ВЕРИФИКАЦИЯ ПРОГРАММ

ДВС

Лектор - С.А. Ивановский

Пример к Лекции 5

Задача о большинстве массива

Пример: массив с большинством

Определение: массив $x_1, x_2, \ldots, x_n$ содержит большинство, если более половины его элементов имеют равные значения.

Например:

1) 1, 2, 1, 2, 1, 2, 1 ;

   n = 7, N(1) = 4

2) 7, 2, 7, 9, 7, 7, 7, 8, 5, 7, 1, 7, 7, 6, 7, 7, 4, 4, 7, 3 ;

   n = 20, N(7) = 11

Разрешена операция сравнения: = или ≠

Требуется: в массиве, содержащем большинство, найти хотя бы одно i (1<=i<=n), такое, что $x_i \in$ большинству значений массива x. [Представитель большинства - x_i]

Нахождение элемента большинства = Finding the Majority Element

Пояснения

Элемент a в массиве X [1 .. n] является элементом большинства ттогда, когда a появляется более $\lfloor n / 2 \rfloor$ раз в массиве X, то есть, по крайней мере

$$\lfloor n/2 \rfloor + 1 = \begin{cases} \lceil n/2 \rceil & \text{если } n \text{ нечётное} \\ \lceil n/2 \rceil + 1 & \text{если } n \text{ чётное} \end{cases}$$

<u>Примечания</u>: элемент большинства является элементом с наибольшей частотой, но обратное не верно.

Решения

1. Простой алгоритм: для каждого элемента подсчитывать число вхождений, пока не встретится элемент, образующий большинство. Сложность $O(n^2)$.
2. Сортировать массив, а затем сосчитать подряд стоящие равные элементы. Это занимает $O(n \log n)$. { ! Здесь используются сравнения}.
3. Найти медиану ($\lceil n/2 \rceil$-наименьший элемент), которая (-ый) является элементом большинства (если оно есть). Это займет в худшем случае $O(n)$, но с большим постоянным множителем, или в среднем $O(n)$. { ! Здесь используются сравнения}.
4. ?

Вариант рандомизированного алгоритма

Случайно выбираем i (равновероятно), а затем проверяем, удовлетворяет ли x_i условию большинства.

```
do
      j = random(n); // Выбор 1<=j<=n
      // проверка:
      c = x[j];
      k = 0;
      for (i = 1; i<=n; i++ )
          if(x[i]==c) k++;
while (k <= n/2);
```

Утверждаем, что сложность по числу сравнений $< 2n$.

Сложность рандомизированного алгоритма

Фиксируем размер входа n.

Множество всех возможных сценариев – множество кортежей $(i_1, i_2, \ldots, i_{p-1}, i_p)$, $p>=1$ и $i_q \in 1..n$,

И при этом $x[i_1], x[i_2], \ldots, x[i_{p-1}]$ – $\notin$ большинству,

а $x[i_p] \in$ большинству.

Вероятности сценариев:

Два события:

- Первая попытка найти нужный индекс j – удачная,
- Первая попытка найти нужный индекс j – Неудачная.

Пусть $P=N/n$, где N –число элементов большинства при входе n.

По условию $P > \frac{1}{2}$.

Рекуррентное уравнение

Средние затраты $\underline{t}=t(n)$ есть

$$\underline{t} = P*n + (1-P)*(\underline{t} + n),$$

где

- P - вероятность удачной попытки; при этом количество действий = n,
- $(1-P)$ - вероятность НЕудачной попытки; при этом количество действий = $(\underline{t} + n)$, т.е. от цикла for - количество действий = n и затем средние затраты $\underline{t}$ следующих попыток.

След.

$$\underline{t} = n/P; \quad \text{и} \quad (P > ½) \Rightarrow \underline{t} < 2n;$$

«Дерандомизация»

Идея: За основу берется рандомизированный алгоритм и случайный выбор заменяется на нерандомизированный (детерминированный)

1. Выбор кандидата
2. Проверка – является ли кандидат элементом большинства

Наблюдения (наводящие соображения)

1. Если *два различных элемента* удаляются из массива X, то большинство старого массива остается большинством и нового. Потому что самое худшее заключается в удалении одного элемента большинства (вместе с элементом меньшинства).

   При этом размер большинства $\lfloor n/2 \rfloor + 1 - 1 > \lfloor (n-2)/2 \rfloor$.

2. Если большинство существует, то как минимум в двух позициях подряд находятся элементы большинства или X[n] является элементом большинства.

M D M D M D M D M D M D M D M D M D M D

M D M D M D M D M D M D M D M D M D M D **M**

M D M D M D M D M D M D M D M D **M M** D M D

Algorithm: Majority

Вход: x[1..n]

Выход: Элемент большинства, если большинство существует, иначе ответ «нет».

```
c = candidate(1);   // находим лишь кандидата
int count = 0;
for (j = 1; j < n; j++)
   if (x[j]==c) count++;
if (count > ⌊n/2⌋ ) return c;
else return "нет";
```

Функция Candidate(m)

```
{
   j = m; c =x[m]; count = 1;
   While ( (j<n) && (count>0)) {
     j ++;
     if (x[j]==c) count++;
     else count--;
   }
   if (j==n) return c;
   else return candidate(j+1);
}
```

Не рекурсивная функция Candid1

```
{
   count = 1; c = x[1];
   for (j = 2; j<n; j++) {
     if (count = 0) {count = 1; c = x[j];}
     else if (x[j] == c) count++;
     else count--;
   }
   return c;
}
```

Не рекурсивная функция Candid2

```
{
   count = 0; // c = x[1];
   for (j = 1; j<n; j++) {
     if (count = 0) {count = 1; c = x[j];}
     else if (x[j] == c) count++;
     else count--;
   }
   return c;
}
```

Не рекурсивная функция Candid3

```
int Majority(int X[], int n)
{// C++, т.е. X[0..n-1]
int Candidate;
int i = 0;
int Count = 0;
while (i < n){
    if ( Count == 0 ) {
     Candidate = X[i];
     Count = 1;
    }
    else if ( Candidate == X[i] )  Count++;
    else Count--;
    i++;
}
return Candidate;
}
```

Примеры

1. x[1..13] = 3131313131313

Count > 0, и имеется большинство x[13]=3

2. x[1..9] = 123456777

Count > 0 , но большинства нет!

3. x[1..9] = 123455555

12

34

5

55555 ⇒ c=5 count=5 > ⌊9/2⌋ = 4 (!)

Продолжение

4. x[1..9] = 555551234

   55555 {count=5}

   555551 {count=4}

   5555512 {count=3}

   55555123 {count=2}

   555551234 {count=1} !!!

5. x[1..9] = 515253545

   51 {count=0}

   52 {count=0}

   … {count=0}

   5 {count=1, j=9, c=5}

Сложность и корректность (!)

Добавить этот пример в задания!