加快速度的想法

- 用迴圈一個一個房間的點數慢慢加，直到大於等於 q_j 是最直接最基本的作法

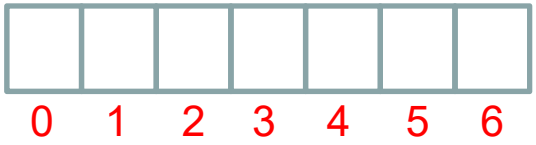
加快速度的想法

- 用迴圈一個一個房間的點數慢慢加，直到大於等於 q_j 是最直接最基本的作法，但是 $q_j \le p_0+p_1\ldots+p_{n-1}$ 的意思是說有可能所有房間都走過一次才能夠滿足一個任務 q_j

加快速度的想法

- 用迴圈一個一個房間的點數慢慢加，直到大於等於 q_j 是最直接最基本的作法，但是 $q_j \le p_0+p_1\ldots+p_{n-1}$ 的意思是說有可能所有房間都走過一次才能夠滿足一個任務 q_j

範例一輸入
7 3



加快速度的想法

- 用迴圈一個一個房間的點數慢慢加，直到大於等於 q_j 是最直接最基本的作法，但是 $q_j \le p_0+p_1\ldots+p_{n-1}$ 的意思是說有可能所有房間都走過一次才能夠滿足一個任務 q_j

範例一輸入
7 3
2 1 5 4 3 5 3

加快速度的想法

範例一輸入
7 3
2 1 5 4 3 5 3
8

- 用迴圈一個一個房間的點數慢慢加，直到大於等於 q_j 是最直接最基本的作法，但是 $q_j \le p_0+p_1\ldots+p_{n-1}$ 的意思是說有可能所有房間都走過一次才能夠滿足一個任務 q_j
- 迴圈過程中可以預期右邊這些數字在加總過程中出現，q_j 比較小的時候，灰色數字並沒有真的算出來

加快速度的想法

範例一輸入
7 3
2 1 5 4 3 5 3
8

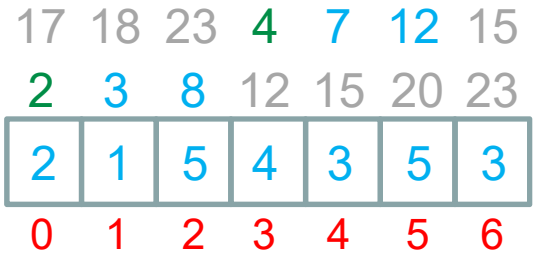
- 用迴圈一個一個房間的點數慢慢加，直到大於等於 q_j 是最直接最基本的作法，但是 $q_j \le p_0+p_1\ldots+p_{n-1}$ 的意思是說有可能所有房間都走過一次才能夠滿足一個任務 q_j
- 迴圈過程中可以預期右邊這些數字在加總過程中出現，q_j 比較小的時候，灰色數字並沒有真的算出來

- 考慮 q_1=**8**，目標是在 2,3,8,…,23 **(數列 S)** 裡找 **≥ 8** 的第一個數，因為是一個遞增數列，可以直接使用二分法來快速搜尋

加快速度的想法

範例一輸入
7 3
2 1 5 4 3 5 3
8 9

- 用迴圈一個一個房間的點數慢慢加，直到大於等於 q_j 是最直接最基本的作法，但是 $q_j \le p_0+p_1\ldots+p_{n-1}$ 的意思是說有可能所有房間都走過一次才能夠滿足一個任務 q_j
- 迴圈過程中可以預期右邊這些數字在加總過程中出現，q_j 比較小的時候，灰色數字並沒有真的算出來

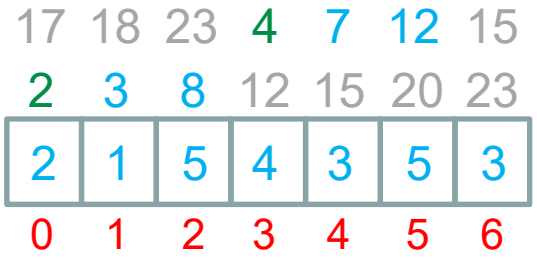


- 考慮 q_1=**8**，目標是在 2,3,8,…,23 **(數列 S)** 裡找 **≥ 8** 的第一個數，因為是一個遞增數列，可以直接使用二分法來快速搜尋
- 考慮 q_2=**9**，目標是在 4,7,12,…,23 裡找 **≥ 9** 的第一個數，因為還是一個遞增數列，仍然可以直接使用二分法

加快速度的想法

範例一輸入
7 3
2 1 5 4 3 5 3
8 9

- 用迴圈一個一個房間的點數慢慢加，直到大於等於 q_j 是最直接最基本的作法，但是 $q_j \le p_0+p_1\ldots+p_{n-1}$ 的意思是說有可能所有房間都走過一次才能夠滿足一個任務 q_j
- 迴圈過程中可以預期右邊這些數字在加總過程中出現，q_j 比較小的時候，灰色數字並沒有真的算出來



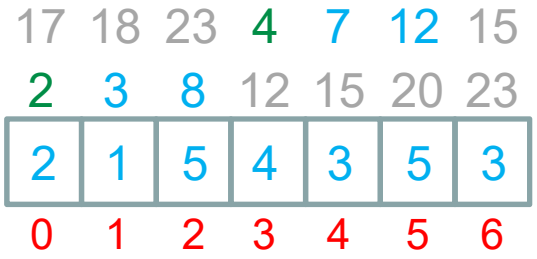
- 考慮 q_1=**8**，目標是在 2,3,8,…,23 **(數列 S)** 裡找 **≥ 8** 的第一個數，因為是一個遞增數列，可以直接使用二分法來快速搜尋
- 考慮 q_2=**9**，目標是在 4,7,12,…,23 裡找 **≥ 9** 的第一個數，因為還是一個遞增數列，仍然可以直接使用二分法

  這個問題可以等效為：在 12,15,20,23,25,26,31 裡用二分法尋找 ≥ 8+**9** 的第一個數，這個數列前半和數列 **S** 後半是一樣的，後半如果 mod 23 的話，就是數列 **S** 的前半

加快速度的想法

範例一輸入
7 3
2 1 5 4 3 5 3
8 9

- 用迴圈一個一個房間的點數慢慢加，直到大於等於 q_j 是最直接最基本的作法，但是 $q_j \le p_0+p_1\ldots+p_{n-1}$ 的意思是說有可能所有房間都走過一次才能夠滿足一個任務 q_j
- 迴圈過程中可以預期右邊這些數字在加總過程中出現，q_j 比較小的時候，灰色數字並沒有真的算出來



- 考慮 q_1=**8**，目標是在 2,3,8,…,23 **(數列 S)** 裡找 **≥ 8** 的第一個數，因為是一個遞增數列，可以直接使用二分法來快速搜尋
- 考慮 q_2=**9**，目標是在 4,7,12,…,23 裡找 **≥ 9** 的第一個數，因為還是一個遞增數列，仍然可以直接使用二分法

這個問題可以等效為：在 12,15,20,23,25,26,31 裡用二分法尋找 ≥ 8+**9** 的第一個數，這個數列前半和數列 **S** 後半是一樣的，後半如果 mod 23 的話，就是數列 **S** 的前半 ⇒ 先算出數列 **S**

加快速度的想法

範例一輸入
7 3
2 1 5 4 3 5 3
8 9 12

- 用迴圈一個一個房間的點數慢慢加，直到大於等於 q_j 是最直接最基本的作法，但是 $q_j \le p_0+p_1\ldots+p_{n-1}$ 的意思是說有可能所有房間都走過一次才能夠滿足一個任務 q_j
- 迴圈過程中可以預期右邊這些數字在加總過程中出現，q_j 比較小的時候，灰色數字並沒有真的算出來
- 考慮 q_1=**8**，目標是在 2,3,8,…,23 **(數列 S)** 裡找 **≥ 8** 的第一個數，因為是一個遞增數列，可以直接使用二分法來快速搜尋
- 考慮 q_2=**9**，目標是在 4,7,12,…,23 裡找 **≥ 9** 的第一個數，因為還是一個遞增數列，仍然可以直接使用二分法

  這個問題可以等效為：在 12,15,20,23,25,26,31 裡用二分法尋找 ≥ 8+**9** 的第一個數，這個數列前半和數列 **S** 後半是一樣的，後半如果 mod 23 的話，就是數列 **S** 的前半 ⇒ 先算出數列 **S**

二分法加速模擬

```
01 #include <cstdio>
02 #include <algorithm>
03 using namespace std;
```

範例一輸入
7 3
2 1 5 4 3 5 3
8 9 12

二分法加速模擬

```
01 #include <cstdio>
02 #include <algorithm>
03 using namespace std;
04 int s0[200001], *s=s0+1;
```

數列 **S**

陣列 s[-1], s[0],…, s[200000]

範例一輸入
7 3
2 1 5 4 3 5 3
8 9 12

二分法加速模擬

範例一輸入
7 3
2 1 5 4 3 5 3
8 9 12

```
01 #include <cstdio>
02 #include <algorithm>
03 using namespace std;
                                  數列 S
04 int s0[200001], *s=s0+1;

05 int main() {
06     int i, n, m, p, q, *end;
07     while (2==scanf("%d%d", &n, &m)) {
08         for (s[-1]=i=0; i<n; i++)
09             scanf("%d", &p), s[i]=s[i-1]+p;
                                  先加出部份和一次
17     }
18     return 0;
19 }
```

二分法加速模擬

```
01 #include <cstdio>
02 #include <algorithm>
03 using namespace std;
                              數列 S
04 int s0[200001], *s=s0+1;

05 int main() {
06     int i, n, m, p, q, *end;
07     while (2==scanf("%d%d", &n, &m)) {
08         for (s[-1]=i=0; i<n; i++)
09             scanf("%d", &p), s[i]=s[i-1]+p;
                                  先加出部份和一次
17     }
18     return 0;
19 }
```

範例一輸入
7 3
2 1 5 4 3 5 3
8 9 12

- s 整數陣列: s[i] = $p_0+p_1\ldots+p_i$

二分法加速模擬

```
01 #include <cstdio>
02 #include <algorithm>
03 using namespace std;
                                  數列 S
04 int s0[200001], *s=s0+1;

05 int main() {
06     int i, n, m, p, q, *end;
07     while (2==scanf("%d%d", &n, &m)) {
08         for (s[-1]=i=0; i<n; i++)
09             scanf("%d", &p), s[i]=s[i-1]+p;
                                  先加出部份和一次
17     }
18     return 0;
19 }
```

範例一輸入
7 3
2 1 5 4 3 5 3
8 9 12

題目保證 $p_0+p_1\ldots+p_{n-1}\leq 10^9$

- s 整數陣列: $s[i] = p_0+p_1\ldots+p_i$

二分法加速模擬

```
01 #include <cstdio>
02 #include <algorithm>
03 using namespace std;
                                  數列 S
04 int s0[200001], *s=s0+1;

05 int main() {
06     int i, n, m, p, q, *end;
07     while (2==scanf("%d%d", &n, &m)) {
08         for (s[-1]=i=0; i<n; i++)
09             scanf("%d", &p), s[i]=s[i-1]+p;
                                  先加出部份和一次
17     }
18     return 0;
19 }
```

範例一輸入
7 3
2 1 5 4 3 5 3
8 9 12

題目保證 $p_0+p_1\ldots+p_{n-1}\leq 10^9$

- s 整數陣列: $s[i] = p_0+p_1\ldots+p_i$
- 令 s[-1]=s[n-1]%s[n-1]=0

二分法加速模擬

```
01 #include <cstdio>
02 #include <algorithm>
03 using namespace std;

04 int s0[200001], *s=s0+1;

05 int main() {
06     int i, n, m, p, q, *end;
07     while (2==scanf("%d%d", &n, &m)) {
08         for (s[-1]=i=0; i<n; i++)
09             scanf("%d", &p), s[i]=s[i-1]+p;
10         for (end=s-1,i=0; i<m; i++) {
11             scanf("%d", &q); 任務 qj




15         }

17     }
18     return 0;
19 }
```

範例一輸入
7 3
2 1 5 4 3 5 3
8 9 12

題目保證 $p_0+p_1\ldots+p_{n-1}\le 10^9$

- s 整數陣列: $s[i] = p_0+p_1\ldots+p_i$
- 令 s[-1]=s[n-1]%s[n-1]=0

end 指向任務開始時的前一個 s[i], 代表前一任務結束時得到的總點數 % s[n-1]

二分法加速模擬

```
01 #include <cstdio>
02 #include <algorithm>
03 using namespace std;

04 int s0[200001], *s=s0+1;

05 int main() {
06     int i, n, m, p, q, *end;
07     while (2==scanf("%d%d", &n, &m)) {
08         for (s[-1]=i=0; i<n; i++)
09             scanf("%d", &p), s[i]=s[i-1]+p;
10         for (end=s-1,i=0; i<m; i++) {
11             scanf("%d", &q);
12             if (*end+q>s[n-1]) 在序列前半嗎?
13                 q-=s[n-1]-*end,

15         }

17     }
18     return 0;
19 }
```

範例一輸入
7 3
2 1 5 4 3 5 3
8 9 12

題目保證 $p_0+p_1\ldots+p_{n-1}\le 10^9$

- s 整數陣列: s[i] = $p_0+p_1\ldots+p_i$
- 令 s[-1]=s[n-1]%s[n-1]=0

end 指向任務開始時的前一個 s[i], 代表前一任務結束時得到的總點數 % s[n-1]

- 此時 ***end+q** 是搜尋的目標
  - 超過 s[n-1] 的部份先由 q 中扣掉

二分法加速模擬

```
01 #include <cstdio>
02 #include <algorithm>
03 using namespace std;

04 int s0[200001], *s=s0+1;

05 int main() {
06     int i, n, m, p, q, *end;
07     while (2==scanf("%d%d", &n, &m)) {
08         for (s[-1]=i=0; i<n; i++)
09             scanf("%d", &p), s[i]=s[i-1]+p;
10         for (end=s-1,i=0; i<m; i++) {
11             scanf("%d", &q);
12             if (*end+q>s[n-1]) 在序列前半嗎?
13                 q-=s[n-1]-*end, q%=s[n-1],


15         }


17     }
18     return 0;
19 }
```

範例一輸入
7 3
2 1 5 4 3 5 3
8 9 12

題目保證 $p_0+p_1\ldots+p_{n-1}\le 10^9$

- s 整數陣列: s[i] = $p_0+p_1\ldots+p_i$
- 令 s[-1]=s[n-1]%s[n-1]=0

end 指向任務開始時的前一個 s[i], 代表前一任務結束時得到的總點數 % s[n-1]

- 此時 ***end+q** 是搜尋的目標
  - 超過 s[n-1] 的部份先由 q 中扣掉
    - 剩餘部份如果超過 s[n-1] 的話，直接扣除 s[n-1]; 其實題目已經保證 q_j<s[n-1]，所以這個敘述沒有作用

二分法加速模擬

```
01 #include <cstdio>
02 #include <algorithm>
03 using namespace std;

04 int s0[200001], *s=s0+1;

05 int main() {
06     int i, n, m, p, q, *end;
07     while (2==scanf("%d%d", &n, &m)) {
08         for (s[-1]=i=0; i<n; i++)
09             scanf("%d", &p), s[i]=s[i-1]+p;
10         for (end=s-1,i=0; i<m; i++) {
11             scanf("%d", &q);
12             if (*end+q>s[n-1])
13                 q-=s[n-1]-*end, q%=s[n-1], end=s-1;


15         }


17     }
18     return 0;
19 }
```

在序列前半嗎?

範例一輸入
7 3
2 1 5 4 3 5 3
8 9 12

題目保證 $p_0+p_1\ldots+p_{n-1}\le 10^9$

- s 整數陣列: s[i] = $p_0+p_1\ldots+p_i$
- 令 s[-1]=s[n-1]%s[n-1]=0

end 指向任務開始時的前一個 s[i], 代表前一任務結束時得到的總點數 % s[n-1]

- 此時 ***end+q** 是搜尋的目標
  - 超過 s[n-1] 的部份先由 q 中扣掉
    - 剩餘部份如果超過 s[n-1] 的話，直接扣除 s[n-1]; 其實題目已經保證 q_j<s[n-1]，所以這個敘述沒有作用
    - 由最前頭 s[-1] 開始搜尋

二分法加速模擬

```
01 #include <cstdio>
02 #include <algorithm>
03 using namespace std;

04 int s0[200001], *s=s0+1;

05 int main() {
06     int i, n, m, p, q, *end;
07     while (2==scanf("%d%d", &n, &m)) {
08         for (s[-1]=i=0; i<n; i++)
09             scanf("%d", &p), s[i]=s[i-1]+p;
10         for (end=s-1,i=0; i<m; i++) {
11             scanf("%d", &q);
12             if (*end+q>s[-1])
13                 q-=s[n-1]-*end, q%=s[n-1], end=s-1;
14             end = lower_bound(end+1, s+n, *end+q);
15         }

17     }
18     return 0;
19 }
```

範例一輸入
7 3
2 1 5 4 3 5 3
8 9 12

題目保證 $p_0+p_1\ldots+p_{n-1}\le 10^9$

- s 整數陣列: s[i] = $p_0+p_1\ldots+p_i$
- 令 s[-1]=s[n-1]%s[n-1]=0

end 指向任務開始時的前一個 s[i], 代表前一任務結束時得到的總點數 % s[n-1]

- 此時 ***end+q** 是搜尋的目標
  - 超過 s[n-1] 的部份先由 q 中扣掉
    - 剩餘部份如果超過 s[n-1] 的話，直接扣除 s[n-1]; 其實題目已經保證 q_j<s[n-1]，所以這個敘述沒有作用
  - 由最前頭 s[-1] 開始搜尋

- 在 [end+1, s+n) 範圍內以二分法搜尋第一個 ≥***end+q** 的元素

二分法加速模擬

範例一輸入
7 3
2 1 5 4 3 5 3
8 9 12

範例一輸出
4

```
01 #include <cstdio>
02 #include <algorithm>
03 using namespace std;

04 int s0[200001], *s=s0+1;

05 int main() {
06     int i, n, m, p, q, *end;
07     while (2==scanf("%d%d", &n, &m)) {
08         for (s[-1]=i=0; i<n; i++)
09             scanf("%d", &p), s[i]=s[i-1]+p;
10         for (end=s-1,i=0; i<m; i++) {
11             scanf("%d", &q);
12             if (*end+q>s[n-1])
13                 q-=s[n-1]-*end, q%=s[n-1], end=s-1;
14             end = lower_bound(end+1, s+n, *end+q);
15         }
16         printf("%d\n", (end+1-s)%n);
17     }
18     return 0;
19 }
```

題目保證 $p_0+p_1\ldots+p_{n-1}\le 10^9$

- s 整數陣列: s[i] = $p_0+p_1\ldots+p_i$
- 令 s[-1]=s[n-1]%s[n-1]=0

end 指向任務開始時的前一個 s[i], 代表前一任務結束時得到的總點數 % s[n-1]

- 此時 ***end+q** 是搜尋的目標
  - 超過 s[n-1] 的部份先由 q 中扣掉
    - 剩餘部份如果超過 s[n-1] 的話，直接扣除 s[n-1]; 其實題目已經保證 q_j<s[n-1]，所以這個敘述沒有作用
  - 由最前頭 s[-1] 開始搜尋
- 輸出陣列的索引值
- 在 [end+1, s+n) 範圍內以二分法搜尋第一個 ≥***end+q** 的元素