

Contents

1	ubuntu	1
1.1	run	1
1.2	cp.sh	1
2	字串	1
2.1	最長迴文子字串	1
3	STL	1
3.1	BIT	1
3.2	deque	2
3.3	map	2
3.4	unordered_map	2
3.5	set	2
3.6	multiset	2
3.7	unordered_set	3
3.8	單調隊列	3
4	sort	3
4.1	大數排序	3
5	math	3
5.1	質數與因數	3
5.2	快速幂	3
5.3	歐拉函數	4
5.4	atan	4
5.5	大步小步	5
6	algorithm	5
6.1	basic	5
6.2	二分搜	5
6.3	三分搜	6
6.4	prefix sum	6
6.5	差分	6
6.6	greedy	6
6.7	floyd warshall	8
6.8	dinic	8
6.9	SegmentTree	9
6.10	Nim Game	9
6.11	Trie	10
6.12	SPFA	10
6.13	dijkstra	11
6.14	SCC Tarjan	11
6.15	SCC Kosaraju	11
6.16	ArticulationPoints Tarjan	11
6.17	最小樹狀圖	12
7	geometry	14
7.1	intersection	14
7.2	半平面相交	14
7.3	凸包	15
8	動態規劃	16
8.1	LCS 和 LIS	16
9	Section2	16
9.1	thm	16
10	DP	16
10.1	字串 DP	16

1 ubuntu

1.1 run

```
1 | ~$ bash cp.sh PA
```

1.2 cp.sh

```
1 #!/bin/bash
2 clear
3 g++ $1.cpp -DDBG -o $1
4 if [[ "$?" == "0" ]]; then
5     echo Running
6     ./$1 < $1.in > $1.out
7     echo END
8 fi
```

2 字串

2.1 最長迴文子字串

```
1 #include<bits/stdc++.h>
2 #define T(x) ((x)%2 ? s[(x)/2] : '. ')
3 using namespace std;
4
5 string s;
6 int n;
7
8 int ex(int l,int r){
9     int i=0;
10    while(l-i>=0&&r+i<n&&T(l-i)==T(r+i)) i++;
11    return i;
12 }
13
14 int main(){
15     cin>>s;
16     n=2*s.size()+1;
17     int mx=0;
18     int center=0;
19     vector<int> r(n);
20     int ans=1;
21     r[0]=1;
22     for(int i=1;i<n;i++){
23         int ii=center-(i-center);
24         int len=mx-i+1;
25         if(i>mx){
26             r[i]=ex(i,i);
27             center=i;
28             mx=i+r[i]-1;
29         }
30         else if(r[ii]==len){
31             r[i]=len+ex(i-len,i+len);
32             center=i;
33             mx=i+r[i]-1;
34         }
35         else r[i]=min(r[ii],len);
36         ans=max(ans,r[i]);
37     }
38     cout<<ans-1<<"\n";
39     return 0;
40 }
```

3 STL

3.1 BIT

```
1 template <class T> class BIT {
2 private:
3     int size;
4     vector<T> bit;
5     vector<T> arr;
6
7 public:
8     BIT(int sz=0): size(sz), bit(sz+1), arr(sz) {}
9
10    /** Sets the value at index idx to val. */
11    void set(int idx, T val) {
12        add(idx, val - arr[idx]);
13    }
14
15    /** Adds val to the element at index idx. */
16    void add(int idx, T val) {
17        arr[idx] += val;
18        for (++idx; idx<=size; idx+=(idx & -idx))
19            bit[idx] += val;
20    }
21
22    /** @return The sum of all values in [0, idx]. */
23    T pre_sum(int idx) {
24        T total = 0;
```

```

25     for (++idx; idx>0; idx--=(idx & -idx))
26         total += bit[idx];
27     return total;
28 }
29 };

```

3.2 deque

```

1 deque 是 C++ 標準模板函式庫
2   (Standard Template Library, STL)
3   中的雙向佇列容器 (Double-ended Queue) ,
4   跟 vector 相似, 不過在 vector
5   中若是要添加新元素至開端,
6   其時間複雜度為 O(N), 但在 deque 中則是 O(1)。
7   同樣也能在我們需要儲存更多元素的時候自動擴展空間,
8   讓我們不必煩惱佇列長度的問題。
9
10 dq.push_back() //在 deque 的最尾端新增元素
11 dq.push_front() //在 deque 的開頭新增元素
12 dq.pop_back() //移除 deque 最尾端的元素
13 dq.pop_front() //移除 deque 最開頭的元素
14 dq.back() //取出 deque 最尾端的元素
15 dq.front() //回傳 deque 最開頭的元素
16 dq.insert()
17 dq.insert(position, n, val)
18   position: 插入元素的 index 值
19   n: 元素插入次數
20   val: 插入的元素值
21 dq.erase()
22   //刪除元素, 需要使用迭代器指定刪除的元素或位置,
23   //同時也會返回指向刪除元素下一元素的迭代器
24 dq.clear() //清空整個 deque 佇列。
25 dq.size() //檢查 deque 的尺寸
26 dq.empty() //如果 deque 佇列為空返回 1;
27   //若是存在任何元素, 則返回 0
28 dq.begin() //返回一個指向 deque 開頭的迭代器
29 dq.end() //指向 deque 結尾,
30   //不是最後一個元素,
31   //而是最後一個元素的下一個位置

```

3.3 map

```

1 map: 存放 key-value pairs 的映射資料結構,
2   會按 key 由小到大排序。
3 元素存取
4 operator[]: 存取指定的[i]元素的資料
5
6 迭代器
7 begin(): 回傳指向map頭部元素的迭代器
8 end(): 回傳指向map末尾的迭代器
9 rbegin(): 回傳一個指向map尾部的反向迭代器
10 rend(): 回傳一個指向map頭部的反向迭代器
11
12 遍歷整個map時, 利用iterator操作:
13 取key: it->first 或 (*it).first
14 取value: it->second 或 (*it).second
15
16 容量
17 empty(): 檢查容器是否為空, 空則回傳true
18 size(): 回傳元素數量
19 max_size(): 回傳可以容納的最大元素個數
20
21 修改器
22 clear(): 刪除所有元素
23 insert(): 插入元素
24 erase(): 刪除一個元素
25 swap(): 交換兩個map
26
27 查找

```

```

28 count(): 回傳指定元素出現的次數
29 find(): 查找一個元素
30
31 //實作範例
32 #include <bits/stdc++.h>
33 using namespace std;
34 int main(){
35     //declaration container and iterator
36     map<string, string> mp;
37     map<string, string>::iterator iter;
38     map<string, string>::reverse_iterator iter_r;
39
40     //insert element
41     mp.insert(pair<string, string>
42         ("r000", "student_zero"));
43     mp["r123"] = "student_first";
44     mp["r456"] = "student_second";
45
46     //traversal
47     for(iter=mp.begin(); iter!=mp.end(); iter++)
48         cout<<iter->first<<" "
49             <<iter->second<<endl;
50     for(iter_r=mp.rbegin(); iter_r!=mp.rend(); iter_r++)
51         cout<<iter_r->first<<"
52             <<iter_r->second<<endl;
53
54     //find and erase the element
55     iter=mp.find("r123");
56     mp.erase(iter);
57     iter=mp.find("r123");
58     if(iter!=mp.end())
59         cout<<"Find, the value is "
60             <<iter->second<<endl;
61     else cout<<"Do not Find"<<endl;
62     return 0;
63 }

```

3.4 unordered_map

```

1 unordered_map: 存放 key-value pairs
2   的「無序」映射資料結構。
3 用法與map相同

```

3.5 set

```

1 set: 集合, 去除重複的元素, 資料由小到大排序。
2
3 取值: 使用iterator
4   x = *st.begin();
5   // set中的第一個元素(最小的元素)。
6   x = *st.rbegin();
7   // set中的最後一個元素(最大的元素)。
8
9 判斷是否為空的set:
10   st.empty() 回傳true
11   st.size() 回傳零
12
13 常用來搭配的member function:
14   st.count(x);
15   auto it = st.find(x);
16   // binary search, O(log(N))
17   auto it = st.lower_bound(x);
18   // binary search, O(log(N))
19   auto it = st.upper_bound(x);
20   // binary search, O(log(N))

```

3.6 multiset

```

1 與 set 用法雷同, 但會保留重複的元素。
2 資料由小到大排序。

```

```

3 宣告：
4      multiset<int> st;
5 刪除資料：
6      st.erase(val);
7      //會刪除所有值為 val 的元素。
8      st.erase(st.find(val));
9      //只刪除第一個值為 val 的元素。

```

3.7 unordered_set

unordered_set 的實作方式通常是用雜湊表(hash table)，資料插入和查詢的時間複雜度很低，為常數級別 $O(1)$ ，相對的代價是消耗較多的記憶體，空間複雜度較高，無自動排序功能。

```

6 unordered_set 判斷元素是否存在
7 unordered_set<int> myunordered_set;
8 myunordered_set.insert(2);
9 myunordered_set.insert(4);
10 myunordered_set.insert(6);
11 cout << myunordered_set.count(4) << "\n"; // 1
12 cout << myunordered_set.count(8) << "\n"; // 0

```

3.8 單調隊列

```

1 //單調隊列
2 "如果一個選手比你小還比你強，你就可以退役了。"--單調隊列
3
4 example
5
6 給出一個長度為 n 的數組，
7 輸出每 k 個連續的數中的最大值和最小值。
8
9 #include <bits/stdc++.h>
10 #define maxn 1000100
11 using namespace std;
12 int q[maxn], a[maxn];
13 int n, k;
14
15 void getmin() {
16     // 得到這個隊列裡的最小值，直接找到最後的就行了
17     int head=0, tail=0;
18     for(int i=1; i<k; i++) {
19         while(head<=tail && a[q[tail]]>=a[i]) tail--;
20         q[++tail]=i;
21     }
22     for(int i=k; i<=n; i++) {
23         while(head<=tail && a[q[tail]]>=a[i]) tail--;
24         q[++tail]=i;
25         while(q[head]<=i-k) head++;
26         cout<<a[q[head]]<<" ";
27     }
28     cout<<endl;
29 }
30
31 void getmax() { // 和上面同理
32     int head=0, tail=0;
33     for(int i=1; i<k; i++) {
34         while(head<=tail && a[q[tail]]<=a[i]) tail--;
35         q[++tail]=i;
36     }
37     for(int i=k; i<=n; i++) {
38         while(head<=tail && a[q[tail]]<=a[i]) tail--;
39         q[++tail]=i;
40         while(q[head]<=i-k) head++;
41         cout<<a[q[head]]<<" ";
42     }
43     cout<<endl;
44 }
45
46 int main(){
47     cin>>n>>k; //每k個連續的數

```

```

48     for(int i=1; i<=n; i++) cin>>a[i];
49     getmin();
50     getmax();
51     return 0;
52 }

```

4 sort

4.1 大數排序

```

1 #python大數排序
2
3 while True:
4     try:
5         n = int(input())          # 有幾筆數字需要排序
6         arr = []                  # 建立空串列
7         for i in range(n):
8             arr.append(int(input())) # 依序將數字存入串列
9         arr.sort()                 # 串列排序
10        for i in arr:
11            print(i)                # 依序印出串列中每個項目
12    except:
13        break

```

5 math

5.1 質數與因數

```

1 埃氏篩法
2 int n;
3 vector<int> isprime(n+1,1);
4 isprime[0]=isprime[1]=0;
5 for(int i=2; i*i<=n; i++){
6     if(isprime[i])
7         for(int j=i*i; j<=n; j+=i) isprime[j]=0;
8 }
9
10 歐拉篩O(n)
11 #define MAXN 47000 //sqrt(2^31)=46,340...
12 bool isPrime[MAXN];
13 int prime[MAXN];
14 int primeSize=0;
15 void getPrimes(){
16     memset(isPrime, true, sizeof(isPrime));
17     isPrime[0]=isPrime[1]=false;
18     for(int i=2; i<MAXN; i++){
19         if(isPrime[i]) prime[primeSize++]=i;
20         for(int
21             j=0; j<primeSize && i*prime[j]<=MAXN; ++j){
22             isPrime[i*prime[j]]=false;
23             if(i%prime[j]==0) break;
24         }
25     }
26
27 最大公因數 O(log(min(a,b)))
28 int GCD(int a, int b){
29     if(b==0) return a;
30     return GCD(b, a%b);
31 }
32
33 質因數分解
34 void primeFactorization(int n){
35     for(int i=0; i<(int)p.size(); ++i){
36         if(p[i]*p[i]>n) break;
37         if(n%p[i]) continue;
38         cout<<p[i]<<" ";
39         while(n%p[i]==0) n/=p[i];
40     }
41     if(n!=1) cout<<n<<" ";

```

```

42     cout<<'\n';
43 }
44
45 擴展歐幾里得算法
46 //ax+by=GCD(a,b)
47 #include <bits/stdc++.h>
48 using namespace std;
49
50 int ext_euc(int a,int b,int &x,int &y){
51     if(b==0){
52         x=1,y=0;
53         return a;
54     }
55     int d=ext_euc(b,a%b,y,x);
56     y-=a/b*x;
57     return d;
58 }
59
60 int main(){
61     int a,b,x,y;
62     cin>>a>>b;
63     ext_euc(a,b,x,y);
64     cout<<x<<' '<<y<<endl;
65     return 0;
66 }
67
68
69
70 歌德巴赫猜想
71 solution : 把偶數 N ( $6 \leq N \leq 10^6$ ) 寫成兩個質數的和。
72 #include <iostream>
73 using namespace std;
74 #define N 2000000
75 int ox[N],p[N],pr;
76 void PrimeTable(){
77     ox[0]=ox[1]=1;
78     pr=0;
79     for(int i=2;i<N;i++){
80         if(!ox[i]) p[pr++]=i;
81         for(int j=0;i*p[j]<N&&j<pr;j++)
82             ox[i*p[j]]=1;
83     }
84 }
85
86 int main(){
87     PrimeTable();
88     int n;
89     while(cin>>n,n){
90         int x;
91         for(x=1;x+=2)
92             if(!ox[x]&&!ox[n-x]) break;
93         printf("%d = %d + %d\n",n,x,n-x);
94     }
95 }
96
97 problem : 給定整數 N ,
98 求 N 最少可以拆成多少個質數的和。
99 如果 N 是質數,則答案為 1。
100 如果 N 是偶數(不含2),則答案為 2 (強歌德巴赫猜想)。
101 如果 N 是奇數且 N-2 是質數,則答案為 2 (2+質數)。
102 其他狀況答案為 3 (弱歌德巴赫猜想)。
103 #include<bits/stdc++.h>
104 using namespace std;
105
106 bool isPrime(int n){
107     for(int i=2;i<n;++i){
108         if(i*i>n) return true;
109         if(n%i==0) return false;
110     }
111     return true;
112 }
113
114 int main(){
115     int n;
116     cin>>n;
117     if(isPrime(n)) cout<<"1\n";
118     else if(n%2==0||isPrime(n-2)) cout<<"2\n";

```

```

118     else cout<<"3\n";
119 }

```

5.2 快速幂

```

1 計算 a^b
2 #include<iostream>
3 #define ll long long
4 using namespace std;
5
6 const ll MOD=1000000007;
7 ll fp(ll a, ll b) {
8     int ans=1;
9     while(b>0){
10         if(b&1) ans=ans*a%MOD;
11         a=a*a%MOD;
12         b>>=1;
13     }
14     return ans;
15 }
16
17 int main() {
18     int a,b;
19     cin>>a>>b;
20     cout<<fp(a,b);
21 }

```

5.3 歐拉函數

```

1 //計算閉區間 [1,n] 中的正整數與 n 互質的個數
2
3 int phi(){
4     int ans=n;
5     for(int i=2;i*i<=n;i++){
6         if(n%i==0){
7             ans=ans-ans/i;
8             while(n%i==0) n/=i;
9         }
10        if(n>1) ans=ans-ans/n;
11        return ans;
12 }

```

5.4 atan

```

1 說明
2     atan() 和 atan2() 函數分別計算 x 和 y/x 的反正切。
3
4 回覆值
5     atan() 函數會傳回介於範圍 - /2 到 /2 弧度之間的值。
6     atan2() 函數會傳回介於 - 至 弧度之間的值。
7     如果 atan2() 函數的兩個引數都是零,
8     則函數會將 errno 設為 EDOM, 並傳回值 0。
9
10 範例
11 #include <math.h>
12 #include <stdio.h>
13
14 int main(void){
15     double a,b,c,d;
16
17     c=0.45;
18     d=0.23;
19
20     a=atan(c);
21     b=atan2(c,d);
22
23     printf("atan(%lf)=%lf/n",c,a);
24     printf("atan2(%lf,%lf)=%lf/n",c,d,b);
25 }
26

```

```

27 |
28 | /*
29 | atan(0.450000)=0.422854
30 | atan2(0.450000,0.230000)=1.098299
31 | */

```

5.5 大步小步

```

1 | 題意
2 | 給定 B,N,P，求出 L 滿足  $B^L \equiv N \pmod P$ 。
3 |
4 | 題解
5 | 餘數的循環節長度必定為 P 的因數，因此
   |  $B^0, B^1, B^2, \dots, B^{P-1}$ ，...，
6 | 也就是說如果有解則  $L < N$ ，枚舉 0,1,2,L-1
   | 能得到結果，但會超時。
7 |
8 | 將 L 拆成  $mx+y$ ，只要分別枚舉 x,y 就能得到答案，
9 | 設  $m=\sqrt{P}$  能保證最多枚舉  $2\sqrt{P}$  次。
10 |
11 |  $B^{mx+y} \equiv N \pmod P$ 
12 |  $B^{mx} B^y \equiv N \pmod P$ 
13 |  $B^y \equiv N(B^{-m})^x \pmod P$ 
14 |
15 | 先求出  $B^0, B^1, B^2, \dots, B^{m-1}$ ，
16 | 再枚舉  $N(B^{-m}), N(B^{-m})^2, \dots$  查看是否有對應的  $B^y$ 。
17 | 這種算法稱為大步小步演算法，
18 | 大步指的是枚舉 x（一次跨 m 步），
19 | 小步指的是枚舉 y（一次跨 1 步）。
20 |
21 | 複雜度分析
22 | 利用 map/unorder_map 存放  $B^0, B^1, B^2, \dots, B^{m-1}$ ，
23 | 枚舉 x 查詢 map/unorder_map 是否有對應的  $B^y$ ，
24 | 存放和查詢最多  $2\sqrt{P}$  次，時間複雜度為  $O(\sqrt{P} \log \sqrt{P}) / O(\sqrt{P})$ 。
25 |
26 |
27 |
28 | #include <bits/stdc++.h>
29 | using namespace std;
30 | using LL = long long;
31 | LL B, N, P;
32 |
33 | LL fpow(LL a, LL b, LL c){
34 |     LL res=1;
35 |     for(;b>=1;b>>=1){
36 |         if(b&1)
37 |             res=(res*a)%c;
38 |         a=(a*a)%c;
39 |     }
40 |     return res;
41 | }
42 |
43 | LL BSGS(LL a, LL b, LL p){
44 |     a%=p, b%=p;
45 |     if(a==0)
46 |         return b==0?1:-1;
47 |     if(b==1)
48 |         return 0;
49 |     map<LL, LL> tb;
50 |     LL sq=ceil(sqrt(p-1));
51 |     LL inv=fpow(a, p-sq-1, p);
52 |     tb[1]=sq;
53 |     for(LL i=1, tmp=1; i<sq; ++i){
54 |         tmp=(tmp*a)%p;
55 |         if(!tb.count(tmp))
56 |             tb[tmp]=i;
57 |     }
58 |     for(LL i=0; i<sq; ++i){
59 |         if(tb.count(b)){
60 |             LL res=tb[b];
61 |             return i*sq+(res==sq?0:res);
62 |         }
63 |         b=(b*inv)%p;
64 |     }

```

```

65 |     return -1;
66 | }
67 |
68 | int main(){
69 |     ios::sync_with_stdio(false);
70 |     cin.tie(0), cout.tie(0);
71 |     while(cin>>P>>B>>N){
72 |         LL ans=BSGS(B,N,P);
73 |         if(ans!=-1)
74 |             cout<<"no solution\n";
75 |         else
76 |             cout<<ans<<"\n";
77 |     }
78 | }

```

6 algorithm

6.1 basic

```

1 | min_element：找尋最小元素
2 | min_element(first, last)
3 | max_element：找尋最大元素
4 | max_element(first, last)
5 | sort：排序，預設由小排到大。
6 | sort(first, last)
7 | sort(first, last, cmp)：可自行定義比較運算子 cmp。
8 | find：尋找元素。
9 | find(first, last, val)
10 | lower_bound：尋找第一個小於 x 的元素位置，
   | 如果不存在，則回傳 last。
11 | lower_bound(first, last, val)
12 | upper_bound：尋找第一個大於 x 的元素位置，
   | 如果不存在，則回傳 last。
13 | upper_bound(first, last, val)
14 | next_permutation：將序列順序轉換成下一個字典序，
   | 如果存在回傳 true，反之回傳 false。
15 | next_permutation(first, last)
16 | prev_permutation：將序列順序轉換成上一個字典序，
   | 如果存在回傳 true，反之回傳 false。
17 | prev_permutation(first, last)

```

6.2 二分搜

```

1 | int binary_search(int target) {
2 |     // For range [ok, ng) or (ng, ok], "ok" is for the
3 |     // index that target value exists, with "ng" doesn't.
4 |     int ok = maxn, ng = -1;
5 |     // For first lower_bound, ok=maxn and ng=-1,
6 |     // for last lower_bound, ok = -1 and ng = maxn
7 |     // (the "check" funtion
8 |     // should be changed depending on it.)
9 |     while(abs(ok - ng) > 1) {
10 |         int mid = (ok + ng) >> 1;
11 |         if(check(mid)) ok = mid;
12 |         else ng = mid;
13 |         // Be careful, "arr[mid]>=target" for first
14 |         // lower_bound and "arr[mid]<=target" for
15 |         // last lower_bound. For range (ng, ok],
16 |         // convert it into (ng, mid] and (mid, ok] than
17 |         // choose the first one, or convert [ok, ng) into
18 |         // [ok, mid) and [mid, ng) and then choose
19 |         // the second one.
20 |     }
21 |     return ok;
22 | }
23 |
24 | lower_bound(arr, arr + n, k); //最左邊 ≥ k 的位置
25 | upper_bound(arr, arr + n, k); //最左邊 > k 的位置
26 | upper_bound(arr, arr + n, k) - 1; //最右邊 ≤ k 的位置
27 | lower_bound(arr, arr + n, k) - 1; //最右邊 < k 的位置

```

```

28 | (lower_bound, upper_bound)           //等於 k 的範圍
29 | equal_range(arr, arr+n, k);

```

6.3 三分搜

```

1 | 題意
2 | 給定兩射線方向和速度，問兩射線最近距離。
3 |
4 | 題解
5 | 假設 F(t) 為兩射線在時間 t 的距離，F(t) 為二次函數，
6 | 可用三分查找二次函數最小值。
7 |
8 | #include <bits/stdc++.h>
9 | using namespace std;
10 |
11 | struct Point{
12 |     double x, y, z;
13 |     Point() {}
14 |     Point(double _x, double _y, double _z):
15 |         x(_x), y(_y), z(_z){}
16 |     friend istream& operator>>(istream& is, Point& p)
17 |     {
18 |         is >> p.x >> p.y >> p.z;
19 |         return is;
20 |     }
21 |     Point operator+(const Point &rhs) const{
22 |         return Point(x+rhs.x, y+rhs.y, z+rhs.z);
23 |     }
24 |     Point operator-(const Point &rhs) const{
25 |         return Point(x-rhs.x, y-rhs.y, z-rhs.z);
26 |     }
27 |     Point operator*(const double &d) const{
28 |         return Point(x*d, y*d, z*d);
29 |     }
30 |     Point operator/(const double &d) const{
31 |         return Point(x/d, y/d, z/d);
32 |     }
33 |     double dist(const Point &rhs) const{
34 |         double res = 0;
35 |         res+=(x-rhs.x)*(x-rhs.x);
36 |         res+=(y-rhs.y)*(y-rhs.y);
37 |         res+=(z-rhs.z)*(z-rhs.z);
38 |         return res;
39 |     }
40 | };
41 | int main(){
42 |     ios::sync_with_stdio(false);
43 |     cin.tie(0), cout.tie(0);
44 |     int T;
45 |     cin>>T;
46 |     for(int ti=1; ti<=T; ++ti){
47 |         double time;
48 |         Point x1, y1, d1, x2, y2, d2;
49 |         cin>>time>>x1>>y1>>x2>>y2;
50 |         d1=(y1-x1)/time;
51 |         d2=(y2-x2)/time;
52 |         double L=0, R=1e8, m1, m2, f1, f2;
53 |         double ans = x1.dist(x2);
54 |         while(abs(L-R)>1e-10){
55 |             m1=(L+R)/2;
56 |             m2=(m1+R)/2;
57 |             f1=((d1*m1)+x1).dist((d2*m1)+x2);
58 |             f2=((d1*m2)+x1).dist((d2*m2)+x2);
59 |             ans = min(ans, min(f1, f2));
60 |             if(f1<f2) R=m2;
61 |             else L=m1;
62 |         }
63 |         cout<<"Case "<<ti<<": ";
64 |         cout<<fixed<<setprecision(4)<<sqrt(ans)<<"\n";
65 |     }
66 | }

```

6.4 prefix sum

```

1 | // 前綴和
2 | 陣列前n項的和。
3 | b[i]=a[0]+a[1]+a[2]+ ... +a[i]
4 | 區間和 [l, r]: b[r]-b[l-1] (要保留b[l]所以-1)
5 |
6 | #include<bits/stdc++.h>
7 | using namespace std;
8 | int main(){
9 |     int n;
10 |    cin>>n;
11 |    int a[n], b[n];
12 |    for(int i=0; i<n; i++) cin>>a[i];
13 |    b[0]=a[0];
14 |    for(int i=1; i<n; i++) b[i]=b[i-1]+a[i];
15 |    for(int i=0; i<n; i++) cout<<b[i]<<" ";
16 |    cout<<"\n";
17 |    int l, r;
18 |    cin>>l>>r;
19 |    cout<<b[r]-b[l-1]; //區間和
20 | }

```

6.5 差分

```

1 | // 差分
2 | 用途：在區間 [l, r] 加上一個數字v。
3 | b[l] += v; (b[0~l] 加上v)
4 | b[r+1] -= v; (b[r+1~n] 減去v (b[r] 仍保留v))
5 | 給的 a[] 是前綴和數列，建構 b[]，
6 | 因為 a[i] = b[0] + b[1] + b[2] + ... + b[i]，
7 | 所以 b[i] = a[i] - a[i-1]。
8 | 在 b[l] 加上 v，b[r+1] 減去 v，
9 | 最後再從 0 跑到 n 使 b[i] += b[i-1]。
10 | 這樣一來，b[] 是一個在某區間加上v的前綴和。
11 |
12 | #include <bits/stdc++.h>
13 | using namespace std;
14 | int a[1000], b[1000];
15 | // a: 前綴和數列, b: 差分數列
16 | int main(){
17 |     int n, l, r, v;
18 |     cin >> n;
19 |     for(int i=1; i<=n; i++){
20 |         cin >> a[i];
21 |         b[i] = a[i] - a[i-1]; //建構差分數列
22 |     }
23 |     cin >> l >> r >> v;
24 |     b[l] += v;
25 |     b[r+1] -= v;
26 |
27 |     for(int i=1; i<=n; i++){
28 |         b[i] += b[i-1];
29 |         cout << b[i] << " ";
30 |     }
31 | }

```

6.6 greedy

```

1 | //貪心
2 | 貪心演算法的核心為，
3 | 採取在目前狀態下最好或最佳（即最有利）的選擇。
4 | 貪心演算法雖然能獲得當前最佳解，
5 | 但不保證能獲得最後（全域）最佳解，
6 | 提出想法後可以先試圖尋找有沒有能推翻原本的想法的反例，
7 | 確認無誤再實作。
8 |
9 |
10 | 刪數字問題
11 | //problem
12 | 給定一個數字 N(≤10^100)，需要刪除 K 個數字，

```


13 請問刪除 K 個數字後最小的數字為何?

14 //solution

15 刪除滿足第 i 位數大於第 i+1 位數的最左邊第 i 位數，
16 扣除高位數的影響較扣除低位數的大。

17 //code

```
20 int main(){
21     string s;
22     int k;
23     cin>>s>>k;
24     for(int i=0;i<k;++i){
25         if((int)s.size()==0) break;
26         int pos =(int)s.size()-1;
27         for(int j=0;j<(int)s.size()-1;++j){
28             if(s[j]>s[j+1]){
29                 pos=j;
30                 break;
31             }
32         }
33         s.erase(pos,1);
34     }
35     while((int)s.size()>0&&s[0]=='0')
36         s.erase(0,1);
37     if((int)s.size()) cout<<s<<'\n';
38     else cout<<0<<'\n';
39 }
```

42 最小區間覆蓋長度

43 //problem

44 給定 n 條線段區間為 [Li,Ri]，
45 請問最少要選幾個區間才能完全覆蓋 [0,S]?

46 //solution

47 先將所有區間依照左界由小到大排序，
48 對於當前區間 [Li,Ri]，要從左界 >Ri 的所有區間中，
49 找到有著最大的右界的區間，連接當前區間。

50 //problem

51 長度 n 的直線中有數個加熱器，
52 在 x 的加熱器可以讓 [x-r,x+r] 內的物品加熱，
53 問最少要幾個加熱器可以把 [0,n] 的範圍加熱。

54 //solution

55 對於最左邊沒加熱的點a，選擇最遠可以加熱a的加熱器，
56 更新已加熱範圍，重複上述動作繼續尋找加熱器。

57 //code

```
60 int main(){
61     int n, r;
62     int a[1005];
63     cin>>n>>r;
64     for(int i=1;i<=n;++i) cin>>a[i];
65     int i=1,ans=0;
66     while(i<=n){
67         int R=min(i+r-1,n),L=max(i-r+1,0)
68         int nextR=-1;
69         for(int j=R;j>=L;--j){
70             if(a[j]){
71                 nextR=j;
72                 break;
73             }
74         }
75         if(nextR==-1){
76             ans=-1;
77             break;
78         }
79         ++ans;
80         i=nextR+r;
81     }
82     cout<<ans<<'\n';
83 }
84 }
```

85 最多不重疊區間

86 //problem

87 給你 n 條線段區間為 [Li,Ri]，
88 請問最多可以選擇幾條不重疊的線段(頭尾可相連)?

89 //solution

90 依照右界由小到大排序，
91 每次取到一個不重疊的線段，答案 +1。

92 //code

```
93 struct Line{
94     int L,R;
95     bool operator<(const Line &rhs)const{
96         return R<rhs.R;
97     }
98 };
99 int main(){
100     int t;
101     cin>>t;
102     Line a[30];
103     while(t--){
104         int n=0;
105         while(cin>>a[n].L>>a[n].R,a[n].L||a[n].R)
106             ++n;
107         sort(a,a+n);
108         int ans=1,R=a[0].R;
109         for(int i=1;i<n;i++){
110             if(a[i].L>=R){
111                 ++ans;
112                 R=a[i].R;
113             }
114         }
115         cout<<ans<<'\n';
116     }
117 }
```

124 最小化最大延遲問題

125 //problem

126 給定 N 項工作，每項工作的需要處理時長為 Ti，
127 期限是 Di，第 i 項工作延遲的時間為 Li=max(0,Fi-Di)，
128 原本Fi 為第 i 項工作的完成時間，
129 求一種工作排序使 maxLi 最小。

130 //solution

131 按照到期時間從早到晚處理。

132 //code

```
133 struct Work{
134     int t, d;
135     bool operator<(const Work &rhs)const{
136         return d<rhs.d;
137     }
138 };
139 int main(){
140     int n;
141     Work a[10000];
142     cin>>n;
143     for(int i=0;i<n;++i)
144         cin>>a[i].t>>a[i].d;
145     sort(a,a+n);
146     int maxL=0,sumT=0;
147     for(int i=0;i<n;++i){
148         sumT+=a[i].t;
149         maxL=max(maxL,sumT-a[i].d);
150     }
151     cout<<maxL<<'\n';
152 }
```

153 最少延遲數量問題

154 //problem

155 給定 N 個工作，每個工作的需要處理時長為 Ti，
156 期限是 Di，求一種工作排序使得逾期工作數量最小。

```

165 //solution
166 期限越早到期的工作越先做。將工作依照到期時間從早到晚排序
167 依序放入工作列表中，如果發現有工作預期，
168 就從目前選擇的工作中，移除耗時最長的工作。
169
170 上述方法為 Moore-Hodgson's Algorithm。
171
172 //problem
173 給定烏龜的重量和可承受重量，問最多可以疊幾隻烏龜？
174
175 //solution
176 和最少延遲數量問題是相同的問題，只要將題敘做轉換。
177 工作處理時長 → 烏龜重量
178 工作期限 → 烏龜可承受重量
179 多少工作不延期 → 可以疊幾隻烏龜
180
181 //code
182 struct Work{
183     int t, d;
184     bool operator<(const Work &rhs)const{
185         return d<rhs.d;
186     }
187 };
188
189 int main(){
190     int n=0;
191     Work a[10000];
192     priority_queue<int> pq;
193     while(cin>>a[n].t>>a[n].d)
194         ++n;
195     sort(a,a+n);
196     int sumT=0,ans=n;
197     for(int i=0;i<n;++i){
198         pq.push(a[i].t);
199         sumT+=a[i].t;
200         if(a[i].d<sumT){
201             int x=pq.top();
202             pq.pop();
203             sumT-=x;
204             --ans;
205         }
206     }
207     cout<<ans<<'\n';
208 }
209
210 任務調度問題
211 //problem
212 給定 N 項工作，每項工作的需要處理時長為  $T_i$ ，
213 期限是  $D_i$ ，如果第  $i$  項工作延遲需要受到  $p_i$  單位懲罰，
214 請問最少會受到多少單位懲罰。
215
216 //solution
217 依照懲罰由大到小排序，
218 每項工作依序嘗試可不可以放在  $D_i - T_i + 1, D_i - T_i, \dots, 1, 0$ ，
219 如果有空間就放進去，否則延後執行。
220
221 //problem
222 給定 N 項工作，每項工作的需要處理時長為  $T_i$ ，
223 期限是  $D_i$ ，如果第  $i$  項工作在期限內完成會獲得  $a_i$ 
    單位獎勵，
224 請問最多會獲得多少單位獎勵。
225
226 //solution
227 和上題相似，這題變成依照獎勵由大到小排序。
228
229 //code
230 struct Work{
231     int d,p;
232     bool operator<(const Work &rhs)const{
233         return p>rhs.p;
234     }
235 };
236
237 int main(){
238     int n;

```

```

239     Work a[100005];
240     bitset<100005> ok;
241     while(cin>>n){
242         ok.reset();
243         for(int i=0;i<n;++i)
244             cin>>a[i].d>>a[i].p;
245         sort(a,a+n);
246         int ans=0;
247         for(int i=0;i<n;++i){
248             int j=a[i].d;
249             while(j--){
250                 if(!ok[j]){
251                     ans+=a[i].p;
252                     ok[j]=true;
253                     break;
254                 }
255             }
256             cout<<ans<<'\n';
257         }
258     }

```

6.7 floyd warshall

```

1 int w[n][n];
2 int d[n][n];
3 int p[n][n];
4 // 由i點到j點的路徑，其中繼點為 p[i][j]。
5
6 void floyd_warshall(){ //O(V^3)
7     for(int i=0;i<n;i++){
8         for(int j=0;j<n;j++){
9             d[i][j]=w[i][j];
10            p[i][j]=-1; // 預設為沒有中繼點
11        }
12    }
13    for(int i=0;i<n;i++) d[i][i]=0;
14    for(int k=0;k<n;k++){
15        for(int i=0;i<n;i++){
16            for(int j=0;j<n;j++){
17                if(d[i][k]+d[k][j]<d[i][j]){
18                    d[i][j]=d[i][k]+d[k][j];
19                    p[i][j]=k; // 由i點走到j點經過了k點
20                }
21            }
22        }
23    }
24    // 這支函式並不會印出起點和終點，必須另行印出。
25    void find_path(int s,int t){ // 印出最短路徑
26        if(p[s][t]==-1) return; // 沒有中繼點就結束
27        find_path(s,p[s][t]); // 前半段最短路徑
28        cout<<p[s][t]; // 中繼點
29        find_path(p[s][t],t); // 後半段最短路徑
30    }

```

6.8 dinic

```

1 const int maxn = 1e5 + 10;
2 const int inf = 0x3f3f3f3f;
3
4 struct Edge {
5     int s, t, cap, flow;
6 };
7
8 int n, m, S, T;
9 int level[maxn], dfs_idx[maxn];
10 vector<Edge> E;
11 vector<vector<int>> G;
12
13 void init() {
14     S = 0;
15     T = n + m;
16     E.clear();
17     G.assign(maxn, vector<int>());
18 }

```



```

19
20 void addEdge(int s, int t, int cap) {
21     E.push_back({s, t, cap, 0});
22     E.push_back({t, s, 0, 0});
23     G[s].push_back(E.size()-2);
24     G[t].push_back(E.size()-1);
25 }
26
27 bool bfs() {
28     queue<int> q({S});
29
30     memset(level, -1, sizeof(level));
31     level[S] = 0;
32
33     while(!q.empty()) {
34         int cur = q.front();
35         q.pop();
36
37         for(int i : G[cur]) {
38             Edge e = E[i];
39             if(level[e.t]==-1 && e.cap>e.flow) {
40                 level[e.t] = level[e.s] + 1;
41                 q.push(e.t);
42             }
43         }
44     }
45     return ~level[T];
46 }
47
48 int dfs(int cur, int lim) {
49     if(cur==T || lim==0) return lim;
50
51     int result = 0;
52     for(int& i=dfs_idx[cur]; i<G[cur].size() && lim; i++) {
53         Edge& e = E[G[cur][i]];
54         if(level[e.s]+1 != level[e.t]) continue;
55
56         int flow = dfs(e.t, min(lim, e.cap-e.flow));
57         if(flow <= 0) continue;
58
59         e.flow += flow;
60         result += flow;
61         E[G[cur][i]^1].flow -= flow;
62         lim -= flow;
63     }
64     return result;
65 }
66
67 int dinic() { //  $O((V^2)E)$ 
68     int result = 0;
69     while(bfs()) {
70         memset(dfs_idx, 0, sizeof(dfs_idx));
71         result += dfs(S, inf);
72     }
73     return result;
74 }

```

6.9 SegmentTree

```

1 #define MAXN 1000
2 int data[MAXN]; //原數據
3 int st[4 * MAXN]; //線段樹
4 int tag[4 * MAXN]; //懶標
5
6 inline int pull(int l, int r) {
7     // 隨題目改變sum、max、min
8     // l、r是左右樹的index
9     return st[l] + st[r];
10 }
11
12 void build(int l, int r, int i) {
13     // 在[l, r]區間建樹，目前根的index為i
14     if (l == r) {
15         st[i] = data[l];

```

```

16         return;
17     }
18     int mid = l + ((r - l) >> 1);
19     build(l, mid, i * 2);
20     build(mid + 1, r, i * 2 + 1);
21     st[i] = pull(i * 2, i * 2 + 1);
22 }
23
24 int query(int ql, int qr, int l, int r, int i) {
25     // [ql, qr]是查詢區間,[l, r]是當前節點包含的區間
26     if (ql <= l && r <= qr)
27         return st[i];
28     int mid = l + ((r - l) >> 1);
29     if (tag[i]) {
30         //如果當前懶標有值則更新左右節點
31         st[i * 2] += tag[i] * (mid - l + 1);
32         st[i * 2 + 1] += tag[i] * (r - mid);
33         tag[i * 2] += tag[i]; //下傳懶標至左節點
34         tag[i*2+1] += tag[i]; //下傳懶標至右節點
35         tag[i] = 0;
36     }
37     int sum = 0;
38     if (ql <= mid)
39         sum += query(ql, qr, l, mid, i * 2);
40     if (qr > mid)
41         sum += query(ql, qr, mid + 1, r, i*2+1);
42     return sum;
43 }
44
45 void update(int ql,int qr,int l,int r,int i,int c) {
46     // [ql, qr]是查詢區間,[l, r]是當前節點包含的區間
47     // c是變化量
48     if (ql <= l && r <= qr) {
49         st[i] += (r - l + 1) * c;
50         //求和,此需乘上區間長度
51         tag[i] += c;
52         return;
53     }
54     int mid = l + ((r - l) >> 1);
55     if (tag[i] && l != r) {
56         //如果當前懶標有值則更新左右節點
57         st[i * 2] += tag[i] * (mid - l + 1);
58         st[i * 2 + 1] += tag[i] * (r - mid);
59         tag[i * 2] += tag[i]; //下傳懶標至左節點
60         tag[i*2+1] += tag[i]; //下傳懶標至右節點
61         tag[i] = 0;
62     }
63     if (ql <= mid) update(ql, qr, l, mid, i * 2, c);
64     if (qr > mid) update(ql, qr, mid+1, r, i*2+1, c);
65     st[i] = pull(i * 2, i * 2 + 1);
66 }
67 //如果是直接改值而不是加值，query與update中的tag與st的
//改值從+=改成=

```

6.10 Nim Game

```

1 //兩人輪流取銅板，每人每次需在某堆取一枚以上的銅板，
2 //但不能同時在兩堆取銅板，直到最後，
3 //將銅板拿光的人贏得此遊戲。
4
5 #include <bits/stdc++.h>
6 #define maxn 23+5
7 using namespace std;
8
9 int SG[maxn];
10 int visited[1000+5];
11 int pile[maxn],ans;
12
13 void calculateSG(){
14     SG[0]=0;
15     for(int i=1;i<=maxn;i++){
16         int cur=0;
17         for(int j=0;j<i;j++)
18             for(int k=0;k<=j;k++)

```

```

19         visited[SG[j]^SG[k]]=i;
20         while(visited[cur]==i) cur++;
21         SG[i]=cur;
22     }
23 }
24
25 int main(){
26     calculateSG();
27     int Case=0,n;
28     while(cin>>n,n){
29         ans=0;
30         for(int i=1;i<=n;i++) cin>>pile[i];
31         for(int i=1;i<=n;i++){
32             if(pile[i]&1) ans^=SG[n-i];
33             cout<<"Game " <<Case<<" ";
34             if(!ans) cout<<"-1 -1 -1\n";
35             else{
36                 bool flag=0;
37                 for(int i=1;i<=n;i++){
38                     if(pile[i]){
39                         for(int j=i+1;j<=n;j++){
40                             for(int k=j;k<=n;k++){
41                                 if((SG[n-i]^SG[n-j]^SG[n-k])==ans){
42                                     cout<<i-1<<" "<<j-1<<" "<<k-1<<endl;
43                                     flag=1;
44                                     break;
45                                 }
46                             }
47                             if(flag) break;
48                         }
49                         if(flag) break;
50                     }
51                 }
52             }
53         }
54         return 0;
55     }
56     /*
57     input
58     4 1 0 1 100
59     3 1 0 5
60     2 2 1
61     0
62     output
63     Game 1: 0 2 3
64     Game 2: 0 1 1
65     Game 3: -1 -1 -1
66     */

```

6.11 Trie

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int maxn = 300000 + 10;
5 const int mod = 20071027;
6
7 int dp[maxn];
8 int mp[4000*100 + 10][26];
9 char str[maxn];
10
11 struct Trie {
12     int seq;
13     int val[maxn];
14
15     Trie() {
16         seq = 0;
17         memset(val, 0, sizeof(val));
18         memset(mp, 0, sizeof(mp));
19     }
20
21     void insert(char* s, int len) {
22         int r = 0;
23         for(int i=0; i<len; i++) {
24             int c = s[i] - 'a';

```

```

25         if(!mp[r][c]) mp[r][c] = ++seq;
26         r = mp[r][c];
27     }
28     val[r] = len;
29     return;
30 }
31
32 int find(int idx, int len) {
33     int result = 0;
34     for(int r=0; idx<len; idx++) {
35         int c = str[idx] - 'a';
36         if(!(r = mp[r][c])) return result;
37         if(val[r])
38             result = (result + dp[idx + 1]) % mod;
39     }
40     return result;
41 }
42 };
43
44 int main() {
45     int n, tc = 1;
46
47     while(~scanf("%s%d", str, &n)) {
48         Trie tr;
49         int len = strlen(str);
50         char word[100+10];
51
52         memset(dp, 0, sizeof(dp));
53         dp[len] = 1;
54
55         while(n--) {
56             scanf("%s", word);
57             tr.insert(word, strlen(word));
58         }
59
60         for(int i=len-1; i>=0; i--)
61             dp[i] = tr.find(i, len);
62         printf("Case %d: %d\n", tc++, dp[0]);
63     }
64     return 0;
65 }
66
67 /******
68 ****Input****
69 * abcd
70 * 4
71 * a b cd ab
72 ****Output***
73 * Case 1: 2
74 ****
75 */

```

6.12 SPFA

```

1 struct Edge
2 {
3     int t;
4     long long w;
5     Edge(){};
6     Edge(int _t, long long _w) : t(_t), w(_w) {}
7 };
8
9 bool SPFA(int st) // 平均O(V + E) 最糟O(VE)
10 {
11     vector<int> cnt(n, 0);
12     bitset<MXV> inq(0);
13     queue<int> q;
14     q.push(st);
15     dis[st] = 0;
16     inq[st] = true;
17     while (!q.empty())
18     {
19         int cur = q.front();
20         q.pop();

```

```

21     inq[cur] = false;
22     for (auto &e : G[cur])
23     {
24         if (dis[e.t] <= dis[cur] + e.w)
25             continue;
26         dis[e.t] = dis[cur] + e.w;
27         if (inq[e.t])
28             continue;
29         ++cnt[e.t];
30         if (cnt[e.t] > n)
31             return false; // negative cycle
32         inq[e.t] = true;
33         q.push(e.t);
34     }
35 }
36 return true;
37 }

```

6.13 dijkstra

```

1 #include <bits/stdc++.h>
2 #define maxn 50000+5
3 #define INF 0x3f3f3f3f
4 using namespace std;
5
6 struct edge{
7     int v,w;
8 };
9
10 struct Item{
11     int u,dis;
12     bool operator<(const Item &rhs)const{
13         return dis>rhs.dis;
14     }
15 };
16
17 vector<edge> G[maxn];
18 int dist[maxn];
19
20 void dijkstra(int s){ // O((V + E)log(E))
21     memset(dist,INF,sizeof(dist));
22     dist[s]=0;
23     priority_queue<Item> pq;
24     pq.push({s,0});
25     while(!pq.empty()){
26         Item now=pq.top();
27         pq.pop();
28         if(now.dis>dist[now.u]) continue;
29         for(edge e:G[now.u]){
30             if(dist[e.v]>dist[now.u]+e.w){
31                 dist[e.v]=dist[now.u]+e.w;
32                 pq.push({e.v,dist[e.v]});
33             }
34         }
35     }
36 }
37
38 int main(){
39     int t,cas=1;
40     cin>>t;
41     while(t--){
42         int n,m,s,t;
43         cin>>n>>m>>s>>t;
44         for(int i=0;i<=n;i++) G[i].clear();
45         int u,v,w;
46         for(int i=0;i<m;i++){
47             cin>>u>>v>>w;
48             G[u].push_back({v,w});
49             G[v].push_back({u,w});
50         }
51         dijkstra(s);
52         cout<<"Case #"<<cas++<<" ";
53         if(dist[t]==INF) cout<<"unreachable\n";
54         else cout<<dist[t]<<endl;
55     }
56 }

```

6.14 SCC Tarjan

```

1 //Strongly Connected Components
2 //Tarjan O(V + E)
3 int dfn[N], low[N], dfncnt, sk[N], in_stack[N], tp;
4 //dfn[u]: dfs時u被visited的順序
5 //low[u]: 在u的dfs子樹中能回到最早已在stack中的節點
6 int scc[N], sc; //節點 u 所在 SCC 的編號
7 int sz[N]; //強連通 u 的大小
8
9 void tarjan(int u) {
10     low[u] = dfn[u] = ++dfncnt, s[++tp] = u,
11     in_stack[u] = 1;
12     for (int i = h[u]; i; i = e[i].nex) {
13         const int &v = e[i].t;
14         if (!dfn[v]) {
15             tarjan(v);
16             low[u] = min(low[u], low[v]);
17         } else if (in_stack[v]) {
18             low[u] = min(low[u], dfn[v]);
19         }
20     }
21     if (dfn[u] == low[u]) {
22         ++sc;
23         while (s[tp] != u) {
24             scc[s[tp]] = sc;
25             sz[sc]++;
26             in_stack[s[tp]] = 0;
27             --tp;
28         }
29         scc[s[tp]] = sc;
30         sz[sc]++;
31         in_stack[s[tp]] = 0;
32         --tp;
33     }
34 }

```

6.15 SCC Kosaraju

```

1 //做兩次dfs, O(V + E)
2 //g 是原圖, g2 是反圖
3 //s是dfs離開的節點
4 void dfs1(int u) {
5     vis[u] = true;
6     for (int v : g[u])
7         if (!vis[v]) dfs1(v);
8     s.push_back(u);
9 }
10
11 void dfs2(int u) {
12     group[u] = sccCnt;
13     for (int v : g2[u])
14         if (!group[v]) dfs2(v);
15 }
16
17 void kosaraju() {
18     sccCnt = 0;
19     for (int i = 1; i <= n; ++i)
20         if (!vis[i]) dfs1(i);
21     for (int i = n; i >= 1; --i)
22         if (!group[s[i]]) {
23             ++sccCnt;
24             dfs2(s[i]);
25         }
26 }

```

6.16 ArticulationPoints Tarjan

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 vector<vector<int>> G;

```

```

5 int N;
6 int timer;
7 bool visited[105];
8 int visTime[105]; // 第一次visit的時間
9 int low[105];
10 // 最小能回到的父節點(不能是自己的parent)的visTime
11 int res;
12 //求割點數量
13 void tarjan(int u, int parent) {
14     int child = 0;
15     bool isCut = false;
16     visited[u] = true;
17     visTime[u] = low[u] = ++timer;
18     for (int v: G[u]) {
19         if (!visited[v]) {
20             ++child;
21             tarjan(v, u);
22             low[u] = min(low[u], low[v]);
23             if (parent != -1 && low[v] >= visTime[u])
24                 isCut = true;
25         }
26         else if (v != parent)
27             low[u] = min(low[u], visTime[v]);
28     }
29     //If u is root of DFS tree->有兩個以上的children
30     if (parent == -1 && child >= 2)
31         isCut = true;
32     if (isCut)
33         ++res;
34 }
35
36 int main()
37 {
38     char input[105];
39     char* token;
40     while (scanf("%d", &N) != EOF && N)
41     {
42         G.assign(105, vector<int>());
43         memset(visited, false, sizeof(visited));
44         memset(low, 0, sizeof(low));
45         memset(visTime, 0, sizeof(visTime));
46         timer = 0;
47         res = 0;
48         getchar(); // for \n
49         while (fgets(input, 105, stdin))
50         {
51             if (input[0] == '0')
52                 break;
53             int size = strlen(input);
54             input[size - 1] = '\0';
55             --size;
56             token = strtok(input, " ");
57             int u = atoi(token);
58             int v;
59             while (token = strtok(NULL, " "))
60             {
61                 v = atoi(token);
62                 G[u].emplace_back(v);
63                 G[v].emplace_back(u);
64             }
65         }
66         tarjan(1, -1);
67         printf("%d\n", res);
68     }
69     return 0;
70 }

```

6.17 最小樹狀圖

```

1 | 定義
2 | 有向圖上的最小生成樹 (Directed Minimum Spanning Tree)
3 | 稱為最小樹形圖。
4
5 | const int maxn = 60 + 10;
6 | const int inf = 0x3f3f3f3f;

```

```

7
8 struct Edge {
9     int s, t, cap, cost;
10 }; // cap 為頻寬 (optional)
11
12 int n, m, c;
13 int inEdge[maxn], idx[maxn], pre[maxn], vis[maxn];
14
15 // 對於每個點，選擇對它入度最小的那條邊
16 // 找環，如果沒有則 return;
17 // 進行縮環並更新其他點到環的距離。
18 int dirMST(vector<Edge> edges, int low) {
19     int result = 0, root = 0, N = n;
20
21     while(true) {
22         memset(inEdge, 0x3f, sizeof(inEdge));
23
24         // 找所有點的 in edge 放進 inEdge
25         // optional: low 為最小 cap 限制
26         for(const Edge& e : edges) {
27             if(e.cap < low) continue;
28             if(e.s!=e.t && e.cost<inEdge[e.t]) {
29                 inEdge[e.t] = e.cost;
30                 pre[e.t] = e.s;
31             }
32         }
33
34         for(int i=0; i<N; i++) {
35             if(i!=root && inEdge[i]==inf)
36                 return -1; //除了root 還有點沒有 in edge
37         }
38
39         int seq = inEdge[root] = 0;
40         memset(idx, -1, sizeof(idx));
41         memset(vis, -1, sizeof(vis));
42
43         // 找所有的 cycle，一起編號為 seq
44         for(int i=0; i<N; i++) {
45             result += inEdge[i];
46             int cur = i;
47             while(vis[cur]!=i && idx[cur]==-1) {
48                 if(cur == root) break;
49                 vis[cur] = i;
50                 cur = pre[cur];
51             }
52             if(cur!=root && idx[cur]==-1) {
53                 for(int j=pre[cur]; j!=cur; j=pre[j])
54                     idx[j] = seq;
55                 idx[cur] = seq++;
56             }
57         }
58
59         if(seq == 0) return result; // 沒有 cycle
60
61         for(int i=0; i<N; i++)
62             // 沒有被縮點的點
63             if(idx[i] == -1) idx[i] = seq++;
64
65         // 縮點並重新編號
66         for(Edge& e : edges) {
67             if(idx[e.s] != idx[e.t])
68                 e.cost -= inEdge[e.t];
69             e.s = idx[e.s];
70             e.t = idx[e.t];
71         }
72         N = seq;
73         root = idx[root];
74     }
75 }
76
77 =====
78
79 Tarjan 的DMST 演算法
80 Tarjan 提出了一種能夠在
81 O(m+nlog n)時間內解決最小樹形圖問題的演算法。
82

```

```

83 流程
84 Tarjan 的演算法分為收縮與伸展兩個過程。
85 接下來先介紹收縮的過程。
86 我們要假設輸入的圖是滿足強連通的，
87 如果不滿足那就加入  $O(n)$  條邊使其滿足，
88 並且這些邊的邊權是無窮大的。
89
90 我們需要一個堆存儲結點的入邊編號，入邊權值，
91 結點總代價等相關信息，由於後續過程中會有堆的合併操作，
92 這裡採用左偏樹 與並查集實現。
93 演算法的每一步都選擇一個任意結點  $v$ ，
94 需要保證  $v$  不是根節點，並且在堆中沒有它的入邊。
95 再將  $v$  的最小入邊加入到堆中，
96 如果新加入的這條邊使堆中的邊形成了環，
97 那麼將構成環的那些結點收縮，
98 我們不妨將這些已經收縮的結點命名為超級結點，
99 再繼續這個過程，如果所有的頂點都縮成了超級結點，
100 那麼收縮過程就結束了。
101 整個收縮過程結束後會得到一棵收縮樹，
102 之後就會對它進行伸展操作。
103
104 堆中的邊總是會形成一條路徑  $v_0 \leftarrow v_1 \leftarrow \dots \leftarrow v_k$ ，
105 由於圖是強連通的，這個路徑必然存在，
106 並且其中的  $v_i$  可能是最初的單一結點，
107 也可能是壓縮後的超級結點。
108
109 最初有  $v_0 = a$ ，其中  $a$  是圖中任意的一個結點，
110 每次都選擇一條最小入邊  $v_k \leftarrow u$ ，
111 如果  $u$  不是  $v_0, v_1, \dots, v_k$  中的一個結點，
112 那麼就將結點擴展到  $v_{k+1} = u$ 。
113 如果  $u$  是他們其中的一個結點  $v_i$ ，
114 那麼就找到了一個關於  $v_i \leftarrow \dots \leftarrow v_k \leftarrow v_i$  的環，
115 再將他們收縮為一個超級結點  $c$ 。
116
117 向隊列  $P$  中放入所有的結點或超級結點，
118 並初始選擇任一節點  $a$ ，只要佇列不為空，就進行以下步驟：
119
120 選擇  $a$  的最小入邊，保證不存在自環，
121 並找到另一頭的結點  $b$ 。
122 如果結點  $b$  沒有被記錄過說明未形成環，
123 令  $a \leftarrow b$ ，繼續目前操作尋找環。
124
125 如果  $b$  被記錄過了，就表示出現了環。
126 總結點數加一，並將環上的所有結點重新編號，對堆進行合併，
127 以及結點/超級結點的總權值的更新。
128 更新權值操作就是將環上所有結點的入邊都收集起來，
129 並減去環上入邊的邊權。
130
131 typedef long long ll;
132 #define maxn 102
133 #define INF 0x3f3f3f3f
134
135 struct UnionFind {
136     int fa[maxn << 1];
137     UnionFind() { memset(fa, 0, sizeof(fa)); }
138     void clear(int n) {
139         memset(fa + 1, 0, sizeof(int) * n);
140     }
141     int find(int x) {
142         return fa[x] ? fa[x] = find(fa[x]) : x;
143     }
144     int operator[](int x) { return find(x); }
145 };
146
147 struct Edge {
148     int u, v, w, w0;
149 };
150
151 struct Heap {
152     Edge *e;
153     int rk, constant;
154     Heap *lch, *rch;
155
156     Heap(Edge *_e):
157         e(_e), rk(1), constant(0), lch(NULL), rch(NULL) {}
158
159     void push() {
160         if (lch) lch->constant += constant;
161         if (rch) rch->constant += constant;
162         e->w += constant;
163         constant = 0;
164     }
165 };
166
167 Heap *merge(Heap *x, Heap *y) {
168     if (!x) return y;
169     if (!y) return x;
170     if (x->e->w + x->constant > y->e->w + y->constant)
171         swap(x, y);
172     x->push();
173     x->rch = merge(x->rch, y);
174     if (!x->lch || x->lch->rk < x->rch->rk)
175         swap(x->lch, x->rch);
176     if (x->rch)
177         x->rk = x->rch->rk + 1;
178     else
179         x->rk = 1;
180     return x;
181 }
182
183 Edge *extract(Heap *&x) {
184     Edge *r = x->e;
185     x->push();
186     x = merge(x->lch, x->rch);
187     return r;
188 }
189
190 vector<Edge> in[maxn];
191 int n, m, fa[maxn << 1], nxt[maxn << 1];
192 Edge *ed[maxn << 1];
193 Heap *Q[maxn << 1];
194 UnionFind id;
195
196 void contract() {
197     bool mark[maxn << 1];
198     //將圖上的每一個節點與其相連的那些節點進行記錄
199     for (int i = 1; i <= n; i++) {
200         queue<Heap *> q;
201         for (int j = 0; j < in[i].size(); j++)
202             q.push(new Heap(&in[i][j]));
203         while (q.size() > 1) {
204             Heap *u = q.front();
205             q.pop();
206             Heap *v = q.front();
207             q.pop();
208             q.push(merge(u, v));
209         }
210         Q[i] = q.front();
211     }
212     mark[1] = true;
213     for (int a = 1, b = 1, p; Q[a]; b = a, mark[b] = true) {
214         //尋找最小入邊以及其端點，保證無環
215         do {
216             ed[a] = extract(Q[a]);
217             a = id[ed[a]->u];
218         } while (a == b && Q[a]);
219         if (a == b) break;
220         if (!mark[a]) continue;
221         //對發現的環進行收縮，以及環內的節點重新編號，
222         //總權值更新
223         for (a = b, n++; a != n; a = p) {
224             id.fa[a] = fa[a] = n;
225             if (Q[a]) Q[a]->constant -= ed[a]->w;
226             Q[n] = merge(Q[n], Q[a]);
227             p = id[ed[a]->u];
228             nxt[p == n ? b : p] = a;
229         }
230     }
231 }
232

```

```

233 ll expand(int x, int r);
234 ll expand_iter(int x) {
235     ll r = 0;
236     for(int u=nxt[x]; u!=x; u=nxt[u]){
237         if (ed[u]->w0 >= INF)
238             return INF;
239         else
240             r+=expand(ed[u]->v,u)+ed[u]->w0;
241     }
242     return r;
243 }
244
245 ll expand(int x, int t) {
246     ll r = 0;
247     for (; x != t; x = fa[x]) {
248         r += expand_iter(x);
249         if (r >= INF) return INF;
250     }
251     return r;
252 }
253
254 void link(int u, int v, int w) {
255     in[v].push_back({u, v, w, w});
256 }
257
258 int main() {
259     int rt;
260     scanf("%d %d %d", &n, &m, &rt);
261     for (int i = 0; i < m; i++) {
262         int u, v, w;
263         scanf("%d %d %d", &u, &v, &w);
264         link(u, v, w);
265     }
266     //保證強連通
267     for (int i = 1; i <= n; i++)
268         link(i > 1 ? i - 1 : n, i, INF);
269     contract();
270     ll ans = expand(rt, n);
271     if (ans >= INF)
272         puts("-1");
273     else
274         printf("%lld\n", ans);
275     return 0;
276 }

```

7 geometry

7.1 intersection

```

1 using LL = long long;
2
3 struct Point2D {
4     LL x, y;
5 };
6
7 struct Line2D {
8     Point2D s, e;
9     LL a, b, c; // L: ax + by = c
10    Line2D(Point2D s, Point2D e): s(s), e(e) {
11        a = e.y - s.y;
12        b = s.x - e.x;
13        c = a * s.x + b * s.y;
14    }
15 };
16
17 // 用克拉馬公式求二元一次解
18 Point2D intersection(Line2D l1, Line2D l2) {
19     LL D = l1.a * l2.b - l2.a * l1.b;
20     LL Dx = l1.c * l2.b - l2.c * l1.b;
21     LL Dy = l1.a * l2.c - l2.a * l1.c;
22
23     if(D) { // intersection
24         double x = 1.0 * Dx / D;
25         double y = 1.0 * Dy / D;

```

```

26     } else {
27         if(Dx || Dy) // Parallel lines
28             else // Same line
29     }
30 }

```

7.2 半平面相交

```

1 // Q: 給定一張凸包(已排序的點),
2 // 找出圖中離凸包外最遠的距離
3
4 const int maxn = 100 + 10;
5 const double eps = 1e-7;
6
7 struct Vector {
8     double x, y;
9     Vector(double x=0.0, double y=0.0): x(x), y(y) {}
10
11     Vector operator+(Vector v) {
12         return Vector(x+v.x, y+v.y);
13     }
14     Vector operator-(Vector v) {
15         return Vector(x-v.x, y-v.y);
16     }
17     Vector operator*(double val) {
18         return Vector(x*val, y*val);
19     }
20     double dot(Vector v) { return x*v.x + y*v.y; }
21     double cross(Vector v) { return x*v.y - y*v.x; }
22     double length() { return sqrt(dot(*this)); }
23     Vector unit_normal_vector() {
24         double len = length();
25         return Vector(-y/len, x/len);
26     }
27 };
28
29 using Point = Vector;
30
31 struct Line {
32     Point p;
33     Vector v;
34     double ang;
35     Line(Point p={}, Vector v={}): p(p), v(v) {
36         ang = atan2(v.y, v.x);
37     }
38     bool operator<(const Line& l) const {
39         return ang < l.ang;
40     }
41     Point intersection(Line l) {
42         Vector u = p - l.p;
43         double t = l.v.cross(u) / v.cross(l.v);
44         return p + v*t;
45     }
46 };
47
48 int n, m;
49 Line narrow[maxn]; // 要判斷的直線
50 Point poly[maxn]; // 能形成半平面交的凸包邊界點
51
52 // return true if point p is on the left of line l
53 bool onLeft(Point p, Line l) {
54     return l.v.cross(p-l.p) > 0;
55 }
56
57 int halfplaneIntersection() {
58     int l, r;
59     Line L[maxn]; // 排序後的向量隊列
60     Point P[maxn]; // s[i] 跟 s[i-1] 的交點
61
62     L[l=r=0] = narrow[0]; // notice: narrow is sorted
63     for(int i=1; i<n; i++) {
64         while(l<r && !onLeft(P[r-1], narrow[i])) r--;
65         while(l<r && !onLeft(P[l], narrow[i])) l++;
66         L[++r] = narrow[i];

```



```

68     if(l < r) P[r-1] = L[r-1].intersection(L[r]);
69 }
70 while(l<r && !onLeft(P[r-1], L[l])) r--;
71 if(r-1 <= 1) return 0;
72 P[r] = L[r].intersection(L[l]);
73
74 int m=0;
75 for(int i=1; i<=r; i++) {
76     poly[m++] = P[i];
77 }
78
79 return m;
80 }
81 }
82
83 Point pt[maxn];
84 Vector vec[maxn];
85 Vector normal[maxn]; // normal[i] = vec[i] 的單位法向量
86
87 double bsearch(double l=0.0, double r=1e4) {
88     if(abs(r-l) < 1e-7) return l;
89
90     double mid = (l + r) / 2;
91
92     for(int i=0; i<n; i++) {
93         narrow[i] = Line(pt[i]+normal[i]*mid, vec[i]);
94     }
95
96     if(halfplaneIntersection())
97         return bsearch(mid, r);
98     else return bsearch(l, mid);
99 }
100 }
101
102 int main() {
103     while(~scanf("%d", &n) && n) {
104         for(int i=0; i<n; i++) {
105             double x, y;
106             scanf("%lf%lf", &x, &y);
107             pt[i] = {x, y};
108         }
109         for(int i=0; i<n; i++) {
110             vec[i] = pt[(i+1)%n] - pt[i];
111             normal[i] = vec[i].unit_normal_vector();
112         }
113
114         printf("%.6lf\n", bsearch());
115     }
116     return 0;
117 }

```

7.3 凸包

```

1 // Q: 平面上給定多個區域，由多個座標點所形成，再給定
2 // 多點(x,y)，判斷有落點的區域(destroyed)的面積總和。
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 const int maxn = 500 + 10;
7 const int maxCoordinate = 500 + 10;
8
9 struct Point {
10     int x, y;
11 };
12
13 int n;
14 bool destroyed[maxn];
15 Point arr[maxn];
16 vector<Point> polygons[maxn];
17
18 void scanAndSortPoints() {
19     int minX = maxCoordinate, minY = maxCoordinate;
20     for(int i=0; i<n; i++) {
21         int x, y;
22         scanf("%d%d", &x, &y);

```

```

23         arr[i] = (Point){x, y};
24         if(y < minY || (y == minY && x < minX)) {
25             // If there are floating points, use:
26             // if(y<minY || (abs(y-minY)<eps && x<minX)) {
27                 minX = x, minY = y;
28             }
29         }
30         sort(arr, arr+n, [minX, minY](Point& a, Point& b){
31             double theta1 = atan2(a.y - minY, a.x - minX);
32             double theta2 = atan2(b.y - minY, b.x - minX);
33             return theta1 < theta2;
34         });
35         return;
36     }
37
38     // returns cross product of u(AB) x v(AC)
39     int cross(Point& A, Point& B, Point& C) {
40         int u[2] = {B.x - A.x, B.y - A.y};
41         int v[2] = {C.x - A.x, C.y - A.y};
42         return (u[0] * v[1]) - (u[1] * v[0]);
43     }
44
45     // size of arr = n >= 3
46     // st = the stack using vector, m = index of the top
47     vector<Point> convex_hull() {
48         vector<Point> st(arr, arr+3);
49         for(int i=3, m=2; i<n; i++, m++) {
50             while(m >= 2) {
51                 if(cross(st[m], st[m-1], arr[i]) < 0)
52                     break;
53                 st.pop_back();
54                 m--;
55             }
56             st.push_back(arr[i]);
57         }
58         return st;
59     }
60
61     bool inPolygon(vector<Point>& vec, Point p) {
62         vec.push_back(vec[0]);
63         for(int i=1; i<vec.size(); i++) {
64             if(cross(vec[i-1], vec[i], p) < 0) {
65                 vec.pop_back();
66                 return false;
67             }
68         }
69         vec.pop_back();
70         return true;
71     }
72
73     1 | x1  x2  x3  x4  x5          xn |
74     A = - | x   x   x   x   x ... x |
75           2 | y1  y2  y3  y4  y5          yn |
76     double calculateArea(vector<Point>& v) {
77         v.push_back(v[0]); // make v[n] = v[0]
78         double result = 0.0;
79         for(int i=1; i<v.size(); i++)
80             result += v[i-1].x*v[i].y - v[i-1].y*v[i].x;
81         v.pop_back();
82         return result / 2.0;
83     }
84
85     int main() {
86         int p = 0;
87         while(~scanf("%d", &n) && (n != -1)) {
88             scanAndSortPoints();
89             polygons[p++] = convex_hull();
90         }
91
92         int x, y;
93         double result = 0.0;
94         while(~scanf("%d%d", &x, &y)) {
95             for(int i=0; i<p; i++) {
96                 if(inPolygon(polygons[i], (Point){x, y}))
97                     destroyed[i] = true;
98             }
99         }

```

```

100     for(int i=0; i<p; i++) {
101         if(destroyed[i])
102             result += calculateArea(polygons[i]);
103     }
104     printf("%.2lf\n", result);
105     return 0;
106 }

```

8 動態規劃

8.1 LCS 和 LIS

```

1 //最長共同子序列 (LCS)
2 給定兩序列 A,B ，求最長的序列 C ，
3   C 同時為 A,B 的子序列。
4
5 //最長遞增子序列 (LIS)
6 給你一個序列 A ，求最長的序列 B ，
7   B 是一個（非）嚴格遞增序列，且為 A 的子序列。
8
9 //LCS 和 LIS 題目轉換
10 LIS 轉成 LCS
11     1. A 為原序列， B=sort(A)
12     2. 對 A,B 做 LCS
13 LCS 轉成 LIS
14     1. A, B 為原本的兩序列
15     2. 最 A 序列作編號轉換，將轉換規則套用在 B
16     3. 對 B 做 LIS
17     4. 重複的數字在編號轉換時後要變成不同的數字，
18        越早出現的數字要越小
19     5. 如果有數字在 B 裡面而不在 A 裡面，
20        直接忽略這個數字不做轉換即可

```

9 Section2

9.1 thm

- 中文測試
- $\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$
- $\binom{x}{y} = \frac{x!}{y!(x-y)!}$
- $\int_0^\infty e^{-x} dx$
- $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$

10 DP

10.1 字串 DP

- Edit distance

S_1 最少需要經過幾次增、刪或換字變成 S_2

$$dp[i][j] = \begin{cases} i+1 & \text{if } j = -1 \\ j+1 & \text{if } i = -1 \\ dp[i-1][j-1] & \text{if } S_1[i] = S_2[j] \\ \min \begin{cases} dp[i-1][j] \\ dp[i][j-1] \\ dp[i-1][j] \end{cases} & \text{if } S_1[i] \neq S_2[j] \end{cases} + 1$$

- Longest Palindromic Subsequence

$$dp[l][r] = \begin{cases} 1 & \text{if } l = r \\ dp[l+1][r-1] & \text{if } S[l] = S[r] \\ \max\{dp[l+1][r], dp[l][r-1]\} & \text{if } S[l] \neq S[r] \end{cases}$$