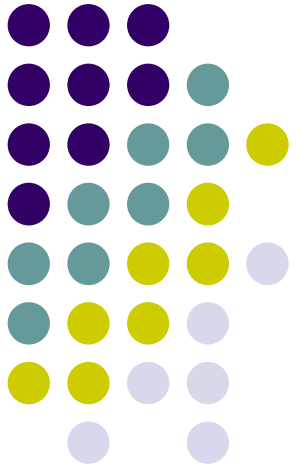
基地台

APCS 106/03 #4

Pei-yih Ting



問題描述

範例一輸入	範例二輸入
5 **2** 5 1 2 8 7	5 **1** 7 5 1 2 8
範例一輸出	範例二輸出
3	**7**

問題描述

範例一輸入	範例二輸入
5 **2** 5 1 2 8 7	5 **1** 7 5 1 2 8
範例一輸出	**範例二輸出**
3	**7**

- 上圖代表一條直線道路, 圖中 1,2,5,7,8 的位置上有建築物, 希望架設 **K** 個服務半徑 **R** 的基地台來滿足所有 **N** 個建築物的需求

問題描述

範例一輸入	範例二輸入
5 **2** 5 1 2 8 7	5 **1** 7 5 1 2 8
範例一輸出	**範例二輸出**
3	**7**

- 上圖代表一條直線道路, 圖中 1,2,5,7,8 的位置上有建築物, 希望架設 **K** 個服務半徑 **R** 的基地台來滿足所有 **N** 個建築物的需求
- 例如：
  - 假設 **K** 為 1, 則基地台應該架設在座標 4.5 的位置, 所有建築物與基地台的距離都在半徑 3.5 以內, 因此基地台的最小服務直徑 **2R** 需要是 7

問題描述

範例一輸入	範例二輸入
5 **2** 5 1 2 8 7	5 **1** 7 5 1 2 8
範例一輸出	**範例二輸出**
3	**7**

- 上圖代表一條直線道路, 圖中 1,2,5,7,8 的位置上有建築物, 希望架設 **K** 個服務半徑 **R** 的基地台來滿足所有 **N** 個建築物的需求
- 例如：
  - 假設 **K** 為 1, 則基地台應該架設在座標 4.5 的位置, 所有建築物與基地台的距離都在半徑 3.5 以內, 因此基地台的最小服務直徑 **2R** 需要是 7
  - 假設 **K** 為 2, 在 0.5 到 2.5 之間架設一個基地台來服務位置 1 與 2 的建築物, 另一個基地台架在 6.5 的位置, 服務位置 5,7,8 的建築物, 基地台的最小服務直徑 **2R** 需要是 3

問題描述

範例一輸入	範例二輸入
5 **2** 5 1 2 8 7	5 **1** 7 5 1 2 8
範例一輸出	**範例二輸出**
3	**7**

- 上圖代表一條直線道路, 圖中 1,2,5,7,8 的位置上有建築物, 希望架設 **K** 個服務半徑 **R** 的基地台來滿足所有 **N** 個建築物的需求
- 例如：
  - 假設 **K** 為 1, 則基地台應該架設在座標 4.5 的位置, 所有建築物與基地台的距離都在半徑 3.5 以內, 因此基地台的最小服務直徑 **2R** 需要是 7
  - 假設 **K** 為 2, 在 0.5 到 2.5 之間架設一個基地台來服務位置 1 與 2 的建築物, 另一個基地台架在 6.5 的位置, 服務位置 5,7,8 的建築物, 基地台的最小服務直徑 **2R** 需要是 3
  - 在 **K** 為 3 時, 基地台的最小服務直徑 **2R** 需要是 1

問題描述

範例一輸入	範例二輸入
5 **2** 5 1 2 8 7	5 **1** 7 5 1 2 8
範例一輸出	**範例二輸出**
3	**7**

- 上圖代表一條直線道路, 圖中 1,2,5,7,8 的位置上有建築物, 希望架設 **K** 個服務半徑 **R** 的基地台來滿足所有 **N** 個建築物的需求
- 例如：
  - 假設 **K** 為 1, 則基地台應該架設在座標 4.5 的位置, 所有建築物與基地台的距離都在半徑 3.5 以內, 因此基地台的最小服務直徑 **2R** 需要是 7
  - 假設 **K** 為 2, 在 0.5 到 2.5 之間架設一個基地台來服務位置 1 與 2 的建築物, 另一個基地台架在 6.5 的位置, 服務位置 5,7,8 的建築物, 基地台的最小服務直徑 **2R** 需要是 3
  - 在 **K** 為 3 時, 基地台的最小服務直徑 **2R** 需要是 1
- 對於指定的 **K** 值,請找出架設時最短的基地台服務直徑 **2R**

問題分析

- 這個問題乍看之下有點像是叢集 (clustering) 分群的問題

問題分析

- 這個問題乍看之下有點像是叢集 (clustering) 分群的問題
  - 仔細想一下, 發現它的距離定義和中心的定義使得很難找到像是 k-means 的演算法來逐步找到各個子群, 也沒有用到各群連續的特點

問題分析

- 這個問題乍看之下有點像是叢集 (clustering) 分群的問題
  - 仔細想一下, 發現它的距離定義和中心的定義使得很難找到像是 k-means 的演算法來逐步找到各個子群, 也沒有用到各群連續的特點
- 不要由資料分群的角度來想, 直接看成尋找基地台涵蓋直徑的問題, 其實這個要尋找的直徑 **2R** (題目限制為整數)

問題分析

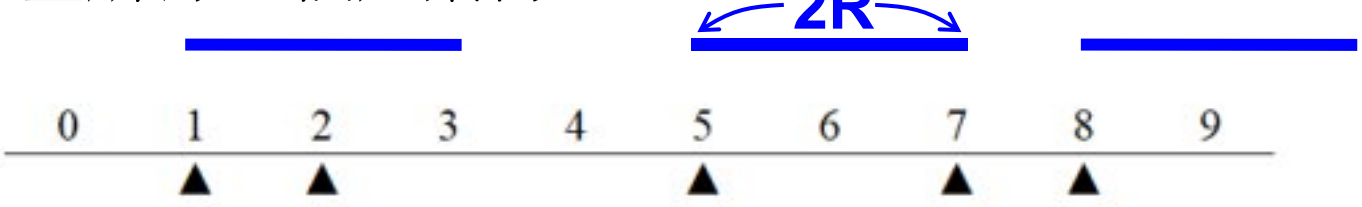
- 這個問題乍看之下有點像是叢集 (clustering) 分群的問題
  - 仔細想一下, 發現它的距離定義和中心的定義使得很難找到像是 k-means 的演算法來逐步找到各個子群, 也沒有用到各群連續的特點
- 不要由資料分群的角度來想, 直接看成尋找基地台涵蓋直徑的問題, 其實這個要尋找的直徑 **2R** (題目限制為整數)
  - 有下限 0 (如果 **K** 值 ≥ 所有位於不同位置的建築物數, **K** < **N**)

問題分析

- 這個問題乍看之下有點像是叢集 (clustering) 分群的問題
  - 仔細想一下, 發現它的距離定義和中心的定義使得很難找到像是 k-means 的演算法來逐步找到各個子群, 也沒有用到各群連續的特點
- 不要由資料分群的角度來想, 直接看成尋找基地台涵蓋直徑的問題, 其實這個要尋找的直徑 **2R** (題目限制為整數)
  - 有下限 0 (如果 **K** 值 ≥ 所有位於不同位置的建築物數, **K** < **N**)
  - 也有上限⌈**T**/**K**⌉(**K** 個基地台沒有重疊的 **2RK** 服務範圍涵蓋全部 0 到 **T** 區間)

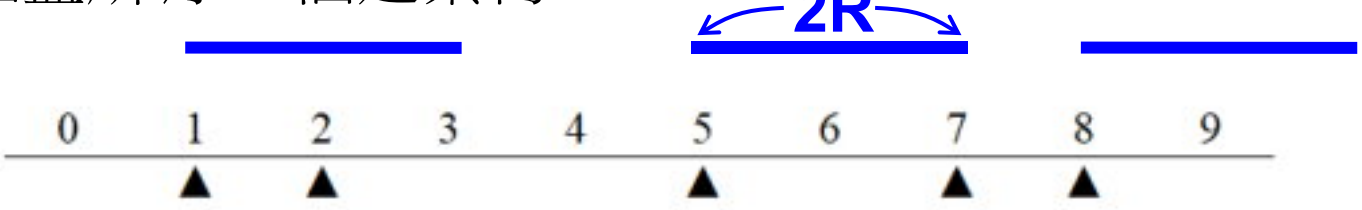
問題分析

- 這個問題乍看之下有點像是叢集 (clustering) 分群的問題
  - 仔細想一下, 發現它的距離定義和中心的定義使得很難找到像是 k-means 的演算法來逐步找到各個子群, 也沒有用到各群連續的特點
- 不要由資料分群的角度來想, 直接看成**尋找基地台涵蓋直徑**的問題, 其實這個要尋找的直徑 **2R** (題目限制為整數)
  - 有下限 0 (如果 **K** 值 ≥ 所有位於不同位置的建築物數, **K** < **N**)
  - 也有上限⌈**T**/**K**⌉(**K** 個基地台沒有重疊的 **2RK** 服務範圍涵蓋全部 0 到 **T** 區間)
- 指定一個 **R** 值, 我們可以很容易地判斷 **K** 個基地台是否可以涵蓋所有 **N** 個建築物



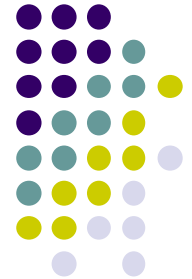
問題分析

- 這個問題乍看之下有點像是叢集 (clustering) 分群的問題
  - 仔細想一下, 發現它的距離定義和中心的定義使得很難找到像是 k-means 的演算法來逐步找到各個子群, 也沒有用到各群連續的特點
- 不要由資料分群的角度來想, 直接看成**尋找基地台涵蓋直徑**的問題, 其實這個要尋找的直徑 **2R** (題目限制為整數)
  - 有下限 0 (如果 **K** 值 ≥ 所有位於不同位置的建築物數, **K** < **N**)
  - 也有上限⌈**T**/**K**⌉(**K** 個基地台沒有重疊的 **2RK** 服務範圍涵蓋全部 0 到 **T** 區間)
- 指定一個 **R** 值, 我們可以很容易地判斷 **K** 個基地台是否可以涵蓋所有 **N** 個建築物



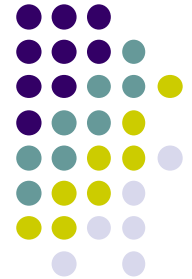
- 線性搜尋的話需要花的時間正比於 **T**, 二分搜尋則只需要花正比於 $\log_2$ **T** 的時間

實作



第 1 子題組 10 分, 座標範圍 T 不超過 100, 1<=K<=2, K<N<=10
第 2 子題組 20 分, 座標範圍 T 不超過 1000, 1<=K<N<=100
第 3 子題組 20 分, 座標範圍 T 不超過 1000000000, 1<=K<N<=500
第 4 子題組 50 分, 座標範圍 T 不超過 1000000000, 1<=K<N<=50000

實作



第 1 子題組 10 分, 座標範圍 T 不超過 100, 1<=K<=2, K<N<=10
第 2 子題組 20 分, 座標範圍 T 不超過 1000, 1<=K<N<=100
第 3 子題組 20 分, 座標範圍 T 不超過 1000000000, 1<=K<N<=500
第 4 子題組 50 分, 座標範圍 T 不超過 1000000000, 1<=K<N<=50000

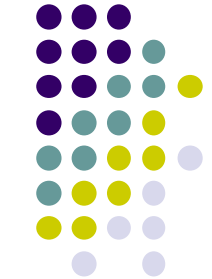
- 要在 **[0,⌈T/K⌉]** 範圍內所有整數中搜尋 **R**，有兩種方法：

實作

第 1 子題組 10 分, 座標範圍 T 不超過 100, 1<=K<=2, K<N<=10
第 2 子題組 20 分, 座標範圍 T 不超過 1000, 1<=K<N<=100
第 3 子題組 20 分, 座標範圍 T 不超過 1000000000, 1<=K<N<=500
第 4 子題組 50 分, 座標範圍 T 不超過 1000000000, 1<=K<N<=50000

- 要在 **[0,⌈T/K⌉]** 範圍內所有整數中搜尋 **R**，有兩種方法：
  1. 做一個大小為⌈T/K⌉的陣列裡面放 0,1,2,…,⌈T/K⌉，撰寫一個比對的函式判斷答案是否小於指定的數值，然後使用 std::lower_bound()

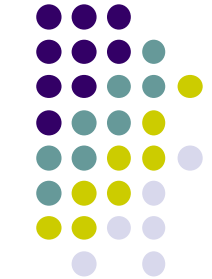
實作



第 1 子題組 10 分, 座標範圍 T 不超過 100, 1<=K<=2, K<N<=10
第 2 子題組 20 分, 座標範圍 T 不超過 1000, 1<=K<N<=100
第 3 子題組 20 分, 座標範圍 T 不超過 1000000000, 1<=K<N<=500
第 4 子題組 50 分, 座標範圍 T 不超過 1000000000, 1<=K<N<=50000

- 要在 **[0,⌈T/K⌉]** 範圍內所有整數中搜尋 **R**，有兩種方法：
  1. 做一個大小為⌈T/K⌉的陣列裡面放 0,1,2,…,⌈T/K⌉，撰寫一個比對的函式判斷答案是否小於指定的數值，然後使用 std::lower_bound()
     - 大部分時候使用 std::lower_bound() 是一個簡單有效的方法，但是第 3,4 子題中 T/K 的數值要求 500MB 以上的記憶體，就不能使用 std::lower_bound() 這種模組化的二分搜尋法了

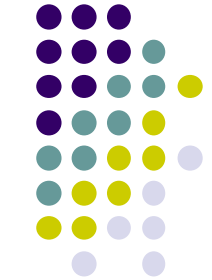
實作



第 1 子題組 10 分, 座標範圍 T 不超過 100, 1<=K<=2, K<N<=10
第 2 子題組 20 分, 座標範圍 T 不超過 1000, 1<=K<N<=100
第 3 子題組 20 分, 座標範圍 T 不超過 1000000000, 1<=K<N<=500
第 4 子題組 50 分, 座標範圍 T 不超過 1000000000, 1<=K<N<=50000

- 要在 **[0,⌈T/K⌉]** 範圍內所有整數中搜尋 **R**，有兩種方法：
  1. 做一個大小為⌈T/K⌉的陣列裡面放 0,1,2,…,⌈T/K⌉，撰寫一個比對的函式判斷答案是否小於指定的數值，然後使用 std::lower_bound()
     - 大部分時候使用 std::lower_bound() 是一個簡單有效的方法，但是第 3,4 子題中 T/K 的數值要求 500MB 以上的記憶體，就不能使用 std::lower_bound() 這種模組化的二分搜尋法了
  2. 類似下面的二分勘根法一樣寫一個二分的迴圈

實作

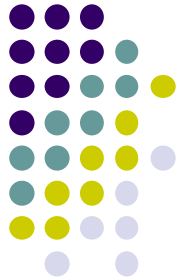


第 1 子題組 10 分, 座標範圍 T 不超過 100, 1<=K<=2, K<N<=10
第 2 子題組 20 分, 座標範圍 T 不超過 1000, 1<=K<N<=100
第 3 子題組 20 分, 座標範圍 T 不超過 1000000000, 1<=K<N<=500
第 4 子題組 50 分, 座標範圍 T 不超過 1000000000, 1<=K<N<=50000

- 要在 **[0,⌈T/K⌉]** 範圍內所有整數中搜尋 **R**，有兩種方法：
  1. 做一個大小為⌈T/K⌉的陣列裡面放 0,1,2,…,⌈T/K⌉，撰寫一個比對的函式判斷答案是否小於指定的數值，然後使用 std::lower_bound()
     - 大部分時候使用 std::lower_bound() 是一個簡單有效的方法，但是第 3,4 子題中 T/K 的數值要求 500MB 以上的記憶體，就不能使用 std::lower_bound() 這種模組化的二分搜尋法了
  2. 類似下面的二分勘根法一樣寫一個二分的迴圈

```
01 f_left = f(x_left), f_right = f(x_right);
02 while (x_right-x_left > 1.0e-10) {
03     x_mid = (x_left + x_right) / 2.0, f_mid = f(x_mid);
04     if (f_left * f_mid < 0.0)
05         x_right = x_mid, f_right = f_mid;
06     else
07         x_left = x_mid, f_left = f_mid;
08 }
```

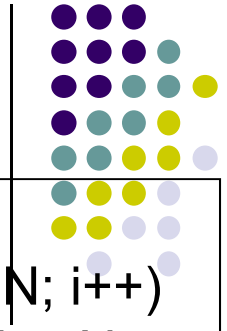
實作 (續)



實作 (續)

- 由於建築物的位置不見得依照順序、也可能相同，所以先要寫一小段程式處理，找到位置不同的建築物有幾個，同時也將位置陣列依照順序排好

```
std::sort(p, p+N);
for (distinctN=i=1; i<N; i++)
    if (p[i]!=p[i-1]) distinctN++;
```



實作 (續)

- 由於建築物的位置不見得依照順序、也可能相同，所以先要寫一小段程式處理，找到位置不同的建築物有幾個，同時也將位置陣列依照順序排好
- 每個建築物都有自己的基地台或是每個位置都有自己的基地台 ⇒ **R** 為 0

```
std::sort(p, p+N);
for (distinctN=i=1; i<N; i++)
    if (p[i]!=p[i-1]) distinctN++;
```

```
if ((distinctN<=K)||
     (p[N-1]-p[0])<K) {
    printf("0\n");
    return 0;
}
```

實作 (續)

- 由於建築物的位置不見得依照順序、也可能相同，所以先要寫一小段程式處理，找到位置不同的建築物有幾個，同時也將位置陣列依照順序排好

```
std::sort(p, p+N);
for (distinctN=i=1; i<N; i++)
    if (p[i]!=p[i-1]) distinctN++;
```

- 每個建築物都有自己的基地台或是每個位置都有自己的基地台 ⇒ **R** 為 0

```
if ((distinctN<=K)||
     (p[N-1]-p[0])<K) {
    printf("0\n");
    return 0;
}
```

- 二分法

```
01 left = 1.0, right = ((double)(p[N-1]-p[0]))/K;
02 while (right-left>1.0-1e-10) {
03     mid = (left+right)/2.0;
04     if (coverAll((int)mid, N, K))
05         right = mid;
06     else
07         left = mid;
08 }
09 printf("%d\n", (int)right);
```

實作 (續)

實作 (續)

- 檢查是否 K 個直徑 d 的基地台能夠服務全部 N 棟建築物

```
int coverAll(int d, int N, int K) {
    int i, *start=p;
    for (i=0; i<K; i++) {
        start = std::upper_bound(start, p+N, *start+d);
        if (start>=p+N) return 1;
    }
    return 0;
}
```

實作 (續)

- 檢查是否 K 個直徑 d 的基地台能夠服務全部 N 棟建築物

```
int coverAll(int d, int N, int K) {
    int i, *start=p;
    for (i=0; i<K; i++) {
        start = std::upper_bound(start, p+N, *start+d);
        if (start>=p+N) return 1;
    }
    return 0;
}
```

- 不論陣列裡是否有相同的元素，std::**upper_bound**(start, end, target) 可以找到陣列 [start, end) 中第一個大於目標數值 target 的元素