long long u = n - 1;
52   int t = 0;
53   for (; u & 1; u >>= 1, ++t)
54     ;
55   for (long long i : chk)
56   {
57     if (!check(i, u, n, t))
58       return false;
59   }
60   return true;
61 }
62 // if (is_prime(int num)) // true == prime
63 // 反之亦然

```

## 6.10 Round(小數)

```

1 double myround(double number, unsigned int
2   bits)
3 {
4   LL integerPart = number;
5   number -= integerPart;
6   for (unsigned int i = 0; i < bits; ++i)
7     number *= 10;
8   number = (LL)(number + 0.5);
9   for (unsigned int i = 0; i < bits; ++i)
10     number /= 10;
11   return integerPart + number;
12 }
13 //printf("%.1f\n", round(3.4515239, 1));

```

## 6.11 二分逼近法

```

1 #define eps 1e-14
2 void half_interval()
3 {
4   double L = 0, R = /*區間*/, M;
5   while (R - L >= eps)
6   {

```

```

7   M = (R + L) / 2;
8   if (/*函數*/ > /*方程式目標*/)
9     L = M;
10   else
11     R = M;
12 }
13 printf("%.31f\n", R);
14 }

```

## 6.12 四則運算

```

1 string s = ""; //開頭是負號要補0
2 long long int DFS(int le, int ri) // (0,
3   string final index)
4 {
5   int c = 0;
6   for (int i = ri; i >= le; i--)
7   {
8     if (s[i] == '+')
9       c++;
10    if (s[i] == '(')
11      c--;
12    if (s[i] == '+' && c == 0)
13      return DFS(le, i - 1) + DFS(i +
14        1, ri);
15    if (s[i] == '-' && c == 0)
16      return DFS(le, i - 1) - DFS(i +
17        1, ri);
18  }
19  for (int i = ri; i >= le; i--)
20  {
21    if (s[i] == '+')
22      c++;
23    if (s[i] == '(')
24      c--;
25    if (s[i] == '*' && c == 0)
26      return DFS(le, i - 1) * DFS(i +
27        1, ri);
28    if (s[i] == '/' && c == 0)
29      return DFS(le, i - 1) / DFS(i +
30        1, ri);
31    if (s[i] == '%' && c == 0)
32      return DFS(le, i - 1) % DFS(i +
33        1, ri);
34  }
35  if ((s[le] == '(' && (s[ri] == ')'))
36    return DFS(le + 1, ri - 1); //去除刮
37    號
38  if (s[le] == ' ' && s[ri] == ' ')
39    return DFS(le + 1, ri - 1); //去除左
40    右兩邊空格
41  if (s[le] == ' ')
42    return DFS(le + 1, ri); //去除左邊空
43    格
44  if (s[ri] == ' ')
45    return DFS(le, ri - 1); //去除右邊空
46    格
47  long long int num = 0;
48  for (int i = le; i <= ri; i++)
49    num = num * 10 + s[i] - '0';
50  return num;

```



41 }

## 6.13 數字乘法組合

```

1 void dfs(int j, int old, int num, vector<int>
  > com, vector<vector<int>> &ans)
2 {
3     for (int i = j; i <= sqrt(num); i++)
4     {
5         if (old == num)
6             com.clear();
7         if (num % i == 0)
8         {
9             vector<int> a;
10            a = com;
11            a.push_back(i);
12            finds(i, old, num / i, a, ans);
13            a.push_back(num / i);
14            ans.push_back(a);
15        }
16    }
17    vector<vector<int>> ans;
18    vector<int> zero;
19    dfs(2, num, num, zero, ans);
20    /*/num 為 input 數字*/
21    for (int i = 0; i < ans.size(); i++)
22    {
23        for (int j = 0; j < ans[i].size() - 1; j
          ++){
24            cout << ans[i][j] << " ";
25            cout << ans[i][ans[i].size() - 1] <<
          endl;
26        }
27    }

```

## 6.14 數字加法組合

```

1 void recur(int i, int n, int m, vector<int>
  &out, vector<vector<int>> &ans)
2 {
3     if (n == 0)
4     {
5         for (int i : out)
6             if (i > m)
7                 return;
8         ans.push_back(out);
9     }
10    for (int j = i; j <= n; j++)
11    {
12        out.push_back(j);
13        recur(j, n - j, m, out, ans);
14        out.pop_back();
15    }
16 }
17 vector<vector<int>> ans;
18 vector<int> zero;
19 recur(1, num, num, zero, ans);
20 /* num 為 input 數字
21 for (int i = 0; i < ans.size(); i++)

```

```

22 {
23     for (int j = 0; j < ans[i].size() - 1; j
          ++){
24         cout << ans[i][j] << " ";
25         cout << ans[i][ans[i].size() - 1] <<
          endl;
26     }

```

## 6.15 羅馬數字

```

1 int romanToInt(string s)
2 {
3     unordered_map<char, int> T;
4     T['I'] = 1;
5     T['V'] = 5;
6     T['X'] = 10;
7     T['L'] = 50;
8     T['C'] = 100;
9     T['D'] = 500;
10    T['M'] = 1000;
11
12    int sum = T[s.back()];
13    for (int i = s.length() - 2; i >= 0; --i)
14    {
15        if (T[s[i]] < T[s[i + 1]])
16            sum -= T[s[i]];
17        else
18            sum += T[s[i]];
19    }
20    return sum;
21 }

```

## 6.16 質因數分解

```

1 void primeFactorization(int n) // 配合質數表
2 {
3     for (int i = 0; i < (int)p.size(); ++i)
4     {
5         if (p[i] * p[i] > n)
6             break;
7         if (n % p[i])
8             continue;
9         cout << p[i] << ' ';
10        while (n % p[i] == 0)
11            n /= p[i];
12    }
13    if (n != 1)
14        cout << n << ' ';
15    cout << '\n';
16 }

```

## 7 Other

### 7.1 binary search 三類變化

```

1 // 查找和目標值完全相等的數
2 int find(vector<int> &nums, int target)
3 {
4     int left = 0, right = nums.size();
5     while (left < right)
6     {
7         int mid = left + (right - left) / 2;
8         if (nums[mid] == target)
9             return mid;
10        else if (nums[mid] < target)
11            left = mid + 1;
12        else
13            right = mid;
14    }
15    return -1;
16 }
17 // 找第一個不小於目標值的數 == 找最後一個小
   於目標值的數
18 /*(lower_bound)*/
19 int find(vector<int> &nums, int target)
20 {
21     int left = 0, right = nums.size();
22     while (left < right)
23     {
24         int mid = left + (right - left) / 2;
25         if (nums[mid] < target)
26             left = mid + 1;
27        else
28            right = mid;
29    }
30    return right;
31 }
32 // 找第一個大於目標值的數 == 找最後一個不大
   於目標值的數
33 /*(upper_bound)*/
34 int find(vector<int> &nums, int target)
35 {
36     int left = 0, right = nums.size();
37     while (left < right)
38     {
39         int mid = left + (right - left) / 2;
40         if (nums[mid] <= target)
41             left = mid + 1;
42        else
43            right = mid;
44    }
45    return right;
46 }

```

### 7.2 heap sort

```

1 void MaxHeapify(vector<int> &array, int root
  , int length)
2 {
3     int left = 2 * root,
4         right = 2 * root + 1,
5         largest;
6     if (left <= length && array[left] >
        array[root])
7         largest = left;
8     else
9         largest = root;

```

```

10     if (right <= length && array[right] >
        array[largest])
11         largest = right;
12     if (largest != root)
13     {
14         swap(array[largest], array[root]);
15         MaxHeapify(array, largest, length);
16     }
17 }
18 void HeapSort(vector<int> &array)
19 {
20     array.insert(array.begin(), 0);
21     for (int i = (int)array.size() / 2; i >=
        1; i--)
22         MaxHeapify(array, i, (int)array.size()
        - 1);
23     int size = (int)array.size() - 1;
24     for (int i = (int)array.size() - 1; i >=
        2; i--)
25     {
26         swap(array[1], array[i]);
27         size--;
28         MaxHeapify(array, 1, size);
29     }
30     array.erase(array.begin());
31 }

```

### 7.3 Merge sort

```

1 void Merge(vector<int> &arr, int front, int
  mid, int end)
2 {
3     vector<int> LeftSub(arr.begin() + front,
        arr.begin() + mid + 1);
4     vector<int> RightSub(arr.begin() + mid +
        1, arr.begin() + end + 1);
5     LeftSub.insert(LeftSub.end(), INT_MAX);
6     RightSub.insert(RightSub.end(), INT_MAX);
7
8     int idxLeft = 0, idxRight = 0;
9     for (int i = front; i <= end; i++)
10    {
11        if (LeftSub[idxLeft] <= RightSub[
            idxRight])
12        {
13            arr[i] = LeftSub[idxLeft];
14            idxLeft++;
15        }
16        else
17        {
18            arr[i] = RightSub[idxRight];
19            idxRight++;
20        }
21    }
22 }
23 void MergeSort(vector<int> &arr, int front,
  int end)
24 {
25     // front = 0, end = arr.size() - 1
26     if (front < end)
27     {

```

```

29     int mid = (front + end) / 2;
30     MergeSort(arr, front, mid);
31     MergeSort(arr, mid + 1, end);
32     Merge(arr, front, mid, end);
33 }
34 }

```

## 7.4 Quick

```

1 int Partition(vector<int> &arr, int front,
  int end)
2 {
3     int pivot = arr[end];
4     int i = front - 1;
5     for (int j = front; j < end; j++)
6     {
7         if (arr[j] < pivot)
8         {
9             i++;
10            swap(arr[i], arr[j]);
11        }
12    }
13    i++;
14    swap(arr[i], arr[end]);
15    return i;
16 }
17 void QuickSort(vector<int> &arr, int front,
  int end)
18 {
19     // front = 0, end = arr.size() - 1
20     if (front < end)
21     {
22         int pivot = Partition(arr, front,
23                               end);
24         QuickSort(arr, front, pivot - 1);
25         QuickSort(arr, pivot + 1, end);
26     }
27 }

```

## 7.5 Weighted Job Scheduling

```

1 struct Job
2 {
3     int start, finish, profit;
4 };
5 bool jobComparataor(Job s1, Job s2)
6 {
7     return (s1.finish < s2.finish);
8 }
9 int latestNonConflict(Job arr[], int i)
10 {
11     for (int j = i - 1; j >= 0; j--)
12     {
13         if (arr[j].finish <= arr[i].start)
14             return j;
15     }
16     return -1;
17 }
18 int findMaxProfit(Job arr[], int n)
19 {

```

```

20     sort(arr, arr + n, jobComparataor);
21     int *table = new int[n];
22     table[0] = arr[0].profit;
23     for (int i = 1; i < n; i++)
24     {
25         int inclProf = arr[i].profit;
26         int l = latestNonConflict(arr, i);
27         if (l != -1)
28             inclProf += table[l];
29         table[i] = max(inclProf, table[i - 1]);
30     }
31     int result = table[n - 1];
32     delete[] table;
33
34     return result;
35 }

```

## 7.6 數獨解法

```

1 int getSquareIndex(int row, int column, int
  n)
2 {
3     return row / n * n + column / n;
4 }
5
6 bool backtracking(vector<vector<int>> &board,
  vector<vector<bool>> &rows, vector<
  vector<bool>> &cols,
7     vector<vector<bool>> &boxes,
  int index, int n)
8 {
9     int n2 = n * n;
10    int rowNum = index / n2, colNum = index
11    % n2;
12    if (index >= n2 * n2)
13        return true;
14
15    if (board[rowNum][colNum] != 0)
16        return backtracking(board, rows,
17                              cols, boxes, index + 1, n);
18
19    for (int i = 1; i <= n2; i++)
20    {
21        if (!rows[rowNum][i] && !cols[colNum]
22            ][i] && !boxes[getSquareIndex(
23                rowNum, colNum, n)][i])
24        {
25            rows[rowNum][i] = true;
26            cols[colNum][i] = true;
27            boxes[getSquareIndex(rowNum,
28                                colNum, n)][i] = true;
29            board[rowNum][colNum] = i;
30            if (backtracking(board, rows,
31                              cols, boxes, index + 1, n))
32                return true;
33            board[rowNum][colNum] = 0;
34            rows[rowNum][i] = false;
35            cols[colNum][i] = false;
36            boxes[getSquareIndex(rowNum,
37                                colNum, n)][i] = false;
38        }
39    }
40    return false;
41 }

```

```

33     return false;
34 }
35 /*用法 main*/
36 int n = sqrt(數獨邊長大小) /*e.g. 9*9 n=3*/
37 vector<vector<int>> board(n * n + 1, vector<
38 int>(n * n + 1, 0));
39 vector<vector<bool>> isRow(n * n + 1, vector<
40 bool>(n * n + 1, false));
41 vector<vector<bool>> isColumn(n * n + 1,
42 vector<bool>(n * n + 1, false));
43 vector<vector<bool>> isSquare(n * n + 1,
44 vector<bool>(n * n + 1, false));
45
46 for (int i = 0; i < n * n; ++i)
47 {
48     for (int j = 0; j < n * n; ++j)
49     {
50         int number;
51         cin >> number;
52         board[i][j] = number;
53         if (number == 0)
54             continue;
55         isRow[i][number] = true;
56         isColumn[j][number] = true;
57         isSquare[getSquareIndex(i, j, n)][
58             number] = true;
59     }
60 }
61 if (backtracking(board, isRow, isColumn,
62 isSquare, 0, n))
63     /*有解答*/
64 else
65     /*解答*/

```

## 8 String

### 8.1 KMP

```

1 // 用在一個 S 內查找一個詞 W 的出現位置
2 void ComputePrefix(string s, int next[])
3 {
4     int n = s.length();
5     int q, k;
6     next[0] = 0;
7     for (k = 0, q = 1; q < n; q++)
8     {
9         while (k > 0 && s[k] != s[q])
10            k = next[k];
11         if (s[k] == s[q])
12            k++;
13         next[q] = k;
14     }
15 }
16 void KMPMatcher(string text, string pattern)
17 {
18     int n = text.length();
19     int m = pattern.length();
20     int next[pattern.length()];
21     ComputePrefix(pattern, next);
22 }

```

```

23 for (int i = 0, q = 0; i < n; i++)
24 {
25     while (q > 0 && pattern[q] != text[i]
26            )
27         q = next[q];
28     if (pattern[q] == text[i])
29         q++;
30     if (q == m)
31     {
32         cout << "Pattern occurs with
33             shift " << i - m + 1 << endl;
34     }
35     q = 0;
36 }
37 // string s = "abcdabcdebcdd";
38 // string p = "bcd";
39 // KMPMatcher(s, p);
40 // cout << endl;

```

## 8.2 Min Edit Distance

```

1 int EditDistance(string a, string b)
2 {
3     vector<vector<int>> dp(a.size() + 1,
4 vector<int>(b.size() + 1, 0));
5     int m = a.length(), n = b.length();
6     for (int i = 0; i < m + 1; i++)
7     {
8         for (int j = 0; j < n + 1; j++)
9         {
10            if (i == 0)
11                dp[i][j] = j;
12            else if (j == 0)
13                dp[i][j] = i;
14            else if (a[i - 1] == b[j - 1])
15                dp[i][j] = dp[i - 1][j - 1];
16            else
17                dp[i][j] = 1 + min(min(dp[i - 1][j],
18                                     dp[i][j - 1]),
19                                     dp[i - 1][j - 1]);
20        }
21    }
22    return dp[m][n];
23 }

```

## 8.3 Sliding window

```

1 string minWindow(string s, string t)
2 {
3     unordered_map<char, int> letterCnt;
4     for (int i = 0; i < t.length(); i++)
5         letterCnt[t[i]]++;
6     int minLength = INT_MAX, minStart = -1;
7     int left = 0, matchCnt = 0;
8     for (int i = 0; i < s.length(); i++)
9     {
10        if (--letterCnt[s[i]] >= 0)
11            matchCnt++;

```

```

12 while (matchCnt == t.length())
13 {
14     if (i - left + 1 < minLength)
15     {
16         minLength = i - left + 1;
17         minStart = left;
18     }
19     if (++letterCnt[s[left]] > 0)
20         matchCnt--;
21     left++;
22 }
23 }
24 return minLength == INT_MAX ? "" : s.
25 substr(minStart, minLength);

```

## 8.4 Split

```

1 vector<string> mysplit(const string &str,
2 const string &delim)
3 {
4     vector<string> res;
5     if (" " == str)
6         return res;
7
8     char *strs = new char[str.length() + 1];
9     char *d = new char[delim.length() + 1];
10    strcpy(strs, str.c_str());
11    strcpy(d, delim.c_str());
12    char *p = strtok(strs, d);
13    while (p)
14    {
15        string s = p;
16        res.push_back(s);
17        p = strtok(NULL, d);
18    }
19    return res;

```

## 9 data structure

### 9.1 Bigint

```

1 //台大
2 struct Bigint
3 {
4     static const int LEN = 60;
5     static const int BIGMOD = 10000;
6     int s;
7     int v1, v[LEN];
8     // vector<int> v;
9     Bigint() : s(1) { v1 = 0; }
10    Bigint(long long a)
11    {
12        s = 1;
13        v1 = 0;
14        if (a < 0)
15        {

```

```

16            s = -1;
17            a = -a;
18        }
19        while (a)
20        {
21            push_back(a % BIGMOD);
22            a /= BIGMOD;
23        }
24    }
25    Bigint(string str)
26    {
27        s = 1;
28        v1 = 0;
29        int stPos = 0, num = 0;
30        if (!str.empty() && str[0] == '-')
31        {
32            stPos = 1;
33            s = -1;
34        }
35        for (int i = str.length() - 1, q = 1; i >= stPos; i--)
36        {
37            num += (str[i] - '0') * q;
38            if ((q *= 10) >= BIGMOD)
39            {
40                push_back(num);
41                num = 0;
42                q = 1;
43            }
44        }
45        if (num)
46            push_back(num);
47        n();
48    }
49    int len() const
50    {
51        return v1; //return SZ(v);
52    }
53    bool empty() const { return len() == 0; }
54    void push_back(int x)
55    {
56        v[v1++] = x; //v.PB(x);
57    }
58    void pop_back()
59    {
60        v1--; //v.pop_back();
61    }
62    int back() const
63    {
64        return v[v1 - 1]; //return v.back();
65    }
66    void n()
67    {
68        while (!empty() && !back())
69            pop_back();
70    }
71    void resize(int n1)
72    {
73        v1 = n1; //v.resize(n1);
74        fill(v, v + v1, 0); //fill(ALL(v),
75        0);
76    }
77    void print() const
78    {
79        if (empty())

```

```

80        {
81            putchar('0');
82            return;
83        }
84        if (s == -1)
85            putchar('-');
86        printf("%d", back());
87        for (int i = len() - 2; i >= 0; i--)
88            printf("%.4d", v[i]);
89    }
90    friend std::ostream &operator<<(std::
91    ostream &out, const Bigint &a)
92    {
93        if (a.empty())
94        {
95            out << "0";
96            return out;
97        }
98        if (a.s == -1)
99            out << "-";
100        out << a.back();
101        for (int i = a.len() - 2; i >= 0; i
102        --)
103        {
104            char str[10];
105            snprintf(str, 5, "%.4d", a.v[i])
106            ;
107            out << str;
108        }
109        return out;
110    }
111    int cp3(const Bigint &b) const
112    {
113        if (s != b.s)
114            return s - b.s;
115        if (s == -1)
116            return -(*this).cp3(-b);
117        if (len() != b.len())
118            return len() - b.len(); //int
119        for (int i = len() - 1; i >= 0; i--)
120            if (v[i] != b.v[i])
121                return v[i] - b.v[i];
122        return 0;
123    }
124    bool operator<(const Bigint &b) const
125    {
126        return cp3(b) < 0;
127    }
128    bool operator<=(const Bigint &b) const
129    {
130        return cp3(b) <= 0;
131    }
132    bool operator==(const Bigint &b) const
133    {
134        return cp3(b) == 0;
135    }
136    bool operator!=(const Bigint &b) const
137    {
138        return cp3(b) != 0;
139    }
140    bool operator>(const Bigint &b) const
141    {

```

```

142        return cp3(b) >= 0;
143    }
144    Bigint operator-() const
145    {
146        Bigint r = (*this);
147        r.s = -r.s;
148        return r;
149    }
150    Bigint operator+(const Bigint &b) const
151    {
152        if (s == -1)
153            return -(*this) + (-b);
154        if (b.s == -1)
155            return (*this) - (-b);
156        Bigint r;
157        int n1 = max(len(), b.len());
158        r.resize(n1 + 1);
159        for (int i = 0; i < n1; i++)
160        {
161            if (i < len())
162                r.v[i] += v[i];
163            if (i < b.len())
164                r.v[i] += b.v[i];
165            if (r.v[i] >= BIGMOD)
166            {
167                r.v[i + 1] += r.v[i] /
168                BIGMOD;
169                r.v[i] %= BIGMOD;
170            }
171        }
172        r.n();
173        return r;
174    }
175    Bigint operator-(const Bigint &b) const
176    {
177        if (s == -1)
178            return -(*this) - (-b);
179        if (b.s == -1)
180            return (*this) + (-b);
181        if ((*this) < b)
182            return -b - (*this);
183        Bigint r;
184        r.resize(len());
185        for (int i = 0; i < len(); i++)
186        {
187            r.v[i] += v[i];
188            if (i < b.len())
189                r.v[i] -= b.v[i];
190            if (r.v[i] < 0)
191            {
192                r.v[i] += BIGMOD;
193                r.v[i + 1]--;
194            }
195        }
196        r.n();
197        return r;
198    }
199    Bigint operator*(const Bigint &b)
200    {
201        Bigint r;
202        r.resize(len() + b.len() + 1);
203        r.s = s * b.s;
204        for (int i = 0; i < len(); i++)
205        {
206            for (int j = 0; j < b.len(); j
207            ++

```

```

206 {
207     r.v[i + j] += v[i] * b.v[j];
208     if (r.v[i + j] >= BIGMOD)
209     {
210         r.v[i + j + 1] += r.v[i
211             + j] / BIGMOD;
212         r.v[i + j] %= BIGMOD;
213     }
214 }
215 r.n();
216 return r;
217 }
218 BigInt operator/(const BigInt &b)
219 {
220     BigInt r;
221     r.resize(max(1, len() - b.len() + 1)
222 );
223 int oriS = s;
224 BigInt b2 = b; // b2 = abs(b)
225 s = b2.s = r.s = 1;
226 for (int i = r.len() - 1; i >= 0; i
227     --)
228 {
229     int d = 0, u = BIGMOD - 1;
230     while (d < u)
231     {
232         int m = (d + u + 1) >> 1;
233         r.v[i] = m;
234         if ((r * b2) > (*this))
235             u = m - 1;
236         else
237             d = m;
238     }
239     r.v[i] = d;
240 }
241 s = oriS;
242 r.s = s * b.s;
243 r.n();
244 return r;
245 }
246 BigInt operator%(const BigInt &b)
247 {
248     return (*this) - (*this) / b * b;
249 }

```

## 9.2 matirx

```

1 template <typename T>
2 struct Matrix
3 {
4     using rt = std::vector<T>;
5     using mt = std::vector<rt>;
6     using matrix = Matrix<T>;
7     int r, c; // [r][c]
8     mt m;
9     Matrix(int r, int c) : r(r), c(c), m(r,
10         rt(c)) {}
11     Matrix(mt a) { m = a, r = a.size(), c =
12         a[0].size(); }
13     rt &operator[](int i) { return m[i]; }
14     matrix operator+(const matrix &a)

```

```

15 {
16     matrix rev(r, c);
17     for (int i = 0; i < r; ++i)
18     for (int j = 0; j < c; ++j)
19         rev[i][j] = m[i][j] + a.m[i
20             ][j];
21     return rev;
22 }
23 matrix operator-(const matrix &a)
24 {
25     matrix rev(r, c);
26     for (int i = 0; i < r; ++i)
27     for (int j = 0; j < c; ++j)
28         rev[i][j] = m[i][j] - a.m[i
29             ][j];
30     return rev;
31 }
32 matrix operator*(const matrix &a)
33 {
34     matrix rev(r, a.c);
35     matrix tmp(a.c, a.r);
36     for (int i = 0; i < a.r; ++i)
37     for (int j = 0; j < a.c; ++j)
38         tmp[j][i] = a.m[i][j];
39     for (int i = 0; i < r; ++i)
40     for (int j = 0; j < a.c; ++j)
41     for (int k = 0; k < c; ++k)
42         rev.m[i][j] += m[i][k] *
43             tmp[j][k];
44     return rev;
45 }
46 bool inverse() //逆矩陣判斷
47 {
48     Matrix t(r, r + c);
49     for (int y = 0; y < r; y++)
50     {
51         t.m[y][c + y] = 1;
52         for (int x = 0; x < c; ++x)
53             t.m[y][x] = m[y][x];
54     }
55     if (!t.gas())
56         return false;
57     for (int y = 0; y < r; y++)
58     for (int x = 0; x < c; ++x)
59         m[y][x] = t.m[y][c + x] / t.
60             m[y][y];
61     return true;
62 }
63 T gas() //行列式
64 {
65     vector<T> lazy(r, 1);
66     bool sign = false;
67     for (int i = 0; i < r; ++i)
68     {
69         if (m[i][i] == 0)
70         {
71             int j = i + 1;
72             while (j < r && !m[j][i])
73                 j++;
74             if (j == r)
75                 continue;
76             m[i].swap(m[j]);
77             sign = !sign;
78         }
79         for (int j = 0; j < r; ++j)

```

```

74 {
75     if (i == j)
76         continue;
77     lazy[j] = lazy[j] * m[i][i];
78     T mx = m[j][i];
79     for (int k = 0; k < c; ++k)
80         m[j][k] = m[j][k] * m[i
81             ][i] - m[i][k] * mx;
82 }
83 }
84 T det = sign ? -1 : 1;
85 for (int i = 0; i < r; ++i)
86 {
87     det = det * m[i][i];
88     det = det / lazy[i];
89     for (auto &j : m[i])
90         j /= lazy[i];
91     return det;
92 }
93 };

```

## 9.3 Trie

```

1 // biginter字典數
2 struct BigInteger{
3     static const int BASE = 100000000;
4     static const int WIDTH = 8;
5     vector<int> s;
6     BigInteger(long long num = 0){
7         *this = num;
8     }
9     BigInteger operator = (long long num){
10         s.clear();
11         do{
12             s.push_back(num % BASE);
13             num /= BASE;
14         }while(num > 0);
15         return *this;
16 }
17 BigInteger operator = (const string& str
18 ){
19     s.clear();
20     int x, len = (str.length() - 1) /
21         WIDTH + 1;
22     for(int i = 0; i < len; i++){
23         int end = str.length() - i*WIDTH
24             ;
25         int start = max(0, end-WIDTH);
26         sscanf(str.substr(start, end-
27             start).c_str(), "%d", &x);
28         s.push_back(x);
29     }
30     return *this;
31 }
32 BigInteger operator + (const BigInteger&
33     b) const{
34     BigInteger c;
35     c.s.clear();
36     for(int i = 0, g = 0; i++){
37         if(g == 0 && i >= s.size() && i
38             >= b.s.size()) break;

```

```

34     int x = g;
35     if(i < s.size()) x+=s[i];
36     if(i < b.s.size()) x+=b.s[i];
37     c.s.push_back(x % BASE);
38     g = x / BASE;
39 }
40 return c;
41 }
42 };
43 ostream& operator << (ostream &out, const
44     BigInteger& x){
45     out << x.s.back();
46     for(int i = x.s.size()-2; i >= 0; i--){
47         char buf[20];
48         sprintf(buf, "%08d", x.s[i]);
49         for(int j = 0; j < strlen(buf); j++){
50             out << buf[j];
51         }
52     }
53     return out;
54 }
55 istream& operator >> (istream &in,
56     BigInteger& x){
57     string s;
58     if(!(in >> s))
59         return in;
60     x = s;
61     return in;
62 }
63 struct Trie{
64     int c[5000005][10];
65     int val[5000005];
66     int sz;
67     int getIndex(char c){
68         return c - '0';
69     }
70     void init(){
71         memset(c[0], 0, sizeof(c[0]));
72         memset(val, -1, sizeof(val));
73         sz = 1;
74     }
75     void insert(BigInteger x, int v){
76         int u = 0;
77         int max_len_count = 0;
78         int firstNum = x.s.back();
79         char firstBuf[20];
80         sprintf(firstBuf, "%d", firstNum);
81         for(int j = 0; j < strlen(firstBuf);
82             j++){
83             int index = getIndex(firstBuf[j
84                 ]);
85             if(!c[u][index]){
86                 memset(c[sz], 0, sizeof(c[
87                     sz]));
88                 val[sz] = v;
89                 c[u][index] = sz++;
90             }
91             u = c[u][index];
92             max_len_count++;
93         }
94     }
95     for(int i = x.s.size()-2; i >= 0; i
96         --){
97         char buf[20];

```

```

94     sprintf(buf, "%08d", x.s[i]);
95     for(int j = 0; j < strlen(buf)
96         && max_len_count < 50; j++){
97         int index = getIndex(buf[j])
98         ;
99         if(!c[u][index]){
100             memset(c[sz], 0 , sizeof
101                 (c[sz]));
102             val[sz] = v;
103             c[u][index] = sz++;
104         }
105         u = c[u][index];
106         max_len_count++;
107     }
108     if(max_len_count >= 50){
109         break;
110     }
111 }
112 int find(const char* s){
113     int u = 0;
114     int n = strlen(s);
115     for(int i = 0 ; i < n; ++i)
116     {
117         int index = getIndex(s[i]);
118         if(!c[u][index]){
119             return -1;
120         }
121         u = c[u][index];
122     }
123     return val[u];
124 }

```

```

23 }
24 fraction operator*(const fraction &b)
25     const
26 {
27     return fraction(n * b.n, d * b.d);
28 }
29 fraction operator/(const fraction &b)
30     const
31 {
32     return fraction(n * b.d, d * b.n);
33 }
34 void print()
35 {
36     cout << n;
37     if (d != 1)
38         cout << "/" << d;
39 }

```

## 9.4 分數

```

1 typedef long long ll;
2 struct fraction
3 {
4     ll n, d;
5     fraction(const ll &n = 0, const ll &d =
6         1) : n(_n), d(_d)
7     {
8         ll t = __gcd(n, d);
9         n /= t, d /= t;
10        if (d < 0)
11            n = -n, d = -d;
12    }
13    fraction operator-() const
14    {
15        return fraction(-n, d);
16    }
17    fraction operator+(const fraction &b)
18        const
19    {
20        return fraction(n * b.d + b.n * d, d * b
21            .d);
22    }
23    fraction operator-(const fraction &b)
24        const
25    {
26        return fraction(n * b.d - b.n * d, d * b
27            .d);
28    }
29 }

```



# TO DO WRITING NOT THINKING

## Contents

<b>1 Basic</b>	<b>1</b>	2.7 LIS . . . . .	1	5.5 Floyd-warshall . . . . .	6	6.16 質因數分解 . . . . .	9
1.1 data range . . . . .	1	2.8 LPS . . . . .	2	5.6 Kruskal . . . . .	6	<b>7 Other</b>	<b>9</b>
1.2 IO_fast . . . . .	1	2.9 Max_subarray . . . . .	2	5.7 Prim . . . . .	7	7.1 binary search 三類變化 . . . . .	9
1.3 常忘記 . . . . .	1	2.10 Money problem . . . . .	2	5.8 Union_find . . . . .	7	7.2 heap sort . . . . .	9
<b>2 DP</b>	<b>1</b>	<b>3 Flow &amp; matching</b>	<b>2</b>	<b>6 Mathematics</b>	<b>7</b>	7.3 Merge sort . . . . .	9
2.1 3 維 DP 思路 . . . . .	1	3.1 Edmonds_karp . . . . .	2	6.1 Combination . . . . .	7	7.4 Quick . . . . .	10
2.2 Knapsack Bounded . . . . .	1	3.2 maximum_matching . . . . .	2	6.2 Extended Euclidean . . . . .	7	7.5 Weighted Job Scheduling . . . . .	10
2.3 Knapsack sample . . . . .	1	3.3 MFlow Model . . . . .	3	6.3 Hex to Dec . . . . .	7	7.6 數獨解法 . . . . .	10
2.4 Knapsack Unbounded . . . . .	1	<b>4 Geometry</b>	<b>3</b>	6.4 log . . . . .	7	<b>8 String</b>	<b>10</b>
2.5 LCIS . . . . .	1	4.1 Line . . . . .	3	6.5 Mod . . . . .	8	8.1 KMP . . . . .	10
2.6 LCS . . . . .	1	4.2 Point . . . . .	4	6.6 Permutation . . . . .	8	8.2 Min Edit Distance . . . . .	10
		4.3 Polygon . . . . .	4	6.7 PI . . . . .	8	8.3 Sliding window . . . . .	10
		4.4 Triangle . . . . .	5	6.8 Prime table . . . . .	8	8.4 Split . . . . .	11
		<b>5 Graph</b>	<b>5</b>	6.9 primeBOOL . . . . .	8	<b>9 data structure</b>	<b>11</b>
		5.1 Bellman-Ford . . . . .	5	6.10 Round(小數) . . . . .	8	9.1 Bigint . . . . .	11
		5.2 BFS-queue . . . . .	5	6.11 二分逼近法 . . . . .	8	9.2 matirx . . . . .	12
		5.3 DFS-rec . . . . .	6	6.12 四則運算 . . . . .	8	9.3 Trie . . . . .	12
		5.4 Dijkstra . . . . .	6	6.13 數字乘法組合 . . . . .	9	9.4 分數 . . . . .	13
				6.14 數字加法組合 . . . . .	9		
				6.15 羅馬數字 . . . . .	9		