

# 我們與二分搜的距離

## BINARY SEARCH

吳邦一



# OUTLINES

- 基本型與常見錯誤
- **STL**內建二分搜、容器與離散化
- 可用可不用二分搜的情形
- 計算函數的根與山谷最低點
- 值域二分搜
- 二分檢測

# 二分搜基本型

- 在一個遞增(或遞減/非遞增/非遞減)搜尋**key**
- 維護好一個搜尋區間，所要的不會被排除在外。每次取一值做一個檢測
  - 找到**key**或是將搜尋區間減半，直到找到**key**或者區間為空
  - 複雜度:  $O(\log n)$
- ```
while (le <= ri) { // 區間 [le, ri]
    mid = le + (ri-le)/2; // (le + ri)/2 避免overflow的寫法
    if (a[mid] == key) return mid;
    else if (a[mid] < key) le = mid+1;
    else ri = mid - 1;
}
```

# 找KEY好寫不容易錯

- 可能的形式：第一個 $\geq$ 、第一個 $>$ 、最後一個 $<$ 、或是最後一個 $\leq$
- 其實是在找(第一個滿足某條件)或(最後一個不滿足某條件)，序列必須對該條件滿足單調性 (**false, false, ..., false, true, true,...,true**)，可能退化為全**false**/**true**
  - 最後一個不滿足的位置是第一個滿足位置的前一個
- 以找第一個  $\geq$  為例
- ```
while (le < ri) { // 區間 [le, ri]
    mid = (le + ri)/2;
    if (a[mid] ? key) _____;
    else _____;
}
```

# 我們與二分搜的距離 人人會寫但又人人都可能寫錯

- ? 中可能填  $<$ ,  $\leq$ ,  $>$ ,  $\geq$ ，不同的區間定義與要求，寫法會不同
- ```
while (le < ri) { // 區間 [le, ri] > l
    mid = (le + ri)/2;
    if (a[mid] < key) le = mid;
    else ri = mid;
}
```
- $ri = le+1$  時， $mid=le$ ，可能無窮迴圈
- 改成  $le = mid+1$
- ```
while (le < ri) { // 區間 [le, ri]
    mid = (le + ri)/2;
    if (a[mid] < key) le = mid+1;
    else ri = mid;
}
```
- 停下來時答案在哪裡？
- 都正確嗎？當  $key > a[ri]$  時

# 基本二分搜

- 寫區間二分搜記得
  - 寫下你的區間定義  $[l_e, r_i]$  或  $[l_e, r_i)$ 。
  - 檢查區間長度剩下2時是否正確。
- **Jumping**的寫法
  - 紀錄目前位置  $p$  與跳躍距離  $jump$
  - 只要不滿足條件就往前跳
  - $jump$  每次減半
- if ( $a[0] \geq key$ ) return 0;  
for ( $p=0, jump=n/2; jump; jump>>=1$ ) {  
    while ( $p < n-jump \&\& a[p+jump]<key$ )  
         $p += jump;$   
    }  
return  $p+1$ ; //  $p$ 是最後一個<  
• 注意 **for**裡面是 **while**不是 **if**
  - 在某些狀況或做2次- 寫成  $p < n-jump$  預防數值形式時 **overflow**

# STL內建二分搜、容器與離散化

- 內建二分搜
  - binary\_search
  - lower\_bound
  - upper\_bound

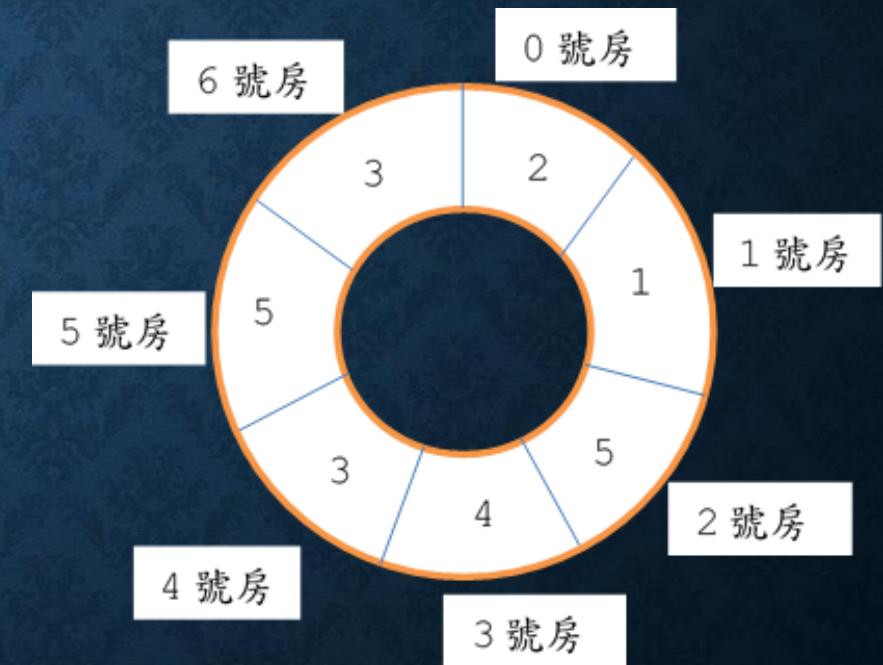
```
8 |     int p[5]={5, 1, 8, 3, 9}, n=5, t=7;
9 |     sort(p, p+n);
10|     // search [first=p, last=p+n) to find the first >=t
11|     int ndx=lower_bound(p, p+n, t) - p;
12|     // for vector
13|     vector<int> v(p, p+n); // copy p to vector v
14|     ndx = lower_bound(v.begin(), v.end(), t) - v.begin();
15|     if (ndx<n) printf("The first >=%d is at [%d]\n",t, ndx);
16|     auto it = upper_bound(v.begin(),v.end(),5); // iterator
17|     if (it!=v.end()) printf("%d\n",*it);
18|     else printf("No upper bound\n");
19|     it = upper_bound(v.begin(),v.end(),9);
20|     if (it!=v.end()) printf("%d\n",*it);
21|     else printf("No upper bound\n");
```

# STL內建二分搜、容器與離散化

- set/map/multiset/multimap
  - binary search tree
  - dynamic data structure, insert/delete
  - lower\_bound/upper\_bound search in  $O(\log n)$  time.
  - can also used to find max/min, but not for ranking/unranking
- 典型常用：相異數、離散化、出現次數
- map<string, int> cnt;  
for (string s: W) cnt[s]++;  
for (auto p: cnt)  
 cout << p.first << ':' << p.second << '\n';
- 小心誤用lower\_bound
  - set<int> S;  
S.lower\_bound(t);  
lower\_bound(S.begin(), S.end(), t);
- 小心在multiset中的刪除一個還是刪除全部同一key的元素

# 範例：圓環出口

- 有 $n$  個房間排列成一個圓環，以順時針方向由0到 $n - 1$ 編號。玩家只能順時針方向依序通過這些房間。
- 每當離開第 $i$ 號房間即可獲得 $p(i)$ 點。玩家必須依序取得 $m$ 把鑰匙，鑰匙編號由0至 $m-1$ ，兌換編號 $i$ 的鑰匙所需的點數為 $Q(i)$ 。
- 一旦玩家點數達到 $Q(i)$ 就會自動獲得編號 $i$ 的鑰匙，而且手中所有的點數就會被「全數收回」，接著要再從當下所在的房間出發，重新收集點數兌換下一把鑰匙。
- 遊戲開始時，玩家位於0號房。請計算玩家拿到最後一把鑰匙時所在的房間編號。



# 圓環出口

- 順便講一下差分與前綴和
  - 數列  $s$  的差分  $d[i] = s[i] - s[i-1]$
- 若  $d$  是  $s$  的差分，則  $s$  是  $d$  的前綴和
  - $s[i] = d[0] + d[1] + \dots + d[i]$
- 本題每個房間的鑰匙數量非負，其前綴和為遞增(非遞減)數列，先做好前綴和之後，在第  $room$  個房間要蒐集  $q$  點的位置，即是找到第一個  $\geq p[room-1] + q$  的位置

```

10  for (i=0; i<n; i++)
11    scanf("%d", &p[i]);
12  for (i=0; i<n; i++) // double array
13    p[n+i] = p[i];
14  // prefix sum
15  for (i=1; i<2*n; i++)
16    p[i] += p[i-1];
17  int room=0, q;
18  for (i=0; i<m; i++) {
19    scanf("%d", &q);
20    if (room != 0) // desired prefix-sum
21      q += p[room-1];
22    // binary search the first >= q
23    room = lower_bound(p+room, p+2*n, q) - p;
24    room = (room+1)%n; // adjust
25  }
26  printf("%d\n", room);

```

# 一些可用可不用的情形

- 連續二分搜不如一路爬過去
- 一群數字裡找兩個數之和等於(或最接近)K。
  - 也可以在兩群數字中各找一數
- ```
for (i=0,j=n-1; i<j ;i++) {  
    while (i<j && a[j]>K-a[i]) j--;  
    if (a[i]+a[j]==K) ans++;  
}
```
- $O(n)$  after sorting; 雙指針法
- 卡通團隊(找相同或互補)
- 一個字母表示一個人物，一個字串表示一個團隊，找互補團隊
- 可以用排序後二分搜
- 也可以用爬行方式
- 或者將互補字串放在一起排序找相同
- hash table也可做 (找exactly match)

|   |   |   |   |   |   |   |    |    |    |
|---|---|---|---|---|---|---|----|----|----|
| 1 | 4 | 5 | 5 | 6 | 9 | 9 | 15 | 17 | 19 |
| ↑ |   |   |   |   | ↑ |   |    |    |    |

# 例題：數位占卜(AP202201Q3)

- 輸入有  $m$  個不相同的字串，要計算有多少對  $X, Y$  滿足  $X+Y$  可以拆成前後兩個完全相同的字串，也就是  $X+Y = P+P$ ，此處字串的相加是串接的意思。
- 例如  $\text{piep} + \text{ie} = \text{piepie}$
- 因為字串皆不相同，所以若  $X+Y = P+P$ ， $X$  與  $Y$  必為一長一短，假設  $X$  比較長，題目中提示短字串一定會出現在長字串之中也就是  $X=A+Y+B$ ，因此我們可以得到  $X+Y = A+Y+B+Y = P+P$ ，結論是  $A = B$ ，也就是  $X$  必定可以拆成  $A+Y+A$  的形式。
- 對每一個  $X$ :  
    對  $X$  的每一個字首  $A$ :  
        if  $X=A+Y+A$  且存在  $Y$  then  
            找到一組答案
- 搜尋  $Y$  的存在可用二分搜或是 hash table

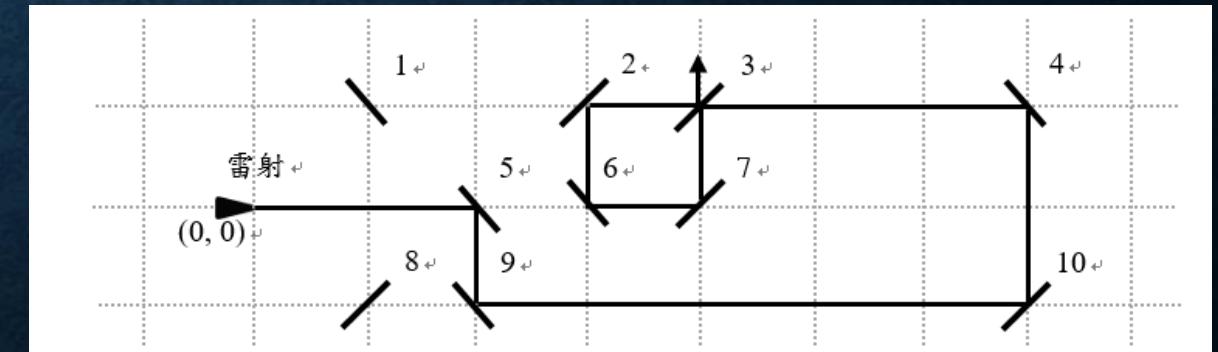
```
// using sorting and binary search+
#include <bits/stdc++.h>
using namespace std;

int main() {
    int i,j,n,k;
    cin >> n;
    vector<string> str(n);
    for (i=0; i<n; i++) {
        cin >> str[i];
    }
    sort(str.begin(), str.end());
    int total =0;
    for (string s: str) {
        // try each suffix, s = x+y+x
        for (int len=1; len+len < s.length(); len++) {
            string pref=s.substr(0, len), suf=s.substr(s.length()-len);
            if (pref != suf) continue;
            string y=s.substr(len, s.length()-len-len);
            if (binary_search(str.begin(), str.end(), y)) {
                total++;
            }
        }
    }
    cout << total << '\n';
    return 0;
}
```

# 例題：雷射測試(AP202206Q3)

- 平面上有若干鏡子，光束由指定位置射入，光束在行進方向中若碰到鏡子則行進方向會轉90度而轉向由鏡子的擺放形式來決定。要計算一共會反射幾次。題目保證不會進入迴圈。
- 我們可以模擬光線的行進路線，記錄著本次出發位置與方向，每次先找到會被擊中的鏡子，這個鏡子的位置就是下次出發位置，然後根據鏡子的擺放形式決定下次的行進方向。如果沒有任何鏡子會被擊中，就表示模擬結束了。

- 二分搜
  - 每一列與每一行都各放一個**vector**
  - 每次用二分搜找碰到鏡子的位置與種類
- 排序後找出鄰居
  - 將所有鏡子的位置以**X**為主排序後找南北鄰居
  - 以**Y**為主排序後找東西鄰居
  - 每次反射方向的鄰居就是下一個擊中的鏡子



```

00 #include <bits/stdc++.h>
01 using namespace std;
02 #define N 300000
03 #define East 0
04 #define South 1
05 #define West 2
06 #define North 3
07 int next_d[2][4]={{North,West,South,East},
08                   {South,East,North,West}}; // 0 = '/',
09 struct T {
10     int x, y, idx;
11 } tem[N];
12 bool cmp_x(T &u, T &v) { // sort by x
13     if (u.x == v.x) return u.y < v.y;
14     return u.x < v.x;
15 }
16
17 bool cmp_y(T &u, T &v) { // sort by y
18     if (u.y == v.y) return u.x < v.x;
19     return u.y < v.y;
20 }
21 int adj[N][4]; // neighbor of 4 direction
22 int mir[N];
23

```

```

33     for (i=0; i<n; i++) for (j=0; j<4; j++) {
34         adj[i][j] = -1; // initial no neighbor
35         // sort by x-major to find North/South neighbor
36         sort(tem, tem+n, cmp_x);
37         for (i=1; i<n; i++) {
38             if (tem[i].x == tem[i-1].x) {
39                 adj[tem[i].idx][South] = tem[i-1].idx;
40                 adj[tem[i-1].idx][North] = tem[i].idx;
41             }
42         }
43         // sort by y-major to find West/East neighbor
44         sort(tem, tem+n, cmp_y);
45         for (i=1; i<n; i++) {
46             if (tem[i].y == tem[i-1].y) {
47                 adj[tem[i].idx][West] = tem[i-1].idx;
48                 adj[tem[i-1].idx][East] = tem[i].idx;
49             }
50         }
51         // start at (0, 0) dir=East
52         int v = adj[0][East], d=East, hit=0;
53         while (v >= 0) {
54             hit++;
55             d = next_d[mir[v]][d]; // next direction
56             v = adj[v][d]; // neighbor of direction d
57         }
58         printf("%d\n", hit);
59     }

```

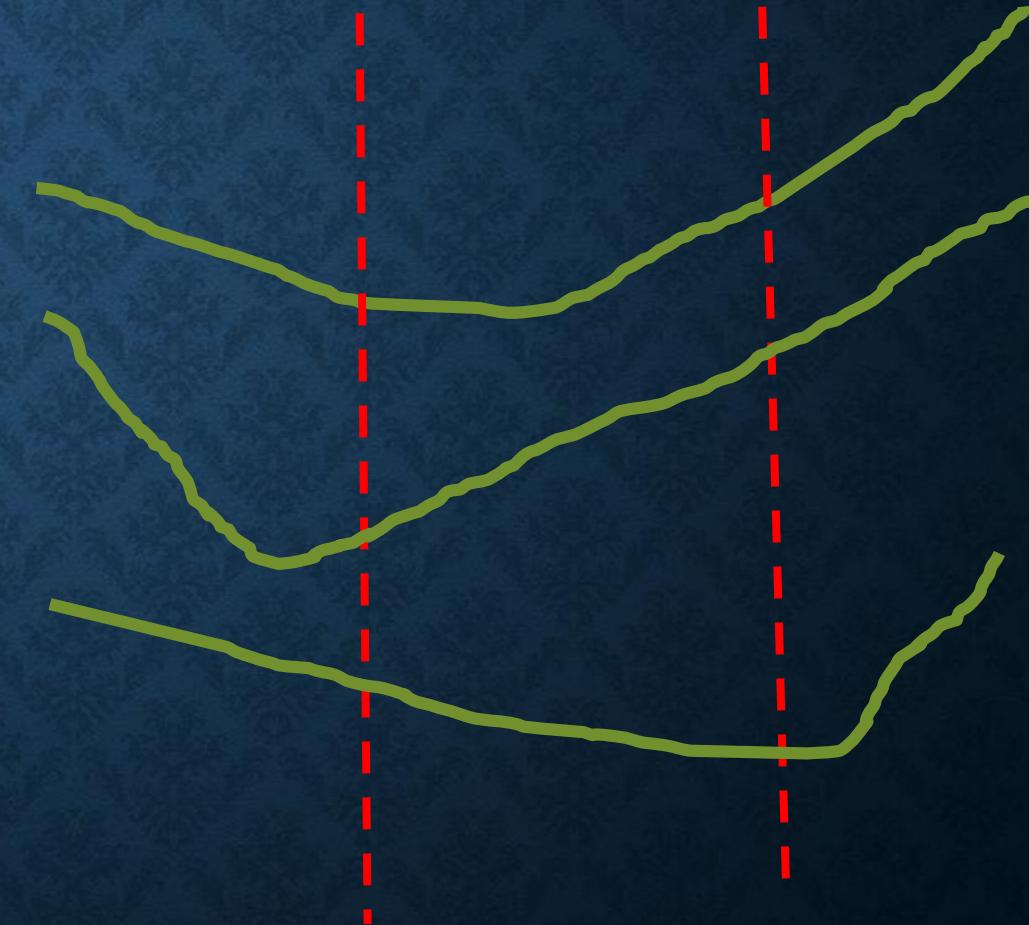
# 曲線交點

- 曲線交點就是計算函數的根
- 勘根定理：假設 $f$ 是連續函數，若 $f(a)f(b) < 0$ 則 $[a,b]$ 區間內必有一根。
- 我們可以用二分搜來逼近這個根。  
`while (b-a > 1e-6) {`
- 有一些更快收斂的方法，如牛頓法

```
3 int main() {
4     int times=0;
5     float low=0.0, up=2.0;
6     while (up-low>1e-6) {
7         times++;
8         float mid=(low+up)/2;
9         float y=mid*mid-2.0;
10        if (y>0) up=mid;
11        else low=mid;
12    }
13    printf("sqrt(2)=%f; iterations=%d\n", low, times);
```

# 山谷低點(山峰高點)

- 一個遞減後遞增的函數
- 三分搜
  - 選兩點均勻切三等份，函數值比較大的一邊可以丟掉 $1/3$
- 差分二分搜
  - 差分有單調性：----[000]+++
  - 小心邊界
- 半邊可以有等號，但不能兩邊都有等號
- 山谷函數來源：例如**max**(遞增,遞減)或一群直線取**max**



# 值域二分搜 (外掛二分搜)

- 例題AP325 Q-4-10：一數列要切成 $m+1$ 段，最大總和的最小值 $f$ 。
  - 用 $m+1$ 個一樣容量的容器去裝，容器的容量至少需要多少
- $f$ 不知如何求，但對於給定容量值，很容易求出需要幾個容器 ---- 一個一個裝，別無選擇，裝不下就要增加一個容器
- 單調性：容器數量越小，容器容量必然越大或一樣，不可能數量變小容量也變小
- 二分搜去猜測容量，檢測數量是否足夠

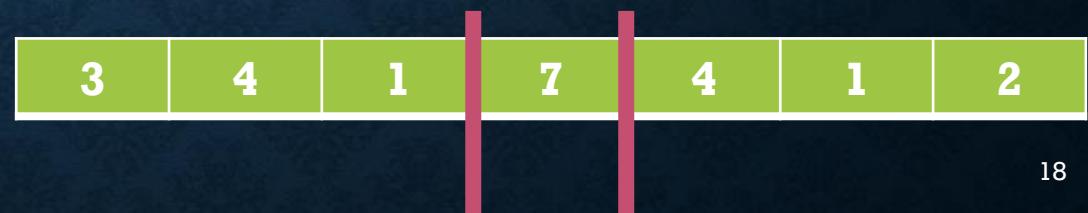
切 3 段

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 3 | 4 | 1 | 7 | 4 | 1 | 2 |
|---|---|---|---|---|---|---|

假設容量 = 7，必須切成4段



假設容量 = 8，可切成3段



```

10 bool enough(LL f) {
11     int remain=m; // remaining segments
12     LL sum = 0;
13     for (int i=0; i<n; i++) {
14         if (p[i]<=f-sum) {
15             sum+=p[i];
16             continue;
17         };
18         if (remain == 0) return false;
19         // use a new box
20         remain--;
21         sum = p[i];
22     }
23     return true;
24 }
```

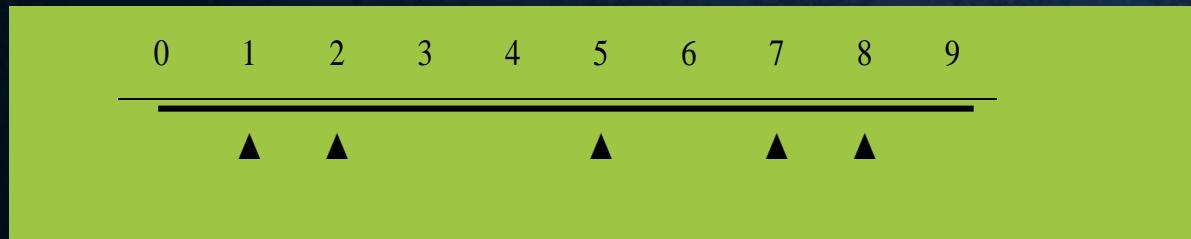
- complexity =  $O(n \log W)$ ,  $W$  is total.
- Another approach  $O(n + (m \log n) \log W)$   
轉prefix sum後用lower\_bound檢測夠不夠

```

26 int main() {
27     scanf("%d%d", &n, &m);
28     LL total=0, mmax=0;
29     for (int i=0; i<n; i++) {
30         scanf("%lld", p+i);
31         total+=p[i];
32         mmax=max(mmax, p[i]);
33     }
34     // binary search, jump to max not-enough capacity
35     if (m==0) {
36         printf("%lld\n", total);
37         return 0;
38     }
39     LL f = mmax-1;
40     for (LL jump=total/2; jump>0; jump>>=1) {
41         while (f+jump<total && !enough(f+jump))
42             f += jump;
43     }
44     printf("%lld\n", f+1);
```

# 基地台(AP201703)

- 直線上有 $N$ 個要服務的點，每架設一座基地台可以涵蓋直徑 $R$ 範圍以內的服務點。輸入服務點的座標位置以及一個正整數 $K$ ，請問：在架設 $K$ 座基地台以及每個基地台的直徑皆相同的條件下，基地台最小直徑 $R$ 為多少？
- 數線上有若干點，用 $k$ 根長度一樣的線段去蓋，求線段長的最小值
- 給定線段長度 $R$ ，需要多少根才夠？
  - Greedy** 用一根左端放在最左邊的點
  - 重複此步驟
- 線段越長，所需數量只會一樣或減少，不可能反而增加
  - 二分搜線段長度



# 基地台

```

01 #include <bits/stdc++.h>
02 using namespace std;
03 // check if k segment of length x is enough
04 int check(int p[], int n, int k, int x) {
05     int endline = -1; // current covered range
06     for (int i=0; i<n; i++) {
07         if (p[i] <= endline) continue;
08         if (k == 0) return 0; // false
09         // use a segment to cover
10         k--;
11         endline = p[i] + x;
12     }
13     return 1; // check ok
14 }
15 int main() {
16     int n, i, k, p[50010];
17     scanf("%d%d", &n, &k);
18     for (i=0; i<n; i++)
19         scanf("%d", p+i);
20     sort(p, p+n);

```

```

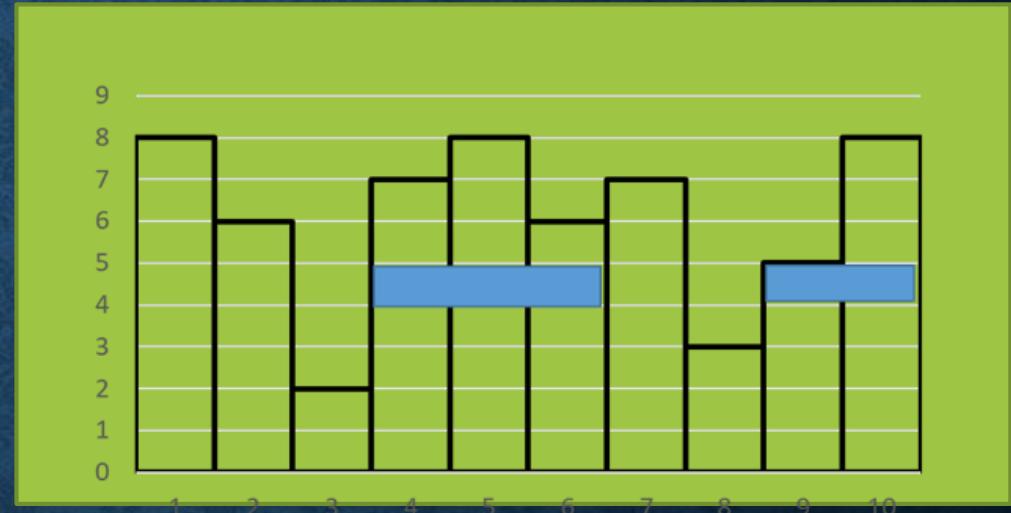
21     int low = 1;
22     int up = p[n-1] - p[0];
23     while (low < up) {
24         int diameter = (low+up)>>1;
25         if (check(p, n, k, diameter) == 0)
26             low = diameter+1;
27         else
28             up = diameter;
29     }
30     printf("%d\n", low);
31     return 0;

```

- Complexity  $O(n\log n + n \log W)$

# 牆上海報(AP202201Q4)

- 圍牆由  $n$  根高高低低的木條排列而成，每根木條的寬度都是 1 單位，高度依序為  $h(1), h(2), \dots, h(n)$ 。
- $k$  幅海報想要掛在這面牆上，每一幅海報的高度都是 1，寬度則依編號順序為  $w(1), w(2), \dots, w(k)$ 。
  - 海報上任何一個位置後面都有木條。
  - 所有海報張貼位置的高度必須相同。
  - 任兩張海報不可以重疊。
  - 海報必須依照編號順序由左而右張貼。
- 請你計算出最高可能張貼的高度。



- 指定高度時，從左往右海報能放就放
- 欄杆底部是一樣的，高度越低可用的越多
- 高度  $h$  可以的話， $< h$  一定也可以，具備單調性 +++++-----

# 牆上海報

```

5 #define oo 1000000000
6 // test if can be cut with height
7 bool test(vector<int> &h, vector<int> &w, int height) {
8     int wi = 0, len=0; // current segment
9     for (int x: h) {
10         if (x < height) {
11             len = 0;
12             continue;
13         }
14         len++;
15         if (len == w[wi]) {
16             if (++wi >= w.size()) return true;
17             len = 0;
18         }
19     }
20     return false;
21 }
```

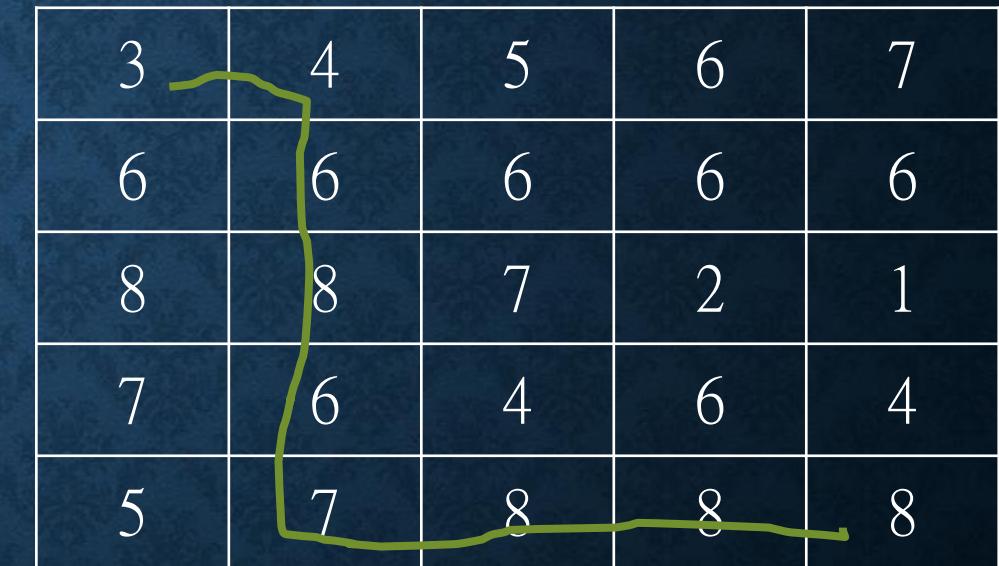
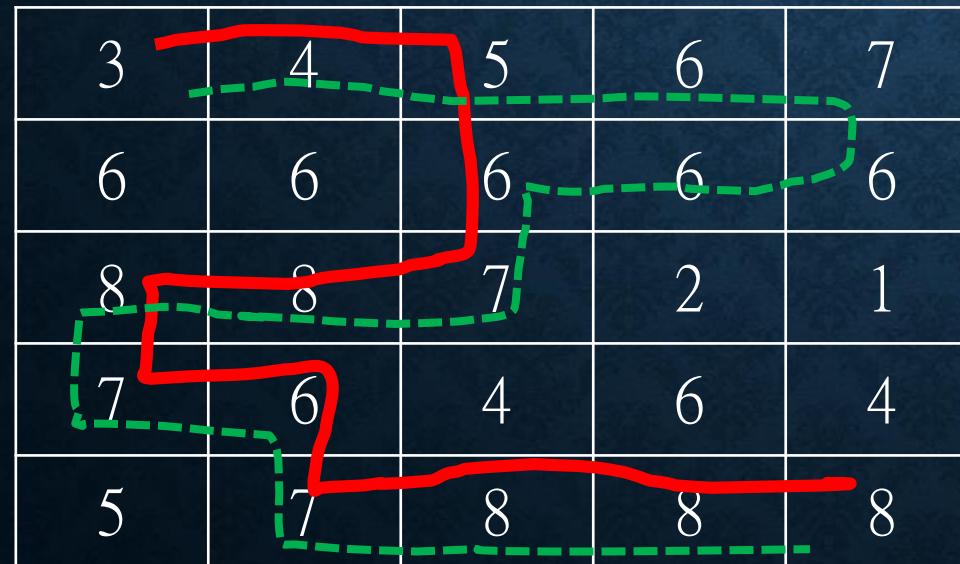
- 答案必為某個柵欄高度，只要針對可能的高度搜尋
- Complexity  $O(n \log n)$ , since  $k \leq n$

```

24 int main() {
25     int i,n,k;
26     cin >> n >> k;
27     vector<int> h(n), w(k);
28     for (i=0; i<n; i++) cin >> h[i];
29     for (i=0; i<k; i++) cin >> w[i];
30     vector<int> allh(h.begin(), h.end());
31     sort(allh.begin(), allh.end());
32     int best = 0; // index of allh
33     // binary search the largest index in allh
34     for (int jump=n/2; jump>0; jump>>=1) {
35         while (best+jump < n && test(h,w,allh[best+jump]))
36             best += jump;
37     }
38     printf("%d\n", allh[best]);
39 }
```

# 蓋步道 APCS-2022-10-Q4 (ZJ-J125)

- 每一個方格的高度，找一條左上角到右下角的步道
  - 每次可走上下左右
  - 相鄰高度差的最大值要最小
  - 在此前題下的最短路徑長度



**h\_difference = 2**

**h\_difference = 1**

綠線高度差也是1但長度較長

# 怎麼做

- 有兩個方法，其中一個超過APCS程度，後面再說。
- 又要找最小高度差，又要最短路徑，不知如何下手，很煩耶
- 反過來想，如果已知最小高度差，怎麼找路徑？
- 先看個高度差=0的情形
  - 有些相鄰的格子可以走
  - 有些相鄰的格子不能走，因為高度差限制

**限制高度差 $h\_difference \leq 0$**

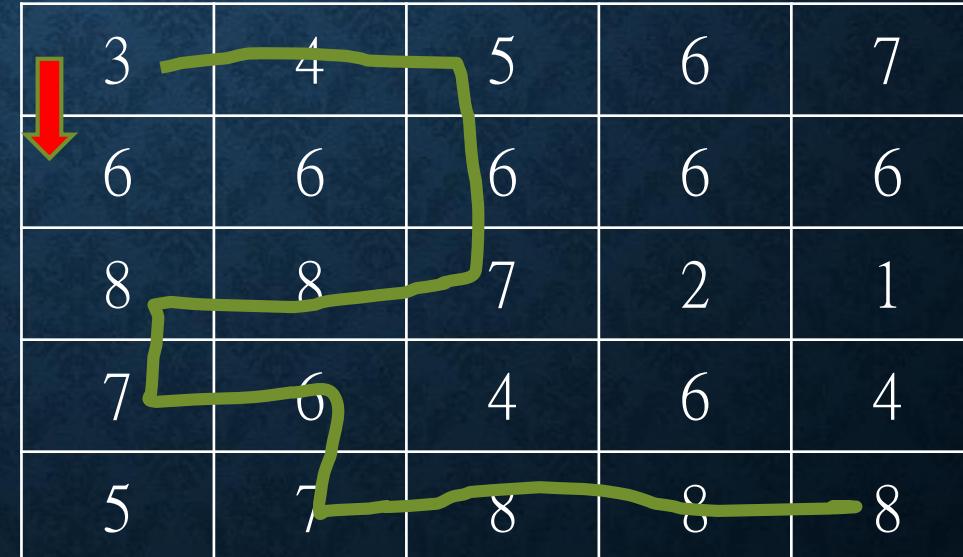
|   |   |   |   |          |
|---|---|---|---|----------|
| 5 | 5 | 5 | 5 | 7        |
| 6 | 6 | 6 | 5 | 5        |
| 5 | 5 | 5 | 5 | 1        |
| 7 | 5 | 4 | 6 | 4        |
| 5 | 5 | 5 | 5 | $5^{25}$ |

# 給定高度差

- 紿定高度差之後，等於修改了任何一點可以走的鄰居的條件
  - 必須在上下左右相鄰的格子
  - 必須兩者高度差  $\leq$  紿定的限制

$h\_difference \leq 1$

雖相鄰但不能走



# BFS (BREADTH FIRST SEARCH)(麵包優先搜尋)

- BFS就像漣漪
  - 從滴落的中心開始，向外一圈一圈擴展
  - 每一圈的距離是一步
- 對於 距離為  $d$  的點來說，它的鄰居
  - $d-1$  (走過的)
  - $d$  (正在波前上的)
  - $d+1$  (下一次的波前)
- 有不同的實現方式
- BFS可以計算距離，如果距離的定義是走幾步(每一步的長度無關)



# BFS

- 白色代表尚未被看到
- 被看到就變灰色
- 被探索完畢就變成黑色 (探索: 鄰居都看完了)
  
- 一開始每個人都純白無瑕
- 起點是灰色
- 每次拿出一個灰色的點，看看它的所有鄰居，讓它變黑

- **BFS**
- 每個人一開始都是純白無瑕的
- 當被人看到時，就變成灰色了
- 當被人看光光之後，就變黑了，變黑就沒人要看了
- (**Cormen, CLRS**)進queue前是白的，進queue變灰的，出queue黑掉了
- 與**BFS**與**Dijkstra/Prim**的差別在於
  - **BFS** 第一眼看到的就是**parent**，永遠不改
  - **Dijkstra** 有奶便是娘，誰給的好處多，就認他做**parent**
- **Dijkstra**如何處理降**key**的問題
  - 用移調夾呀，你是說吉他嗎？(誤)
  - 嶄惰是城市人一個好的天性

```
BFS(G, s) {  
    initialize vertices;  
    Q = {s}; // Q is a queue; initialize to s  
    while (Q not empty) {  
        u = RemoveTop(Q);  
        for each v ∈ u->adj {  
            if (v->color == WHITE)  
                v->color = GREY;  
                v->d = u->d + 1;  
                v->p = u;  
                Enqueue(Q, v);  
        }  
        u->color = BLACK;  
    }  
}
```

# 回到蓋步道

- 如果高度差已知，可以計算起點到終點的距離
  - 也可以算是否可以到達
- 太好了？但我們不知道高度差
- 我們可以用猜的
  - 高度差由 $0, 1, 2, 3, \dots$ ，對每個高度差限制，做一次**BFS**，第一次可以到達者就是最小高度差，**BFS**同時也求出距離了
- 高度差100萬，做100萬次**BFS**太慢，二分搜
  - 因為當高度差放寬時，可以走的還是可以走，所以在最小高度差以上，都是走得到
- 結論：**BFS** + 二分搜

```

1 // BFS + binary search
2 #include <bits/stdc++.h>
3 using namespace std;
4 typedef long long ll;
5 define N 310
6 define oo 2000001
7 struct POS {
8     int r,c;
9 };
10 int grid[N][N], dr[4]={0,1,0,-1}, dc[4]={1,0,-1,0};
11 // BFS, source=(1,1), dest=(n,n), return distance or
12 int bfs(int n, int slope) {
13     int i,j, d[N][N];
14     for (i=1;i<=n;i++) for (j=1;j<=n;j++)
15         d[i][j] = oo;
16     d[1][1] = 0;
17     queue<POS> Q;

```

```

18     Q.push({1,1});
19     while (!Q.empty() && d[n][n]==oo) {
20         auto e=Q.front();
21         Q.pop();
22         for (int i=0; i<4; i++) {
23             int r=e.r+dr[i], c=e.c+dc[i];
24             if (d[r][c]==oo && \
25                 abs(grid[r][c]-grid[e.r][e.c])<=slope) {
26                 Q.push({r,c});
27                 d[r][c] = d[e.r][e.c]+1;
28             }
29         }
30     }
31     return d[n][n];

```

```
34 int main() {
35     int i, j, n, ans=-1;
36     scanf("%d", &n);
37     for (i=0; i<=n+1; i++) grid[0][i]=grid[i][0]=oo;
38     for (i=0; i<=n+1; i++) grid[n+1][i]=grid[i][n+1]=oo;
39     for (i=1; i<=n; i++) for (j=1; j<=n; j++) {
40         scanf("%d", &grid[i][j]);
41     }
42     // find the max slope cannot reach
43     for (int jump=oo/2; jump>0; jump>>=1) {
44         while (ans+jump<oo && bfs(n, ans+jump)==oo) {
45             ans += jump;
46         }
47     }
48     printf("%d\n%d\n", ans+1, bfs(n, ans+1));
49     return 0;
50 }
```

# 再訪蓋步道 -- 進階作法

- 第二步求最短路徑沒甚麼好說的了，**BFS**就是簡單又快的方法。
- 第一步找最小高度差是有其他方法可以做的
  - 可以視為找將起點連通到終點的 **mini-max spanning tree**
    - **Prim algorithm**的變化形
    - 也可以看成距離是**mini-max**形式的最短路徑
      - **Dijkstra algroithm**的變化形
  - 事實上**Prim**與**Dijkstra**本來就只有**cost function**不同而已
  - 與**BFS**也極其相似

# DIJKSTRA/PRIM ALGORITHM

(因為超過APCS程度，這裡僅略述，可參考AP325)

- 與BFS相似，差別在於
  - BFS中，第一眼看到的就是**parent**，永遠不改
  - Dijkstra中，有奶便是娘，誰給的好處多，就認他做**parent**
- 在變黑之前點的**d**值會改變，因此我們必須把**queue**換成**priority\_queue**(優先佇列，PQ)，與**queue**不同，**key**小的會先出來。
  - C++ 與Python中都有PQ可叫用
- 麻煩點：在PQ中的點**key**值改變，但這些現成的資料結構都不提供變更**key**值的方法！
- 真是糟糕了，難到要自己寫？

# 懶惰是城市人一個好的天性 --- 除了宅之外

- 我們不必太勤勞去PQ中改變**key**值，傻傻地把新的**key**丟進去就好了
- 反正小的(我們要的)會先出來，大的沉在裡面做肥料無所謂
- 但是，在出口要檢查，出來的是想要的本尊，還是本尊已經出來所剩下的分身
- 這樣的懶惰概念適用的場合不少。
- 看程式碼就可以具體了解。

# 二分檢測

- 當有大量的檢測要做，陽性的機率很低時，我們可以將多個檢測放在一起測，檢測必須滿足：
  - 多個陰性放在一起還是陰性
  - 一群中只要有一個陽性就會是陽性
- Covid-19核酸檢測據說有類似的做法

有 1000 个瓶子，其中 999 瓶是水，1 瓶是毒药，外观无法区别。现有 10 只小白鼠和无限多的干净试管，你怎么找出那瓶毒药？

(微软中国 2012 面试题，

杨超越吧

17 792 2049

全部回复 只看楼主 热门

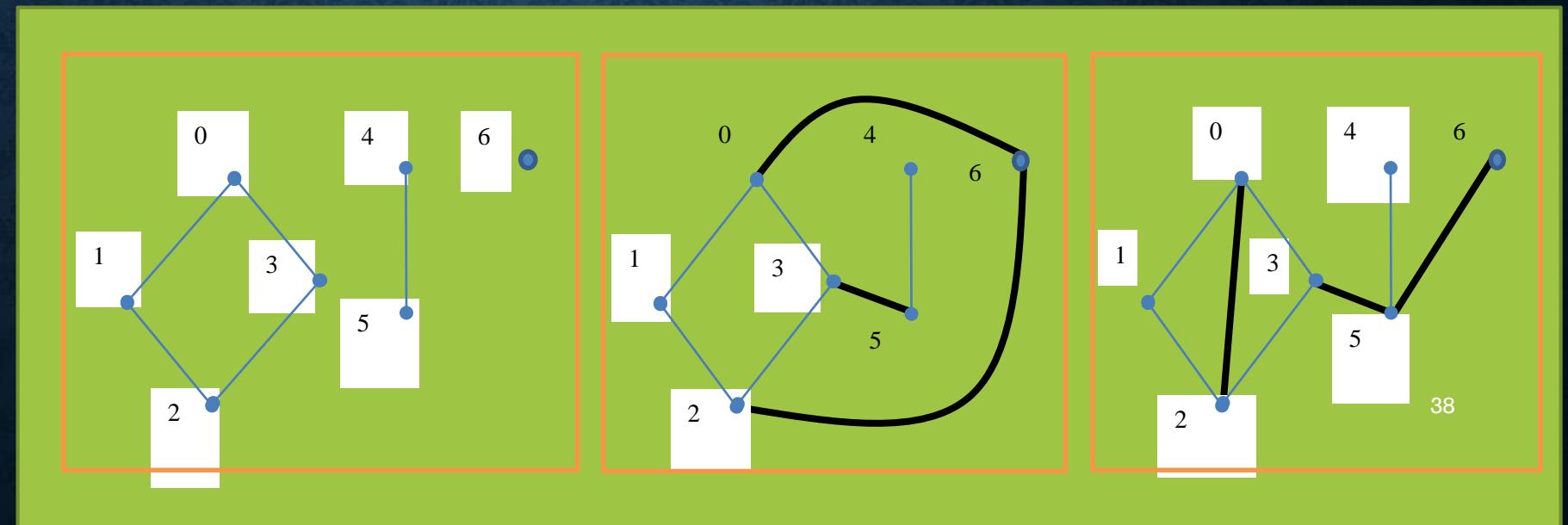
s石小天 10-20

老题型了

直接倒掉一瓶，然后弄死一直小白鼠。就说已经处理有毒的了。如果后续有人喝出问题就说造谣。

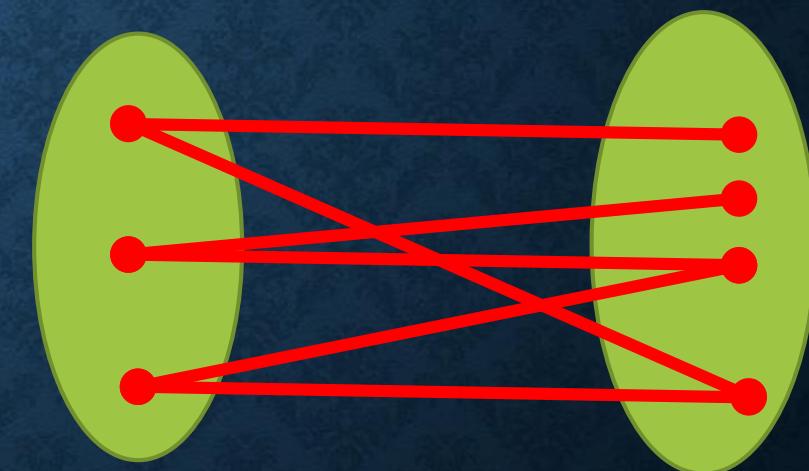
# 真假子圖(AP202111Q4)

- 有一些邊的集合(子圖)取自一張二分圖(**bipartite graph**)，其中一張大子圖與若干小子圖，另外有不超過3張假的子圖。真子圖取自原圖，所以聯集起來都是二分圖，但任何一張假子圖與大子圖聯集都不是二分圖。
- 要找出哪些小圖是假圖



# 二分圖

- 一個**graph**的所有點可以分成兩個獨立集，則此圖稱為二分圖
- 圖的著色：將圖的點著色，滿足相鄰的點不同色，稱為正確的著色。也就是將塗分成最少的獨立集。二分圖就是可以兩色著色的圖。
- 二分圖檢測：任選一點開始給予0色，鄰居給1，重複此步驟將以著色點的鄰居給予不同色，如果可以做完就是二分圖，否則就一定不是。
- 如果圖不連通，對每一個連通區塊各別做。
- 容易證明，因為步驟中沒有可變化的餘地。
- 同一獨立集的點之間沒有邊



# 真假子圖

- 一個圖如果是二分圖，刪除邊也必然還是；  
一個圖如果不是二分圖，再加邊也必然不是。
- 因為題目保證所有真圖的聯集是個二分圖，  
而任一假圖與大圖的聯集是非二分圖。
- 因為假圖很少，所以可以組合再一起測，  
計算成本比較低。

```
3 using namespace std;
4 #define N 150010
5 #define M 10010
6 vector<int> g0[N];
7 vector<pair<int,int>> gi[M]; // edge list
8 int col[N], n;
9 vector<int> ans;
10 bool dfs(int v, int c) {
11     col[v]=c;
12     for (int u: g0[v]) {
13         if (col[u]==c) return false;
14         if (col[u]==-c) continue;
15         // color[u]==0
16         if (!dfs(u,-c)) return false;
17     }
18     return true;
19 }
```

```
21 bool bipartite() {
22     for (int i=0;i<n;i++) col[i]=0; // 0:unvisit, 1, -1
23     for (int i=0;i<n;i++) {
24         if (col[i]==0) {
25             if (!dfs(i, 1)) return false;
26         }
27     }
28     return true;
29 }
```

```
31 void bsearch(int left, int right) {
32     for (int i=left; i<=right; i++) {
33         for (auto e: gi[i]) {
34             g0[e.first].push_back(e.second);
35             g0[e.second].push_back(e.first);
36         }
37     }
38     bool none=bipartite();
39     // recover
40     for (int i=left; i<=right; i++) {
41         for (auto e: gi[i]) {
42             g0[e.first].pop_back();
43             g0[e.second].pop_back();
44         }
45     }
46     if (none) return;
47 }
```

```
47 if (left==right) {
48     ans.push_back(left);
49 } else {
50     int mid=(left+right)/2;
51     bsearch(left,mid);
52     bsearch(mid+1,right);
53 }
54 return;
55 }
```

```
57 int main() {
58     int i, j, m, u, v, p, k;
59     scanf("%d%d", &n, &m);
60     for (i=0; i<m; i++) {
61         scanf("%d%d", &u, &v);
62         g0[u].push_back(v);
63         g0[v].push_back(u);
64     }
65     scanf("%d%d", &p, &k); // p subgraph with k edges
66     for (i=0; i<p; i++) {
67         for (j=0; j<k; j++) {
68             scanf("%d%d", &u, &v);
69             gi[i].push_back({u, v});
70         }
71     }
72     bsearch(0, p-1);
73     sort(ans.begin(), ans.end());
74     for (int x: ans) {
75         printf("%d\n", x+1);
76     }
77     return 0;
78 }
```

# NHSPC109 PG矩陣相乘

- 給兩個 $N \times N$ 的方陣  $A$  與  $B$ ，已知  $C = A \times B$  最多只有  $2N$  個非零項，求  $C$ 。
  - 模  $P$  運算， $? < P \leq 5e7$  是質數
  - 提示不可用太多 `mod`，一次內積只用一個 `mod`， $N \times P^2 < 2^{63}$
- 子題
  - 子題1.  $C$  的每一列恰有一個非零項
  - 子題2. 每一列兩個
  - 子題3. 一列可能多個，總共  $2N$  個

$C[i][j] = A$  的第  $i$  列 與  $B$  的第  $j$  行 內積

# 第一子題，C的每一列恰有一個非零

- Let  $A[i]$  be the  $i$ -th row vector of  $A$  and  $B[j]$  the  $j$ -th column vector of  $B$
- 分配率： $A[i] * \text{sum}(B[j]) = \text{sum}(A[i] * B[j]) = x_i$ , the non-zero in  $i$ -th row of  $C$
- Weighing  $j$ -th column by  $j$ :
  - $A[i] * \text{sum}(j*B[j]) = \text{sum}(A[i] * (j*B[j])) = j*x_i$
- 做2個內積可以找到  $x_i$  與  $j*x_i$ ，然後可以算出  $j$  與  $x_i$  (using inverse)
- Total complexity  $O(n^2 + n \log P)$

# 第一子題，C的每一列恰有一個非零

- Another approach
- 先看看網路上一個笑話
  - 如何找出一千個瓶子中的一瓶毒藥
  - 對於一個列向量  $A[i]$ ，要在  $N$  個行向量找出其中一個  $B[j]$ ，使得  
 $A[i] * B[j] \neq 0$ 
    - 找  $B[j]$  其實就是找毒藥
  - 對於一個區間  $[c1, c2]$ ，若  
 $A[i] * \text{sum}(B[c1:c2]) \neq 0$ ，則毒藥在  $[c1:c2]$  中



# 二分搜 SEARCH( $C_1, C_2$ ) FOR SUB

- 利用前綴和可以快速求出任意 $[c_1, c_2]$  區間之和向量 $B[c_1:c_2]$ ，做一次內積可檢定毒藥是否在其中
- 只需要 $\log(n)$ 次向量內積，可以找出所對應的行
- 整體複雜度 $O(n^2 \log(n))$



# 第二子題：每列恰好(至多)兩個

- 二分搜的大哥：二分繼續搜
- 二分繼續搜 `search(c1, c2)`
  - If ( $A[i] * \text{sum}(B[c1:c2]) == 0$ ) then return;
  - if ( $c1 == c2$ ) then found and return;
  - $\text{mid} = (c1+c2)/2;$
  - `search(c1, mid);`
  - `search(mid+1, c2);`

B的轉置矩陣



[ $c1, c2$ ] 區間向量之和

$$\left[ \begin{array}{c} \text{[c1, c2] 区间向量之和} \\ \text{*} \end{array} \right] \times \left[ \begin{array}{c} A[i] \end{array} \right]$$

$= 0$ , 我看你沒有  
 $\neq 0$ , 二分繼續搜  
`search(c1, mid);`  
`search(mid+1, c2);`

## 第二子題：每列恰好(至多)兩個

- 二分繼續搜用在第一子題沒問題。用在第二子題複雜度還是 $O(n^2 \log(n))$ ，因為每個被找出的位置只需要花 $\log(n)$ 次測試。此法成功的機率很大，但不能保證成功，因為：
  - 因為可能有壞人，讓以下狀況發生：  
 $A[i] * (B[j] + B[k]) = 0$  但  $A[i] * B[j] \neq 0$  且  $A[i] * B[k] \neq 0$ ！  
有  $1/P$  的機會，有壞人就會這麼巧
  - 你看沒有的區間可能藏匿了兩個要找的對象
- 對抗壞人：加權，將每個  $B[j]$  乘上  $j$ ，再做一次
  - 若  $x + y = 0$  且  $jx + ky = 0$ ，則  $x = y = 0$ , unless  $j=k \pmod P$
- 將原始行向量搜一次，加權後再搜一次。若  $A[i] * B[j] \neq 0$  則壞人無法讓兩次都搜不到
  - 只要先搜位置，最後再做內積計算值就可以了

# 每列不定多少個，但總共 $2N$ 個

- Randomized algorithm：
  - 每次加權用隨機數做加權
  - 二分繼續搜是單邊誤差的隨機演算法
    - 測到非零時不會錯，測到0時(false negative)有 $1/P$ 的機率是錯的。
- 可以重複測試來降低錯誤的機率。計算順序很重要
- 比較好的計算順序：檢測區間時，針對所有列與所有加權
  - $P \geq 37$ ，False negative的機率是 $1/37$ ，加權做五次錯誤的機率是 $1.5e-8$ ， $n=2800$ 最多要做1400個檢測，失敗的機率大約是5萬分之一
  - 真的那麼倒楣？再submit一次，失敗的機率是25億分之一透了
- 比較差的計算順序：每加權一次做一次二分繼續搜，重複執行
  - 時間足夠做12次以上。  
(其實只要能做5~6次，送幾次就會過，無人能擋 XD)。

# 我們與二分搜的距離

謝謝

