Esercizi – Tecniche algoritmiche di base

Anno Accademico 2022/2023

Dott. Staccone Simone

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Introduzione

Nell’ambito dell’analisi degli algoritmi è importante saper partire da un problema reale e ragionare su diverse possibili soluzioni.

Un approccio utile è quello di ricondurre i problemi analizzati a problemi noti cercando di applicare le stesse tecniche di risoluzione algoritmica per gli stessi problemi (brute force, dividi et impera, ecc..)

Per riuscire in questo è importante realizzare algoritmi scritti in pseudocodice prima che in codice vero e proprio, utilizzando delle istanze di test (test cases) per verificare la validità del proprio algoritmo.

*Ricorda che per essere certi della validità di un algoritmo bisogna fornire una dimostrazione (almeno informale ai fini del corso), andando a valutare anche la complessità dell’algoritmo stesso!

Esercizio 1

Nel 2017, New York ha festeggiato il numero più basso di crimini degli ultimi 60 anni. Il sindaco di NY, è curioso: qual è stato il periodo di 7 giorni consecutivi con il più alto numero di crimini? Oppure 50 giorni? Oppure 36 ore? Per soddisfare la curiosità del sindaco, ti è stato chiesto di scrivere una funzione che prenda in input:

- un vettore A di n interi positivi, dove A[i] rappresenta l'istante temporale in cui è stato registrato l'i-esimo crimine, misurato come numero intero di minuti trascorsi dalla scoperta dell'isola di Manhattan (3/9/1609). I crimini sono memorizzati in ordine cronologico, ovvero il vettore è ordinato.

Nota: nello stesso minuto, possono essere stati registrati più omicidi.

- un valore intero t, che rappresenta la lunghezza del periodo temporale considerato, misurato in numero di minuti.

Scrivere un algoritmo int maxCrime(int[ ] A, int n, int t) che prenda in input il vettore degli istanti temporali e il parametro t e restituisca il numero massimo di crimini scoperti in un qualsiasi periodo di t minuti consecutivi.

Esercizio 1 - 2

Discutere informalmente la correttezza della soluzione proposta e calcolare la complessità computazionale.

Esempio: se A = [10, 17, 19, 23, 24, 26] e t = 4, l'algoritmo restituisce 3, in quanto il periodo lungo 4 minuti con il maggior numero di crimini è compreso fra 23 e 26 (infatti, 26-23+1)

Utilizzare prima lo pseudocodice e poi il codice C per realizzare l'algoritmo

Esercizio 1 - 3

Esistono varie soluzioni, una con costo O(n^2) (basata su un doppio ciclo, brute force), una con costo O(n log n) (basata su divide-et-impera) e una con costo O(n), presentata qui.

```
int maxcrime(int[] A, int n, int t)
    int i, j, count, maxsofar
    i = j = 1
    maxsofar = 0
    while j ≤ n do
        if A[j] − A[i] + 1 ≤ t then
            maxsofar = max(maxsofar, j − i + 1)
            j = j + 1
        else
            i = i + 1
    return maxsofar
```

Esercizio 1 - 4

L'idea è semplice: si definiscono due indici, i e j, che rappresentano gli estremi del periodo che stiamo considerando.

- Se A[j] − A[i] + 1 è più piccolo o uguale del periodo t, si può tentare di allargare gli estremi, incrementando l'indice j. Quando questo viene fatto, viene contato un nuovo crimine tramite il contatore count ed eventualmente aggiornato il numero massimo di crimini trovati finora tramite la variabile maxsofar .
- Se A[j] − A[i] + 1 è più grande del periodo t, si restringono gli estremi incrementando i; bisogna anche togliere 1 da count.

L'algoritmo termina quando l'estremo j raggiunge la fine dell'array. All'inizio, gli estremi sono entrambi pari a 1, ovvero contengono solo il primo elemento. La complessità è pari a O(n), come detto.

*Nota: il vettore d'ingresso è ordinato per ipotesi! Un ipotesi non banale. (Vedremo che esistono algoritmi chiamati algoritmi di ordinamento per imporre questa condizione)

Esercizio 2

Dato un array di interi positivi e un intero positivo k, determinare il numero di coppie (i,j) dove i<j e arr[i] + arr[j] è divisibile per k.

Esempio:
arr = [1,2,3,4,5,6]
k = 5

Quali sono le coppie?

Esercizio 2 - 1

Dato un array di interi positivi e un intero positivo k, determinare il numero di coppie (i,j) dove i<j e arr[i] + arr[j] è divisibile per k.

Esempio:
arr = [1,2,3,4,5,6]
k = 5

Quali sono le coppie?

Risposta: [1,4] , [2,3], [4,6]

Perché:
1+4=5 → 5 mod 5 = 0;

2+3=5 → 5 mod 5 = 0;

4+6=10 → 10 mod 5 = 0;

Esercizio 2 - 2

Dato un array di interi positivi e un intero positivo k, determinare il numero di coppie (i,j) dove i<j e arr[i] + arr[j] è divisibile per k.

Come facciamo a trovare un algoritmo che risulta valido per tutte le istanze del problema?

Utilizziamo un approccio brute force, ovvero proviamole tutte!

*Nota: i valori nell'array sono ordinati! Data una coppia [i,j] i<j !!!
[j,i] non è una coppia valida!

*Nota 2: i != j, un singolo numero non può essere una coppia!

Esercizio 2 - 3

Dato un array di interi positivi e un intero positivo k, determinare il numero di coppie (i,j) dove i<j e arr[i] + arr[j] è divisibile per k.

```
fun es_divisibile_k(int n, int k, int arr[]) {
    counter ← 0
    i←0
    j ←0
    while(i<n){
      while(j<n){
          if( (ar[i]+ar[j]) mod k == 0 ){
                      counter++;
          }
          i←i+1
        }
        j← j+1
      }
   return counter;
}
```

Complessità?

Esercizio 2 - 4

Dato un array di interi positivi e un intero positivo k, determinare il numero di coppie (i,j) dove i<j e arr[i] + arr[j] è divisibile per k.

```
fun es_divisibile_k(int n, int k, int arr[]) {
    counter ← 0
    i←0
    j ←0
    while(i<n){
      while(j<n){
         if( (ar[i]+ar[j]) mod k == 0 ){
                        counter++;
         }
         i←i+1
       }
       j← j+1
      }
   return counter;
}
```

Complessità? O(n^2)

Esercizio 2 - 5

Dato un array di interi positivi e un intero positivo k, determinare il numero di coppie (i,j) dove i<j e arr[i] + arr[j] è divisibile per k.

Implementiamo il codice e proviamolo con queste istanze:

1) arr = [1,3,2,6,1,2]; n = 6; k = 3; -→ output: 5
2) arr = [8,10]; n = 2; k = 2; -→ output: 1
3) arr = [1,3,4,13,15,6,7];n=7;k=7; →output: 5

Ricorda che superare tutti i test cases non implica la correttezza dell'algoritmo! Servirebbe una dimostrazione matematica per essere certi che un algoritmo risolvi un problema in qualsiasi sua istanza.

Si può fare di meglio?

Esercizio 2 - 6

Dato un array di interi positivi e un intero positivo k, determinare il numero di coppie (i,j) dove i<j e arr[i] + arr[j] è divisibile per k.

Un po' di algebra!

Ricordiamo che:
(a+b) mod n = (a mod n + b mod n) mod n

Quindi nel nostro caso:
(arr[i] + arr[j]) mod k = (a mod k + b mod k) mod k

Noi dobbiamo trovare i due elementi per i quali il resto della divisione per k sia 0. Quindi scorriamo tutti i numeri dell'array e vediamo le classi di resto che si formano (partizioniamo gli elementi dell'array in classi di resto)

Esercizio 2 - 7

Dato un array di interi positivi e un intero positivo k, determinare il numero di coppie (i,j) dove i<j e arr[i] + arr[j] è divisibile per k.

Prendiamo la prima istanza come esempio:
arr = [1,2,3,4,5,6]
k = 5

Individuiamo le classi di resto:

[5] → 0 mod 5
[1,6] → 1 mod 5
[2] → 2 mod 5
[3] → 3 mod 5
[4] → 4 mod 5

(arr[i] + arr[j]) mod k = (a mod k + b mod k) mod k

(2 + 3) mod 5 = (2 mod 5 + 3 mod 5) mod 5 = 0 Corretto!

Esercizio 2 - 8

Dato un array di interi positivi e un intero positivo k, determinare il numero di coppie (i,j) dove i<j e arr[i] + arr[j] è divisibile per k.

Dobbiamo semplicemente sommare le classi di resto, non concentrandoci più sul numero originale (Nota che possiamo fare questa assunzione perché la richiesta si basa sul numero delle coppie e non su quali coppie)

[5] → 0 mod 5
[1,6] → 1 mod 5
[2] → 2 mod 5
[3] → 3 mod 5
[4] → 4 mod 5

#(numeri congrui a 1 mod 5) * #(numeri congrui a 4 mod 5) →
→ 2 * 1 = 2

#(numeri congrui a 2 mod 5) * #(numeri congrui a 3 mod 5) →
→ 1 * 1 = 2
Totale: 2 + 1 = 3   Corretto!!!

Esercizio 2 - 9

Il ragionamento sembra contorto e inutilmente complesso. Andiamo ad analizzare il codice e la sua complessità.

```
int divisibleSumPairs(int n, int k, int* arr) {
   int sum=0;
   int m[k];
    for(int i=0; i<k; i++)
        m[i]=0;                    //Inizializzo il vettore che conta gli elementi di ogni classe 
di resto
    for(int i = 0; i < n; i++){
        m[arr[i]%k]++;    //Conto gli elementi per ogni classe di resto
    }
    sum+=(m[0]*(m[0]-1))/2;  //Aggiungo al contatore il # di elementi che hanno 
resto 0
    for(int i=1; i<=k/2 && i!=k-i; i++){
        sum+=m[i]*m[k-i]; //Aggiungo al contatore il # di elementi il cui resto da 
somma 5
    }
    if(k%2==0)  // A cosa serve questa condizione?
        sum+=(m[k/2]*(m[k/2]-1))/2;
```

Esercizio 2 - 10

Il ragionamento sembra contorto e inutilmente complesso. Andiamo ad analizzare il codice e la sua complessità.

```
int divisibleSumPairs(int n, int k, int* arr) {
   int sum=0;
   int m[k];
    for(int i=0; i<k; i++)
        m[i]=0;                  //Inizializzo il vettore che conta gli elementi di ogni classe 
di resto
    for(int i = 0; i < n; i++){
        m[arr[i]%k]++;    //Conto gli elementi per ogni classe di resto
    }
    sum+=(m[0]*(m[0]-1))/2;  //Aggiungo al contatore il # di elementi che hanno 
resto 0
    for(int i=1; i<=k/2 && i!=k-i; i++){
        sum+=m[i]*m[k-i]; //Aggiungo al contatore il # di elementi il cui resto da 
somma 5
    }
    if(k%2==0)  // Se k è pari, allora dobbiamo sommare il # degli elementi 
appartenenti alla classe di resto k/2
```

Esercizio 2 - 11

Esegui con n = 6; k = 4; arr = [1,3,2,6,1,2] per capire meglio

```
int divisibleSumPairs(int n, int k, int* arr) {
   int sum=0;
   int m[k];
    for(int i=0; i<k; i++)
        m[i]=0;                  //Inizializzo il vettore che conta gli elementi di ogni classe di resto
    for(int i = 0; i < n; i++){
        m[arr[i]%k]++;    //Conto gli elementi per ogni classe di resto
    }
    sum+=(m[0]*(m[0]-1))/2;  //Aggiungo al contatore il # di elementi che hanno resto 0
    for(int i=1; i<=k/2 && i!=k-i; i++){
        sum+=m[i]*m[k-i]; //Aggiungo al contatore il # di elementi il cui resto da somma 5
    }
    if(k%2==0)
        sum+=(m[k/2]*(m[k/2]-1))/2;
    return sum;
```

Esercizio 2 - 12

```
int divisibleSumPairs(int n, int k, int* arr) {
   int sum=0;
   int m[k];
    for(int i=0; i<k; i++)
        m[i]=0;                  //Inizializzo il vettore che conta gli elementi di ogni classe
di resto
    for(int i = 0; i < n; i++){
        m[arr[i]%k]++;    //Conto gli elementi per ogni classe di resto
    }
    sum+=(m[0]*(m[0]-1))/2;  //Aggiungo al contatore il # di elementi che hanno
resto 0
    for(int i=1; i<=k/2 && i!=k-i; i++){
        sum+=m[i]*m[k-i]; //Aggiungo al contatore il # di elementi il cui resto da
somma 5
    }
    if(k%2==0)
        sum+=(m[k/2]*(m[k/2]-1))/2;
    return sum;
}
```

Esercizio 2 - 13

Complessità? $T(n) = k + n + k/2 = O(n + k)$

Attenzione! La situazione si complica e non di poco: quando posso considerare k trascurabile rispetto ad n?

Dipende dalla grandezza di k rispetto ad n (Es. array lungo 5 elementi, k = 120, k>>n)

Con k troppo grande rispetto ad n avrei un array ‘lunghissimo’, ma pieno di zeri!

Inoltre per diminuire la complessità temporale abbiamo aumentato la complessità spaziale. In questo corso non la consideriamo, però è bene essere consapevoli che bisogna allocare sullo stack di memoria un array di grandezza k (nella realtà spesso la memoria a disposizione è tanta e quindi la consideriamo trascurabile, ma non è cosi in tutti i casi)