Лекция 7. Работа с массивами

Некоторые задачи и трудоемкость алгоритмов

В конце прошлой лекции было замечание о трудоемкости алгоритмов. При работе с массивами всегда имеет смысл стремиться к использованию быстрых алгоримов. Наилучший вариант, когда трудоемкость пропорциональна длине массива, т.е. есть $O(N)$, где N — количество элементов массива. Такую трудоемкость не всегда можно обеспечить, и следующая "контрольная точка" — это трудоемкость порядка $O(N \log N)$. Как правило, такую трудоемкость можно получить, если задача может быть реализована для отдельных частей массива (например, половинной длины — метод "разделяй и властвуй"), а потом общий результат получается объединением "половинных" результатов с затратой $O(N)$ действий. Если и этого достичь не получается, то приходится довольствоваться более трудоемкими алгоритмами ($O(N^2)$). Есть математические результаты по поводу наилучших и наихудщих трудоемкостей определенных классов алгоритмов (например, сортировок). Об этом разговор уже скоро.

Сейчас разберем несколько задач из семестровых заданий для иллюстрации некоторых стандартных подходов к построению алгоритмов работы с массивами.

2. Переставить элементы массива в обратном порялке.

Тривиальная задача, но использует стандартный прием — обмен значений двух переменных.

```
void Invert(int n, double *a)
{
   int i, m = n/2;
   double c;
   for (i = 0; i < m; i++) {
      c = a[i];  a[i] = a[n-i-1];  a[n-i-1] = c;
   }
}
```

На просторах Интернета часто пропагандируется забавный трюк с обменом значений двух чисел a и b без использования дополнительной переменной:

```
                    a = 6   b = 3
  a = a + b;            9       3
  b = a - b;            9       6
  a = a - b;            3       6
```

Однако на практике такой метод никуда не годится.

— не годится для нечисловых объектов

— может случиться переполнение при арифметической операции

— арифметические операции могут вносить погрешность (float, double)

— помимо присваиваний требует дополнительных арифметических операций

9. Сократить подряд идущие одинаковые значения ...

Прием "двух позиций" — одна позиция (индекс или указатель) отмечает позицию обработанной (преобразованной) части массива, а другая показывает на очередной элемент, который надо рассмотреть и обработать.

```
int ClearCopies(int n, int *a)   // возвращает длину преобразованой части массива
{
```

```
    int i; // индекс (длина) обработаной части
           // инвариант: для всех a[k], k<=i требуемое условие выполнено
    int j; // индекс текущего элемента для обработки

    i = 0;
    for (j = 1; j<n; j++) {
       if (a[i] != a[j]) {
          a[++i] = a[j];  // возможно лишнее присваивание ...
       }
    }
    return i+1;
 }
-----------------------------------
    for (j = 1; j<n; j++) {
       if (a[i] != a[j] && i+1 != j) {
          a[++i] = a[j];
       }
    }
```

11. Циклически сдвинуть массив на K позиций с трудоемкостью $O(N)$
Неправильный вариант — зациклить K раз циклический сдвиг на 1 позицию

```
 void Rotate1 (int n, double *a)
 {
    int i, c = a[0];
    for (i = 0; i < n - 1; i++) {
    {
       a[i] = a[i+1];
    }
    a[n-1] = c;
 }
 .....
 while (k--) { Rotate1(n,a); }
 .....
```

Можно предложить два варианта с затратой примерно N присваиваний и примерно $3N$ присваиваний.
Вариант с проходом по цепочке (циклу) `a[i] = a[i+k];`

```
 void Rotate (int n, double *a, int k)
 {
    int i0, i, j;
    int cnt = 0;

    // перебор всех возможных цепочек
    for (i0 = 0; cnt < n; i0++) {
    {
       // цикл по цепочке
       i = i0;
       c = a[i];
```

```
      for (j = (i+k)%n;  j != i0; j = (j + k)%n  )
      {
          a[i] = a[j];
          cnt++;
          i = j;
      }
      a[i] = c;
      cnt++;
   }
}
```

Вариант в переворотом

```
void Rotate (int n, double *a, int k)
{
    Invert(k, a);
    Invert(n-k, a+k);                  3N
    Invert(n, a);
}
```

19. Бинарный поиск (биссекция)
Классика алгоритмов.

```
bool BinSearch (int n, int *a, int x, int *index)
{
    int left = 0, right = n-1, mid;
    // инвариант:  a[left] < x <= a[right]

    if (x > a[right]) { *index = n; return false; }
    if (x <= a[left]) { *index = 0; return x == a[0]; }
    while (right - left > 1)
    {
       mid = (left + right)/2;
       if (a[mid] < x) {
          left = mid;
       } else {
          right = mid;
       }
    }
    *index = right;
    return x == a[right];
}
```

20. Слияние упорядоченных массивов

```
void Merge (int na, double *a, int nb, double *b, double *c)
{
    int ia, ib, ic;
    for (ia = ib = ic = 0; ia < na && ib < nb;  ) {
       if (a[ia] < b[ib]) {
```

```
            c[ic++] = a[ia++];
        } else {
            c[ic++] = b[ib++];
        }
        // c[ic++] = (a[ia] < b[ib]) ?  a[ia++] : b[ib++];
    }
    while (ia < n) { c[ic++] = a[ia++]; }
    while (ib < n) { c[ic++] = b[ib++]; }
}
```

Слияние требует дополнительного массива для получения результата. Этот факт мы дополнительно обсудим при рассмотрении сортировок, поскольку там бывает нужно выполнять слияние для частей одного и того же массива. И это приводит к необходимости использования дополнительной памяти. Можно ли реализовать слияние двух упорядоченных частей массива на их же исходом месте без дополнительной памяти — это достаточно интересный вопрос. Ответ — да, но за это придется заплатить усложнением алгоритма. Обсудим при знакомстве с сортировками.

21, 22 Перемешивание/группировка по четным/нечетным индесам

Идея "разделяй и властвуй" ($N \log N$) Для примера, группировка элементов с четными индексами в начале. Делим массив пополам. Пусть для половинок это преобразование выполнили. Получили группы E1 O1 | E2 O2 выполняем циклический сдвиг части O1 E2 и получаем E1 E2 O1 O2, т.е. то, что надо. Циклический сдвиг — $O(N)$. Далее рекурсия с глубиной $\log_2 N$.

Примерно так:

```
void Mixer (int n, double a)
{
    int m = n/2;
    if (n < 3) return;
    Mixer(m, a);
    Mixer(n - m, a + m);
    // надо аккуратно определить размеры O1 и E2
    int k, ko1, ke2;
    ko1 = m/2;
    ke2 = n - m - m/2;
    Rotate (ko1 + ke2, a + m - ko1, ko1);
}
```

Перестановка.

Пусть перестановка массива задана явно массивом perm[], т.е. a[perm[i]] = a[i]; Можно ли реализовать такую перестановку со сложностью $O(n)$?