

# Contents

<b>1 Basic</b>	
1.1 Syntax	
1.2 Linux Command	
1.3 Substring	
1.4 BigInteger	
<b>2 Data Structure</b>	
2.1 Tree	
2.2 Disjoint Set	
2.3 Segment Tree	
2.4 Tree Policy	
2.5 KMP	
2.6 LCA	
<b>3 Divide and Conquer</b>	
3.1 MaximumSubArray	
3.2 Closet Set Pair	
<b>4 Dynamic Programming</b>	
4.1 LCS	
4.2 LIS	
4.3 Knapsack	
4.4 ChangeCoin	
4.5 String Edition	
4.6 Chain Matrix Mul	
<b>5 Search</b>	
5.1 Binary Search	
<b>6 Sequence</b>	
6.1 RSQ(Prefix Sum)	
6.2 RSQ(2DPrefix Sum)	
6.3 RSQ(Fenwick Tree)	
<b>7 Sorting</b>	
7.1 Counting Sort	
7.2 Topology Sort	
7.3 Topology Sort with DFS(check 有無環)	
<b>8 Graph</b>	
8.1 DFS I	
8.2 DFS II	
8.3 DFS Tree	
8.4 BFS	
8.5 AOE	
8.6 Dijkstra	
8.7 SPFA	
8.8 BellmanFord	
8.9 FloydWarshall	
8.10Kruskal	
8.11Articulation Point	
8.12Bipartite Matching	
8.13Bipartite Check1	
8.14CLE Directed MST	
8.15Dinic	
8.16MCMF	
<b>9 Number</b>	
9.1 Sieve	
9.2 Power	
9.3 Euler	
9.4 Factors	
9.5 Extend Euclidean	
9.6 Matrix	
9.7 GaussElimination	
<b>10 Geometry</b>	
10.1Geometry	
10.2Lines Intersection1	
10.3Lines Intersection2	
10.4Polygon Inside Or Outside	
10.5Convex Hull	

## 1 Basic

### 1.1 Syntax

```

1 // 加速cin, cout
2 #define IOS cin.tie(nullptr); cout.tie(nullptr);
   ios_base::sync_with_stdio(false);
3 int main(int argc, char const *argv[])
4 {
5     // String to Integer
6     char str[30] = {'-', '1', '2', '3', '4', '5', '\0'};
7     printf("%d\n", stoi(str));
8     // Integer to String
9     int x = 185;
10    char temp[30];
11    int base = 10;
12    itoa(x, temp, base);
13    printf("%s\n", temp);
14    // String to Double
15    char strd[30] = {'0', '.', '6', '0', '2', '9', '\0'};
16    printf("%Lf\n", stod(strd));
17    // Double to String
18    double y = 3.1415926;
19    string dstr = to_string(y);
20    cout << dstr << endl;
21    // String initialize
22    char null[30] = {'\0'};
23    char A[30];
24    strcpy(A, null);
25    // String Length
26    char strl[30] = {'H', 'E', 'L', 'L', 'O', '\0'};
27    printf("%d\n", strlen(strl));
28    // String Reverse
29    char a[] = {'a', 'b', 'c', 'd', 'e', 'f', '\0'};
30    strrev(a); reverse(a, a + 6);
31    string s = "abcdefg";
32    reverse(s.begin(), s.end());
33    /* Complexity
34     O(N) 大概 N 可以到 1億
35     O(N log N) 大概 N 可以到數百萬~千萬
36     O(N^1.5) 大概可以到數萬
37     O(N^2) 大概 5000~10000
38     */
39    return 0;
40 }

```

### 1.2 Linux Command

```

1 1. 創一個.in檔案
2 touch PA.in
3 2. 執行exe檔案
4 ./PA.exe > PA.in < PA.out
5 3. 打開.out或.in檔
6 cat PA.in
7 4. 比較答案
8 diff PA.out ans.txt

```

### 1.3 Substring

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 bool isSubstring(string a, string b){
4     bool is = 0;
5     if(b.find(a) != std::string::npos){
6         is = 1;
7     }
8     return is;
9 }
10 //check if string a is substring of b
11 int main(){
12     string a = "123", b = "12345";
13     // "123" 是不是 substring "12345"
14     if(isSubstring(a,b)) cout << "yes"<<endl;
15     else cout << "no"<<endl;
16     return 0;
17 }

```

## 1.4 BigInteger

```

1 import java.math.BigInteger;
2 import java.util.Scanner;
3 class Main {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         BigInteger n = input.nextBigInteger();
7         BigInteger m = input.nextBigInteger();
8         n.add(m); a.subtract(m); n.multiply(m); n.
            divide(m); n.mod(m);
9         n.pow(m.intValue()); n.gcd(m); n.negate(); n.
            abs();
10    }
11 }

```

## 2 Data Structure

### 2.1 Tree

```

1 struct node
2 {
3     int data;
4     node* left;
5     node* right;
6 };
7
8 void preOrder(node *root) {
9
10    cout << root->data << " ";
11    if(root->left != NULL) preOrder(root->left);
12    if(root->right != NULL) preOrder(root->right);
13 }
14
15 void postOrder(node *root) {
16
17    if(root->left != NULL) postOrder(root->left);
18    if(root->right != NULL) postOrder(root->right);
19    cout << root->data << " ";
20 }
21
22 void inOrder(node *root) {
23
24    if(root->left != NULL) inOrder(root->left);
25    cout << root->data << " ";
26    if(root->right != NULL) inOrder(root->right);
27 }

```

### 2.2 Disjoint Set

```

1 const int n = 6; // number of nodes
2 int p[n+10];
3 void init()
4 {
5     for(int i = 0; i < n; i++){
6         p[i] = -1;
7     }
8 }
9 int find(int x){
10    int root, trail, lead;
11    for (root = x; p[root] >= 0; root = p[root]);
12    for (trail = x; trail != root; trail = lead) {
13        lead = p[trail];
14        p[trail] = root;
15    }
16    return root;
17 }
18 void uni(int x, int y)
19 {
20     int xRoot = find(x), yRoot = find(y);
21     if(xRoot != yRoot){
22         if(p[xRoot] < p[yRoot]){
23             p[xRoot] += p[yRoot];

```

```

24         p[yRoot] = xRoot;
25     }
26     else{
27         p[yRoot] += p[xRoot];
28         p[xRoot] = yRoot;
29     }
30 }
31 }

```

### 2.3 Segment Tree

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int n = 8;
4 int B[n] = {18, 17, 13, 19, 15, 11, 20, 87};
5 typedef vector<int> vi;
6 vi A (B, B + 8);
7 vi ST;
8 void ST_Build(vi &ST, const vi &A, int vertex, int L,
    int R)
9 {
10    if(L == R) ST[vertex] = L;
11    else
12    {
13        int nL = vertex * 2, nR = vertex * 2 + 1;
14        ST_Build(ST, A, nL, L, L + (R - L) / 2);
15        ST_Build(ST, A, nR, L + (R - L) / 2 + 1, R);
16        int indexL = ST[nL], indexR = ST[nR];
17        int valueL = A[indexL], valueR = A[indexR];
18        ST[vertex] = valueL <= valueR ? indexL : indexR;
19    }
20 }
21
22 void ST_Creation(vi &ST, const vi &A)
23 {
24     int len = 4 * A.size();
25     ST.assign(len, 0);
26     ST_Build(ST, A, 1, 0, A.size()-1);
27 }
28 int query(vi &ST, const vi &A, int vertex, int L, int R
    , int qL, int qR)
29 {
30     int temp, mid = (L + R) / 2;
31     if(qL <= L && R <= qR) return ST[vertex];
32     if(qR <= mid)
33     { //all we want at the left child
34         return query(ST, A, vertex * 2, L, mid, qL, qR);
35     }
36     if(qL > mid)
37     { // all we want at the right child
38         return query(ST, A, vertex * 2 + 1, mid + 1, R, qL,
            qR);
39     }
40     return A[query(ST, A, vertex * 2, L, mid, qL, qR)] <=
        A[query(ST, A, vertex * 2 + 1, mid + 1, R, qL,
            qR)]
41         ? query(ST, A, vertex * 2, L, mid, qL, qR) :
            query(ST, A, vertex * 2 + 1, mid + 1, R, qL,
                qR);
42 }
43
44 void update(vi &ST, vi &A, int x, int L, int R, int p,
    int v)
45 {
46     // p is the index where you want to update
47     // v is the value will be update in A[p];
48     int mid = L + (R - L) / 2;
49     if(L == R) A[ST[x]] = v;
50     else
51     {
52         if(p <= mid) update(ST, A, x*2, L, mid, p, v);
53         else update(ST, A, x*2+1, mid+1, R, p, v);
54         ST[x] = (A[ST[x*2]] <= A[ST[x*2+1]]) ? ST[x*2] : ST
            [x*2+1];
55     }
56 }

```

```

57 int main(int argc, char const *argv[])
58 {
59     ST_Creation(ST, A);
60     printf("%d\n", query(ST, A, 1, 0, n-1, 3, 7));
61     // query return the index
62     printf("%d\n", A[query(ST, A, 1, 0, n-1, 3, 7)]);
63     update(ST, A, 1, 0, n-1, 5, 18);
64     // query and update first to fifth parameter dont
        change
65     // ST, A, 1, 0, n-1
66     // last two would be
67     // query: the range(array index) you want to query
68     // update: first is the index you want to update,
        second is the value will be
69     return 0;
70 }

```

## 2.4 Tree Policy

```

1 #include <bits/stdc++.h>
2 #include <ext/pb_ds/assoc_container.hpp> // Common file
3 #include <ext/pb_ds/tree_policy.hpp>
4 #include <functional> // for less
5 using namespace std;
6 using namespace __gnu_pbds;
7 typedef tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> new_data_set;
8 new_data_set t;
9 int main()
10 {
11     t.insert(5);
12     t.insert(6);
13     t.insert(3);
14     t.insert(1);
15     // the smallest is (0), biggest is (n-1), kth small
        is (k-1)
16     int num = *t.find_by_order(0);
17     printf("%d\n", num); // print 1
18     num = *t.find_by_order(t.size()-1);
19     printf("%d\n", num); // print 6
20     // find the index
21     int index = t.order_of_key(6);
22     printf("%d\n", index); // print 3
23     // check if there exist x
24     int x = 5;
25     int check = t.erase(x);
26     if(check == 0) printf("t not contain 5\n");
27     else if(check == 1) printf("t contain 5\n");
28     //tree policy like set
29     t.insert(5); t.insert(5);
30     // get the size of t
31     printf("%d\n", t.size()); // print 4
32     return 0;
33 }

```

## 2.5 KMP

```

1 int kmp(string text, string pattern){
2     if(pattern.size()==0) return -1;
3     int patLen=pattern.size();
4     int textLen=text.size();
5     int LPS[patLen]={0};
6     int i=1,j=0;
7
8     while(i<patLen){
9         if(pattern[i]==pattern[j]){
10             LPS[i++]=++j;
11         }
12         else{
13             if(j) j=LPS[j-1];
14             else LPS[i++]=0;
15         }
16     }

```

```

17     i=j=0;
18     while(i<textLen){
19         if(pattern[j]==text[i]){
20             i++;j++;
21         }
22         if(j==patLen) return i-j;
23         else{
24             if(i<textLen && pattern[j]!=text[i]){
25                 if(j) j=LPS[j-1];
26                 else i++;
27             }
28         }
29     }
30     return -1;
31 }
32
33 int main(){
34     string text,pattern;
35     getline(cin, text);
36     getline(cin, pattern);
37     int index=kmp(text,pattern);
38     if(index>0){
39         cout<<"\nPattern found at : "<<index<<"\n";
40     }
41     else{
42         cout<<"\nPattern not found!\n";
43     }
44 }

```

## 2.6 LCA

```

1 #define max 100
2 #define lg_max 7
3 vector<int> graph[max];
4 int parent[max][lg_max], level[max], lg[max];
5 int n;
6 void log()
7 {
8     for (int i = 2; i < max; i++)
9         lg[i] = lg[i / 2] + 1;
10 }
11 void dfs(int u, int p)
12 {
13     for (auto v : graph[u]){
14         if (v != p){
15             level[v] = level[u] + 1;
16             parent[v][0] = u;
17             dfs(v, u);
18         }
19     }
20 }
21 void build()
22 {
23     for (int j = 1; j <= lg[n]; j++)
24     {
25         for (int i = 1; i <= n; i++)
26         {
27             parent[i][j] = parent[parent[i][j-1]][j-1];
28         }
29     }
30 }
31 int lca(int u, int v)
32 {
33     if (level[u] < level[v]) return lca(v, u);
34     for (int i = lg[n]; i >= 0; i--){
35         if (level[u] - (1 << i) >= level[v]){
36             u = parent[u][i];
37         }
38     }
39     if (u == v) return u;
40     for (int i = lg[n]; i >= 0; i--){
41         if (parent[u][i] != parent[v][i]){
42             u = parent[u][i];
43             v = parent[v][i];
44         }

```

```

45     }
46     return parent[u][0];
47 }
48 int main()
49 {
50     log();
51     int x, y;
52     scanf("%d", &n);
53     for (int i = 0; i < n - 1; i++){
54         scanf("%d%d", &x, &y);
55         graph[x].push_back(y);
56         graph[y].push_back(x);
57     }
58     dfs(1, 1);
59     build();
60     scanf("%d%d", &x, &y);
61     printf("%d\n", lca(x, y));
62 }

```

## 3 Divide and Conquer

### 3.1 MaximumSubArray

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int n = 16;
4 int arr[n] = {13, -3, -25, 20, -3, -16, -23,
5              18, 20, -7, 12, -5, -22, 15, -4, 7};
6
7 int findMaxCrossing(int left, int mid, int right){
8     int maxl = 0x80000000;
9     int sum = 0;
10    for(int i = mid; i >= left; i--){
11        sum += arr[i];
12        if(sum > maxl) maxl = sum;
13    }
14    int maxr = 0x80000000;
15    sum = 0;
16    for(int i = mid + 1; i <= right; i++){
17        sum += arr[i];
18        if(sum > maxr) maxr = sum;
19    }
20
21    return (maxl + maxr);
22 }
23
24 int findMaxSub(int left, int right)
25 {
26     if(left == right){
27         return arr[left];
28     }
29     else{
30         int mid = left + (right - left) / 2;
31         int maxl = findMaxSub(left, mid);
32         int maxr = findMaxSub(mid + 1, right);
33         int res = max(maxl, maxr);
34         res = max(res, findMaxCrossing(left, mid, right));
35         return res;
36     }
37 }
38
39
40 int main(int argc, char const *argv[])
41 {
42     printf("%d\n", findMaxSub(0, n-1));
43     return 0;
44 }

```

### 3.2 Closet Set Pair

```

1 struct point2D
2 {

```

```

3     double x, y;
4     bool operator< (point2D const other) const{
5         return x < other.x;
6     }
7     bool operator> (point2D const other) const{
8         return y > other.y;
9     }
10 };
11 point2D p[10000+10];
12
13 double dis(point2D p1, point2D p2)
14 {
15     return sqrt(((p1.x - p2.x) * (p1.x - p2.x)) + ((p1.y
16         - p2.y) * (p1.y - p2.y)));
17 }
18 double bruteforce(int start, int n){
19     double mind = 2e9;
20     for(int i = start; i < n - 1; i++){
21         for(int j = i + 1; j < n; j++){
22             mind = min(mind, dis(p[i], p[j]));
23         }
24     }
25     return mind;
26 }
27 double findcp(int left, int right, int n)
28 {
29     if(n <= 3){
30         return bruteforce(left, n);
31     }
32     double mind;
33     int mid = left + (right - left) / 2;
34     double cl = findcp(left, mid, mid - left + 1);
35     double cr = findcp(mid + 1, right, right - mid);
36     mind = min(cl, cr);
37     vector<point2D> v;
38     for(int i = left; i <= right; i++){
39         if(p[i].x <= p[mid].x + mind && p[i].x >= p[mid].x
40             - mind)
41             v.push_back(p[i]);
42     }
43     sort(v.begin(), v.end(), greater<point2D>());
44
45     for(vector<point2D>::iterator it = v.begin(); it != v
46         .end()-1; it++){
47         for(vector<point2D>::iterator jt = it + 1; jt != v.
48             end(); jt++){
49             mind = min(mind, dis(*it, *jt));
50         }
51     }
52     return mind;
53 }
54
55 int main(int argc, char const *argv[])
56 {
57     int n;
58     double min;
59     while(cin >> n && n)
60     {
61         for(int i = 0; i < n; i++){
62             cin >> p[i].x >> p[i].y;
63         }
64         sort(p, p + n);
65         min = findcp(0, n-1, n);
66         if(min < 10000) printf("%.4lf\n", min);
67         else printf("INFINITY\n");
68     }
69     return 0;
70 }

```

## 4 Dynamic Programming

### 4.1 LCS

```

1 const int maxn = 10000; // maxn is maximum length of
   arrp and arrq
2 int arrp[maxn], arrq[maxn];
3 int dp[maxn+5][maxn+5];
4 int p, q; // p is the length of arrp, q is the length
   of arrq
5 void LCS()
6 {
7     memset(dp, 0, sizeof(dp));
8
9     for(int i = 1; i <= p; i++){
10         for(int j = 1; j <= q; j++){
11             if(arrp[i] == arrq[j]){
12                 dp[i][j] = 1 + dp[i-1][j-1];
13             }
14             else{
15                 dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
16             }
17         }
18     }
19     // dp[p][q] is the answer
20 }

```

## 4.2 LIS

```

1 int LIS(vector<int>& s)
2 {
3     if (s.size() == 0) return 0;
4
5     vector<int> v;
6     v.push_back(s[0]);
7
8     for (int i = 1; i < s.size(); ++i)
9     {
10         int n = s[i];
11
12         if (n > v.back())
13             v.push_back(n);
14         else
15             *lower_bound(v.begin(), v.end(), n) = n;
16     }
17
18     return v.size();
19 }

```

## 4.3 Knapsack

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 int dp[1005][1005];
4 int track[1005][1005];
5 struct Item{
6     int value, weight;
7 };
8 vector <Item> item;
9 int main(){
10     int n, W, t, t1, t2, temp_w, temp_v;
11     vector <int> ans_item;
12     cin >> n;
13     while(n--){
14         cin >> W >> t; //W = total weight, t = # of item
15         item.clear(); ans_item.clear();
16         item.push_back(Item{0, 0});
17         for(int i = 0 ; i<t ; i++){
18             cin >> t1 >> t2;
19             item.push_back(Item{t1, t2});
20         }
21         memset(track, 0, sizeof(track));
22         for(int i = 0 ; i<=t ; i++){//row - i
23             temp_w = item[i].weight;
24             temp_v = item[i].value;
25             for(int w = 0 ; w<=W ; w++){
26                 if(i == 0 || w == 0){

```

```

27                 dp[w][i] = 0;
28                 continue;
29             }
30             if(temp_w <= w){
31                 //dp[w][i] = max(dp[w][i-1], dp[w - temp_w][i
32                     -1] + temp_v);
33                 if((dp[w - temp_w][i-1] + temp_v) > dp[w][i
34                     -1]){
35                     dp[w][i] = dp[w - temp_w][i-1] + temp_v;
36                     track[w][i] = true; //true=有放
37                 }
38                 else{
39                     dp[w][i] = dp[w][i-1];
40                 }
41             }
42             else{
43                 dp[w][i] = dp[w][i-1];
44             }
45             cout << dp[W][t] << endl;
46             //backtracking
47             int ii = t-1, ww = W;
48             while(ii != 0){
49                 if(track[ww][ii]){
50                     ww -= item[ii].weight;
51                     ans_item.push_back(ii);
52                 }
53                 ii -= 1;
54             }
55         }
56     }

```

## 4.4 ChangeCoin

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 long long dp[30005][5];
4 int cents[5] = {1, 5, 10, 25, 50};
5 int main(){
6     long long N, ans;
7     while(cin >> N){
8         for(int k = 0 ; k<5 ; k++){
9             for(int n = 0 ; n<=N ; n++){
10                 if(k == 0 || n == 0){
11                     dp[n][k] = 1;
12                     continue;
13                 }
14                 if(n < cents[k]){
15                     dp[n][k] = dp[n][k-1];
16                 }
17                 else dp[n][k] = dp[n][k-1] + dp[n - cents[k]][k
18                     ];
19             }
20             ans = dp[N][4];
21             printf("There are %lld ways to produce %lld cents
22                 change.\n", ans, N);
23         }
24     }

```

## 4.5 String Edition

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int maxn = 90;
4 char s1[maxn], s2[maxn];
5 int dp[maxn][maxn];
6 struct Coord
7 {
8     int x, y;
9 };

```

```

10 Coor backtracking[maxn][maxn];
11 vector<Coor>ans;
12 int main(int argc, char const *argv[])
13 {
14     bool begining = true;
15     while(gets(s1)){
16         gets(s2);
17         if(begining) begining = false;
18         else printf("\n");
19         memset(dp, 0, sizeof(dp));
20         memset(backtracking, 0, sizeof(backtracking));
21         ans.clear();
22         for(int i = 1; i <= strlen(s2); i++) {
23             dp[0][i] = dp[0][i-1] + 1;
24             backtracking[0][i].x = 0;
25             backtracking[0][i].y = i-1;
26         }
27         for(int i = 1; i <= strlen(s1); i++) {
28             dp[i][0] = dp[i-1][0] + 1;
29             backtracking[i][0].x = i-1;
30             backtracking[i][0].y = 0;
31         }
32         for(int i = 1; i <= strlen(s1); i++){
33             for(int j = 1; j <= strlen(s2); j++){
34                 if(s1[i-1] == s2[j-1]) {
35                     dp[i][j] = dp[i-1][j-1];
36                     backtracking[i][j] = Coor{i-1, j-1};
37                 }
38                 else{
39                     dp[i][j] = min(dp[i][j-1], min(dp[i-1][j-1],
40                     dp[i-1][j]));
41                     if(dp[i][j] == dp[i][j-1]){
42                         backtracking[i][j] = Coor{i, j-1};
43                     }
44                     else if(dp[i][j] == dp[i-1][j-1]){
45                         backtracking[i][j] = Coor{i-1, j-1};
46                     }
47                     else if(dp[i][j] == dp[i-1][j]){
48                         backtracking[i][j] = Coor{i-1, j};
49                     }
50                     dp[i][j]++;
51                 }
52             }
53         }
54         printf("%d\n", dp[strlen(s1)][strlen(s2)]);
55         int curi = strlen(s1), curj = strlen(s2);
56         ans.push_back(Coor{curi, curj});
57         while(curj != 0 || curi != 0){
58             int tempi = curi, tempj = curj;
59             curi = backtracking[tempi][tempj].x; curj =
60             backtracking[tempi][tempj].y;
61             ans.push_back(Coor{curi, curj});
62         }
63         int offset = 0, cnt = 1;
64         for(int i = ans.size()-2; i >= 0; i--){
65             if(dp[ans[i].x][ans[i].y] != dp[ans[i+1].x][ans[i
66             +1].y]){
67                 if((ans[i].x - ans[i+1].x) == 1 && (ans[i].y -
68                 ans[i+1].y) == 1){
69                     printf("%d Replace %d,%c\n", cnt++, ans[i].x
70                     + offset, s2[ans[i].y-1]);
71                 }
72                 else if((ans[i].x - ans[i+1].x) == 1 && (ans[i
73                 ].y - ans[i+1].y) == 0){
74                     printf("%d Delete %d\n", cnt++, ans[i].x+
75                     offset);
76                     offset--;
77                 }
78                 else if((ans[i].x - ans[i+1].x) == 0 && (ans[i
79                 ].y - ans[i+1].y) == 1){
80                     printf("%d Insert %d,%c\n", cnt++, ans[i].x+
81                     offset+1, s2[ans[i].y-1]);
82                     offset++;
83                 }
84             }
85         }
86     }
87 }

```

```

78     }
79 }
80 return 0;
81 }

```

String Edition

## 4.6 Chain Matrix Mul

```

1 //intut matrix的矩陣大小, output最少需做幾次乘法
2 int M[1005][1005];
3 int P[1005][1005];
4 vector<int> d;
5 int do_dp(int i, int j){
6     if(M[i][j] > 0) return M[i][j];
7     if(i == j) return 0;
8     int minx = 0xFFFFFFFF;
9     for(int k = i; k < j; k++){
10         if((do_dp(i, k) + do_dp(k+1, j) + d[i-1]*d[k]*d[j])
11             < minx){
12             minx = do_dp(i, k) + do_dp(k+1, j) + d[i-1]*d[k]*
13             d[j];
14             P[i][j] = k;
15         }
16         //如果不用紀錄k是誰
17         //minx = min(minx, do_dp(i, k) + do_dp(k+1, j) + d[
18         //i-1]*d[k]*d[j]);
19     }
20     return M[i][j] = minx;
21 }
22 int main(){
23     int n, temp, s, ans;
24     cin >> n;
25     stringstream s1;
26     string str;
27     cin.ignore();
28     while(n--){
29         getline(cin, str);
30         s1.clear();
31         s1.str("");
32         s1 << str;
33         d.clear();
34         while(s1 >> temp){
35             d.push_back(temp);
36         }
37         s = d.size() - 1;
38         memset(M, 0, sizeof(M));
39         memset(P, 0, sizeof(P));
40         ans = do_dp(1, s);
41         printf("%d\n", ans);
42     }
43 }

```

## 5 Search

### 5.1 Binary Search

```

1 int L = 0; // Left boundary
2 int R = ans; // right boundary
3 // check using L = 3, R = 4, ans = 4
4 while(L < R){
5     int M = L + (R - L + 1) / 2; // Left + half distance
6     if(ok(M)) L = M; // ok() method is to find
7     // whether the M can qualify the demand
8     else R = M - 1;
9 }
10 while(L < R){
11     int M = L + (R - L) / 2; // Left + half distance
12     if(ok(M)) R = M; // ok() method is to find
13     // whether the M can qualify the demand
14     else L = M + 1;
15 }

```



## 6 Sequence

### 6.1 RSQ(Prefix Sum)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int maxn = 10;
4 int arr[maxn] = {5, -2, 3, 10, -7, 1, -4, 8, -9};
5 int query[maxn];
6 void init()
7 {
8     // every query is the sum of all previos element,
9     // include it self
10    // example query[3] = arr[0] + arr[1] + arr[2] + arr
11    [3]
12    query[0] = arr[0];
13    for(int i = 1; i < maxn; i++){
14        query[i] = arr[i];
15        query[i] += query[i-1];
16    }
17 }
18 int RangeSumQuery(int s, int e)
19 {
20     //Prefix Sum Algorithm
21     if(s >= 1) return query[e] - query[s-1];
22     else return query[e];
23 }
24 int main(int argc, char const *argv[])
25 {
26     init();
27     int start = 2, end = 5;
28     printf("RangeSumQuery(%d, %d): %d\n", start, end,
29         RangeSumQuery(start, end));
30     return 0;
31 }

```

### 6.2 RSQ(2DPrefix Sum)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int arr[110][110];
4 int query[110][110];
5 int n;
6
7 int main(int argc, char const *argv[])
8 {
9     while(cin >> n){
10        // input
11        for(int i = 0; i < n; i++){
12            for(int j = 0; j < n; j++){
13                cin >> arr[i][j];
14            }
15        }
16        // bulid prefix query
17        for(int i = 0; i < n; i++){
18            for(int j = 0; j < n; j++){
19                query[i][j] = arr[i][j];
20                if(i - 1 >= 0) query[i][j] += query[i-1][j];
21                if(j - 1 >= 0) query[i][j] += query[i][j-1];
22                if(i - 1 >= 0 && j - 1 >= 0) query[i][j] -=
23                    query[i-1][j-1];
24            }
25        }
26        int temp;
27        int maximum = 0x80000000;
28        // find the maximum sum in any range
29        for(int i = 0; i < n; i++){
30            for(int j = 0; j < n; j++){
31                for(int k = i; k < n; k++){
32                    for(int t = j; t < n; t++){
33                        temp = query[k][t];
34                        if(i - 1 >= 0) temp -= query[i-1][t];
35                        if(j - 1 >= 0) temp -= query[k][j-1];

```

```

36                        if(i - 1 >= 0 && j - 1 >= 0) temp += query[
37                            i-1][j-1];
38                        if(maximum < temp) maximum = temp;
39                    }
40                }
41            }
42            printf("%d\n", maximum);
43        }
44    }
45    return 0;
46 }

```

### 6.3 RSQ(Fenwick Tree)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int maxn = 10;
4 int arr[maxn] = {5, -2, 3, 10, -7, 1, -4, 8, -9};
5 int FenwickTree[maxn];
6 int ANDlowbit(int src)
7 {
8     // src & -src will get the lowbit
9     // example: 6 & -6 = 0110 & 1010 = 0010 = 2
10    return src & -src;
11 }
12 void init()
13 {
14     memset(FenwickTree, 0, sizeof(FenwickTree));
15     // Notice that we start in 1
16     for(int i = 1; i <= maxn; i++){
17         int index = i;
18         FenwickTree[i] += arr[i-1];
19         int temp = arr[i-1];
20         while(index + ANDlowbit(index) <= maxn){
21             index += ANDlowbit(index);
22             FenwickTree[index] += temp;
23         }
24     }
25 }
26 void Modify(int src, int val)
27 {
28     // Modify arr[src] to val
29     int gap = val - arr[src];
30     arr[src] = val;
31     int index = src + 1;
32     FenwickTree[index] += gap;
33     while(index + ANDlowbit(index) <= maxn){
34         index += ANDlowbit(index);
35         FenwickTree[index] += gap;
36     }
37 }
38 int SequenceQuery(int src)
39 {
40     //src is the index of the array which we want to know
41     // the Sequence Query
42     int res = FenwickTree[src];
43     int index = src;
44     while(index - ANDlowbit(index) > 0){
45         index -= ANDlowbit(index);
46         res += FenwickTree[index];
47     }
48     return res;
49 }
50 int RangeSumQuery(int s, int e)
51 {
52     return SequenceQuery(e) - SequenceQuery(s - 1);
53 }
54 int main(int argc, char const *argv[])
55 {
56     init();
57     int start = 2, end = 5;
58     // for Fenwick index is 3, 6 for array index is 2, 5

```

```

59   printf("RangeSumQuery(%d, %d): %d\n", start, end,
        RangeSumQuery(start + 1, end + 1));
60   Modify(2, 5);
61   // Modify arr[2] from 3 to 5
62   printf("RangeSumQuery(%d, %d): %d\n", start, end,
        RangeSumQuery(start + 1, end + 1));
63   return 0;
64 }

```

## 7 Sorting

### 7.1 Counting Sort

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const int maxn = 50;
4  const int maxDigit = 1050;
5  int unsorted[maxn] = {0, 3, 7, 6, 5}, sorted[maxn], aux
    [maxDigit];
6  // aux size is depends on the max digit in sorting
7  int main(int argc, char const *argv[])
8  {
9      int n = 4;
10     // array index start with 1
11     memset(aux, 0, sizeof(aux));
12     for(int i = 1; i <= n; i++){
13         aux[unsorted[i]]++;
14     }
15     for(int i = 1; i < maxDigit; i++){
16         aux[i] += aux[i-1];
17     }
18     for(int i = n; i > 0; i--){
19         sorted[aux[unsorted[i]]] = unsorted[i];
20         aux[unsorted[i]]--;
21     }
22     for(int i = 1; i <= n; i++){
23         printf("%d ", sorted[i]);
24     }
25     return 0;
26 }

```

### 7.2 Topology Sort

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const int maxn = 100;
4  vector<int> ans;
5  vector<int> adj[maxn];
6  int refs[maxn];
7  int n = 5;
8
9  // refs 紀錄這個點被幾個邊連到
10 void TopologyOrder()
11 {
12     for(int i = 0; i < n; i++){
13         int s = 0;
14         while(s < n && refs[s] != 0) {
15             s++;
16         }
17         if(s == n) break;
18         refs[s] = -1;
19         ans.push_back(s);
20         for(auto j : adj[s]){
21             refs[j]--;
22         }
23     }
24 }
25 int main(int argc, char const *argv[])
26 {
27     memset(refs, 0, sizeof(refs));
28     ans.clear();
29     // adj[from].push_back(to); refs[to]++;

```

```

30     adj[4].push_back(1); refs[1]++;
31     adj[1].push_back(3); refs[3]++;
32     adj[1].push_back(0); refs[0]++;
33     adj[2].push_back(0); refs[0]++;
34     adj[3].push_back(0); refs[0]++;
35     TopologyOrder();
36     for(int i = 0; i < ans.size(); i++){
37         if(i == ans.size()-1) printf("%d\n", ans[i]);
38         else printf("%d ", ans[i]);
39     }
40     return 0;
41 }

```

### 7.3 Topology Sort with DFS(有無環)

```

1  const int maxn = 5000+50;
2  vector<int> adj[maxn];
3  stack<int> ans;
4  int state[maxn];
5  bool head[maxn];
6  bool valid;
7  int n, m;
8
9  void dfs(int src)
10 {
11     state[src] = 1;
12
13     for (auto next : adj[src])
14         if (!state[next]) dfs(next);
15         else if (state[next] == 1){
16             // 有環
17             valid = false;
18             return;
19         }
20
21     state[src] = 2;
22     ans.push(src);
23 }
24
25 void topology_sort()
26 {
27     for (int i = 0; i < n; i++){
28         // 從 (0 ~ n-1) 找一個頭沒有被任何人連到的開始
29         做dfs
30         if (valid && head[i]) dfs(i);
31     }
32
33     if (!valid)
34     {
35         cout << "Cycle!" << endl;
36         return;
37     }
38
39     while (!ans.empty())
40     {
41         cout << ans.top() << endl;
42         ans.pop();
43     }
44 }
45
46 int main()
47 {
48     cin >> n >> m;
49
50     memset(head, true, sizeof(head));
51     // make adjacent list
52     for (int i = 0; i < m; i++)
53     {
54         int a, b;
55         cin >> a >> b;
56
57         head[b] = false;
58         //
59         adj[a].push_back(b);

```



```

60 }
61
62 memset(state, 0, sizeof(state));
63 valid = true;
64 //如果 valid = false代表有還
65 topology_sort();
66
67 return 0;
68 }

```

## 8 Graph

### 8.1 DFS I

```

1 //implement by adjacent list
2 //functional dfs
3 void dfs(int now, int fa, int layer){
4     for (auto j : adj[now])
5         if(j != fa ) dfs(j, now, layer + 1);
6 }
7 //stack dfs
8 stack<int> st;
9 bool vis[maxn];
10 memset(vis, false, sizeof(vis));
11 int src;
12 st.push(src);
13 while(!st.empty())
14 {
15     int now = st.top(); st.pop();
16     vis[now] = true;
17     for(auto i : adj[now])
18         if(!vis[i]) st.push(i);
19 }

```

### 8.2 DFS II

```

1 const int maxn = 10;
2 struct Node{
3     int d, f, color;
4     // d: discover time, f: finish time, color: 0 ==
5         // white, 1 == gray, 2 == black
6 };
7 vector<int> adj[maxn];
8 Node node[maxn];
9 int times;
10 void DFS(int src)
11 {
12     node[src].d = times++;
13     node[src].color = 1;
14     for(auto i : adj[src]){
15         if(node[i].color == 0) DFS(i);
16     }
17     node[src].color = 2;
18     node[src].f = times++;
19 }
20 void DFS_Start(int n, int sp)
21 {
22     for(int i = 0; i < n; i++){
23         node[i].color = 0;
24     }
25     times = 0;
26     DFS(sp);
27 }
28 int main(int argc, char const *argv[])
29 {
30     int n, m, x, y;
31     cin >> n >> m;
32     for(int i = 0; i < n; i++) adj[i].clear();
33     for(int i = 0; i < m; i++){
34         cin >> x >> y;
35         adj[x].push_back(y);

```

```

36 }
37 DFS_Start(6, 0);
38 for(int i = 0; i < n; i++){
39     printf("%d: d: %d f: %d color: %d\n", i, node[i].d,
40         node[i].f, node[i].color);
41 }
42 return 0;
43 }

```

### 8.3 DFS Tree

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int maxn = 100000+5;
4 vector<int> adj[maxn];
5 int blocks[maxn];
6 void dfs(int cur, int fa)
7 {
8     blocks[cur] = 1;
9     for(auto i : adj[cur]){
10         if(i != fa) {
11             dfs(i, cur);
12             blocks[cur] += blocks[i];
13         }
14     }
15 }
16 int main(int argc, char const *argv[])
17 {
18     int n, x;
19     while(cin >> n){
20         for(int i = 0; i <= n; i++) adj[i].clear();
21         memset(blocks, 0, sizeof(blocks));
22         // blocks 為包含自己，自己的子節點數量
23         // 建一個無環的圖
24         for(int i = 1; i < n; i++){
25             cin >> x;
26             adj[i].push_back(x);
27             adj[x].push_back(i);
28         }
29         // 從0當Root做dfs
30         dfs(0, -1);
31         for(int i = 0; i < n; i++) {
32             int ans = 0;
33             for(auto j : adj[i]){
34                 if(blocks[i] > blocks[j]) ans = max(ans, blocks
35                     [j]);
36             }
37             ans = max(ans, n - blocks[i]);
38             printf("%d\n", ans);
39         }
40     }
41     return 0;
42 }

```

### 8.4 BFS

```

1 queue<int> st;
2 bool vis[maxn];
3 memset(vis, false, sizeof(vis));
4 int src;
5 st.push(src);
6 while(!st.empty())
7 {
8     int now = st.front(); st.pop();
9     vis[now] = true;
10     for(auto i : adj[now])
11         if(!vis[i]) st.push(i);
12 }

```

### 8.5 AOE

```

1 struct AOE {
2     // zero base
3     const int INF = 1e9;
4     struct Edge {
5         int at;
6         int cost;
7     };
8
9     struct Vertex {
10        int early;
11        int late;
12        vector<Edge> from;
13        vector<Edge> to;
14    };
15
16    int n;
17    vector<Vertex> vertices;
18
19    void init(int _n) {
20        n = _n;
21        vertices.clear();
22        vertices.resize(_n);
23        for (int i = 0; i < n; i++) {
24            vertices[i].early = -1;
25            vertices[i].late = INF;
26        }
27    }
28
29    void addEdge(int from, int to, int cost) {
30        // zero base
31        vertices[from].to.push_back({to, cost});
32        vertices[to].from.push_back({from, cost});
33    }
34
35    void dfsEarly(int now) {
36        for (auto e : vertices[now].to) {
37            if (vertices[e.at].early < vertices[now].
38                early + e.cost) {
39                vertices[e.at].early = vertices[now].
40                    early + e.cost;
41                dfsEarly(e.at);
42            }
43        }
44
45    void dfsLate(int now) {
46        for (auto e : vertices[now].from) {
47            if (vertices[e.at].late > vertices[now].
48                late - e.cost) {
49                vertices[e.at].late = vertices[now].
50                    late - e.cost;
51                dfsLate(e.at);
52            }
53        }
54
55    // may be slow?
56    void printCritical(int now, vector<int> path) {
57        if (vertices[now].to.size() == 0) {
58            // critical path found
59            for (auto i : path) {
60                cout << i << ' ';
61            }
62            cout << '\n';
63            return;
64        }
65        for (auto e : vertices[now].to) {
66            if (vertices[e.at].early == vertices[e.at].
67                late) {
68                vector<int> tmp = path;
69                tmp.push_back(e.at);
70                printCritical(e.at, tmp);
71            }
72        }
73
74    int run() {

```

```

75        for (int i = 0; i < n; i++) {
76            if (vertices[i].from.size() == 0) {
77                vertices[i].early = 0;
78                dfsEarly(i);
79            }
80
81            int ans = 0;
82            for (int i = 0; i < n; i++) {
83                if (vertices[i].to.size() == 0) {
84                    vertices[i].late = vertices[i].early;
85                    ans = max(ans, vertices[i].late);
86                    dfsLate(i);
87                }
88            }
89
90            for (int i = 0; i < n; i++) {
91                cout << "i = " << i << " early = " <<
92                    vertices[i].early << " late = " <<
93                    vertices[i].late << "\n";
94            }
95
96            for (int i = 0; i < n; i++) {
97                if (vertices[i].from.size() == 0) {
98                    vector<int> path;
99                    path.push_back(i);
100                    printCritical(i, path);
101                }
102            }
103
104            return ans;
105        }
106    };
107
108    int main() {
109        AOE aoetest;
110        int n, m;
111        cin >> n >> m;
112        aoetest.init(n);
113
114        int a, b, w;
115        for (int i = 0; i < m; i++) {
116            cin >> a >> b >> w;
117            aoetest.addEdge(a, b, w);
118        }
119
120        int res = aoetest.run();
121        cout << "res = " << res << endl;
122    }

```

## 8.6 Dijkstra

```

1 #define MP make_pair
2 #define PII pair<int, int>
3 #define maxn 50000 + 5
4
5 int dis[maxn]; // 預設都是 INF
6 vector<PII> adj[maxn]; // (連到的點, 邊的距離)
7
8 void dijk(int cur) // dijk(起點)
9 {
10     int d;
11     priority_queue<PII, vector<PII>, greater<PII>> q; //
12     // 放 (距離, 點編號), 每次會拿距離最小的點出來
13     q.push(MP(0, cur));
14
15     while (!q.empty())
16     {
17         tie(d, cur) = q.top(); q.pop();
18         if (dis[cur] != 1e9) continue; // 如果之前就拜訪
19         // 過, 無視
20
21         dis[cur] = d;
22
23         for (auto i : adj[cur]){

```

```

22     if (dis[i.first] == 1e9) q.push(MP(d + i.second,
23         i.first));
24     }
25 }
26 }
27
28 void init(void)
29 {
30     fill(dis, dis + maxn, 1e9);
31
32     for (int i = 0; i < maxn; i++){
33         adj[i].clear();
34     }
35 }

```

## 8.7 SPFA

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define INF 0x3f3f3f3f
5 const int maxn = 10000+5;
6
7 int n, m;
8 int dist[maxn], out[maxn];
9 bool inq[maxn];
10 //dist = distance, inq = inqit, out
11 vector<pair<int, int>> adj[maxn];
12
13 void init()
14 {
15     memset(dist, INF, sizeof(dist));
16     memset(inq, 0, sizeof(inq));
17     memset(out, 0, sizeof(out));
18     for(int i = 0; i <= n; i++){
19         adj[i].clear();
20     }
21 }
22
23 bool spfa(int sp, int n)
24 {
25     queue<int> q;
26     q.push(sp);
27     dist[sp] = 0; inq[sp] = true;
28
29     while(!q.empty())
30     {
31         int u = q.front(); q.pop();
32         inq[u] = false; // pop point
33         out[u]++;
34         if(out[u] > n) return false; // negative cycle
35             occurs
36
37         for(int j = 0; j < adj[u].size(); j++){
38             int v = adj[u][j].first; // first is point,
39                 second is weight
40             if(dist[v] > dist[u] + adj[u][j].second){
41                 dist[v] = dist[u] + adj[u][j].second;
42                 if(inq[v]) continue;
43
44                 inq[v] = true; //push point
45                 q.push(v);
46             }
47         }
48     }
49     return true;
50 }
51
52 int main(int argc, char const *argv[])
53 {
54     // n nodes and m edges
55     scanf("%d%d", &n, &m);
56     init();
57     // make adjacent list
58     int a, b, w;

```

```

57     for(int i = 0; i < m; i++){
58         scanf("%d%d%d", &a, &b, &w);
59         adj[a].push_back(make_pair(b, w));
60     }
61     int sp = 0; // start point
62
63     if(spfa(sp, n))
64         for (int i = 0; i < n; i++) printf("dist %d: %d\n",
65             i, dist[i]);
66     else printf("can't reach.\n");
67     return 0;
68 }

```

## 8.8 BellmanFord

```

1 int main(int argc, char const *argv[])
2 {
3     //initialize dis[] with 1e9
4     //make an adjacent list
5     call bellman_ford(src);
6     return 0;
7 }
8
9 void bellman_ford(int src)
10 {
11     dis[src] = 0; //initialize source
12     //with distance 0
13     for (int k = 0; k < n - 1; ++k){ //do n-1
14         //times
15         for (int i = 0; i < n; ++i){
16             for(auto j : v[i]){
17                 if(dis[i] != 1e9) dis[j] = min(dis[j], dis[i] +
18                     w[i][j]);
19             }
20         }
21     }
22     bool negativeCycle()
23     {
24         for(i = 0; i < n; ++i){
25             for(auto j : v[i]){
26                 if(dis[j] > dis[i] + w[i][j]) return true //has
27                     negative cycle
28             }
29         }
30     }
31     return false;
32 }

```

## 8.9 FloydWarshall

```

1 //dis[i][j] is the distance of node i to node j
2 int dis[n+5][n+5];
3 void init()
4 {
5     memset(dis, 0x3f, sizeof(dis));
6     for(int i = 0; i < n; i++) d[i][i] = 0;
7 }
8 void floyd(){
9     for (int k = 0; k < n; ++k)
10         for(int i = 0; i < n; ++i)
11             for(int j = 0; j < n; ++j)
12                 dis[i][j] = dis[j][i] = min(dis[i][j], dis[i][
13                     k] + dis[k][j]);
14 }
15
16 int main(int argc, char const *argv[])
17 {
18     //If we got n nodes, label from 0 to (n-1)
19     init();
20     //Set the dis
21     floyd();
22 }

```

## 8.10 Kruskal

```

1 const int maxn = 1000+5;
2 struct Edge
3 {
4     int from, to;
5     double cost;
6     bool operator<(const Edge other){
7         return cost < other.cost;
8     }
9 }E[maxn*maxn];
10 int p[maxn];
11 vector<Edge> G[maxn];
12 int find(int x){
13     return p[x] < 0 ? x : (p[x] = find(p[x]));
14 }
15 bool uni(int x ,int y)
16 {
17     int xRoot = find(x), yRoot = find(y);
18     if(xRoot != yRoot){
19         if(p[xRoot] < p[yRoot]){
20             p[xRoot] += p[yRoot];
21             p[yRoot] = xRoot;
22         }
23         else{
24             p[yRoot] += p[xRoot];
25             p[xRoot] = yRoot;
26         }
27         return true;
28     }
29     else return false;
30 }
31 double kruskal(int n, int m)
32 {
33     // n is the numbers of node, m is the numbers of edge
34     for(int i = 0; i <= n; i++){
35         G[i].clear();
36         p[i] = -1;
37     }
38     sort(E, E + m);
39     double ans = 0;
40     int edge_cnt = 0;
41     for(int i = 0; i < m; i++){
42         if(uni(E[i].from, E[i].to)){
43             int from = E[i].from, to = E[i].to;
44             ans += E[i].cost;
45             G[from].push_back(Edge{from, to, E[i].cost});
46             G[to].push_back(Edge{to, from, E[i].cost});
47             if(++edge_cnt == n-1) break;
48         }
49     }
50     if(edge_cnt == n-1) return ans;
51     else return -1; // means can't found spanning tree
52 }
53 // find max segment in MST graph
54 int maxcost[maxn][maxn];
55 vector<int> visited;
56 void dfs(int pre, int now, int w){
57     for(auto x : visited){
58         maxcost[x][now] = maxcost[now][x] = max(w, maxcost[
59             pre][x]);
60     }
61     visited.push_back(now);
62     for(auto i : G[now]){
63         if(pre != i.to) dfs(now, i.to, i.cost);
64     }
65 }
66 void findMaxPtah(int sp, int ep){
67     memset(maxcost, 0, sizeof(maxcost));
68     visited.clear();
69     dfs(-1, sp, 0);

```

## 8.11 Articulation Point

```

1 #define NIL -1
2 // A class that represents an undirected graph
3 class Graph
4 {
5     int V; // No. of vertices
6     list<int> *adj; // A dynamic array of adjacency
7         lists
8     void APUtil(int v, bool visited[], int disc[], int
9         low[], int parent[], bool ap[]);
10 public:
11     Graph(int V); // Constructor
12     void addEdge(int v, int w); // function to add an
13         edge to graph
14     void AP(); // prints articulation
15         points
16 };
17 Graph::Graph(int V)
18 {
19     this->V = V;
20     adj = new list<int>[V];
21 }
22 void Graph::addEdge(int v, int w)
23 {
24     adj[v].push_back(w);
25     adj[w].push_back(v); // Note: the graph is
26         undirected
27 }
28 // A recursive function that find articulation points
29     using DFS traversal
30 // u --> The vertex to be visited next
31 // visited[] --> keeps tract of visited vertices
32 // disc[] --> Stores discovery times of visited
33     vertices
34 // parent[] --> Stores parent vertices in DFS tree
35 // ap[] --> Store articulation points
36 void Graph::APUtil(int u, bool visited[], int disc[],
37     int low[], int parent[], bool ap[])
38 {
39     // A static variable is used for simplicity, we can
40         avoid use of static
41     // variable by passing a pointer.
42     static int time = 0;
43
44     // Count of children in DFS Tree
45     int children = 0;
46
47     // Mark the current node as visited
48     visited[u] = true;
49
50     // Initialize discovery time and low value
51     disc[u] = low[u] = ++time;
52
53     // Go through all vertices adjacent to this
54     list<int>::iterator i;
55     for (i = adj[u].begin(); i != adj[u].end(); ++i)
56     {
57         int v = *i; // v is current adjacent of u
58
59         // If v is not visited yet, then make it a
60             child of u
61         // in DFS tree and recur for it
62         if (!visited[v])
63         {
64             children++;
65             parent[v] = u;
66             APUtil(v, visited, disc, low, parent, ap);
67
68             // Check if the subtree rooted with v has a
69                 connection to
70             // one of the ancestors of u
71             low[u] = min(low[u], low[v]);

```

```

68 // u is an articulation point in following
69 // cases
70 // (1) u is root of DFS tree and has two or
71 // more children.
72 if (parent[u] == NIL && children > 1)
73     ap[u] = true;
74 // (2) If u is not root and low value of
75 // one of its child is more
76 // than discovery value of u.
77 if (parent[u] != NIL && low[v] >= disc[u])
78     ap[u] = true;
79 }
80 // Update low value of u for parent function
81 // calls.
82 else if (v != parent[u])
83     low[u] = min(low[u], disc[v]);
84 }
85 }
86 // The function to do DFS traversal. It uses recursive
87 // function APUtil()
88 void Graph::AP()
89 {
90     // Mark all the vertices as not visited
91     bool *visited = new bool[V];
92     int *disc = new int[V];
93     int *low = new int[V];
94     int *parent = new int[V];
95     bool *ap = new bool[V]; // To store articulation
96     // points
97     // Initialize parent and visited, and ap(
98     // articulation point) arrays
99     for (int i = 0; i < V; i++)
100     {
101         parent[i] = NIL;
102         visited[i] = false;
103         ap[i] = false;
104     }
105     // Call the recursive helper function to find
106     // articulation points
107     // in DFS tree rooted with vertex 'i'
108     for (int i = 0; i < V; i++)
109     {
110         if (visited[i] == false)
111             APUtil(i, visited, disc, low, parent, ap);
112     }
113     // Now ap[] contains articulation points, print
114     // them
115     for (int i = 0; i < V; i++)
116     {
117         if (ap[i] == true)
118             cout << i << " ";
119     }
120 }
121 int main()
122 {
123     Graph g(7);
124     g.addEdge(0, 1);
125     g.addEdge(1, 2);
126     g.addEdge(2, 0);
127     g.addEdge(1, 3);
128     g.addEdge(1, 4);
129     g.addEdge(1, 6);
130     g.addEdge(3, 5);
131     g.addEdge(4, 5);
132     g.AP();
133     return 0;
134 }

```

```

2 int W[maxn][maxn], n;
3 int Lx[maxn], Ly[maxn];
4 int Lef[maxn];
5 bool S[maxn], T[maxn];
6 bool match(int i)
7 {
8     S[i] = true;
9     for (int j = 1; j <= n; ++j)
10     {
11         if (Lx[i] + Ly[j] == W[i][j] && !T[j])
12         {
13             T[j] = true;
14             if (!Lef[j] || match(Lef[j]))
15             {
16                 Lef[j] = i;
17                 return true;
18             }
19         }
20     }
21     return false;
22 }
23 void update()
24 {
25     int a = 0x3f3f3f3f;
26     for (int i = 1; i <= n; i++)
27     {
28         if (S[i])
29         {
30             for (int j = 1; j <= n; j++)
31             {
32                 if (!T[j]) a = min(a, Lx[i] + Ly[j] - W[i][j]);
33             }
34         }
35     }
36     for (int i = 1; i <= n; i++)
37     {
38         if (S[i]) Lx[i] -= a;
39         if (T[i]) Ly[i] += a;
40     }
41 }
42 void KM()
43 {
44     for (int i = 1; i <= n; ++i)
45     {
46         Lef[i] = Lx[i] = Ly[i] = 0;
47         for (int j = 1; j <= n; j++){
48             Lx[i] = max(Lx[i], W[i][j]);
49         }
50     }
51     for (int i = 1; i <= n; ++i)
52     {
53         for (;;) {
54             for (int j = 1; j <= n; j++) {
55                 S[j] = T[j] = 0;
56             }
57             if (match(i)) break;
58             else update();
59         }
60     }
61 }
62 }
63 int main(int argc, char const *argv[])
64 {
65     for (int i = 1; i <= n; i++) {
66         for (int j = 1; j <= n; j++) {
67             scanf("%d", &W[i][j]);
68         }
69     }
70     KM();
71     int ans = 0;
72     for (int i = 1; i <= n; i++) {
73         ans += Ly[i];
74         ans += Lx[i];
75     }
76 }

```

## 8.12 Bipartite Matching

1 | const int maxn = 500+5;

```

79 }
80
81 for(int i = 1; i <= n; i++){
82     if(i != n) printf("%d ", Lx[i]);
83     else printf("%d\n", Lx[i]);
84 }
85
86 for(int i = 1; i <= n; i++){
87     if(i != n) printf("%d ", Ly[i]);
88     else printf("%d\n", Ly[i]);
89 }
90
91 printf("%d\n", ans);
92 return 0;
93 }

```

### 8.13 Bipartite Check1

```

1 #define S 50050
2
3 vector<int> map[S];
4 int visit[S];
5 bool valid;
6
7 void check(int start)
8 {
9     stack<int> st;
10    st.push(start);
11    visit[start] = 1;
12
13    while (valid && !st.empty())
14    {
15        int cur = st.top();
16        st.pop();
17
18        for (int i = 0; i < map[cur].size(); i++)
19        {
20            int next = map[cur][i];
21
22            if (visit[next] == -1)
23            {
24                st.push(next);
25
26                if (visit[cur] == 1)
27                    visit[next] = 2;
28                else
29                    visit[next] = 1;
30            }
31            else if (visit[cur] == visit[next])
32                valid = false;
33        }
34    }
35 }
36
37 int main()
38 {
39     int n, m;
40     cin >> n >> m;
41
42     for (int i = 0; i < m; i++)
43     {
44         int a, b;
45         cin >> a >> b;
46
47         map[a].push_back(b);
48         map[b].push_back(a);
49     }
50
51     // -1 : not visit, 1 : tsudere, 2 : proud
52     memset(visit, -1, sizeof(visit));
53     valid = true;
54
55     for (int i = 1; i <= n; i++)
56     {
57         if (valid && visit[i] == -1)
58         {

```

```

59             check(i);
60         }
61     }
62
63     if (valid)
64         cout << "yes" << endl;
65     else
66         cout << "no" << endl;
67
68     return 0;
69 }

```

### 8.14 CLE Directed MST

```

1 const int maxn = 60+5;
2 const int INF = 0x3f3f3f3f;
3 struct Edge
4 {
5     int from, to, cost;
6 };
7 Edge E[maxn * maxn], e[maxn * maxn];
8 int n, m, c;
9 int in[maxn], pre[maxn], id[maxn], vis[maxn];
10 int CLE(int root, int n, int m)
11 {
12     int res = 0;
13     while(1)
14     {
15         for(int i = 0; i < n; i++){
16             in[i] = INF;
17         }
18         //Find in edge
19         for(int i = 0; i < m; i++){
20             int from = e[i].from, to = e[i].to;
21             if(from != to && e[i].cost < in[to]){
22                 in[to] = e[i].cost;
23                 pre[to] = from;
24             }
25         }
26         //Check in edge
27         for(int i = 0; i < n; i++){
28             if(i == root) continue;
29             if(in[i] == INF) return -1;
30         }
31
32         int num = 0;
33         memset(id, -1, sizeof(id));
34         memset(vis, -1, sizeof(vis));
35         in[root] = 0;
36
37         //Find cycles
38         for(int i = 0; i < n; i++){
39             res += in[i];
40             int v = i;
41             while(vis[v] != i && id[v] == -1 && v != root)
42             {
43                 vis[v] = i;
44                 v = pre[v];
45             }
46             if(v != root && id[v] == -1)
47             {
48                 for(int j = pre[v]; j != v; j = pre[j]){
49                     id[j] = num;
50                 }
51                 id[v] = num++;
52             }
53         }
54         //No cycle
55         if(num == 0) break;
56         for(int i = 0; i < n; i++){
57             if(id[i] == -1) id[i] = num++;
58         }
59         //Grouping the vertices
60         for(int i = 0; i < m; i++){
61             int from = e[i].from, to = e[i].to;
62             e[i].from = id[from]; e[i].to = id[to];

```



```

63     if(id[from] != id[to]) e[i].cost -= in[to];
64 }
65 n = num;
66 root = id[root];
67 }
68 return res;
69 }
70 int main(int argc, char const *argv[])
71 {
72     int n, m;
73     // n nodes and m edges
74     scanf("%d%d", &n, &m);
75     for(int i = 0; i < m; i++){
76         scanf("%d%d%d", &E[i].from, &E[i].to, &E[i].cost)
77         ;
78     }
79     int sp = 0; // start point
80     int ans = CLE(sp, n, m);
81     if(ans == -1) printf("No Directed Minimum Spanning
82     Tree.\n");
83     else printf("%d\n", ans);
84     return 0;
85 }

```

## 8.15 Dinic

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int maxn = 50+5;
4 const int INF = 0x3f3f3f3f;
5 template <typename T>
6 struct Dinic
7 {
8     int n, s, t, level[maxn], now[maxn];
9     struct Edge
10     {
11         int v;
12         T rf; // rf: residual flow
13         int re;
14     };
15     vector<Edge> e[maxn];
16     void init(int _n, int _s, int _t)
17     {
18         n = _n;
19         s = _s;
20         t = _t;
21         for (int i = 0; i <= n; i++)
22         {
23             e[i].clear();
24         }
25     }
26     void add_edge(int u, int v, T f)
27     {
28         e[u].push_back({v, f, (int)e[v].size()});
29         e[v].push_back({u, f, (int)e[u].size() - 1});
30         // for directional graph
31         // e[v].push_back({u, 0, (int)e[u].size() - 1});
32     }
33     bool bfs()
34     {
35         fill(level, level + n + 1, -1);
36         queue<int> q;
37         q.push(s);
38         level[s] = 0;
39         while (!q.empty())
40         {
41             int u = q.front();
42             q.pop();
43             for (auto it : e[u])
44             {
45                 if (it.rf > 0 && level[it.v] == -1)
46                 {
47                     level[it.v] = level[u] + 1;
48                     q.push(it.v);
49                 }

```

```

50             }
51         }
52         return level[t] != -1;
53     }
54     T dfs(int u, T limit)
55     {
56         if (u == t)
57             return limit;
58         T res = 0;
59         while (now[u] < (int)e[u].size())
60         {
61             Edge &it = e[u][now[u]];
62             if (it.rf > 0 && level[it.v] == level[u] +
63                 1)
64             {
65                 T f = dfs(it.v, min(limit, it.rf));
66                 res += f;
67                 limit -= f;
68                 it.rf -= f;
69                 e[it.v][it.re].rf += f;
70                 if (limit == 0)
71                 {
72                     return res;
73                 }
74             }
75             else
76             {
77                 ++now[u];
78             }
79         }
80         if (!res)
81         {
82             level[u] = -1;
83         }
84         return res;
85     }
86     T flow(T res = 0)
87     {
88         while (bfs())
89         {
90             T tmp;
91             memset(now, 0, sizeof(now));
92             do{
93                 tmp = dfs(s, INF);
94                 res += tmp;
95             }while(tmp);
96         }
97         return res;
98     };
99     /*
100     usage
101     Dinic<int> dinic; // declare, flow type is int
102     dinic.init(n, s, t); // initialize, n vertexs, start
103     from s to t
104     dinic.add_edge(x, y, z); // add edge from x to y,
105     weight is z
106     dinic.flow() // calculate max flow
107     */

```

## 8.16 MCMF

```

1 struct Edge
2 {
3     int v;
4     T cost;
5     int cap;
6     Edge(int _v, int _cost, int _cap) : v(_v), cost(
7         _cost), cap(_cap) {}
8 };
9 vector<int> dis(MXV), pre(MXV);
10 vector<vector<int>> G(MXV);
11 int n;
12 vector<Edge> edges;
13 bitset<MXV> inque;

```

```

13 queue<int> q;
14 void init(int _n)
15 {
16     n = _n;
17     edges.clear();
18     for (int i = 0; i <= MXV; ++i)
19     {
20         G[i].clear();
21     }
22 }
23 void addEdge(int u, int v, T cost, int cap)
24 {
25     G[u].push_back((int)edges.size());
26     edges.push_back(Edge(v, cost, cap));
27     G[v].push_back((int)edges.size());
28     edges.push_back(Edge(u, -cost, 0));
29 }
30 bool spfa(int s, int t)
31 {
32     FOR(i, 0, MXV) { dis[i] = INF; }
33     inque.reset();
34     while (!q.empty())
35     {
36         q.pop();
37     }
38     dis[s] = 0;
39     q.push(s);
40     while (!q.empty())
41     {
42         int u = q.front();
43         q.pop();
44         inque[u] = false;
45         FOR(i, 0, G[u].size())
46         {
47             Edge &e = edges[G[u][i]];
48             if (e.cap > 0 && dis[e.v] > dis[u] + e.cost)
49             {
50                 dis[e.v] = dis[u] + e.cost;
51                 pre[e.v] = G[u][i];
52                 if (!inque[e.v])
53                 {
54                     q.push(e.v);
55                     inque[e.v] = true;
56                 }
57             }
58         }
59     }
60     return dis[t] != INF;
61 }
62 void update(int s, int t, int bottleneck)
63 {
64     for (int u = t; u != s;)
65     {
66         int pos = pre[u];
67         edges[pos].cap -= bottleneck;
68         edges[pos ^ 1].cap += bottleneck;
69         u = edges[pos ^ 1].v;
70     }
71 }
72 void sol(int s, int t)
73 {
74     int mnCost = 0;
75     while (spfa(s, t))
76     {
77         update(s, t, 1);
78         mnCost += dis[t];
79     }
80     cout << mnCost << '\n';
81 }
82
83 int main()
84 {
85     IOS;
86     int n, m;
87     while (cin >> n >> m)
88     {

```

```

89         init(n);
90         for (int i = 0, f, t, w; i != m; ++i)
91         {
92             cin >> f >> t >> w;
93             addEdge(f, t, w, 1);
94             addEdge(t, f, w, 1);
95         }
96         int s = 0, t = n + 1;
97         addEdge(s, 1, 0, 2);
98         addEdge(n, t, 0, 2);
99         sol(s, t);
100     }
101 }

```

## 9 Number

### 9.1 Sieve

```

1 const int maxn = 500+10;
2 bool visit[maxn];
3 int primes[maxn];
4 int sieve(int src)
5 {
6     memset(visit, false, sizeof(visit));
7     for(int i = 2; i <= sqrt(src + 0.5); i++){
8         if(!visit[i]){
9             for(int j = i * i; j <= src; j += i){
10                 visit[j] = true;
11             }
12         }
13     }
14     int cnt = 0;
15     for(int i = 2; i <= src; i++){
16         if(!visit[i]) primes[cnt++] = i;
17     }
18     return cnt;
19 }

```

### 9.2 Power

```

1 double Power(double x, int n)
2 {
3     if (n == 0) return 1.00;
4     if (n == 1) return x;
5     double ans = Power(x, n / 2);
6     if (n % 2 == 0) return ans * ans;
7     else if (n < 0) return ans * ans / x;
8     else return ans * ans * x;
9 }

```

### 9.3 Euler

```

1 const int maxn = 50000;
2 int F[maxn+5];
3 void Euler(){
4     memset(F, 0, sizeof(F));
5     F[1] = 1;
6     for(int i=2; i<maxn; i++){
7         if(!F[i]){
8             for(int j=i; j<maxn; j+=i){
9                 if(!F[j]) F[j] = j;
10                 F[j] = F[j] / i*(i-1);
11             }
12         }
13     }
14 }

```

## 9.4 Factors

```

1 vector<int> getDivisions(int x){
2     vector<int> res;
3     int sq = (int) sqrt(x + 0.5);
4     for(int i = 1; i <= sq; i++){
5         if(x % i == 0) {
6             int j = x / i;
7             res.push_back(i);
8             if(i != j) res.push_back(j);
9         }
10    }
11    return res;
12 }
```

## 9.5 Extend Euclidean

```

1 int extgcd(int a, int b, int &x, int &y)
2 {
3     int d = a;
4     if (b)
5     {
6         d = extgcd(b, a % b, y, x), y -= (a / b) * x;
7     }
8     else
9         x = 1, y = 0;
10    return d;
11 } // ax+by=1 ax同餘 1 mod b
```

## 9.6 Matrix

```

1 const int maxn = 20;
2 struct Matrix {
3     long long a[maxn][maxn];
4     void zeroM() // 清空矩陣
5     {
6         memset(a, 0, sizeof(a));
7     }
8     void identityM()
9     {
10        memset(a, 0, sizeof a);
11        for(int i = 0; i < maxn; i++)
12            a[i][i] = 1;
13    }
14 };
15 int n; // n * n matrix
16
17 Matrix operator*(const Matrix &a, const Matrix &b) //
    矩陣乘法
18 {
19     Matrix ret;
20     ret.all_0();
21     for (int i = 0; i < n; i++){
22         for (int j = 0; j < n; j++){
23             for (int k = 0; k < n; k++){
24                 ret.a[i][j] += a.a[i][k] * b.a[k][j];
25                 ret.a[i][j] %= m;
26             }
27         }
28     }
29     return ret;
30 }
31
32 Matrix mpower(Matrix a, int n)
33 {
34     Matrix ret;
35     ret.identityM();
36     if (n==0) return ret;
37     if (n==1) return a;
38     ret = mpower(a, n/2);
39     ret = ret * ret;
40     if (n % 2 == 1) ret = ret * a;
41     return ret;
```

```

42 }
43 int main(int argc, char const *argv[])
44 {
45     cin >> n; // n * n matrix
46     Matrix A; // declare
47     return 0;
48 }
```

## 9.7 GaussElimination

```

1 const int MAXN = 300;
2 const double EPS = 1e-8;
3 int n;
4 double A[MAXN][MAXN];
5 void Gauss()
6 {
7     for (int i = 0; i < n; i++)
8     {
9         bool ok = 0;
10        for (int j = i; j < n; j++)
11        {
12            if (fabs(A[j][i]) > EPS)
13            {
14                swap(A[j], A[i]);
15                ok = 1;
16                break;
17            }
18        }
19        if (!ok)
20            continue;
21        double fs = A[i][i];
22        for (int j = i + 1; j < n; j++)
23        {
24            double r = A[j][i] / fs;
25            for (int k = i; k < n; k++)
26            {
27                A[j][k] -= A[i][k] * r;
28            }
29        }
30    }
31 }
```

## 10 Geometry

### 10.1 Geometry

```

1 using f64 = double;
2 using dvt = f64;
3 const double eps = 1e-9;
4 struct dot
5 {
6     dvt x, y;
7 };
8
9 struct line
10 {
11     dot start, end;
12 };
13
14 dot operator+(dot a, dot b) { return {a.x + b.x, a.y + b.y}; }
15 dot operator-(dot a, dot b) { return {a.x - b.x, a.y - b.y}; }
16 dot operator*(dot a, dvt c) { return {a.x * c, a.y * c}; }
17 dot operator*(dvt c, dot a) { return a * c; }
18 dot operator/(dot a, dvt c) { return {a.x / c, a.y / c}; }
19 bool operator<(dot a, dot b)
20 {
21     return std::tie(a.x, a.y) < std::tie(b.x, b.y);
22 }
```

```

23 |
24 | dvt iproduct(dot a, dot b)
25 | {
26 |     return a.x * b.x + a.y * b.y;
27 | }
28 |
29 | dvt cross(dot a, dot b)
30 | {
31 |     return a.x * b.y - a.y * b.x;
32 | }
33 |
34 | int side(line L, dot a)
35 | {
36 |     dvt cross_value = cross(a - L.start, L.end - L.start)
37 |     ;
38 |     if (cross_value > eps)
39 |     {
40 |         return 1;
41 |     }
42 |     else if (cross_value < -eps)
43 |     {
44 |         return -1;
45 |     }
46 |     return 0;
47 | }
48 | bool has_jiao(line AB, line CD)
49 | {
50 |     int c = side(AB, CD.start);
51 |     int d = side(AB, CD.end);
52 |     // 0 代表在線上
53 |     if (c == 0 || d == 0)
54 |         return true;
55 |     // 正負號不同=>異側=>相交
56 |     return c == -d;
57 | }

```

## 10.2 Lines Intersection1

```

1 | #include <iostream>
2 | #include <cmath>
3 | #include <cstring>
4 |
5 | using namespace std;
6 |
7 | struct pt {
8 |     double x, y;
9 | };
10 |
11 | struct line {
12 |     double a, b, c;
13 |     line(pt p1, pt p2) {
14 |         a = p2.y - p1.y;
15 |         b = p1.x - p2.x;
16 |         c = -a * p1.x - b * p1.y;
17 |     }
18 | };
19 |
20 | const double EPS = 1e-9;
21 |
22 | double det (double a, double b, double c, double d) {
23 |     return a * d - b * c;
24 | }
25 |
26 | bool intersect (line m, line n, pt & res) {
27 |     double zn = det (m.a, m.b, n.a, n.b);
28 |     if (abs (zn) < EPS)
29 |         return false;
30 |     res.x = - det (m.c, m.b, n.c, n.b) / zn;
31 |     res.y = - det (m.a, m.c, n.a, n.c) / zn;
32 |     return true;
33 | }
34 |
35 | bool parallel (line m, line n) {
36 |     return abs (det (m.a, m.b, n.a, n.b)) < EPS;

```

```

37 | }
38 |
39 | bool equivalent (line m, line n) {
40 |     return abs (det (m.a, m.b, n.a, n.b)) < EPS
41 |         && abs (det (m.a, m.c, n.a, n.c)) < EPS
42 |         && abs (det (m.b, m.c, n.b, n.c)) < EPS;
43 | }
44 |
45 | void solve(line a, line b) {
46 |     if (equivalent(a, b)) {
47 |         cout << "LINE\n";
48 |         return ;
49 |     }
50 |     if (parallel(a, b)) {
51 |         cout << "NONE\n";
52 |         return ;
53 |     }
54 |     pt res;
55 |     intersect(a, b, res);
56 |     cout.precision(2);
57 |     cout << "POINT " << fixed << res.x << " " << res.y <<
58 |         "\n";
59 | }
60 |
61 | int main() {
62 |     int t;
63 |     cin >> t;
64 |     cout << "INTERSECTING LINES OUTPUT\n";
65 |     while (t--) {
66 |         pt p1, p2;
67 |         cin >> p1.x >> p1.y >> p2.x >> p2.y;
68 |         line a = line(p1, p2);
69 |         cin >> p1.x >> p1.y >> p2.x >> p2.y;
70 |         line b = line(p1, p2);
71 |
72 |         solve(a, b);
73 |     }
74 |     cout << "END OF OUTPUT\n";
75 |     return 0;

```

## 10.3 Lines Intersection2

```

1 | #include<iostream>
2 | #include<cstdio>
3 | using namespace std;
4 |
5 | struct point {
6 |     int x, y;
7 | };
8 |
9 | int direction (point p1, point p2, point p3) {
10 |     return (p2.y - p1.y) * (p3.x - p2.x) - (p3.y - p2.y)
11 |         * (p2.x - p1.x);
12 | }
13 |
14 | bool onSeg (point P1, point P2, point P3) {
15 |     if ( min(P1.x, P2.x) <= P3.x && P3.x <= max(P1.x,
16 |         P2.x) )
17 |         if ( min(P1.y, P2.y) <= P3.y && P3.y <= max(P1.
18 |             y, P2.y) )
19 |             return true;
20 |     return false;
21 | }
22 |
23 | bool segIntersect (point p1, point p2, point p3, point
24 |     p4) {
25 |     int d1 = direction(p3,p4,p1); int d2 = direction(p3
26 |         ,p4,p2);
27 |     int d3 = direction(p1,p2,p3); int d4 = direction(p1
28 |         ,p2,p4);
29 |
30 |     if ( ((d1>0 && d2<0) || (d1<0 && d2>0)) && ((d3>0
31 |         && d4<0) || (d3<0 && d4>0)) )
32 |         return true;

```

```

27     else if ( d1 == 0 && onSeg(p3,p4,p1) ) return true; 40
28     else if ( d2 == 0 && onSeg(p3,p4,p2) ) return true; 41
29     else if ( d3 == 0 && onSeg(p1,p2,p3) ) return true; 42
30     else if ( d4 == 0 && onSeg(p1,p2,p4) ) return true; 43
31     else return false;
32
33 }
34
35 int main () {
36     // We will have four point to check the lines
37     // 2 for one lines edges, 2 for the other one
38     point line1p1;
39     point line1p2;
40     point line2p1;
41     point line2p2;
42
43     // Inputs for points
44     cin >> line1p1.x >> line1p1.y >> line1p2.x >>
45         line1p2.y;
46     cin >> line2p1.x >> line2p1.y >> line2p2.x >>
47         line2p2.y;
48     if ( segIntersect( line1p1, line1p2, line2p1,
49         line2p2 ) )
50         cout << "YES" << endl ;
51     else
52         cout << "NO" << endl ;
53 }

```

## 10.4 Polygon Inside Or Outside

```

1 #include<iostream>
2 #define INF 10000
3 using namespace std;
4
5 struct point {
6     int x,y;
7 };
8
9 int dir (point p1, point p2, point p3) {
10
11     int val = (p3.x - p2.x) * (p2.y - p1.y) - (p3.y - p2.
12         y) * (p2.x - p1.x);
13
14     if (val == 0) return 0;
15     return (val > 0) ? 1 : 2;
16 }
17
18 bool onSeg (point p1, point p2, point p3) {
19
20     if ((min(p1.x , p2.x) <= p3.x && p3.x <= max(p1.x, p2
21         .x)) && (min(p1.y, p2.y) <= p3.y && p3.y <= max(
22         p1.y, p2.y)) ) return true;
23     return false;
24 }
25
26 bool doIntersect (point p1, point p2, point p3, point
27     p4) {
28
29     int d1 = dir(p3, p4, p1); int d2 = dir(p3, p4, p2);
30     int d3 = dir(p1, p2, p3); int d4 = dir(p1, p2, p4);
31
32     if (d1 != d2 && d3 != d4) return true;
33     if (d1 == 0 && onSeg(p3, p4, p1)) return true;
34     if (d2 == 0 && onSeg(p3, p4, p2)) return true;
35     if (d3 == 0 && onSeg(p1, p2, p3)) return true;
36     if (d4 == 0 && onSeg(p1, p2, p4)) return true;
37     return false;
38 }
39
40 bool inInside ( point * polygon, int n, point p) {

```

```

41     if ( n < 3 ) return false;
42
43     point extreme = { INF , p.y };
44
45     int count = 0, i = 0;
46
47     do{
48
49         int next = (i+1)%n;
50
51         if(doIntersect(polygon[i], polygon[next], p,
52             extreme)) {
53             if(dir(polygon[i], p, polygon[next]) == 0)
54                 return onSeg(polygon[i], p, polygon[next]);
55             // Might be wrong I will check
56             count++;
57         }
58         i = next;
59     } while(i != 0);
60     return count&1;
61 }
62
63 int main () {
64
65     point Polygon[4] = {{0,0},{0,4},{4,4},{4,0}};
66     int size = sizeof(Polygon)/sizeof(point);
67
68     point p = {20, 20};
69     inInside(Polygon, size, p) ? cout << "Yes\n" : cout
70         << "No\n" ;
71
72     point p2 = {2, 2};
73     inInside(Polygon, size, p2) ? cout << "Yes\n" :
74         cout << "No\n" ;
75 }
76

```

## 10.5 Convex Hull

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 struct point{
5     int x;
6     int y;
7     int d;
8 }p[600],ch[600];
9
10 int dist(point a, point b) {
11     return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y);
12 }//若點的angle一樣，則比較遠的點
13
14 bool find_small_vertex(point a, point b) {
15     return (a.y < b.y) || (a.y == b.y && a.x < b.x);
16 }
17
18 int cross(point o, point a, point b) {
19     return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x
20         - o.x);
21 }
22
23 bool compare_angle(point a, point b){
24     double c = cross( p[0], a, b );
25     if ( !c ) return a.d < b.d;
26     else return c > 0;
27 }
28
29 void GrahamScan(int k){
30     sort(p+0, p+k, find_small_vertex);
31     for(int i=1; i<k; i++){

```

```
31     p[i].d = dist(p[0], p[i]);
32 }
33 sort(p+1, p+k, compare_angle);
34
35 int m=0;
36 for(int i=0; i<k; i++){
37     while(m>=2 && cross(ch[m-2], ch[m-1], p[i]) <= 0){
38         m--;
39     }
40     ch[m++] = p[i];
41 }
42 // Convex Hull find m nodes and print them out
43 printf("%d\n", m+1);
44 for(int j=0; j<m; j++){
45     printf("%d %d\n", ch[j].x, ch[j].y);
46 }
47 printf("%d %d\n", ch[0].x, ch[0].y);
48 }
```