

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 ios_base::sync_with_stdio(0);
2 cin.tie(0);
```

2 DP

2.1 KMP

```
1 void ComputePrefix(string s, int next[])
2 {
3     int n = s.length();
4     int q, k;
5     next[0] = 0;
6     for (k = 0, q = 1; q < n; q++)
7     {
8         while (k > 0 && s[k] != s[q])
9             k = next[k];
10        if (s[k] == s[q])
11            k++;
12        next[q] = k;
13    }
14 }
15 void KMPMatcher(string text, string pattern)
16 {
17     int n = text.length();
18     int m = pattern.length();
19     int next[pattern.length()];
20     ComputePrefix(pattern, next);
21
22     for (int i = 0, q = 0; i < n; i++)
23     {
24         while (q > 0 && pattern[q] != text[i])
25             q = next[q];
26         if (pattern[q] == text[i])
27             q++;
28         if (q == m)
29         {
30             cout << "Pattern occurs with
31               shift " << i - m + 1 << endl;
32         }
33     }
```

```
31         q = 0;
32     }
33 }
34 }
35 // string s = "abcdabcdebc";
36 // string p = "bcd";
37 // KMPMatcher(s, p);
38 // cout << endl;
```

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
5 void knapsack(int n, int w)
6 {
7     for (int i = 0; i < n; ++i)
8     {
9         int num = min(number[i], w / weight[i]);
10        for (int k = 1; num > 0; k *= 2)
11        {
12            if (k > num)
13                k = num;
14            num -= k;
15            for (int j = w; j >= weight[i] * k; --j)
16                c[j] = max(c[j], c[j - weight[i] * k] + cost[i] * k);
17        }
18    }
19    cout << "Max Prince" << c[w];
20 }
```

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[i - 1][j];
18            else dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weight[i]] + value[i]);
19        }
20    }
```

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
8     for (int i = 0; i < n; ++i)
9         for (int j = weight[i]; j <= w; ++j)
10             c[j] = max(c[j], c[j - weight[i]] + cost[i]);
11
12    cout << "最高的價值為" << c[w];
13 }
```

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
9     for (int i = 0; i < n; i++)
10    {
11        int current = 0;
12
13        for (int j = 0; j < m; j++)
14        {
15            if (arr1[i] == arr2[j])
16                if (current + 1 > table[j])
17                    table[j] = current + 1;
18
19            if (arr1[i] > arr2[j])
20                if (table[j] > current)
21                    current = table[j];
22        }
23    }
24    int result = 0;
25    for (int i = 0; i < m; i++)
26        if (table[i] > result)
27            result = table[i];
28
29    return result;
30 }
```

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], LCS[i][j - 1]);
15        }
16    }
17    cout << LCS[N][M] << '\n';
18    //列印 LCS
19    int n = N, m = M;
20    vector<string> k;
21    while (n && m)
22    {
23        if (LCS[n][m] != max(LCS[n - 1][m], LCS[n][m - 1]))
24        {
25            k.push_back(Ans[n - 1]);
26            n--;
27            m--;
28        }
29        else if (LCS[n][m] == LCS[n - 1][m])
30            n--;
31        else if (LCS[n][m] == LCS[n][m - 1])
32            m--;
33    }
34    reverse(k.begin(), k.end());
35    for (auto i : k)
36        cout << i << " ";
37    cout << endl;
38    return LCS[N][M];
39 }
```

2.7 LIC

```
1 void getMaxElementAndPos(vector<int> &LISTbl
2   , vector<int> &LISLen, int tNum,
3   int tlen, int tStart, int &num, int &pos)
4 {
5     int max = numeric_limits<int>::min();
6     int maxPos;
7     for (int i = tStart; i <= tlen; i++)
8     {
9         if (LISLen[i] == tlen && LISTbl[i] < tNum)
10        {
11            if (LISTbl[i] > max)
12                max = LISTbl[i];
13        }
14    }
```

```

13         maxPos = i;
14     }
15 }
16 }
17 num = max;
18 pos = maxPos;
19 }
20 int LIS(vector<int> &LISTbl)
21 {
22     if (LISTbl.size() == 0)
23         return 0;
24     vector<int> LISlen(LISTbl.size(), 1);
25     for (int i = 1; i < LISTbl.size(); i++)
26     {
27         for (int j = 0; j < i; j++)
28         {
29             if (LISTbl[j] < LISTbl[i])
30                 LISlen[i] = max(LISlen[i],
31                                 LISlen[j] + 1);
32         }
33     }
34     int maxlen = *max_element(LISlen.begin(),
35                               LISlen.end());
36     int num, pos;
37     vector<int> buf;
38     getMaxElementAndPos(LISTbl, LISlen,
39                         numeric_limits<int>
40                         ::max(),
41                         maxlen, LISTbl.size()
42                         - 1, num, pos
43                         );
44     buf.push_back(num);
45     for (int len = maxlen - 1; len >= 1; len--)
46     {
47         int tnum = num;
48         int tpos = pos;
49         getMaxElementAndPos(LISTbl, LISlen,
50                             tnum, len, tpos
51                             - 1, num,
52                             pos);
53         buf.push_back(num);
54     }
55     reverse(buf.begin(), buf.end());
56     for (int k = 0; k < buf.size(); k++) //
57         列印
58     {
59         if (k == buf.size() - 1)
60             cout << buf[k] << endl;
61         else
62             cout << buf[k] << ",";
63     }
64     return maxlen;
65 }

```

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
9             - x >= 0) && (i + x < n)) //odd
10             length
11             x++;
12         x--;
13         if (2 * x + 1 > maxlen)
14         {
15             maxlen = 2 * x + 1;
16             l = i - x;
17             r = i + x;
18         }
19         x = 0;
20         while ((s[i - x] == s[i + 1 + x]) &&
21             (i - x >= 0) && (i + 1 + x < n)) //even length
22             x++;
23         if (2 * x > maxlen)
24         {
25             maxlen = 2 * x;
26             l = i - x + 1;
27             r = i + x;
28         }
29     }
30     cout << maxlen << '\n'; // 最後長度
31     cout << l + 1 << ' ' << r + 1 << '\n';
32     //頭到尾

```

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], nums[i] +
7         local_max);
8         global_max = max(local_max,
9         global_max);
10    }
11    return global_max;
12 }

```

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;
9             ++j) // 由低價位逐步到高價位
10             c[j] = c[j] || c[j - price[i]];
11         // 湊、湊、湊

```

```

9     if (c[limit]) cout << "YES\n";
10    else cout << "NO\n";
11 }
12 // 湊得某個價位的湊法總共幾種
13 void change(vector<int> price, int limit)
14 {
15     vector<bool> c(limit + 1, 0);
16     c[0] = true;
17     for (int i = 0; i < price.size(); ++i)
18         for (int j = price[i]; j <= limit;
19             ++j)
20             c[j] += c[j - price[i]];
21     cout << c[limit] << '\n';
22 }
23 // 湊得某個價位的最少錢幣用量
24 void change(vector<int> price, int limit)
25 {
26     vector<bool> c(limit + 1, 0);
27     c[0] = true;
28     for (int i = 0; i < price.size(); ++i)
29         for (int j = price[i]; j <= limit;
30             ++j)
31             c[j] = min(c[j], c[j - price[i]]
32                 + 1);
33     cout << c[limit] << '\n';
34 }
35 // 湊得某個價位的錢幣用量，有幾種可能性
36 void change(vector<int> price, int limit)
37 {
38     vector<bool> 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;
42             ++j)
43             c[j] |= c[j - price[i]] << 1; //
44             錢幣數量加一，每一種可能性都
45             加一。
46     for (int i = 1; i <= 63; ++i)
47         if (c[i] & (1 << i))
48             cout << "用" << i << "個錢幣可湊
49             得價位" << i;
50 }

```

3 Geometry

3.1 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
45    { // 兩線段距離平方
46        return min({dis2(l.p1, 1), dis2(l.p2
47            , 1), l.dis2(p1, 1), l.dis2(p2,
48            1)});
49    }
50    point<T> projection(const point<T> &p)
51    const
52    { // 點對直線的投影
53        point<T> n = (p2 - p1).normal();
54        return p - n * (p - p1).dot(n) / n.
55            abs2();
56    }
57    point<T> mirror(const point<T> &p) const
58    {
59        // 點對直線的鏡射，要先呼叫pton轉成一般式
60        point<T> R;
61        T d = a * a + b * b;
62        R.x = (b * b * p.x - a * a * p.x - 2
63            * a * b * p.y - 2 * a * c) / d;
64        R.y = (a * a * p.y - b * b * p.y - 2
65            * a * b * p.x - 2 * b * c) / d;
66        return R;
67    }
68    bool equal(const line &l) const
69    { // 直線相等
70        return ori(l.p1) == 0 && ori(l.p2)
71            == 0;
72    }
73    bool parallel(const line &l) const
74    {

```

```

64     return (p1 - p2).cross(l.p1 - l.p2)
65         == 0;
66 }
67 bool cross_seg(const line &l) const
68 {
69     return (p2 - p1).cross(l.p1 - p1) *
70         (p2 - p1).cross(l.p2 - p1) <= 0;
71     // 直線是否交線段
72 }
73 int line_intersect(const line &l) const
74 { // 直線相交情況 · -1無限多點、1交於一
75     // 點、0不相交
76     return parallel(l) ? (ori(l.p1) == 0
77         ? -1 : 0) : 1;
78 }
79 int seg_intersect(const line &l) const
80 {
81     T c1 = ori(l.p1), c2 = ori(l.p2);
82     T c3 = l.ori(p1), c4 = l.ori(p2);
83     if (c1 == 0 && c2 == 0)
84     { // 共線
85         bool b1 = btw(l.p1) >= 0, b2 =
86             btw(l.p2) >= 0;
87         T a3 = l.btw(p1), a4 = l.btw(p2);
88         if (b1 && b2 && a3 == 0 && a4 ==
89             0)
90             return 2;
91         if (b1 && b2 && a3 >= 0 && a4 ==
92             0)
93             return 3;
94         if (b1 && b2 && a3 >= 0 && a4 >=
95             0)
96             return 0;
97         return -1; // 無限交點
98     }
99     else if (c1 * c2 <= 0 && c3 * c4 <=
100         0)
101         return 1;
102     return 0; // 不相交
103 }
104 point<T> line_intersection(const line &l
105     ) const
106 { /* 直線交點 */
107     point<T> a = p2 - p1, b = l.p2 - l.
108         p1, s = l.p1 - p1;
109     // if(a.cross(b)==0) return INF;
110     return p1 + a * (s.cross(b) / a.
111         cross(b));
112 }
113 point<T> seg_intersection(const line &l)
114     const
115 { // 線段交點
116     int res = seg_intersect(l);
117     if (res <= 0)
118         assert(0);
119     if (res == 2)
120         return p1;
121     if (res == 3)
122         return p2;
123     return line_intersection(l);
124 }
125 }
126 }

```

3.2 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    {
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    {
50        // 對x軸的弧度
51        // 超過180度會變負
52        T A = atan2(y, x);
53        if (A <= -PI / 2)
54            A += PI * 2;
55        return A;
56    }
57 };

```

3.3 Polygon

```

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
31        bool c = 0;
32        for (int i = 0, j = p.size() - 1; i
33            < p.size(); j = i++)
34            if (line<T>(p[i], p[j]).
35                point_on_segment(t))
36                return -1;
37            else if ((p[i].y > t.y) != (p[j]
38                .y > t.y) &&
39                t.x < (p[j].x - p[i].x) *
40                    (t.y - p[i].y) /
41                    (p[j].y - p[i].y) +
42                    p[i].x)
43                c = !c;
44        return c;
45    }
46    char point_in_convex(const point<T> &x)
47        const
48    {
49        int l = 1, r = (int)p.size() - 2;
50        while (l <= r)
51        { // 點是否在凸多邊形內 · 是的話回傳1
52            // 在邊上回傳-1、否則回傳0
53            int mid = (l + r) / 2;
54            T a1 = (p[mid] - p[0]).cross(x -
55                p[0]);
56            T a2 = (p[mid + 1] - p[0]).cross
57                (x - p[0]);
58            if (a1 >= 0 && a2 <= 0)
59            {
60                T res = (p[mid + 1] - p[mid]
61                    ).cross(x - p[mid]);
62                return res > 0 ? 1 : (res >=
63                    0 ? -1 : 0);
64            }
65        }
66    }
67 }

```

```

49     else if (a1 < 0)
50         r = mid - 1;
51     else
52         l = mid + 1;
53 }
54 return 0;
55 }
56 vector<T> getA() const
57 { // 凸包邊對x軸的夾角
58     vector<T> res; // 一定是遞增的
59     for (size_t i = 0; i < p.size(); ++i)
60         res.push_back((p[(i + 1) % p.
61             size()] - p[i]).getA());
62     return res;
63 }
64 bool line_intersect(const vector<T> &A,
65     const line<T> &l) const
66 { // O(logN)
67     int f1 = upper_bound(A.begin(), A.
68         end(), (l.p1 - l.p2).getA()) - A.
69         begin();
70     int f2 = upper_bound(A.begin(), A.
71         end(), (l.p2 - l.p1).getA()) - A.
72         begin();
73     return l.cross_seg(line<T>(p[f1], p[
74         f2]));
75 }
76 polygon cut(const line<T> &l) const
77 { // 凸包對直線切割 · 得到直線l左側的凸包
78     polygon ans;
79     for (int n = p.size(), i = n - 1, j
80         = 0; j < n; i = j++)
81     {
82         if (l.ori(p[i]) >= 0)
83         {
84             ans.p.push_back(p[i]);
85             if (l.ori(p[j]) < 0)
86                 ans.p.push_back(l.
87                     line_intersection(
88                         line<T>(p[i], p[j]))
89                     );
90         }
91         else if (l.ori(p[j]) > 0)
92             ans.p.push_back(l.
93                 line_intersection(line<T>
94                     (>p[i], p[j])));
95     }
96     return ans;
97 }
98 static bool graham_cmp(const point<T> &a
99     , const point<T> &b)
100 { // 凸包排序函數 // 起點點不同
101     // return (a.x < b.x) || (a.x == b.x
102         && a.y < b.y); // 最左下角開始
103     return (a.y < b.y) || (a.y == b.y &&
104         a.x < b.x); // 最小開始
105 }
106 void graham(vector<point<T>> &s)
107 { // 凸包 Convexhull 2D
108     sort(s.begin(), s.end(), graham_cmp);
109     ;
110     p.resize(s.size() + 1);
111     int m = 0;

```

```

95 // cross >= 0 順時針。cross <= 0 逆
    時針旋轉
96 for (size_t i = 0; i < s.size(); ++i)
97 {
98     while (m >= 2 && (p[m - 1] - p[m
        - 2]).cross(s[i] - p[m -
        2]) <= 0)
99         --m;
100     p[m++] = s[i];
101 }
102 for (int i = s.size() - 2, t = m +
    1; i >= 0; --i)
103 {
104     while (m >= t && (p[m - 1] - p[m
        - 2]).cross(s[i] - p[m -
        2]) <= 0)
105         --m;
106     p[m++] = s[i];
107 }
108 if (s.size() > 1) // 重複頭一次需扣
    掉
109     --m;
110     p.resize(m);
111 }
112 T diam()
113 { //直徑
114     int n = p.size(), t = 1;
115     T ans = 0;
116     p.push_back(p[0]);
117     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;
        ans = min(ans, tmp);
    }
    return p.pop_back(), ans;
}
T dis2(polygon &p1)
{ //凸包最近距離平方
    vector<point<T>> &P = p, &Q = p1.p;
    int n = P.size(), m = Q.size(), l =
        0, r = 0;
    for (int i = 0; i < n; ++i)
        if (P[i].y < P[l].y)
            l = i;
    for (int i = 0; i < m; ++i)
        if (Q[i].y < Q[r].y)
            r = i;
    P.push_back(P[0]), Q.push_back(Q[0]);
    ;
    T ans = 1e99;
    for (int i = 0; i < n; ++i)
    {
        while ((P[l] - P[l + 1]).cross(Q
            [r + 1] - Q[r]) < 0)
            r = (r + 1) % m;
        ans = min(ans, line<T>(P[l], P[l
            + 1]).seg_dis2(line<T>(Q[r]
            , Q[r + 1])));
        l = (l + 1) % n;
    }
    return P.pop_back(), Q.pop_back(),
        ans;
}
static char sign(const point<T> &t)
{
    return (t.y == 0 ? t.x : t.y) < 0;
}
static bool angle_cmp(const line<T> &A,
    const line<T> &B)
{
    point<T> a = A.p2 - A.p1, b = B.p2 -
        B.p1;
    return sign(a) < sign(b) || (sign(a)
        == sign(b) && a.cross(b) > 0);
}
int halfplane_intersection(vector<line<T
    >> &s)
{
    //半平面交
    sort(s.begin(), s.end(), angle_cmp);
    //線段左側為該線段半平面
    int L, R, n = s.size();
    vector<point<T>> px(n);
    vector<line<T>> q(n);
    q[L = R = 0] = s[0];
    for (int i = 1; i < n; ++i)
    {
        while (L < R && s[i].ori(px[R -
            1]) <= 0)
            --R;
        while (L < R && s[i].ori(px[L])
            <= 0)
            ++L;
        q[++R] = s[i];
        if (q[R].parallel(q[R - 1]))

```

```

    {
        --R;
        if (q[R].ori(s[i].p1) > 0)
            q[R] = s[i];
    }
    if (L < R)
        px[R - 1] = q[R - 1].
            line_intersection(q[R]);
    }
    while (L < R && q[L].ori(px[R - 1])
        <= 0)
        --R;
    p.clear();
    if (R - L <= 1)
        return 0;
    px[R] = q[R].line_intersection(q[L]);
    ;
    for (int i = L; i <= R; ++i)
        p.push_back(px[i]);
    return R - L + 1;
}

```

3.4 Triangle

```

1 template <typename T>
2 struct triangle
3 {
4     point<T> a, b, c;
5     triangle() {}
6     triangle(const point<T> &a, const point<
7         T> &b, const point<T> &c) : a(a), b(b),
8         c(c) {}
9     T area() const
10     {
11         T t = (b - a).cross(c - a) / 2;
12         return t > 0 ? t : -t;
13     }
14     point<T> barycenter() const
15     { //重心
16         return (a + b + c) / 3;
17     }
18     point<T> circumcenter() const
19     { //外心
20         static line<T> u, v;
21         u.p1 = (a + b) / 2;
22         u.p2 = point<T>(u.p1.x - a.y + b.y,
23             u.p1.y + a.x - b.x);
24         v.p1 = (a + c) / 2;
25         v.p2 = point<T>(v.p1.x - a.y + c.y,
26             v.p1.y + a.x - c.x);
27         return u.line_intersection(v);
28     }
29     point<T> incenter() const
30     { //內心
31         T A = sqrt((b - c).abs2()), B = sqrt
32             ((a - c).abs2()), C = sqrt((a -
33             b).abs2());
34         return point<T>(A * a.x + B * b.x +
35             C * c.x, A * a.y + B * b.y + C *
36             c.y) / (A + B + C);
37     }
38 }

```

```

30 point<T> perpcenter() const
31 { //垂心
32     return barycenter() * 3 -
        circumcenter() * 2;
33 }
34 };

```

4 Graph

4.1 Bellman-Ford

```

1 /*SPA - Bellman-Ford*/
2 #include<bits/stdc++.h>
3 #define inf 99999 //define by you maximum
    edges weight
4 using namespace std;
5 vector<vector<int>> > edges;
6 vector<int> dist;
7 vector<int> ancestor;
8 void BellmanFord(int start,int node){
9     dist[start] = 0;
10    for(int it = 0; it < node-1; it++){
11        for(int i = 0; i < node; i++){
12            for(int j = 0; j < node; j++){
13                if(edges[i][j] != -1){
14                    if(dist[i] + edges[i][j]
15                        < dist[j]){
16                        dist[j] = dist[i] +
17                            edges[i][j];
18                        ancestor[j] = i;
19                    }
20                }
21            }
22        }
23    }
24    for(int i = 0; i < node; i++) //
        negative cycle detection
25    for(int j = 0; j < node; j++){
26        if(dist[i] + edges[i][j] < dist[
27            j]){
28            cout<<"Negative cycle!"<<
                endl;
29            return;
30        }
31    }
32    int main(){
33        int node;
34        cin>>node;
35        edges.resize(node,vector<int>(node,inf))
36        ;
37        dist.resize(node,inf);
38        ancestor.resize(node,-1);
39        int a,b,d;
40        while(cin>>a>>b>>d){
41            /*input: source destination weight*/
42            if(a == -1 && b == -1 && d == -1)
43                break;
44            edges[a][b] = d;
45        }

```

```

44 int start;
45 cin>>start;
46 BellmanFord(start,node);
47 return 0;
48 }

```

4.2 BFS-queue

```

1 /*BFS - queue version*/
2 #include<bits/stdc++.h>
3 using namespace std;
4 void BFS(vector<int> &result,vector<pair<int
,int> > edges,int node,int start){
5     vector<int> pass(node, 0);
6     queue<int> q;
7     queue<int> p;
8     q.push(start);
9     int count = 1;
10    vector<pair<int, int>> newedges;
11    while(!q.empty()){
12        pass[q.front()] = 1;
13        for (int i = 0; i < edges.size(); i
        ++){
14            if(edges[i].first == q.front()
            && pass[edges[i].second] ==
            0){
15                p.push(edges[i].second);
16                result[edges[i].second] =
                count;
17            }
18            else if(edges[i].second == q.
            front() && pass[edges[i].
            first] == 0){
19                p.push(edges[i].first);
20                result[edges[i].first] =
                count;
21            }
22            else
23                newedges.push_back(edges[i])
                ;
24        }
25        edges = newedges;
26        newedges.clear();
27        q.pop();
28        if(q.empty() == true){
29            q = p;
30            queue<int> tmp;
31            p = tmp;
32            count++;
33        }
34    }
35 }
36 int main(){
37     int node;
38     cin >> node;
39     vector<pair<int, int>> edges;
40     int a, b;
41     while(cin>>a>>b){
42         /*a = b = -1 means input edges ended
         */
43         if(a == -1 && b == -1)
44             break;

```

```

45         edges.push_back(pair<int, int>(a, b)
        );
46     }
47     vector<int> result(node, -1);
48     BFS(result, edges, node, 0);
49
50     return 0;
51 }

```

4.3 DFS-rec

```

1 /*DFS - Recursive version*/
2 #include<bits/stdc++.h>
3 using namespace std;
4 map<pair<int,int>,int> edges;
5 vector<int> pass;
6 vector<int> route;
7 void DFS(int start){
8     pass[start] = 1;
9     map<pair<int,int>,int>::iterator iter;
10    for(iter = edges.begin(); iter != edges.
    end(); iter++){
11        if((*iter).first.first == start &&
        (*iter).second == 0 && pass[(
        *iter).first.second] == 0){
12            route.push_back((*iter).first.
            second);
13            DFS((*iter).first.second);
14        }
15        else if((*iter).first.second ==
        start && (*iter).second == 0 &&
        pass[(*)iter).first.first] == 0){
16            route.push_back((*iter).first.
            first);
17            DFS((*iter).first.first);
18        }
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>
        >(pair<int,int>(a,b),0));
30     }
31     int start;
32     cin>>start;
33     route.push_back(start);
34     DFS(start);
35     return 0;
36 }

```

4.4 Dijkstra

```

1 /*SPA - Dijkstra*/
2 #include<bits/stdc++.h>

```

```

3 #define inf INT_MAX
4 using namespace std;
5 vector<vector<int> > weight;
6 vector<int> ancestor;
7 vector<int> dist;
8 void dijkstra(int start){
9     priority_queue<pair<int,int>,vector<
    pair<int,int> >,greater<pair<int,
    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
        (); i++){
15            if(dist[i] > dist[cur] + weight[
            cur][i] && weight[cur][i] !=
            -1){
16                dist[i] = dist[cur] + weight
                [cur][i];
17                ancestor[i] = cur;
18                pq.push(make_pair(dist[i],i)
                );
19            }
20        }
21    }
22 }
23 int main(){
24     int node;
25     cin>>node;
26     int a,b,d;
27     weight.resize(node,vector<int>(node,-1))
        ;
28     while(cin>>a>>b>>d){
29         /*input: source destination weight*/
30         if(a == -1 && b == -1 && d == -1)
31             break;
32         weight[a][b] = d;
33     }
34     ancestor.resize(node,-1);
35     dist.resize(node,inf);
36     int start;
37     cin>>start;
38     dist[start] = 0;
39     dijkstra(start);
40     return 0;
41 }

```

4.5 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
    , int s, int t, int n){
8     int ans = 0;
9     vector<vector<int>> residual(n+1, vector<
    int>(n+1, 0)); //residual network
10    while(true){
11        vector<int> bottleneck(n+1, 0);

```

```

12        bottleneck[s] = inf;
13        queue<int> q;
14        q.push(s);
15        vector<int> pre(n+1, 0);
16        while(!q.empty() && bottleneck[t] == 0){
17            int cur = q.front();
18            q.pop();
19            for(int i = 1; i <= n ; i++){
20                if(bottleneck[i] == 0 && capacity[
                cur][i] > residual[cur][i]){
21                    q.push(i);
22                    pre[i] = cur;
23                    bottleneck[i] = min(bottleneck[cur
                    ], capacity[cur][i] - residual
                    [cur][i]);
24                }
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)
41             break;
42         vector<vector<int>> capacity(n+1, vector
            <int>(n+1, 0));
43         int s, t, c;
44         cin >> s >> t >> c;
45         int a, b, bandwidth;
46         for(int i = 0 ; i < c ; ++i){
47             cin >> a >> b >> bandwidth;
48             capacity[a][b] += bandwidth;
49             capacity[b][a] += bandwidth;
50         }
51         cout << "Network " << testcase++ << endl
            ;
52         cout << "The bandwidth is " <<
            getMaxFlow(capacity, s, t, n) << ".
            "
53         << endl;
54         cout << endl;
55     }
56     return 0;

```

4.6 Floyd-warshall

```

1 /*SPA - Floyd-Warshall*/
2 #include<bits/stdc++.h>
3 #define inf 99999
4 using namespace std;

```



```

5 void floyd_warshall(vector<vector<int>>&
  distance, vector<vector<int>>& ancestor,
  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
          [k][j] < distance[i][j]){
10            distance[i][j] =
              distance[i][k] +
              distance[k][j];
11            ancestor[i][j] =
              ancestor[k][j];
12          }
13        }
14      }
15    }
16  }
17  int main(){
18    int n;
19    cin >> n;
20    int a, b, d;
21    vector<vector<int>> distance(n, vector<
      int>(n,99999));
22    vector<vector<int>> ancestor(n, vector<
      int>(n,-1));
23    while(cin>>a>>b>>d){
24      if(a == -1 && b == -1 && d == -1)
25        break;
26      distance[a][b] = d;
27      ancestor[a][b] = a;
28    }
29    for (int i = 0; i < n; i++){
30      distance[i][i] = 0;
31      floyd_warshall(distance, ancestor, n);
32      /*Negative cycle detection*/
33      for (int i = 0; i < n; i++){
34        if(distance[i][i] < 0){
35          cout << "Negative cycle!" <<
            endl;
36          break;
37        }
38      }
39      return 0;
40 }

```

4.7 Kruskal

```

1 /*mst - Kruskal*/
2 #include<bits/stdc++.h>
3 using namespace std;
4 struct edges{
5   int from;
6   int to;
7   int weight;
8   friend bool operator < (edges a, edges b
9   ){
10     return a.weight > b.weight;
11   }
12 };
13 int find(int x,vector<int>& union_set){
  if(x != union_set[x])

```

```

14   union_set[x] = find(union_set[x],
      union_set);
15   return union_set[x];
16 }
17 void merge(int a,int b,vector<int>&
  union_set){
18   int pa = find(a, union_set);
19   int pb = find(b, union_set);
20   if(pa != pb)
21     union_set[pa] = pb;
22 }
23 void kruskal(priority_queue<edges> pq,int n)
  {
24   vector<int> union_set(n, 0);
25   for (int i = 0; i < n; i++)
26     union_set[i] = i;
27   int edge = 0;
28   int cost = 0; //evaluate cost of mst
29   while(!pq.empty() && edge < n - 1){
30     edges cur = pq.top();
31     int from = find(cur.from, union_set)
      ;
32     int to = find(cur.to, union_set);
33     if(from != to){
34       merge(from, to, union_set);
35       edge += 1;
36       cost += cur.weight;
37     }
38     pq.pop();
39   }
40   if(edge < n-1)
41     cout << "No mst" << endl;
42   else
43     cout << cost << endl;
44 }
45 int main(){
46   int n;
47   cin >> n;
48   int a, b, d;
49   priority_queue<edges> pq;
50   while(cin>>a>>b>>d){
51     if(a == -1 && b == -1 && d == -1)
52       break;
53     edges tmp;
54     tmp.from = a;
55     tmp.to = b;
56     tmp.weight = d;
57     pq.push(tmp);
58   }
59   kruskal(pq, n);
60   return 0;
61 }

```

4.8 Prim

```

1 /*mst - Prim*/
2 #include<bits/stdc++.h>
3 #define inf 99999
4 using namespace std;
5 struct edges{
6   int from;
7   int to;
8   int weight;

```

```

9   friend bool operator < (edges a, edges b
      ){
10     return a.weight > b.weight;
11   }
12 };
13 void Prim(vector<vector<int>> gp,int n,int
  start){
14   vector<bool> pass(n,false);
15   int edge = 0;
16   int cost = 0; //evaluate cost of mst
17   priority_queue<edges> pq;
18   for (int i = 0; i < n; i++){
19     if(gp[start][i] != inf){
20       edges tmp;
21       tmp.from = start;
22       tmp.to = i;
23       tmp.weight = gp[start][i];
24       pq.push(tmp);
25     }
26   }
27   pass[start] = true;
28   while(!pq.empty() && edge < n-1){
29     edges cur = pq.top();
30     pq.pop();
31     if(!pass[cur.to]){
32       for (int i = 0; i < n; i++){
33         if(gp[cur.to][i] != inf){
34           edges tmp;
35           tmp.from = cur.to;
36           tmp.to = i;
37           tmp.weight = gp[cur.to][
              i];
38           pq.push(tmp);
39         }
40       }
41       pass[cur.to] = true;
42       edge += 1;
43       cost += cur.weight;
44     }
45   }
46   if(edge < n-1)
47     cout << "No mst" << endl;
48   else
49     cout << cost << endl;
50 }
51 int main(){
52   int n;
53   cin >> n;
54   int a, b, d;
55   vector<vector<int>> gp(n,vector<int>(n,
      inf));
56   while(cin>>a>>b>>d){
57     if(a == -1 && b == -1 && d == -1)
58       break;
59     if(gp[a][b] > d)
60       gp[a][b] = d;
61   }
62   Prim(gp,n,0);
63   return 0;
64 }

```

4.9 Union_find

```

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

```

5 Mathematics

5.1 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   }

```

5.2 Extended Euclidean

```

1 // ax + by = gcd(a,b)
2 pair<long long, long long> extgcd(long long
  a, long long b)
3 {
4     if (b == 0)
5         return {1, 0};
6     long long k = a / b;
7     pair<long long, long long> p = extgcd(b,
  a - k * b);
8     //cout << p.first << " " << p.second <<
  endl;
9     //cout << "商數(k)= " << k << endl <<
  endl;
10    return {p.second, p.first - k * p.second
  };
11 }
12
13 int main()
14 {
15     int a, b;
16     cin >> a >> b;
17     pair<long long, long long> xy = extgcd(a
  , b); //(x0,y0)
18     cout << xy.first << " " << xy.second <<
  endl;
19     cout << xy.first << " * " << a << " + "
  << xy.second << " * " << b << endl;
20     return 0;
21 }
22 // ax + by = gcd(a,b) * r
23 /*find |x|+|y| -> min*/
24 int main()
25 {
26     long long r, p, q; /*px+qy = r*/
27     int cases;
28     cin >> cases;
29     while (cases--)
30     {
31         cin >> r >> p >> q;
32         pair<long long, long long> xy =
  extgcd(q, p); //(x0,y0)
33         long long ans = 0, tmp = 0;
34         double k, k1;
35         long long s, s1;
36         k = 1 - (double)(r * xy.first) / p;
37         s = round(k);
38         ans = llabs(r * xy.first + s * p) +
  llabs(r * xy.second - s * q);
39         k1 = -(double)(r * xy.first) / p;
40         s1 = round(k1);
41         /*cout << k << endl << k1 << endl;
42          cout << s << endl << s1 << endl;
43          */
44         tmp = llabs(r * xy.first + s1 * p) +
  llabs(r * xy.second - s1 * q);
45         ans = min(ans, tmp);
46         cout << ans << endl;
47     }
48     return 0;
49 }

```

5.3 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         if (num[i] == '0' && num[i] == '9')
8         {
9             temp += (num[i] - 48) base;
10            base = base * 16;
11        }
12        else if (num[i] == 'A' && num[i] == 'F'
  )
13        {
14            temp += (num[i] - 55) base;
15            base = base * 16;
16        }
17    }
18    return temp;
19 }
20 void DecToHex(int p_intValue) //10 to 16
21 {
22     char l_pCharRes = new (char);
23     sprintf(l_pCharRes, %X, p_intValue);
24     int l_intResult = stoi(l_pCharRes);
25     cout << l_pCharRes << endl;
26     return l_intResult;
27 }

```

5.4 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 +
  (b + c)) % p;
17 // 乘法結合律: ((a * b) % p * c) % p = (a *
  (b * c)) % p;
18 // 加法交換律: (a + b) % p = (b + a) % p;
19 // 乘法交換律: (a * b) % p = (b * a) % p;
20 // 結合律: ((a + b) % p * c) = ((a * c) % p
  + (b * c) % p) % p;
21

```

```

22 // 如果 a ≡ b(mod m) · 我們會說 a,b 在模 m
  下同餘。
23 // 整除性: a ≡ b(mod m) ⇔ c | m = a - b, c
  | Z ⇔ a ≡ b (mod m) ⇔ m|a-b
24 // 遞移性: 若 a ≡ b (mod c), b ≡ d(mod c) 則
  a ≡ d (mod c)
25 /****基本運算****/
26 // a ≡ b (mod m) ⇔ { a ± c ≡ b ± d (mod m) }
27 // c ≡ d (mod m) ⇔ { a * c ≡ b * d (mod m) }
28 // 放大縮小模數: k ∈ Z+, a ≡ b (mod m) ⇔ k | a
  ≡ k | b (mod k|m)

```

5.5 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());

```

5.6 PI

```

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

```

5.7 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(); ++j)
13         {
14             if (i * p[j] > maxn)
15                 break;
16             is_notp[i * p[j]] = 1;
17             if (i % p[j] == 0)
18                 break;
19         }
20     }
21 }

```

5.8 primeBOOL

```

1 // n < 4759123141    chk = [2, 7, 61]
2 // n < 1122004669633  chk = [2, 13, 23,
  1662803]
3 // n < 2^64          chk = [2, 325, 9375,
  28178, 450775, 9780504, 1795265022]
4 long long fmul(long long a, long long n,
  long long mod)
5 {
6     long long ret = 0;
7     for (; n; n >>= 1)
8     {
9         if (n & 1)
10             (ret += a) %= mod;
11         (a += a) %= mod;
12     }
13     return ret;
14 }
15
16 long long fpow(long long a, long long n,
  long long mod)
17 {
18     long long ret = 1LL;
19     for (; n; n >>= 1)
20     {
21         if (n & 1)
22             ret = fmul(ret, a, mod);
23         a = fmul(a, a, mod);
24     }
25     return ret;
26 }
27 bool check(long long a, long long u, long
  long n, int t)
28 {
29     a = fpow(a, u, n);
30     if (a == 0)
31         return true;
32     if (a == 1 || a == n - 1)
33         return true;
34     for (int i = 0; i < t; ++i)
35     {
36         a = fmul(a, a, n);
37         if (a == 1)
38             return false;
39         if (a == n - 1)
40             return true;
41     }
42     return false;
43 }
44 bool is_prime(long long n)
45 {
46     if (n < 2)
47         return false;
48     if (n % 2 == 0)
49         return n == 2;
50     long long u = n - 1;
51     int t = 0;
52     for (; u & 1; u >>= 1, ++t)
53         ;
54     for (long long i : chk)
55     {
56         if (!check(i, u, n, t))
57             return false;
58     }
59     return true;
60 }
61

```

```
62 // if (is_prime(int num)) // true == prime
    反之亦然
```

5.9 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("%.3lf\n", R);
14 }
```

5.10 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,
    string final index)
3 {
4     int c = 0;
5     for (int i = ri; i >= le; i--)
6     {
7         if (s[i] == ')')
8             c++;
9         if (s[i] == '(')
10            c--;
11        if (s[i] == '+' && c == 0)
12            return DFS(le, i - 1) + DFS(i + 1, ri);
13        if (s[i] == '-' && c == 0)
14            return DFS(le, i - 1) - DFS(i + 1, ri);
15    }
16    for (int i = ri; i >= le; i--)
17    {
18        if (s[i] == ')')
19            c++;
20        if (s[i] == '(')
21            c--;
22        if (s[i] == '*' && c == 0)
23            return DFS(le, i - 1) * DFS(i + 1, ri);
24        if (s[i] == '/' && c == 0)
25            return DFS(le, i - 1) / DFS(i + 1, ri);
26        if (s[i] == '%' && c == 0)
27            return DFS(le, i - 1) % DFS(i + 1, ri);
28    }
29    if ((s[le] == '(' && (s[ri] == ')'))
30        return DFS(le + 1, ri - 1); //去除刮號
```

```
31 if (s[le] == ' ' && s[ri] == ' ')
32     return DFS(le + 1, ri - 1); //去除左
    右兩邊空格
33 if (s[le] == ' ')
34     return DFS(le + 1, ri); //去除左邊空
    格
35 if (s[ri] == ' ')
36     return DFS(le, ri - 1); //去除右邊空
    格
37 long long int num = 0;
38 for (int i = le; i <= ri; i++)
39     num = num * 10 + s[i] - '0';
40 return num;
41 }
```

5.11 數字乘法組合

```
1 void toans(vector<vector<int>> &ans, vector<
    int> com)
2 {
3     // sort(com.begin(), com.end());
4     ans.push_back(com);
5     // for (auto i : com)
6         cout << i << ' ';
7     // cout << endl;
8 }
9 void finds(int j, int old, int num, vector<
    int> com, vector<vector<int>> &ans)
10 {
11     for (int i = j; i <= sqrt(num); i++)
12     {
13         if (old == num)
14             com.clear();
15         if (num % i == 0)
16         {
17             vector<int> a;
18             a = com;
19             a.push_back(i);
20             finds(i, old, num / i, a, ans);
21             a.push_back(num / i);
22             toans(ans, a);
23         }
24     }
25 }
26 int main()
27 {
28     vector<vector<int>> ans;
29     vector<int> zero;
30     finds(2, num, num, zero, ans);
31     // num 為 input 數字
32     for (int i = 0; i < ans.size(); i++)
33     {
34         for (int j = 0; j < ans[i].size() - 1; j++)
35             cout << ans[i][j] << " ";
36         cout << ans[i][ans[i].size() - 1] << endl;
37     }
38 }
```

5.12 數字加法組合

```
1 void printCombination(vector<int> const &out
    , int m, vector<vector<int>> &ans)
2 {
3     for (int i : out)
4         if (i > m)
5             return;
6     ans.push_back(out);
7 }
8 void recur(int i, int n, int m, vector<int>
    &out, vector<vector<int>> &ans)
9 {
10    if (n == 0)
11        printCombination(out, m, ans);
12    for (int j = i; j <= n; j++)
13    {
14        out.push_back(j);
15        recur(j, n - j, m, out, ans);
16        out.pop_back();
17    }
18 }
19 int main()
20 {
21     vector<vector<int>> ans;
22     vector<int> zero;
23     recur(1, num, num, zero, ans);
24     // num 為 input 數字
25     for (int i = 0; i < ans.size(); i++)
26     {
27         for (int j = 0; j < ans[i].size() - 1; j++)
28             cout << ans[i][j] << " ";
29         cout << ans[i][ans[i].size() - 1] << endl;
30     }
31 }
32 }
```

5.13 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 | }

5.14 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 }
```

6 Other

6.1 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     return result;
34 }
35 }

```

6.2 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>> &boxs
  , int index, int n)
8 {
9     int n2 = n * n;
10    int rowNum = index / n2, colNum = index
    % n2;
11    if (index >= n2 * n2)
12        return true;
13
14    if (board[rowNum][colNum] != 0)
15        return backtracking(board, rows,
    cols, boxs, index + 1, n);
16
17    for (int i = 1; i <= n2; i++)
18    {
19        if (!rows[rowNum][i] && !cols[colNum]
    ][i] && !boxs[getSquareIndex(
    rowNum, colNum, n)][i])
20        {
21            rows[rowNum][i] = true;
22            cols[colNum][i] = true;
23            boxs[getSquareIndex(rowNum,
    colNum, n)][i] = true;
24            board[rowNum][colNum] = i;
25            if (backtracking(board, rows,
    cols, boxs, index + 1, n))
26                return true;
27            board[rowNum][colNum] = 0;
28            rows[rowNum][i] = false;
29            cols[colNum][i] = false;
30            boxs[getSquareIndex(rowNum,
    colNum, n)][i] = false;
31        }
32    }
33    return false;
34 }
35
36 /*用法 main*/
37 int n = sqrt(數獨邊長大小) /*e.g. 9*9 n=3*/
38 vector<vector<int>> board(n * n + 1, vector<
  int>(n * n + 1, 0));
39 vector<vector<bool>> isRow(n * n + 1, vector<
  bool>(n * n + 1, false));
40 vector<vector<bool>> isColumn(n * n + 1,
  vector<bool>(n * n + 1, false));
41 vector<vector<bool>> isSquare(n * n + 1,
  vector<bool>(n * n + 1, false));

```

```

42 for (int i = 0; i < n * n; ++i)
43 {
44     for (int j = 0; j < n * n; ++j)
45     {
46         int number;
47         cin >> number;
48         board[i][j] = number;
49         if (number == 0)
50             continue;
51         isRow[i][number] = true;
52         isColumn[j][number] = true;
53         isSquare[getSquareIndex(i, j, n)][
    number] = true;
54     }
55 }
56 if (backtracking(board, isRow, isColumn,
  isSquare, 0, n))
57     /*有解答*/
58 else
59     /*解答*/

```

7 String

7.1 Sliding window

```

1 string minWindow(string s, string t) {
2     unordered_map<char, int> letterCnt;
3     for (int i = 0; i < t.length(); i++)
4         letterCnt[t[i]]++;
5     int minLength = INT_MAX, minStart = -1;
6     int left = 0, matchCnt = 0;
7     for (int i = 0; i < s.length(); i++)
8     {
9         if (--letterCnt[s[i]] >= 0)
10             matchCnt++;
11         while (matchCnt == t.length())
12         {
13             if (i - left + 1 < minLength)
14             {
15                 minLength = i - left + 1;
16                 minStart = left;
17             }
18             if (++letterCnt[s[left]] > 0)
19                 matchCnt--;
20             left++;
21         }
22     }
23     return minLength == INT_MAX ? "" : s.
    substr(minStart, minLength);
24 }

```

7.2 Split

```

1 vector<string> mysplit(const string& str,
  const string& delim)
2 {
3     vector<string> res;

```

```

4     if (" " == str)
5         return res;
6
7     char *strs = new char[str.length() + 1];
8     strcpy(strs, str.c_str());
9
10    char *d = new char[delim.length() + 1];
11    strcpy(d, delim.c_str());
12
13    char *p = strtok(strs, d);
14    while (p)
15    {
16        string s = p;
17        res.push_back(s);
18        p = strtok(NULL, d);
19    }
20    return res;
21 }

```

8 data structure

8.1 Bigint

```

1 //台大
2 struct Bigint{
3     static const int LEN = 60;
4     static const int BIGMOD = 10000;
5     int s;
6     int vl, v[LEN];
7     // vector<int> v;
8     Bigint() : s(1) { vl = 0; }
9     Bigint(long long a) {
10         s = 1; vl = 0;
11         if (a < 0) { s = -1; a = -a; }
12         while (a) {
13             push_back(a % BIGMOD);
14             a /= BIGMOD;
15         }
16     }
17     Bigint(string str) {
18         s = 1; vl = 0;
19         int stPos = 0, num = 0;
20         if (!str.empty() && str[0] == '-') {
21             stPos = 1;
22             s = -1;
23         }
24         for (int i = str.length() - 1, q=1; i
    >=stPos; i--) {
25             num += (str[i] - '0') * q;
26             if ((q *= 10) >= BIGMOD) {
27                 push_back(num);
28                 num = 0; q = 1;
29             }
30         }
31         if (num) push_back(num);
32         n();
33     }
34     int len() const {
35         return vl; //return SZ(v);
36     }

```

```

37     bool empty() const { return len() == 0;
    }
38     void push_back(int x) {
39         v[vl++] = x; //v.PB(x);
40     }
41     void pop_back() {
42         vl--; //v.pop_back();
43     }
44     int back() const {
45         return v[vl-1]; //return v.back();
46     }
47     void n() {
48         while (!empty() && !back()) pop_back
    ();
49     }
50     void resize(int nl) {
51         vl = nl; //v.resize(nl);
52         fill(v, v+vl, 0); //fill(ALL(v), 0);
53     }
54     void print() const {
55         if (empty()) { putchar('0'); return;
    }
56         if (s == -1) putchar('-');
57         printf("%d", back());
58         for (int i=len()-2; i>=0; i--)
59             printf("%.4d", v[i]);
60     }
61     friend std::ostream& operator << (std:::
    ostream& out, const Bigint &a) {
62         if (a.empty()) { out << "0"; return
    out; }
63         if (a.s == -1) out << "-";
64         out << a.back();
65         for (int i=a.len()-2; i>=0; i--) {
66             char str[10];
67             snprintf(str, 5, "%.4d", a.v[i])
    ;
68             out << str;
69         }
70         return out;
71     }
72     int cp3(const Bigint &b) const {
73         if (s != b.s) return s - b.s;
74         if (s == -1) return -(*this).cp3(-b
    );
75         if (len() != b.len()) return len()-b
    .len(); //int
76         for (int i=len()-1; i>=0; i--)
77             if (v[i]!=b.v[i]) return v[i]-b.
    v[i];
78         return 0;
79     }
80     bool operator<(const Bigint &b) const
    { return cp3(b)<0; }
81     bool operator<=(const Bigint &b) const
    { return cp3(b)<=0; }
82     bool operator==(const Bigint &b) const
    { return cp3(b)==0; }
83     bool operator!=(const Bigint &b) const
    { return cp3(b)!=0; }
84     bool operator>(const Bigint &b) const
    { return cp3(b)>0; }
85     bool operator>=(const Bigint &b) const
    { return cp3(b)>=0; }
86     Bigint operator - () const {
87         Bigint r = (*this);
88         r.s = -r.s;
89         return r;
90     }
91     Bigint operator + () const {
92         Bigint r = (*this);
93         r.s = r.s;
94         return r;
95     }
96     Bigint operator * (int x) const {
97         Bigint r = (*this);
98         r *= x;
99         return r;
100    }
101    Bigint operator / (int x) const {
102        Bigint r = (*this);
103        r /= x;
104        return r;
105    }
106    Bigint operator % (int x) const {
107        Bigint r = (*this);
108        r %= x;
109        return r;
110    }
111    Bigint operator %> (int x) const {
112        Bigint r = (*this);
113        r %= x;
114        return r;
115    }
116    Bigint operator %< (int x) const {
117        Bigint r = (*this);
118        r %= x;
119        return r;
120    }
121    Bigint operator %>= (int x) const {
122        Bigint r = (*this);
123        r %= x;
124        return r;
125    }
126    Bigint operator %<= (int x) const {
127        Bigint r = (*this);
128        r %= x;
129        return r;
130    }
131    Bigint operator %>= (int x) const {
132        Bigint r = (*this);
133        r %= x;
134        return r;
135    }
136    Bigint operator %<= (int x) const {
137        Bigint r = (*this);
138        r %= x;
139        return r;
140    }
141    Bigint operator %>= (int x) const {
142        Bigint r = (*this);
143        r %= x;
144        return r;
145    }
146    Bigint operator %<= (int x) const {
147        Bigint r = (*this);
148        r %= x;
149        return r;
150    }
151    Bigint operator %>= (int x) const {
152        Bigint r = (*this);
153        r %= x;
154        return r;
155    }
156    Bigint operator %<= (int x) const {
157        Bigint r = (*this);
158        r %= x;
159        return r;
160    }
161    Bigint operator %>= (int x) const {
162        Bigint r = (*this);
163        r %= x;
164        return r;
165    }
166    Bigint operator %<= (int x) const {
167        Bigint r = (*this);
168        r %= x;
169        return r;
170    }
171    Bigint operator %>= (int x) const {
172        Bigint r = (*this);
173        r %= x;
174        return r;
175    }
176    Bigint operator %<= (int x) const {
177        Bigint r = (*this);
178        r %= x;
179        return r;
180    }
181    Bigint operator %>= (int x) const {
182        Bigint r = (*this);
183        r %= x;
184        return r;
185    }
186    Bigint operator %<= (int x) const {
187        Bigint r = (*this);
188        r %= x;
189        return r;
190    }
191    Bigint operator %>= (int x) const {
192        Bigint r = (*this);
193        r %= x;
194        return r;
195    }
196    Bigint operator %<= (int x) const {
197        Bigint r = (*this);
198        r %= x;
199        return r;
200    }
201    Bigint operator %>= (int x) const {
202        Bigint r = (*this);
203        r %= x;
204        return r;
205    }
206    Bigint operator %<= (int x) const {
207        Bigint r = (*this);
208        r %= x;
209        return r;
210    }
211    Bigint operator %>= (int x) const {
212        Bigint r = (*this);
213        r %= x;
214        return r;
215    }
216    Bigint operator %<= (int x) const {
217        Bigint r = (*this);
218        r %= x;
219        return r;
220    }
221    Bigint operator %>= (int x) const {
222        Bigint r = (*this);
223        r %= x;
224        return r;
225    }
226    Bigint operator %<= (int x) const {
227        Bigint r = (*this);
228        r %= x;
229        return r;
230    }
231    Bigint operator %>= (int x) const {
232        Bigint r = (*this);
233        r %= x;
234        return r;
235    }
236    Bigint operator %<= (int x) const {
237        Bigint r = (*this);
238        r %= x;
239        return r;
240    }
241    Bigint operator %>= (int x) const {
242        Bigint r = (*this);
243        r %= x;
244        return r;
245    }
246    Bigint operator %<= (int x) const {
247        Bigint r = (*this);
248        r %= x;
249        return r;
250    }
251    Bigint operator %>= (int x) const {
252        Bigint r = (*this);
253        r %= x;
254        return r;
255    }
256    Bigint operator %<= (int x) const {
257        Bigint r = (*this);
258        r %= x;
259        return r;
260    }
261    Bigint operator %>= (int x) const {
262        Bigint r = (*this);
263        r %= x;
264        return r;
265    }
266    Bigint operator %<= (int x) const {
267        Bigint r = (*this);
268        r %= x;
269        return r;
270    }
271    Bigint operator %>= (int x) const {
272        Bigint r = (*this);
273        r %= x;
274        return r;
275    }
276    Bigint operator %<= (int x) const {
277        Bigint r = (*this);
278        r %= x;
279        return r;
280    }
281    Bigint operator %>= (int x) const {
282        Bigint r = (*this);
283        r %= x;
284        return r;
285    }
286    Bigint operator %<= (int x) const {
287        Bigint r = (*this);
288        r %= x;
289        return r;
290    }
291    Bigint operator %>= (int x) const {
292        Bigint r = (*this);
293        r %= x;
294        return r;
295    }
296    Bigint operator %<= (int x) const {
297        Bigint r = (*this);
298        r %= x;
299        return r;
300    }
301    Bigint operator %>= (int x) const {
302        Bigint r = (*this);
303        r %= x;
304        return r;
305    }
306    Bigint operator %<= (int x) const {
307        Bigint r = (*this);
308        r %= x;
309        return r;
310    }
311    Bigint operator %>= (int x) const {
312        Bigint r = (*this);
313        r %= x;
314        return r;
315    }
316    Bigint operator %<= (int x) const {
317        Bigint r = (*this);
318        r %= x;
319        return r;
320    }
321    Bigint operator %>= (int x) const {
322        Bigint r = (*this);
323        r %= x;
324        return r;
325    }
326    Bigint operator %<= (int x) const {
327        Bigint r = (*this);
328        r %= x;
329        return r;
330    }
331    Bigint operator %>= (int x) const {
332        Bigint r = (*this);
333        r %= x;
334        return r;
335    }
336    Bigint operator %<= (int x) const {
337        Bigint r = (*this);
338        r %= x;
339        return r;
340    }
341    Bigint operator %>= (int x) const {
342        Bigint r = (*this);
343        r %= x;
344        return r;
345    }
346    Bigint operator %<= (int x) const {
347        Bigint r = (*this);
348        r %= x;
349        return r;
350    }
351    Bigint operator %>= (int x) const {
352        Bigint r = (*this);
353        r %= x;
354        return r;
355    }
356    Bigint operator %<= (int x) const {
357        Bigint r = (*this);
358        r %= x;
359        return r;
360    }
361    Bigint operator %>= (int x) const {
362        Bigint r = (*this);
363        r %= x;
364        return r;
365    }
366    Bigint operator %<= (int x) const {
367        Bigint r = (*this);
368        r %= x;
369        return r;
370    }
371    Bigint operator %>= (int x) const {
372        Bigint r = (*this);
373        r %= x;
374        return r;
375    }
376    Bigint operator %<= (int x) const {
377        Bigint r = (*this);
378        r %= x;
379        return r;
380    }
381    Bigint operator %>= (int x) const {
382        Bigint r = (*this);
383        r %= x;
384        return r;
385    }
386    Bigint operator %<= (int x) const {
387        Bigint r = (*this);
388        r %= x;
389        return r;
390    }
391    Bigint operator %>= (int x) const {
392        Bigint r = (*this);
393        r %= x;
394        return r;
395    }
396    Bigint operator %<= (int x) const {
397        Bigint r = (*this);
398        r %= x;
399        return r;
400    }
401    Bigint operator %>= (int x) const {
402        Bigint r = (*this);
403        r %= x;
404        return r;
405    }
406    Bigint operator %<= (int x) const {
407        Bigint r = (*this);
408        r %= x;
409        return r;
410    }
411    Bigint operator %>= (int x) const {
412        Bigint r = (*this);
413        r %= x;
414        return r;
415    }
416    Bigint operator %<= (int x) const {
417        Bigint r = (*this);
418        r %= x;
419        return r;
420    }
421    Bigint operator %>= (int x) const {
422        Bigint r = (*this);
423        r %= x;
424        return r;
425    }
426    Bigint operator %<= (int x) const {
427        Bigint r = (*this);
428        r %= x;
429        return r;
430    }
431    Bigint operator %>= (int x) const {
432        Bigint r = (*this);
433        r %= x;
434        return r;
435    }
436    Bigint operator %<= (int x) const {
437        Bigint r = (*this);
438        r %= x;
439        return r;
440    }
441    Bigint operator %>= (int x) const {
442        Bigint r = (*this);
443        r %= x;
444        return r;
445    }
446    Bigint operator %<= (int x) const {
447        Bigint r = (*this);
448        r %= x;
449        return r;
450    }
451    Bigint operator %>= (int x) const {
452        Bigint r = (*this);
453        r %= x;
454        return r;
455    }
456    Bigint operator %<= (int x) const {
457        Bigint r = (*this);
458        r %= x;
459        return r;
460    }
461    Bigint operator %>= (int x) const {
462        Bigint r = (*this);
463        r %= x;
464        return r;
465    }
466    Bigint operator %<= (int x) const {
467        Bigint r = (*this);
468        r %= x;
469        return r;
470    }
471    Bigint operator %>= (int x) const {
472        Bigint r = (*this);
473        r %= x;
474        return r;
475    }
476    Bigint operator %<= (int x) const {
477        Bigint r = (*this);
478        r %= x;
479        return r;
480    }
481    Bigint operator %>= (int x) const {
482        Bigint r = (*this);
483        r %= x;
484        return r;
485    }
486    Bigint operator %<= (int x) const {
487        Bigint r = (*this);
488        r %= x;
489        return r;
490    }
491    Bigint operator %>= (int x) const {
492        Bigint r = (*this);
493        r %= x;
494        return r;
495    }
496    Bigint operator %<= (int x) const {
497        Bigint r = (*this);
498        r %= x;
499        return r;
500    }
501    Bigint operator %>= (int x) const {
502        Bigint r = (*this);
503        r %= x;
504        return r;
505    }
506    Bigint operator %<= (int x) const {
507        Bigint r = (*this);
508        r %= x;
509        return r;
510    }
511    Bigint operator %>= (int x) const {
512        Bigint r = (*this);
513        r %= x;
514        return r;
515    }
516    Bigint operator %<= (int x) const {
517        Bigint r = (*this);
518        r %= x;
519        return r;
520    }
521    Bigint operator %>= (int x) const {
522        Bigint r = (*this);
523        r %= x;
524        return r;
525    }
526    Bigint operator %<= (int x) const {
527        Bigint r = (*this);
528        r %= x;
529        return r;
530    }
531    Bigint operator %>= (int x) const {
532        Bigint r = (*this);
533        r %= x;
534        return r;
535    }
536    Bigint operator %<= (int x) const {
537        Bigint r = (*this);
538        r %= x;
539        return r;
540    }
541    Bigint operator %>= (int x) const {
542        Bigint r = (*this);
543        r %= x;
544        return r;
545    }
546    Bigint operator %<= (int x) const {
547        Bigint r = (*this);
548        r %= x;
549        return r;
550    }
551    Bigint operator %>= (int x) const {
552        Bigint r = (*this);
553        r %= x;
554        return r;
555    }
556    Bigint operator %<= (int x) const {
557        Bigint r = (*this);
558        r %= x;
559        return r;
560    }
561    Bigint operator %>= (int x) const {
562        Bigint r = (*this);
563        r %= x;
564        return r;
565    }
566    Bigint operator %<= (int x) const {
567        Bigint r = (*this);
568        r %= x;
569        return r;
570    }
571    Bigint operator %>= (int x) const {
572        Bigint r = (*this);
573        r %= x;
574        return r;
575    }
576    Bigint operator %<= (int x) const {
577        Bigint r = (*this);
578        r %= x;
579        return r;
580    }
581    Bigint operator %>= (int x) const {
582        Bigint r = (*this);
583        r %= x;
584        return r;
585    }
586    Bigint operator %<= (int x) const {
587        Bigint r = (*this);
588        r %= x;
589        return r;
590    }
591    Bigint operator %>= (int x) const {
592        Bigint r = (*this);
593        r %= x;
594        return r;
595    }
596    Bigint operator %<= (int x) const {
597        Bigint r = (*this);
598        r %= x;
599        return r;
600    }
601    Bigint operator %>= (int x) const {
602        Bigint r = (*this);
603        r %= x;
604        return r;
605    }
606    Bigint operator %<= (int x) const {
607        Bigint r = (*this);
608        r %= x;
609        return r;
610    }
611    Bigint operator %>= (int x) const {
612        Bigint r = (*this);
613        r %= x;
614        return r;
615    }
616    Bigint operator %<= (int x) const {
617        Bigint r = (*this);
618        r %= x;
619        return r;
620    }
621    Bigint operator %>= (int x) const {
622        Bigint r = (*this);
623        r %= x;
624        return r;
625    }
626    Bigint operator %<= (int x) const {
627        Bigint r = (*this);
628        r %= x;
629        return r;
630    }
631    Bigint operator %>= (int x) const {
632        Bigint r = (*this);
633        r %= x;
634        return r;
635    }
636    Bigint operator %<= (int x) const {
637        Bigint r = (*this);
638        r %= x;
639        return r;
640    }
641    Bigint operator %>= (int x) const {
642        Bigint r = (*this);
643        r %= x;
644        return r;
645    }
646    Bigint operator %<= (int x) const {
647        Bigint r = (*this);
648        r %= x;
649        return r;
650    }
651    Bigint operator %>= (int x) const {
652        Bigint r = (*this);
653        r %= x;
654        return r;
655    }
656    Bigint operator %<= (int x) const {
657        Bigint r = (*this);
658        r %= x;
659        return r;
660    }
661    Bigint operator %>= (int x) const {
662        Bigint r = (*this);
663        r %= x;
664        return r;
665    }
666    Bigint operator %<= (int x) const {
667        Bigint r = (*this);
668        r %= x;
669        return r;
670    }
671    Bigint operator %>= (int x) const {
672        Bigint r = (*this);
673        r %= x;
674        return r;
675    }
676    Bigint operator %<= (int x) const {
677        Bigint r = (*this);
678        r %= x;
679        return r;
680    }
681    Bigint operator %>= (int x) const {
682        Bigint r = (*this);
683        r %= x;
684        return r;
685    }
686    Bigint operator %<= (int x) const {
687        Bigint r = (*this);
688        r %= x;
689        return r;
690    }
691    Bigint operator %>= (int x) const {
692        Bigint r = (*this);
693        r %= x;
694        return r;
695    }
696    Bigint operator %<= (int x) const {
697        Bigint r = (*this);
698        r %= x;
699        return r;
700    }
701    Bigint operator %>= (int x) const {
702        Bigint r = (*this);
703        r %= x;
704        return r;
705    }
706    Bigint operator %<= (int x) const {
707        Bigint r = (*this);
708        r %= x;
709        return r;
710    }
711    Bigint operator %>= (int x) const {
712        Bigint r = (*this);
713        r %= x;
714        return r;
715    }
716    Bigint operator %<= (int x) const {
717        Bigint r = (*this);
718        r %= x;
719        return r;
720    }
721    Bigint operator %>= (int x) const {
722        Bigint r = (*this);
723        r %= x;
724        return r;
725    }
726    Bigint operator %<= (int x) const {
727        Bigint r = (*this);
728        r %= x;
729        return r;
730    }
731    Bigint operator %>= (int x) const {
732        Bigint r = (*this);
733        r %= x;
734        return r;
735    }
736    Bigint operator %<= (int x) const {
737        Bigint r = (*this);
738        r %= x;
739        return r;
740    }
741    Bigint operator %>= (int x) const {
742        Bigint r = (*this);
743        r %= x;
744        return r;
745    }
746    Bigint operator %<= (int x) const {
747        Bigint r = (*this);
748        r %= x;
749        return r;
750    }
751    Bigint operator %>= (int x) const {
752        Bigint r = (*this);
753        r %= x;
754        return r;
755    }
756    Bigint operator %<= (int x) const {
757        Bigint r = (*this);
758        r %= x;
759        return r;
760    }
761    Bigint operator %>= (int x) const {
762        Bigint r = (*this);
763        r %= x;
764        return r;
765    }
766    Bigint operator %<= (int x) const {
767        Bigint r = (*this);
768        r %= x;
769        return r;
770    }
771    Bigint operator %>= (int x) const {
772        Bigint r = (*this);
773        r %= x;
774        return r;
775    }
776    Bigint operator %<= (int x) const {
777        Bigint r = (*this);
778        r %= x;
779        return r;
780    }
781    Bigint operator %>= (int x) const {
782        Bigint r = (*this);
783        r %= x;
784        return r;
785    }
786    Bigint operator %<= (int x) const {
787        Bigint r = (*this);
788        r %= x;
789        return r;
790    }
791    Bigint operator %>= (int x) const {
792        Bigint r = (*this);
793        r %= x;
794        return r;
795    }
796    Bigint operator %<= (int x) const {
797        Bigint r = (*this);
798        r %= x;
799        return r;
800    }
801    Bigint operator %>= (int x) const {
802        Bigint r = (*this);
803        r %= x;
804        return r;
805    }
806    Bigint operator %<= (int x) const {
807        Bigint r = (*this);
808        r %= x;
809        return r;
810    }
811    Bigint operator %>= (int x) const {
812        Bigint r = (*this);
813        r %= x;
814        return r;
815    }
816    Bigint operator %<= (int x) const {
817        Bigint r = (*this);
818        r %= x;
819        return r;
820    }
821    Bigint operator %>= (int x) const {
822        Bigint r = (*this);
823        r %= x;
824        return r;
825    }
826    Bigint operator %<= (int x) const {
827        Bigint r = (*this);
828        r %= x;
829        return r;
830    }
831    Bigint operator %>= (int x) const {
832        Bigint r = (*this);
833        r %= x;
834        return r;
835    }
836    Bigint operator %<= (int x) const {
837        Bigint r = (*this);
838        r %= x;
839        return r;
840    }
841    Bigint operator %>= (int x) const {
842        Bigint r = (*this);
843        r %= x;
844        return r;
845    }
846    Bigint operator %<= (int x) const {
847        Bigint r = (*this);
848        r %= x;
849        return r;
850    }
851    Bigint operator %>= (int x) const {
852        Bigint r = (*this);
853        r %= x;
854        return r;
855    }
856    Bigint operator %<= (int x) const {
857        Bigint r = (*this);
858        r %= x;
859        return r;
860    }
861    Bigint operator %>= (int x) const {
862        Bigint r = (*this);
863        r %= x;
864        return r;
865    }
866    Bigint operator %<= (int x) const {
867        Bigint r = (*this);
868        r %= x;
869        return r;
870    }
871    Bigint operator %>= (int x) const {
872        Bigint r = (*this);
873        r %= x;
874        return r;
875    }
876    Bigint operator %<= (int x) const {
877        Bigint r = (*this);
878        r %= x;
879        return r;
880    }
881    Bigint operator %>= (int x) const {
882        Bigint r = (*this);
883        r %= x;
884        return r;
885    }
886    Bigint operator %<= (int x) const {
887        Bigint r = (*this);
888        r %= x;
889        return r;
890    }
891    Bigint operator %>= (int x) const {
892        Bigint r = (*this);
893        r %= x;
894        return r;
895    }
896    Bigint operator %<= (int x) const {
897        Bigint r = (*this);
898        r %= x;
899        return r;
900    }
901    Bigint operator %>= (int x) const {
902        Bigint r = (*this);
903        r %= x;
904        return r;
905    }
906    Bigint operator %<= (int x) const {
907        Bigint r = (*this);
908        r %= x;
909        return r;
910    }
911    Bigint operator %>= (int x) const {
912        Bigint r = (*this);
913        r %= x;
914        return r;
915    }
916    Bigint operator %<= (int x) const {
917        Bigint r = (*this);
918        r %= x;
919        return r;
920    }
921    Bigint operator %>= (int x) const {
922        Bigint r = (*this);
923        r %= x;
924        return r;
925    }
926    Bigint operator %<= (int x) const {
927        Bigint r = (*this);
928        r %= x;
929        return r;
930    }
931    Bigint operator %>= (int x) const {
932        Bigint r = (*this);
933        r %= x;
934        return r;
935    }
936    Bigint operator %<= (int x) const {
937        Bigint r = (*this);
938        r %= x;
939        return r;
940    }
941    Bigint operator %>= (int
```

```

93     r.s = -r.s;
94     return r;
95 }
96 Bigint operator + (const Bigint &b)
97 {
98     const {
99         if (s == -1) return -(*this)+(-b);
100     };
101     if (b.s == -1) return (*this)-(-b);
102     Bigint r;
103     int nl = max(len(), b.len());
104     r.resize(nl + 1);
105     for (int i=0; i<nl; i++) {
106         if (i < len()) r.v[i] += v[i];
107         if (i < b.len()) r.v[i] += b.v[i];
108     };
109     if (r.v[i] >= BIGMOD) {
110         r.v[i+1] += r.v[i] / BIGMOD;
111         r.v[i] %= BIGMOD;
112     }
113     r.n();
114     return r;
115 }
116 Bigint operator - (const Bigint &b)
117 {
118     const {
119         if (s == -1) return -(*this)-(-b);
120     };
121     if (b.s == -1) return (*this)+(-b);
122     if ((*this) < b) return -(b-(*this));
123     Bigint r;
124     r.resize(len());
125     for (int i=0; i<len(); i++) {
126         r.v[i] += v[i];
127         if (i < b.len()) r.v[i] -= b.v[i];
128     };
129     if (r.v[i] < 0) {
130         r.v[i] += BIGMOD;
131         r.v[i+1]--;
132     }
133     r.n();
134     return r;
135 }
136 Bigint operator * (const Bigint &b) {
137     Bigint r;
138     r.resize(len() + b.len() + 1);
139     r.s = s * b.s;
140     for (int i=0; i<len(); i++) {
141         for (int j=0; j<b.len(); j++) {
142             r.v[i+j] += v[i] * b.v[j];
143             if (r.v[i+j] >= BIGMOD) {
144                 r.v[i+j+1] += r.v[i+j] / BIGMOD;
145                 r.v[i+j] %= BIGMOD;
146             }
147         }
148     }
149     r.n();
150     return r;
151 }
152 Bigint operator / (const Bigint &b) {
153     Bigint r;
154     r.resize(max(1, len()-b.len()+1));
155     int oriS = s;
156     Bigint b2 = b; // b2 = abs(b)

```

```

151     s = b2.s = r.s = 1;
152     for (int i=r.len()-1; i>=0; i--) {
153         int d=0, u=BIGMOD-1;
154         while(d<u) {
155             int m = (d+u+1)>>1;
156             r.v[i] = m;
157             if((r*b2) > (*this)) u = m-1;
158             else d = m;
159         }
160         r.v[i] = d;
161     }
162     s = oriS;
163     r.s = s * b.s;
164     r.n();
165     return r;
166 }
167 Bigint operator % (const Bigint &b) {
168     return (*this)-(*this)/b*b;
169 }
170 };

```

8.2 MFlow

```

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]] + 1;
50                    que[++tail] = v[i];
51                }
52            }
53            ++head;
54        }
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        for (int i = first[x]; i != -1; i = next[i])
63        {
64            if (dis[x] + 1 == dis[v[i]] && (
65                f = dfs(v[i], min(a, cap[i] - flow[i])) > 0))
66            {
67                flow[i] += f;
68                flow[i ^ 1] -= f;
69                a -= f;
70                flw += f;
71                if (a == 0) break;
72            }
73        }
74        return flw;
75    }
76    ll MaxFlow(int s, int t)
77    {
78        this->s = s;
79        this->t = t;
80        ll flw = 0;
81        while (bfs())
82        {
83            for (int i = 1; i <= n; ++i)
84                cur[i] = 0;
85            flw += dfs(s, oo);
86        }
87        return flw;
88    }
89 }
90 // MF Net;
91 // Net.n = n;
92 // Net.Clear();
93 // a 到 b (注意從1開始!!!!)
94 // Net.Add(a, b, w, 0);
95 // Net.MaxFlow(s, d)
96 // s 到 d 的 MF

```

8.3 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) / WIDTH + 1;
21        for(int i = 0; i < len; i++){
22            int end = str.length() - i*WIDTH;
23            int start = max(0, end-WIDTH);
24            sscanf(str.substr(start, end-start).c_str(), "%d", &x);
25            s.push_back(x);
26        }
27        return *this;
28    }
29    BigInteger operator + (const BigInteger& b) const{
30        BigInteger c;
31        c.s.clear();
32        for(int i = 0, g = 0; i < s.size() || b.s.size() > i; i++){
33            if(g == 0 && i >= s.size() && i >= 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
44 ostream& operator << (ostream &out, const 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 }
56

```

```

57 istream& operator >> (istream &in,
58     BigInteger& x){
59     string s;
60     if(!(in >> s))
61         return in;
62
63     x = s;
64     return in;
65 }
66
67 struct Trie{
68     int c[5000005][10];
69     int val[5000005];
70     int sz;
71     int getIndex(char c){
72         return c - '0';
73     }
74     void init(){
75         memset(c[0], 0, sizeof(c[0]));
76         memset(val, -1, sizeof(val));
77         sz = 1;
78     }
79     void insert(BigInteger x, int v){
80         int u = 0;
81         int max_len_count = 0;
82         int firstNum = x.s.back();
83         char firstBuf[20];
84         sprintf(firstBuf, "%d", firstNum);
85         for(int j = 0; j < strlen(firstBuf);
86             j++){
87             int index = getIndex(firstBuf[j]);
88             if(!c[u][index]){
89                 memset(c[sz], 0, sizeof(c[
90                     sz]));
91                 val[sz] = v;
92                 c[u][index] = sz++;
93             }
94             u = c[u][index];
95             max_len_count++;
96         }
97         for(int i = x.s.size()-2; i >= 0; i
98             --){
99             char buf[20];
100             sprintf(buf, "%08d", x.s[i]);
101             for(int j = 0; j < strlen(buf)
102                 && max_len_count < 50; j++){
103                 int index = getIndex(buf[j]);
104                 if(!c[u][index]){
105                     memset(c[sz], 0, sizeof
106                         (c[sz]));
107                     val[sz] = v;
108                     c[u][index] = sz++;
109                 }
110                 u = c[u][index];
111                 max_len_count++;
112             }
113             if(max_len_count >= 50){
114                 break;
115             }
116         }
117     }
118 }

```

```

115 int find(const char* s){
116     int u = 0;
117     int n = strlen(s);
118     for(int i = 0; i < n; ++i)
119     {
120         int index = getIndex(s[i]);
121         if(!c[u][index]){
122             return -1;
123         }
124         u = c[u][index];
125     }
126     return val[u];
127 }
128 }

```

8.4 分數

```

1 class Rational
2 {
3     friend istream &operator>>(istream &,
4         Rational &);
5     friend ostream &operator<<(ostream &,
6         const Rational &);
7 public:
8     Rational() //constructor one
9     {
10         m_numerator = 0;
11         m_denominator = 1;
12     }
13     Rational(int a, int b) //constructor two
14     {
15         if (b < 0 || b == 0) //avoids negative
16             denominators. && prevents a 0
17             denominator
18         {
19             cout << "This Rational number can't be
20                 used.\n\n";
21             m_numerator = 0;
22             m_denominator = 0;
23         }
24         else
25         {
26             cout << "This Rational number can be
27                 used.\n\n";
28             m_numerator = a;
29             m_denominator = b;
30         }
31     }
32     Rational operator+(const Rational& a); //
33     Rational operator-(const Rational& a); //
34     Rational operator*(const Rational& a); //
35     Rational operator/(const Rational& a); //
36     bool operator==(const Rational& a); //相
37     void reduce(); //化簡
38 private:
39     int m_numerator;
40     int m_denominator;

```

```

35 };
36 istream &operator>>(istream &input, Rational
37     &test )
38 {
39     char temp;
40     input >> test.m_numerator;
41     input >> temp;
42     input >> test.m_denominator;
43     Rational final(test.m_numerator, test.
44         m_denominator); //final用來告訴使用者
45         這數字不符合!
46     if (test.m_denominator < 0 || test.
47         m_denominator == 0) //不符合(再輸入
48         一次)
49     {
50         while (test.m_denominator < 0 || test.
51             m_denominator == 0) //有可能輸入的
52             東西還是不符合,所以用迴圈
53     {
54         cout << "Enter another Rational number
55             (n/d): ";
56         input >> test.m_numerator;
57         input >> temp;
58         input >> test.m_denominator;
59         Rational final(test.m_numerator, test.
60             m_denominator); //final用來告訴使
61             用者這數字不符合!
62     }
63     return input;
64 }
65 else
66     return input;
67 }
68 ostream &operator<<(ostream &output, const
69     Rational &test )
70 {
71     output << test.m_numerator;
72     if(test.m_numerator == 0)
73         return output;
74     if (test.m_denominator == 1)
75         return output;
76     else
77     {
78         output << "/";
79         output << test.m_denominator;
80     }
81     return output;
82 }
83 Rational Rational::operator+(const Rational&
84     a)
85 {
86     Rational c;
87     c.m_denominator = this->m_denominator * a.
88         m_denominator; //通分(同乘)
89     c.m_numerator = (this->m_numerator * a.
90         m_denominator) + (a.m_numerator * this
91         ->m_denominator);
92     c.reduce();
93     return c;
94 }
95 Rational Rational::operator-(const Rational&
96     a)
97 {
98     Rational c;
99     c.m_denominator = this->m_denominator * a.
100         m_denominator;
101     c.m_numerator = (this->m_numerator * a.
102         m_denominator) - (a.m_numerator * this
103         ->m_denominator);
104     c.reduce();
105     return c;
106 }
107 Rational Rational::operator*(const Rational&
108     a)
109 {
110     Rational c;
111     c.m_denominator = this->m_denominator * a.
112         m_denominator;
113     c.m_numerator = (this->m_numerator * a.
114         m_numerator);
115     c.reduce();
116     return c;
117 }
118 Rational Rational::operator/(const Rational&
119     a)
120 {
121     Rational c;
122     c.m_denominator = this->m_denominator * a.
123         m_denominator;
124     c.m_numerator = (this->m_numerator * a.
125         m_denominator) / a.m_numerator;
126     c.reduce();
127     return c;
128 }

```

```

83 Rational c;
84 c.m_denominator = this->m_denominator * a.
85     m_denominator;
86 c.m_numerator = (this->m_numerator * a.
87     m_denominator) - (a.m_numerator * this
88     ->m_denominator);
89 c.reduce();
90 return c;
91 }
92 Rational Rational::operator*(const Rational&
93     a)
94 {
95     Rational c;
96     c.m_denominator = this->m_denominator * a.
97         m_denominator;
98     c.m_numerator = (this->m_numerator * a.
99         m_numerator);
100     c.reduce();
101     return c;
102 }
103 Rational Rational::operator/(const Rational&
104     a)
105 {
106     Rational c;
107     c.m_denominator = this->m_denominator * a.
108         m_denominator;
109     c.m_numerator = (this->m_numerator * a.
110         m_denominator) / a.m_numerator;
111     c.reduce();
112     return c;
113 }
114 bool Rational::operator==(const Rational& a)
115 {
116     if (m_numerator == a.m_numerator)
117     {
118         if (m_denominator == a.m_denominator)
119             return true;
120         else
121             return false;
122     }
123     else
124         return false;
125 }
126 void Rational::reduce()
127 {
128     int i;
129     int max;
130     if(m_numerator > m_denominator)
131         max = m_numerator;
132     else
133         max = m_denominator;
134     for (i = 2; i <= max; i++)
135     {
136         if (m_denominator % i == 0 && m_numerator
137             % i == 0)
138         {
139             m_denominator /= i;
140             m_numerator /= i;
141             i = 1;
142             max = m_denominator;
143             continue;
144         }
145     }
146 }

```

TO DO WRITING NOT THINKING

Contents

1 Basic	1	2.4 Knapsack Unbounded	1	4.4 Dijkstra	5	5.11 數字乘法組合	8
1.1 data range	1	2.5 LCIS	1	4.5 Edmonds_karp	5	5.12 數字加法組合	8
1.2 IO_fast	1	2.6 LCS	1	4.6 Floyd-warshall	5	5.13 羅馬數字	8
2 DP	1	2.7 LIC	1	4.7 Kruskal	6	5.14 質因數分解	8
2.1 KMP	1	2.8 LPS	2	4.8 Prim	6	6 Other	8
2.2 Knapsack Bounded	1	2.9 Max_subarray	2	4.9 Union_find	6	6.1 Weighted Job Scheduling	8
2.3 Knapsack sample	1	2.10 Money problem	2	5 Mathematics	6	6.2 數獨解法	9
		3 Geometry	2	5.1 Combination	6	7 String	9
		3.1 Line	2	5.2 Extended Euclidean	7	7.1 Sliding window	9
		3.2 Point	3	5.3 Hex to Dec	7	7.2 Split	9
		3.3 Polygon	3	5.4 Mod	7	8 data structure	9
		3.4 Triangle	4	5.5 Permutation	7	8.1 Bigint	9
		4 Graph	4	5.6 PI	7	8.2 MFlow	10
		4.1 Bellman-Ford	4	5.7 Prime table	7	8.3 Trie	10
		4.2 BFS-queue	5	5.8 primeBOOL	7	8.4 分數	11
		4.3 DFS-rec	5	5.9 二分逼近法	8		
				5.10 四則運算	8		