

# 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 Disjoint Set	
2.2 Segment Tree	
2.3 Tree Policy	
2.4 KMP	
2.5 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	
8.13CLE Directed MST	
8.14Dinic	
8.15Convex Hull	
<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 Lines Intersection	
<b>10 other</b>	
10.1BigNum	
10.2DP + Dijkstra	
10.3MST + Enumeration	
10.4Binary Search Example	
10.5Binary Search Example	
10.6BFS Number Transform	

## 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.
9       divide(m); n.mod(m);
10      n.pow(m.intValue()); n.gcd(m); n.negate(); n.
11      abs();
12  }

```

## 2 Data Structure

### 2.1 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 }

```

### 2.2 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,
9  int R)
10 {
11     if(L == R) ST[vertex] = L;
12     else
13     {
14         int nL = vertex * 2, nR = vertex * 2 + 1;
15         ST_Build(ST, A, nL, L, L + (R - L) / 2);
16         ST_Build(ST, A, nR, L + (R - L) / 2 + 1, R);
17         int indexL = ST[nL], indexR = ST[nR];
18         int valueL = A[indexL], valueR = A[indexR];
19         ST[vertex] = valueL <= valueR ? indexL : indexR;
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
29 , int qL, int qR)
30 {
31     int temp, mid = (L + R) / 2;
32     if(qL <= L && R <= qR) return ST[vertex];
33     if(qR <= mid)
34     { //all we want at the left child
35         return query(ST, A, vertex * 2, L, mid, qL, qR);
36     }
37     if(qL > mid)
38     { // all we want at the right child
39         return query(ST, A, vertex * 2 + 1, mid + 1, R, qL,
40 qR);
41     }
42     return A[query(ST, A, vertex * 2, L, mid, qL, qR)] <=
43 A[query(ST, A, vertex * 2 + 1, mid + 1, R, qL,
44 qR)]
45 ? query(ST, A, vertex * 2, L, mid, qL, qR) :
46 query(ST, A, vertex * 2 + 1, mid + 1, R, qL,
47 qR);
48 }
49 void update(vi &ST, vi &A, int x, int L, int R, int p,
50 int v)
51 {
52     // p is the index where you want to update
53     // v is the value will be update in A[p];
54     int mid = L + (R - L) / 2;
55     if(L == R) A[ST[x]] = v;
56     else
57     {
58         if(p <= mid) update(ST, A, x*2, L, mid, p, v);
59         else update(ST, A, x*2+1, mid+1, R, p, v);
60         ST[x] = (A[ST[x*2]] <= A[ST[x*2+1]]) ? ST[x*2] : ST
61 [x*2+1];
62     }
63 }
64 int main(int argc, char const *argv[])
65 {
66     ST_Creation(ST, A);
67     printf("%d\n", query(ST, A, 1, 0, n-1, 3, 7));
68     // query return the index
69     printf("%d\n", A[query(ST, A, 1, 0, n-1, 3, 7)]);
70     update(ST, A, 1, 0, n-1, 5, 18);
71     // query and update first to fifth parameter dont
72     // change
73     // ST, A, 1, 0, n-1
74     // last two would be
75     // query: the range(array index) you want to query
76     // update: first is the index you want to update,
77     // second is the value will be
78     return 0;
79 }

```

### 2.3 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,
8  tree_order_statistics_node_update> new_data_set;
9  new_data_set t;
10 int main()
11 {
12     t.insert(5);
13     t.insert(6);
14     t.insert(3);
15     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 }

```

## 2.4 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 }

```

## 2.5 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 }

```

```

38     if(p[i].x <= p[mid].x + mind && p[i].x >= p[mid].x
        - mind)
39         v.push_back(p[i]);
40 }
41 sort(v.begin(), v.end(), greater<point2D>());
42
43 for(vector<point2D>::iterator it = v.begin(); it != v
        .end()-1; it++){
44     for(vector<point2D>::iterator jt = it + 1; jt != v.
        end(); jt++){
45         mind = min(mind, dis(*it, *jt));
46     }
47 }
48 return mind;
49 }
50
51 int main(int argc, char const *argv[])
52 {
53     int n;
54     double min;
55     while(cin >> n && n)
56     {
57         for(int i = 0; i < n; i++){
58             cin >> p[i].x >> p[i].y;
59         }
60         sort(p, p + n);
61         min = findcp(0, n-1, n);
62         if(min < 10000) printf("%.4lf\n", min);
63         else printf("INFINITY\n");
64     }
65     return 0;
66 }
67 }

```

## 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
        - p2.y) * (p1.y - p2.y)));
16 }
17 double bruteforce(int start, int n){
18     double mind = 2e9;
19     for(int i = start; i < n - 1; i++){
20         for(int j = i + 1; j < n; j++){
21             mind = min(mind, dis(p[i], p[j]));
22         }
23     }
24     return mind;
25 }
26 double findcp(int left, int right, int n)
27 {
28     if(n <= 3){
29         return bruteforce(left, n);
30     }
31     double mind;
32     int mid = left + (right - left) / 2;
33     double cl = findcp(left, mid, mid - left + 1);
34     double cr = findcp(mid + 1, right, right - mid);
35     mind = min(cl, cr);
36     vector<point2D> v;
37     for(int i = left; i <= right; i++){

```

## 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     vector<int> v;
5     v.push_back(s[0]);
6
7     for (int i = 1; i < s.size(); ++i)
8     {

```



```

47         backtracking[i][j] = Coord{i-1, j};
48     }
49     dp[i][j]++;
50 }
51 }
52 }
53
54 printf("%d\n", dp[strlen(s1)][strlen(s2)]);
55 int curi = strlen(s1), curj = strlen(s2);
56 ans.push_back(Coord{curi, curj});
57 while(cur_i != 0 || cur_j != 0){
58     int tempi = curi, tempj = curj;
59     curi = backtracking[tempi][tempj].x; curj =
60         backtracking[tempi][tempj].y;
61     ans.push_back(Coord{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 }
88 return 0;
89 }

```

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 = 0xFFFFFFF;
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     }
17     //如果不用紀錄k是誰
18     //minx = min(minx, do_dp(i, k) + do_dp(k+1, j) + d[
19         i-1]*d[k]*d[j]);
20 }
21 return M[i][j] = minx;
22 }
23 int main(){
24     int n, temp, s, ans;
25     cin >> n;
26     stringstream s1;
27     string str;
28     cin.ignore();
29     while(n--){
30         getline(cin, str);

```

```

27     s1.clear();
28     s1.str("");
29     s1 << str;
30     d.clear();
31     while(s1 >> temp){
32         d.push_back(temp);
33     }
34     s = d.size() - 1;
35     memset(M, 0, sizeof(M));
36     memset(P, 0, sizeof(P));
37     ans = do_dp(1, s);
38     printf("%d\n", ans);
39 }
40 }

```

## 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 }
31 return 0;

```



29| }

## 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             // bulid prefix query
16             for(int i = 0; i < n; i++){
17                 for(int j = 0; j < n; j++){
18                     query[i][j] = arr[i][j];
19                     if(i - 1 >= 0) query[i][j] += query[i-1][j];
20                     if(j - 1 >= 0) query[i][j] += query[i][j-1];
21                     if(i - 1 >= 0 && j - 1 >= 0) query[i][j] -=
22                         query[i-1][j-1];
23                 }
24             }
25             int temp;
26             int maximum = 0x80000000;
27             // find the maximum sum in any range
28             for(int i = 0; i < n; i++){
29                 for(int j = 0; j < n; j++){
30                     for(int k = i; k < n; k++){
31                         for(int t = j; t < n; t++){
32                             temp = query[k][t];
33                             if(i - 1 >= 0) temp -= query[i-1][t];
34                             if(j - 1 >= 0) temp -= query[k][j-1];
35                             if(i - 1 >= 0 && j - 1 >= 0) temp += query[
36                                 i-1][j-1];
37                             if(maximum < temp) maximum = temp;
38                         }
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;

```

```

19         FenwickTree[index] += arr[i-1];
20         int temp = arr[i-1];
21         while(index + ANDlowbit(index) <= maxn){
22             index += ANDlowbit(index);
23             FenwickTree[index] += temp;
24         }
25     }
26 }
27 void Modify(int src, int val)
28 {
29     // Modify arr[src] to val
30     int gap = val - arr[src];
31     arr[src] = val;
32     int index = src + 1;
33     FenwickTree[index] += gap;
34     while(index + ANDlowbit(index) <= maxn){
35         index += ANDlowbit(index);
36         FenwickTree[index] += gap;
37     }
38 }
39 int SequenceQuery(int src)
40 {
41     //src is the index of the array which we want to know
42     //the Sequence Query
43     int res = FenwickTree[src];
44     int index = src;
45     while(index - ANDlowbit(index) > 0){
46         index -= ANDlowbit(index);
47         res += FenwickTree[index];
48     }
49     return res;
50 }
51 int RangeSumQuery(int s, int e)
52 {
53     return SequenceQuery(e) - SequenceQuery(s - 1);
54 }
55 int main(int argc, char const *argv[])
56 {
57     init();
58     int start = 2, end = 5;
59     // for Fenwick index is 3, 6 for array index is 2, 5
60     printf("RangeSumQuery(%d, %d): %d\n", start, end,
61         RangeSumQuery(start + 1, end + 1));
62     Modify(2, 5);
63     // Modify arr[2] from 3 to 5
64     printf("RangeSumQuery(%d, %d): %d\n", start, end,
65         RangeSumQuery(start + 1, end + 1));
66     return 0;
67 }

```

## 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
6     [maxDigit];
7 // aux size is depends on the max digit in sorting
8 int main(int argc, char const *argv[])
9 {
10     int n = 4;
11     // array index start with 1
12     memset(aux, 0, sizeof(aux));
13     for(int i = 1; i <= n; i++){
14         aux[unsorted[i]]++;
15     }
16     for(int i = 1; i < maxDigit; i++){
17         aux[i] += aux[i-1];
18     }
19     for(int i = n; i > 0; i--){
20         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(check 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     state[src] = 2;
21
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 ==
        // white, 1 == gray, 2 == black
5 };
6 vector<int> adj[maxn];
7 Node node[maxn];
8 int times;
9 void DFS(int src)
10 {
11     node[src].d = times++;
12     node[src].color = 1;
13     for(auto i : adj[src]){
14         if(node[i].color == 0) DFS(i);
15     }
16     node[src].color = 2;
17     node[src].f = times++;
18 }
19 void DFS_Start(int n, int sp)
20 {
21     for(int i = 0; i < n; i++){
22         node[i].color = 0;
23     }
24     times = 0;
25     DFS(sp);
26 }
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,
            node[i].f, node[i].color);
40     }
41     return 0;
42 }

```

## 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
                [j]);
35         }
36         ans = max(ans, n - blocks[i]);
37         printf("%d\n", ans);
38     }
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].
                early + e.cost) {

```

```

38         vertices[e.at].early = vertices[now].
           early + e.cost;
39         dfsEarly(e.at);
40     }
41 }
42 }
43
44 void dfsLate(int now) {
45     for (auto e : vertices[now].from) {
46         if (vertices[e.at].late > vertices[now].
           late - e.cost) {
47             vertices[e.at].late = vertices[now].
           late - e.cost;
48             dfsLate(e.at);
49         }
50     }
51 }
52
53 // may be slow?
54 void printCritical(int now, vector<int> path) {
55     if (vertices[now].to.size() == 0) {
56         // critical path found
57         for (auto i : path) {
58             cout << i << ' ';
59         }
60         cout << '\n';
61         return;
62     }
63     for (auto e : vertices[now].to) {
64         if (vertices[e.at].early == vertices[e.at].
           late) {
65             vector<int> tmp = path;
66             tmp.push_back(e.at);
67             printCritical(e.at, tmp);
68         }
69     }
70 }
71
72 int run() {
73     for (int i = 0; i < n; i++) {
74         if (vertices[i].from.size() == 0) {
75             vertices[i].early = 0;
76             dfsEarly(i);
77         }
78     }
79
80     int ans = 0;
81     for (int i = 0; i < n; i++) {
82         if (vertices[i].to.size() == 0) {
83             vertices[i].late = vertices[i].early;
84             ans = max(ans, vertices[i].late);
85             dfsLate(i);
86         }
87     }
88
89     for (int i = 0; i < n; i++) {
90         cout << "i = " << i << " early = " <<
           vertices[i].early << " late = " <<
           vertices[i].late << "\n";
91     }
92
93     for (int i = 0; i < n; i++) {
94         if (vertices[i].from.size() == 0) {
95             vector<int> path;
96             path.push_back(i);
97             printCritical(i, path);
98         }
99     }
100
101     return ans;
102 }
103 };
104 int main() {
105     AOE aoetest;
106     int n, m;
107     cin >> n >> m;
108     aoetest.init(n);

```

```

109     int a, b, w;
110     for (int i = 0; i < m; i++) {
111         cin >> a >> b >> w;
112         aoetest.addEdge(a, b, w);
113     }
114
115     int res = aoetest.run();
116     cout << "res = " << res << endl;
117 }

```

## 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; //
        放 (距離, 點編號), 每次會拿距離最小的點出來
12     q.push(MP(0, cur));
13
14     while (!q.empty())
15     {
16         tie(d, cur) = q.top(); q.pop();
17         if (dis[cur] != 1e9) continue; // 如果之前就拜訪
            過, 無視
18
19         dis[cur] = d;
20
21         for (auto i : adj[cur]){
22             if (dis[i.first] == 1e9) q.push(MP(d + i.second,
                i.first));
23         }
24     }
25 }
26
27 void init(void)
28 {
29     fill(dis, dis + maxn, 1e9);
30
31     for (int i = 0; i < maxn; i++){
32         adj[i].clear();
33     }
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], vis[maxn], out[maxn];
9 //dist = distance, vis = visit, out
10 vector< pair< int, int > > adj[maxn];
11
12 void init()
13 {
14     memset(dist, INF, sizeof(dist));
15     memset(vis, 0, sizeof(vis));
16     memset(out, 0, sizeof(out));
17     for(int i = 0; i <= n; i++){
18         adj[i].clear();
19     }

```

```

20 }
21
22 bool spfa(int sp, int n)
23 {
24     queue<int> q;
25     q.push(sp);
26
27     while(!q.empty())
28     {
29         int u = q.front(); q.pop();
30         vis[u] = 0; // pop point
31         out[u]++;
32         if(out[u] > n) return false; // negative cycle occurs
33
34         for(int j = 0; j < adj[u].size(); j++){
35             int v = adj[u][j].first; // first is point,
36                 second is weight
37             if(dist[v] > dist[u] + adj[u][j].second){
38                 dist[v] = dist[u] + adj[u][j].second;
39                 if(vis[v]) continue;
40
41                 vis[v] = 1; //push point
42                 q.push(v);
43             }
44         }
45         return true;
46     }
47
48 int main(int argc, char const *argv[])
49 {
50     // n nodes and m edges
51     scanf("%d%d", &n, &m);
52     init();
53     // make adjacent list
54     int a, b, w;
55     for(int i = 0; i < m; i++){
56         scanf("%d%d%d", &a, &b, &w);
57         adj[a].push_back(make_pair(b, w));
58     }
59     int sp = 0; // start point
60     dist[sp] = 0; vis[sp] = 1;
61     if(spfa(sp, n))
62         for (int i = 0; i < n; i++) printf("dist %d: %d\n",
63             i, dist[i]);
64     else printf("can't reach.\n");
65     return 0;
66 }

```

## 8.8 BellmanFord

```

1 int main(int argc, char const *argv[])
2 {
3     //initialize dis[] with 1e9
4     //make an adjecnt 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 }

```

```

22 for(i = 0; i < n; ++i){
23     for(auto j : v[i]){
24         if(dis[j] > dis[i] + w[i][j]) return true //has
25             negative cycle
26     }
27 }
28 return false;

```

## 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 int main(int argc, char const *argv[])
16 {
17     //If we got n nodes, label from 0 to (n-1)
18     init();
19     //Set the dis
20     floyd();

```

## 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     int root, trail, lead;
14     for (root = x ; p[root] >= 0; root = p[root]);
15     for (trail = x ; trail != root; trail = lead) {
16         lead = p[trail];
17         p[trail]= root;
18     }
19     return root;
20 }
21 bool uni(int x ,int y)
22 {
23     int xRoot = find(x), yRoot = find(y);
24     if(xRoot != yRoot){
25         if(p[xRoot] > p[yRoot]){
26             p[xRoot] += p[yRoot];
27             p[yRoot] = xRoot;
28         }
29         else{
30             p[yRoot] += p[xRoot];
31             p[xRoot] = yRoot;
32         }
33     }
34     return true;
35 }
36 else return false;
37 }
38 double kruskal(int n, int m)
39 {

```

```

39 // n is the numbers of node, m is the numbers of edge
40 for(int i = 0; i <= n; i++){
41     G[i].clear();
42     p[i] = -1;
43 }
44 sort(E, E + m);
45 double ans = 0;
46 int edge_cnt = 0;
47 for(int i = 0; i < m; i++){
48     if(uni(E[i].from, E[i].to)){
49         int from = E[i].from, to = E[i].to;
50         ans += E[i].cost;
51         G[from].push_back(Edge{from, to, E[i].cost});
52         G[to].push_back(Edge{to, from, E[i].cost});
53         if(++edge_cnt == n-1) break;
54     }
55 }
56 if(edge_cnt == n-1) return ans;
57 else return -1; // means can't found spanning tree
58 }
59 // find max segment in MST graph
60 int maxcost[maxn][maxn];
61 vector<int> visited;
62 void dfs(int pre, int now, int w){
63     for(auto x : visited){
64         maxcost[x][now] = maxcost[now][x] = max(w, maxcost[
65             pre][x]);
66     }
67     visited.push_back(now);
68     for(auto i : G[now]){
69         if(pre != i.to) dfs(i.from, i.to, i.cost);
70     }
71 }
72 void findMaxPtah(int sp, int ep){
73     memset(maxcost, 0, sizeof(maxcost));
74     visited.clear();
75     dfs(-1, sp, 0);
76 }

```

## 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     void APUtil(int v, bool visited[], int disc[], int
8         low[], int parent[], bool ap[]);
9
10 public:
11     Graph(int V); // Constructor
12     void addEdge(int v, int w); // function to add an
13     void AP(); // prints articulation
14 };
15
16 Graph::Graph(int V)
17 {
18     this->V = V;
19     adj = new list<int>[V];
20 }
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 // parent[] --> Stores parent vertices in DFS tree
34 // ap[] --> Store articulation points
35 void Graph::APUtil(int u, bool visited[], int disc[],
36     int low[], int parent[], bool ap[])
37 {
38     // A static variable is used for simplicity, we can
39     // avoid use of static
40     // variable by passing a pointer.
41     static int time = 0;
42
43     // Count of children in DFS Tree
44     int children = 0;
45
46     // Mark the current node as visited
47     visited[u] = true;
48
49     // Initialize discovery time and Low value
50     disc[u] = low[u] = ++time;
51
52     // Go through all vertices adjacent to this
53     list<int>::iterator i;
54     for (i = adj[u].begin(); i != adj[u].end(); ++i)
55     {
56         int v = *i; // v is current adjacent of u
57
58         // If v is not visited yet, then make it a
59         // child of u
60         // in DFS tree and recur for it
61         if (!visited[v])
62         {
63             children++;
64             parent[v] = u;
65             APUtil(v, visited, disc, low, parent, ap);
66
67             // Check if the subtree rooted with v has a
68             // connection to
69             // one of the ancestors of u
70             low[u] = min(low[u], low[v]);
71
72             // u is an articulation point in following
73             // cases
74
75             // (1) u is root of DFS tree and has two or
76             // more children.
77             if (parent[u] == NIL && children > 1)
78                 ap[u] = true;
79
80             // (2) If u is not root and Low value of
81             // one of its child is more
82             // than discovery value of u.
83             if (parent[u] != NIL && low[v] >= disc[u])
84                 ap[u] = true;
85         }
86
87         // Update Low value of u for parent function
88         // calls.
89         else if (v != parent[u])
90             low[u] = min(low[u], disc[v]);
91     }
92 }
93
94 // The function to do DFS traversal. It uses recursive
95 // function APUtil()
96 void Graph::AP()
97 {
98     // Mark all the vertices as not visited
99     bool *visited = new bool[V];
100     int *disc = new int[V];
101     int *low = new int[V];
102     int *parent = new int[V];
103     bool *ap = new bool[V]; // To store articulation
104     // points

```

```

96 // Initialize parent and visited, and ap(
    articulation point) arrays
97 for (int i = 0; i < V; i++)
98 {
99     parent[i] = NIL;
100     visited[i] = false;
101     ap[i] = false;
102 }
103
104 // Call the recursive helper function to find
    articulation points
105 // in DFS tree rooted with vertex 'i'
106 for (int i = 0; i < V; i++)
107     if (visited[i] == false)
108         APUtil(i, visited, disc, low, parent, ap);
109
110 // Now ap[] contains articulation points, print
    them
111 for (int i = 0; i < V; i++)
112     if (ap[i] == true)
113         cout << i << " ";
114 }
115
116 int main()
117 {
118     Graph g(7);
119     g.addEdge(0, 1);
120     g.addEdge(1, 2);
121     g.addEdge(2, 0);
122     g.addEdge(1, 3);
123     g.addEdge(1, 4);
124     g.addEdge(1, 6);
125     g.addEdge(3, 5);
126     g.addEdge(4, 5);
127     g.AP();
128
129     return 0;
130 }

```

## 8.12 Bipartite Matching

```

1 const int maxn = 500+5;
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
18                 return true;
19             }
20         }
21     }
22     return false;
23 }
24 void update()
25 {
26     int a = 0x3f3f3f3f;
27     for (int i = 1; i <= n; i++)
28     {
29         if (S[i])
30         {
31             for (int j = 1; j <= n; j++)
32             {
33                 if (!T[j]) a = min(a, Lx[i] + Ly[j] - W[i][j]);
34             }
35         }
36     }

```

```

36 }
37 for (int i = 1; i <= n; i++)
38 {
39     if (S[i]) Lx[i] -= a;
40     if (T[i]) Ly[i] += a;
41 }
42 }
43 void KM()
44 {
45     for (int i = 1; i <= n; ++i)
46     {
47         Lef[i] = Lx[i] = Ly[i] = 0;
48         for (int j = 1; j <= n; j++){
49             Lx[i] = max(Lx[i], W[i][j]);
50         }
51     }
52     for (int i = 1; i <= n; ++i)
53     {
54         for (;;){
55             for (int j = 1; j <= n; j++){
56                 S[j] = T[j] = 0;
57             }
58             if (match(i)) break;
59             else update();
60         }
61     }
62 }
63
64 int main(int argc, char const *argv[])
65 {
66     for (int i = 1; i <= n; i++){
67         for (int j = 1; j <= n; j++){
68             scanf("%d", &W[i][j]);
69         }
70     }
71 }
72
73 KM();
74 int ans = 0;
75
76 for (int i = 1; i <= n; i++){
77     ans += Ly[i];
78     ans += Lx[i];
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 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     int sp = 0; // start point
79     int ans = CLE(sp, n, m);
80     if(ans == -1) printf("No Directed Minimum Spanning
81         Tree.\n");
82     else printf("%d\n", ans);
83     return 0;
84 }

```

## 8.14 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     }
81 }

```



```

79     if (!res)
80     {
81         level[u] = -1;
82     }
83     return res;
84 }
85 T flow(T res = 0)
86 {
87     while (bfs())
88     {
89         T tmp;
90         memset(now, 0, sizeof(now));
91         do{
92             tmp = dfs(s, INF);
93             res += tmp;
94         }while(tmp);
95     }
96     return res;
97 }
98 };
99
100 /*
101 usage
102 Dinic<int> dinic; // declare, flow type is int
103 dinic.init(n, s, t); // initialize, n vertexs, start
104   from s to t
105 dinic.add_edge(x, y, z); // add edge from x to y,
106   weight is z
107 dinic.flow() // calculate max flow
108 */

```

## 8.15 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 bool compare_angle(point a, point b){
23     double c = cross( p[0], a, b );
24     if ( !c ) return a.d < b.d;
25     else return c > 0;
26 }
27
28 void GrahamScan(int k){
29     sort(p+0, p+k, find_small_vertex);
30     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 }

```

## 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> getDivisioris(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  template <typename T, int N = 2> struct Mat
2  { // Matrix
3      unsigned long long v[N][N];
4      Mat operator*(Mat b) const
5      {
6          Mat val;
7          for (int i = 0; i < N; i++)
8          {
9              for (int j = 0; j < N; j++)
10             {
11                 val.v[i][j] = 0;
12                 for (int k = 0; k < N; k++)
13                 {
14                     val.v[i][j] += v[i][k] * b.v[k][j];
15                 }
16             }
17         }
18         return val;
19     }
20 };

```

## 9.7 Lines Intersection

```

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;
76 }

```

## 10 other

### 10.1 BigNum

```

1  #define ll long long
2
3  const int size = 1000;
4  const int carrySys = 10;
5
6
7  struct BigNum{
8
9      int len;
10     int bgNum[size];
11     bool sign;
12
13     void reset(){
14
15         len = 1;
16         memset(bgNum, 0, sizeof(bgNum));
17     }

```

```

18
19
20 BigNum add(const BigNum lhs, const BigNum rhs){
21
22     BigNum sum;
23     sum.reset();
24
25     int l = std::max(rhs.len, lhs.len);
26
27     for (int i = 0; i < l; i++)
28     {
29         sum.bgNum[i] += lhs.bgNum[i] + rhs.bgNum[i];
30         if (sum.bgNum[i] >= carrySys)
31         {
32             sum.bgNum[i + 1]++;
33             sum.bgNum[i] -= carrySys;
34         }
35     }
36     if (sum.bgNum[l])
37         l++;
38     sum.len = l;
39
40     if (!lhs.sign && !rhs.sign)
41         sum.sign = false;
42     else
43         sum.sign = true;
44
45     return sum;
46 }
47
48
49 BigNum sub(const BigNum lhs, const BigNum rhs, bool
50 ){
51
52     BigNum ans;
53     ans.reset();
54
55     int l = max(rhs.len, lhs.len);
56     int tmp[size];
57     memset(tmp, 0, sizeof(tmp));
58     copy(lhs.bgNum, lhs.bgNum + lhs.len, tmp);
59     for (int i = 0; i < l; i++)
60     {
61         if (tmp[i] < rhs.bgNum[i] && i != l - 1)
62         {
63             tmp[i + 1] -= 1;
64             tmp[i] += carrySys;
65         }
66         ans.bgNum[i] = tmp[i] - rhs.bgNum[i];
67     }
68     if (ans.bgNum[l - 1] < 0)
69     {
70         ans.bgNum[l - 1] = abs(ans.bgNum[l - 1]);
71         ans.sign = false;
72     }
73     else
74         ans.sign = true;
75
76     ans.len = l;
77
78     while (ans.len > 1 && !ans.bgNum[ans.len - 1])
79     {
80         ans.len--;
81     }
82
83     ans.sign = s;
84
85     return ans;
86 }
87
88 void intToBigNum(ll x){
89
90
91
92
93

```

```

94     if(x < 0){
95         sign = false;
96         x *= -1;
97     }
98     else
99         sign = true;
100
101     reset();
102     if(x == 0)
103         return;
104
105     len = 0;
106     while(x){
107         bgNum[len++] = x % 10;
108         x /= 10;
109     }
110 }
111 void strToBigNum(char x[]){
112
113     reset();
114     len = strlen(x);
115     int l = 0;
116     int a = -1;
117     if(x[0] == '-'){
118         sign = false;
119         a++;
120     }
121     else{
122         sign = true;
123     }
124
125     for(int i = len-1 ; i > a ; i--){
126         bgNum[l++] = x[i] - '0';
127     }
128     if(!sign){
129         len--;
130     }
131 }
132
133 void strToBigNum(string x){
134
135     reset();
136     if(x[0] == '-')
137         sign = false;
138     else
139         sign = true;
140
141     reverse(x.begin(), x.end());
142     len = x.size();
143     if(!sign)
144         len--;
145
146     for(int i = 0 ; i < len ; i++){
147         bgNum[i] = x[i] - '0';
148     }
149 }
150
151 BigNum operator+(const BigNum &rhs){
152
153     BigNum a = *this;
154     BigNum b = rhs;
155
156     if(sign && rhs.sign)
157         return add(*this, rhs);
158     else if(!sign && rhs.sign){
159         a.sign = true;
160         return (a > b ? sub(a, b, false) : sub(b, a, true
161 ));
162

```

```

170     }
171     else if(sign && !rhs.sign){
172         b.sign = true;
173         return (a > b ? sub(a, b, true) : sub(b, a, false));
174     }
175 }
176 else if(sign && !rhs.sign)
177     return add(*this, rhs);
178 }
179
180 BigNum operator-(const BigNum &rhs){
181     BigNum a = *this;
182     BigNum b = rhs;
183
184     if(sign && rhs.sign)
185         return ((*this >= rhs) ? sub(*this, rhs, true) : sub(rhs, *this, false));
186     else if(!sign && rhs.sign){
187         b.sign = false;
188         return add(a, b);
189     }
190     else if(sign && !rhs.sign){
191         b.sign = true;
192         return add(a, b);
193     }
194     else{
195         a.sign = true;
196         b.sign = true;
197         if(a > b){
198             return sub(a, b, false);
199         }
200         else{
201             return sub(b, a, true);
202         }
203     }
204 }
205
206 // BigNum operator * (const BigNum &rhs){
207 //     cout << "mul" << endl;
208 //     BigNum ans;
209 //     ans.reset();
210 //     for(int i = 0 ; i < Len ; i++){
211 //         for(int j = 0 ; j < rhs.Len ; j++){
212 //             int l = i + j;
213 //             ans.bgNum[l] += bgNum[i] * rhs.bgNum[j];
214 //             while(ans.bgNum[l] >= carrySys){
215 //                 ans.bgNum[l+1] += ans.bgNum[l] / carrySys;
216 //                 ans.bgNum[l] = ans.bgNum[l] % carrySys;
217 //             }
218 //         }
219 //     }
220 //     ans.Len = Len + rhs.Len;
221 //     if(!ans.bgNum[ans.Len-1]){
222 //         ans.Len--;
223 //     }
224 //     return ans;
225 // }
226
227 friend bool operator < (const BigNum &lhs, const BigNum &rhs){
228     // cout << lhs.Len << rhs.Len << endl;
229 }
230
231 if(lhs.sign < rhs.sign)
232     return true;
233 else if(lhs.sign > rhs.sign)
234     return false;
235 else{
236     if(lhs.len < rhs.len)
237         return true;
238     else if(lhs.len == rhs.len){
239         for(int i = 0 ; i < lhs.len ; i++){
240             if(lhs.bgNum[i] < rhs.bgNum[i])
241                 return true;
242             return false;
243         }
244     }
245     else
246         return false;
247 }
248
249 friend bool operator > (const BigNum &lhs, const BigNum &rhs){
250     if(lhs.sign > rhs.sign)
251         return true;
252     else if(lhs.sign < rhs.sign)
253         return false;
254     else{
255         if (lhs.len > rhs.len)
256             return true;
257         else if (lhs.len == rhs.len){
258             for (int i = 0; i < lhs.len; i++){
259                 if (lhs.bgNum[i] > rhs.bgNum[i])
260                     return true;
261             }
262             return false;
263         }
264         else
265             return false;
266     }
267 }
268
269 friend bool operator >= (const BigNum &lhs, const BigNum &rhs){
270     return !(lhs < rhs);
271 }
272
273 friend bool operator <= (const BigNum &lhs, const BigNum &rhs){
274     return !(lhs > rhs);
275 }
276
277 BigNum operator = (const BigNum &rhs){
278     len = rhs.len;
279     copy(rhs.bgNum, rhs.bgNum+rhs.len, bgNum);
280     sign = rhs.sign;
281 }
282
283 friend ostream& operator<<(ostream &out, const BigNum &num){
284     if(!num.sign){
285         cout << "-";
286     }
287     out << num.bgNum[num.len-1];
288     for(int i = num.len-2 ; i >= 0 ; i--){
289         out << num.bgNum[i];

```

```

317     return out;
318 }
319
320 BigNum(){ reset(); }
321 BigNum(int x){ reset(); intToBigNum(x); }
322 BigNum(ll x){ reset(); intToBigNum(x); }
323 BigNum(string x){
324
325     reset();
326     strToBigNum(x);
327 }
328 BigNum(char x[]){
329
330     reset();
331     strToBigNum(x);
332 }
333 };
334
335 int main(){
336
337     #ifdef DBG
338     freopen("1.in", "r", stdin);
339     freopen("2.out", "w", stdout);
340     #endif // DEBUG
341
342     // char a[] = "12345";
343     // char b[] = "-2345";
344
345     // BigNum x = a;
346     // BigNum y = b;
347
348     // cout << x << " " << y << endl;
349
350     string a, b;
351
352     while(cin >> a >> b){
353
354         BigNum x, y;
355         // cout << "aaa: ";
356         x = a;
357         y = b;
358
359         BigNum z = x - y;
360         cout << z << endl;
361         // cout << z.len << endl;
362     }
363
364     return 0;
365 }

```

## 10.2 DP + Dijkstra

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int maxn = 1000+10;
4 int n, m, u, v, d, c, s, e, Q;
5 int cost[maxn];
6 int weight[maxn][maxn];
7 vector<int> adj[maxn];
8 int dp[maxn][100+10];
9 struct Car
10 {
11     int cur, left, costSum;
12     bool operator<(const Car &other) const{
13         return costSum > other.costSum;
14     }
15 };
16 int dij()
17 {
18     if(s == e) return 0;
19     for(int i = 0; i < n; i++)
20         for(int j = 0; j <= c; j++)
21             dp[i][j] = 1e9;

```

```

22     priority_queue<Car> pq;
23     pq.push(Car{s, 0, 0});
24     while(!pq.empty()){
25         auto top = pq.top(); pq.pop();
26         if(dp[top.cur][top.left] < top.costSum) continue;
27         if(top.cur == e && top.left == 0) return dp[top.cur][top.left];
28         for(auto i : adj[top.cur]){
29             if(c < weight[top.cur][i]) continue;
30             for(int j = top.left; j <= c; j++){
31                 if(j < weight[top.cur][i]) continue;
32                 int OilCost = cost[top.cur] * (j - top.left);
33                 int temp = j - weight[top.cur][i];
34                 if(dp[i][temp] > top.costSum + OilCost){
35                     dp[i][temp] = top.costSum + OilCost;
36                     pq.push(Car{i, j - weight[top.cur][i], dp[i][temp]});
37                 }
38             }
39         }
40     }
41     return 1e9;
42 }
43
44 int main(int argc, char const *argv[])
45 {
46     while(scanf("%d%d", &n, &m) != EOF){
47         for(int i = 0; i < n; i++){
48             adj[i].clear();
49             scanf("%d", &cost[i]);
50         }
51         for(int i = 0; i < m; i++){
52             scanf("%d%d%d", &u, &v, &d);
53             adj[u].push_back(v);
54             adj[v].push_back(u);
55             weight[u][v] = d;
56             weight[v][u] = d;
57         }
58         scanf("%d", &Q);
59         while(Q--){
60             scanf("%d%d%d", &c, &s, &e);
61             int res = dij();
62             if(res == 1e9) printf("impossible\n");
63             else printf("%d\n", res);
64         }
65     }
66     return 0;
67 }

```

## 10.3 MST + Enumeration

## 10.4 Binary Search Example

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int n, gift[100000+10], L, R;
4 int leftn[100000+10], rightn[100000+10];
5 bool ok(int src)
6 {
7     int l = gift[0], r = src - gift[0];
8     leftn[0] = gift[0], rightn[0] = 0;
9     for(int i = 1; i < n; i++)
10     {
11         if(i % 2 == 1){
12             leftn[i] = min(l - leftn[i-1], gift[i]);
13             rightn[i] = gift[i] - leftn[i];
14         }
15         else{
16             rightn[i] = min(r - rightn[i-1], gift[i]);
17             leftn[i] = gift[i] - rightn[i];
18         }
19     }
20     return leftn[n-1] == 0;
21 }

```

```

22 int main(int argc, char const *argv[])
23 {
24     while(cin >> n && n){
25         L = R = 0;
26         for(int i = 0; i < n; i++){
27             cin >> gift[i];
28         }
29
30         if(n == 1) {
31             printf("%d\n", gift[0]);
32             continue;
33         }
34
35         gift[n] = gift[0];
36         for(int i = 1; i <= n; i++){
37             L = max(L, gift[i-1] + gift[i]);
38         }
39         if(n % 2 == 0){
40             printf("%d\n", L);
41         }
42     }
43     else{
44         for(int i = 0; i < n; i++){
45             R = max(R, 3 * gift[i]);
46         }
47         while(L < R){
48             int M = L + (R - L) / 2;
49             if(ok(M)) R = M;
50             else L = M + 1;
51         }
52         printf("%d\n", R);
53     }
54 }
55 return 0;
56 }

```

## 10.5 Binary Search Example

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int n, gift[100000+10], L, R;
4 int leftn[100000+10], rightn[100000+10];
5 bool ok(int src)
6 {
7     int l = gift[0], r = src - gift[0];
8     leftn[0] = gift[0], rightn[0] = 0;
9     for(int i = 1; i < n; i++)
10     {
11         if(i % 2 == 1){
12             leftn[i] = min(l - leftn[i-1], gift[i]);
13             rightn[i] = gift[i] - leftn[i];
14         }
15         else{
16             rightn[i] = min(r - rightn[i-1], gift[i]);
17             leftn[i] = gift[i] - rightn[i];
18         }
19     }
20     return leftn[n-1] == 0;
21 }
22 int main(int argc, char const *argv[])
23 {
24     while(cin >> n && n){
25         L = R = 0;
26         for(int i = 0; i < n; i++){
27             cin >> gift[i];
28         }
29
30         if(n == 1) {
31             printf("%d\n", gift[0]);
32             continue;
33         }
34
35         gift[n] = gift[0];
36         for(int i = 1; i <= n; i++){
37             L = max(L, gift[i-1] + gift[i]);

```

```

39         }
40         if(n % 2 == 0){
41             printf("%d\n", L);
42         }
43         else{
44             for(int i = 0; i < n; i++){
45                 R = max(R, 3 * gift[i]);
46             }
47             while(L < R){
48                 int M = L + (R - L) / 2;
49                 if(ok(M)) R = M;
50                 else L = M + 1;
51             }
52             printf("%d\n", R);
53         }
54     }
55     return 0;
56 }

```

## 10.6 BFS Number Transform

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 struct node
4 {
5     int val, layer;
6 };
7 int visit[1000+5];
8 int isPrime[1000];
9
10 node BFS(int src, int des)
11 {
12     queue<node> q;
13     q.push(node{src, 0});
14     memset(visit, 0, sizeof(visit));
15     visit[src] = 0;
16     while(!q.empty()){
17         auto front = q.front();
18         if(front.val == des) break;
19         q.pop();
20         if(front.val > des) continue;
21         for(int i = 2; i < front.val; i++){
22             if(isPrime[i] && front.val % i == 0 && visit[
23                 front.val + i] <= front.layer) {
24                 q.push(node{front.val + i, front.layer + 1});
25                 visit[front.val + i] = front.layer + 1;
26             }
27         }
28         if(!q.empty()) return q.front();
29         else return node{-1, -1};
30     }
31 }
32 int main(int argc, char const *argv[])
33 {
34     int s, t, ctr = 0;
35     for(int i = 2; i <= 1000; i++){
36         bool ok = true;
37         for(int j = 2; j <= sqrt(i); j++){
38             if(i % j == 0) ok = false;
39         }
40         if(ok) isPrime[i] = true;
41         else isPrime[i] = false;
42     }
43     int cas = 1;
44     while(scanf("%d%d", &s, &t))
45     {
46         if(s == 0 && t == 0) break;
47         node ans = BFS(s, t);
48         printf("Case %d: %d\n", cas++, ans.layer);
49     }
50     return 0;
51 }

```