УДК 519.683.6
ББК 32.973.26-018.2

ПРИМЕНЕНИЕ СОРТИРОВКИ МЕТОДОМ ПРЕОБРАЗОВАНИЯ ШВАРЦА В ЗАДАЧАХ ВЫЧИСЛИТЕЛЬНОЙ ГЕОМЕТРИИ

Клячин Владимир Александрович

Доктор физико-математических наук,
заведующий кафедрой компьютерных наук и экспериментальной математики
Волгоградского государственного университета
klchnv@mail.ru, kiem@volsu.ru
просп. Университетский, 100, 400062 г. Волгоград, Российская Федерация

Аннотация. В статье рассматривается применение преобразования Шварца для сортировки применительно к объектам вычислительной геометрии. На примерах иллюстрируется временной выигрыш использования указанного преобразования при сортировке.

Ключевые слова: сортировка, приемы сортировки, преобразование Шварца, сортировка ребер графа, языки программирования Perl и Python.

1. Задача сортировки. Пусть X — произвольное множество и $A = (a_1, a_2, ..., a_n) \subset \subset X$ — его конечное подмножество и пусть на X задана некоторая функция

$$g : X \to \mathbb{R}.$$

Задача сортировки множества A состоит в том, чтобы найти такую перестановку номеров $(i_1, i_2, ..., i_n)$, чтобы было выполнено условие

$$g(a_{i_1}) \leq g(a_{i_2}) \leq ... \leq g(a_{i_n}).$$

Приведем небольшой обзор имеющихся программных средств для сортировки массивов произвольной природы.

Язык программирования C. В стандартной библиотеке stdlib.h имеется описание функции

```
void qsort ( void * base, size_t num, size_t size,
int ( * comp ) ( const void *, const void * ) );
```

где base — ссылка на первый элемент массива, который следует отсортировать, num — количество элементов в массиве, size — размер в байтах одного элемента массива, comp — указатель на функцию сравнения, согласно которой будет произведена сортировка элементов в массиве. Эта функция принимает два аргумента — два элемента сортируемого массива и возвращает значение -1,0 или 1 в зависимости от отношения между переданными параметрами.

Язык программирования C++ (STL). Функция sort из библиотеки <algorithm> сортирует контейнер на полуинтервале [first, last). Вот ее описание

```
template
void sort ( RandomAccessIterator first,
            RandomAccessIterator last );
```

или

```
template
void sort ( RandomAccessIterator first,
            RandomAccessIterator last,
            Compare comp );
```

где first — итератор на первый элемент, last — итератор на последний элемент, comp — функция сравнения, согласно которому будет произведена сортировка элементов в контейнере, итераторы которого указаны в первых двух аргументах функции.

Язык программирования Python. Списки сортируются с помощью вызова метода sort. Метод list.sort принимает три необязательных параметра: key — функция, которая принимает элемент списка и возвращает объект, который будет использоваться при сравнении во время сортировки вместо оригинального элемента списка; cmp — функция, которая принимает два элемента списка и возвращает -1, 0 или 1 в зависимости от отношения между переданными параметрами; reversed — если True, то список будет отсортирован в обратном порядке.

Язык программирования Perl. Описание функции sort. Sort — это встроенная функция Perl. Простейший способ ее вызова выглядит так

```
sort LIST
```

По умолчанию sort сравнивает элементы переданного списка как строки, сортирует их в алфавитном порядке и возвращает отсортированный список. Если задана директива use locale, при сортировке будут учитываться национальные установки. В следующих способах вызова функции задаются правила сравнения элементов списка

```
sort USERSUB LIST
sort BLOCK LIST
```

Здесь USERSUB или BLOCK задают функции сравнения как дополнительную процедуру в первом случае, или встроенный блок операторов — во втором.

Заметим, что во всех перечисленных выше случаях указывается функция сравнения, которая может быть сведена к сравнению некоторой скалярной величины. В нашем случае — это функция $g : X \to \mathbb{R}$. Например, на языке Python реализация выглядит так

```
def g(x):
    ...#вычисление значения y
    return y

def comp(a,b):
    return cmp(g(a),g(b))
```

2. Преобразование Шварца. Суть этого преобразования состоит в том, чтобы вместо сортировки элементов множества $A \subset X$ осуществлять сортировку массива $A' \subset \subset X \times \mathbb{R}$, где

$$A' = \{(a, g(a)) : a \in A\}.$$

Если обозначить через $p_i : X \times \mathbb{R} \to \mathbb{R}, i = 1, 2$ естественную проекцию на i-й множитель произведения $X \times \mathbb{R}$, то функция сравнения для сортировки множества A' будет выглядеть так

```
def comp(a,b):
    return cmp(p2(a),p2(b))
```

Таким образом, сначала вычисляются все значения функции g на элементах множества A, а пары $(a, g(a))$ сохраняются в дополнительный массив. Затем этот массив сортируется по уже вычисленным значениям в каждой паре. И наконец, проекция $p_1 : A' \to A$ вернет перестановку исходного массива (рис. 1).

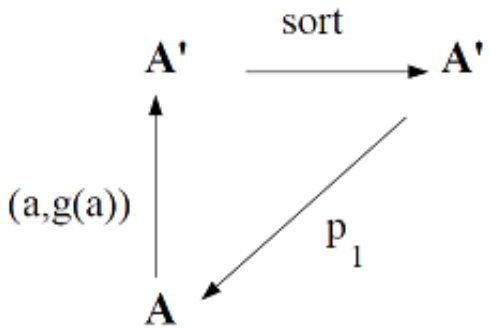

Рис. 1. Общая схема преобразования Шварца

Оценим алгоритмическую сложность обычной сортировки и сортировки с использованием преобразования Шварца. Если через $T(n)$ обозначить число сравнений в алгоритме сортировки, а через k — число операций для вычисления значения функции g для одного элемента множества A, то общее число операций для сортировки массива будет равно

$$T_1(n, k) = kT(n)$$

при простом способе сортировки и

$$T_2(n, k) = nk + T(n)$$

при использовании преобразования Шварца. Поэтому, получаем

$$\frac{T_2(n, k)}{T_1(n, k)} = \frac{1}{k} + \frac{1}{T(n)}.$$

Произвести сортировку с использованием преобразования Шварца оказывается не сложно. Приведем реализацию такого преобразования средствами некоторых языков программирования.

Язык программирования Perl. Для входного массива **@input** оператор **map** строит список пар вида $(a, g(a))$, который сортируется функцией **sort**. Последний оператор **map** осуществляет проекцию множества пар в исходное множество. Впервые реализация была сделана Реналдом Шварцем [3]. Также это преобразование описано в книге [2].

```
@output=map $_->[0],
             sort{$a->[1]<=>$b->[1]
             } map [$_,g($_)], @input;
```

Или в виде отдельной процедуры, в которую необходимо передать две ссылки: одна на сортируемый массив, другая на процедуру вычисления функции $g : X \to \mathbb{R}$.

```
sub shvartz_transform
{
my ($input,$g)=@_;
my @output=map $_->[0],
           sort{$a->[1]<=>$b->[1]
           } map [$_,$g->($_)], @$input;
return @output;
}
```

Для сравнения, обычная сортировка осуществляется так

```
@output=sort {g($a)<=>g($b)} @input;
```

Язык программирования Python. Так же как и выше приводим реализацию преобразования Шварца в виде функции с теми же что и выше обозначениями.

```
def shvartz_transform(input,g):
    temp=map(None,input,map(g,input))
    temp.sort(cmp=lambda a,b:cmp(a[1],b[1]))
    return map(lambda a:a[0],temp)
```

Однако, начиная с 3-й версии Python, для метода **sort** не определен именованный параметр **cmp**. Вместо него введен новый параметр **key**. Значение этого параметра является функцией вычисления величины, которая будет участвовать в сортировке. Другими словами, начиная с версии Python 3, в функцию сортировки необходимо передать процедуру вычисления функции $g : X \to \mathbb{R}$. Поэтому, в этом случае реализация преобразования Шварца несколько другая.

```
def shvartz_transform(input,g):
    tmp=list(map(lambda a,b: [a,b],input,map(g,input)))
    tmp.sort(key=lambda a: a[1])
    return map(lambda a:a[0],tmp)
```

Однако, ввиду указанного выше смысла именованного параметра **key**, для Python 3.x преобразование Шварца, по всей видимости, встроено в реализацию метода **list.sort()**. Неявно этот вывод подтверждается на страничке [4].

Пример решения модельной задачи. Рассмотрим задачу сортировки строк числовой матрицы $n \times k$ по возрастанию их минимальных элементов. Матрицу будем задавать как массив строк чисел, разделенных пробелами. Так что функция вычисления минимального элемента отдельной строки матрицы выглядит так

```
sub min
{
my $a=shift;
my @coords=split " ", $a;

my $min=$coords[0];
for my $x (@coords){
if($x<$min){$min=$x;}
}
return $min;
}
```

При этом обычная сортировка получается так

```
@output=sort {min($a)<=>min($b)} @input;
```

Приведем результаты измерения времени (в секундах) выполнения сортировки.

n	Обычная сортировка (с)	Сортировка с преобразованием Шварца (с)
120	0.063	0.015
2600	1.906	0.313
5200	3.767	0.672

Данные приведены для случая $k = 3$.

3. Сортировка в задачах вычислительной геометрии. Первой рассмотрим задачу построения выпуклой оболочки конечного множества точек $p_1, ..., p_n$ на плоскости. Некоторые алгоритмы (см., например, [1]) основаны на предварительной сортировке этого набора точек по возрастанию величины полярного угла с полюсом в некоторой точке O, заведомо принадлежащей выпуклой оболочке. Приведем результаты измерения времени выполнения такой операции.

n	Обычная сортировка (секунд)	Сортировка с преобразованием Шварца (с)
2500	0.063	0.031
10000	0.285	0.094
40000	1.361	0.454
90000	3.281	1.145
250000	9.664	3.094

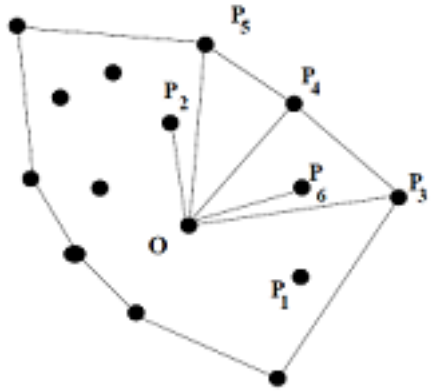

Рис. 2. Сортировка точек по полярному углу

Второй рассмотрим задачу сортировки против часовой стрелки ребер плоского графа для каждой его вершины. Такая операция используется в процессе построения структуры для хранения графа в виде реберного списка с двойными связями [1]. Структуру графа в программе будем задавать в виде массива вида

```
@graph=([1,3,[1,4,2]],[3,1,[0,3,2]],[4,4,[1,3,4]],
        [5,2,[1,2,4]],[4,6,[0,2,3]],...);
```

Здесь в каждом внутреннем массиве первые два числа задают координаты вершины графа, а целочисленный массив содержит номера соседних вершин с данной. Для решения задачи нам необходимо ввести две процедуры. Первая из них **phi** вычисляет величину полярного угла наклона ребра графа по отношению к оси Ox. В процедуру передаются номера вершин — концов ребра.

```
sub phi
{
my $dy=$graph[$_[1]]->[1]-$graph[$_[0]]->[1];
my $dx=$graph[$_[1]]->[0]-$graph[$_[0]]->[0];
return atan2($dy,$dx);
}
```

Вторая процедура выполняет первый шаг преобразования Шварца для всех вершин графа.

```
sub add_phi
{
my $k=shift;
[$graph[$k]->[0],$graph[$k]->[1],
           [map{
[$_,phi($k,$_)];
   }@{$graph[$k]->[2]}
 ]
];
}
```

Тогда сортировка ребер графа обычным способом осуществляется так

```
@sorted=map{
my @s=sort{phi($a,$_)<=>phi($b,$_)} @{$graph[$_]->[2]};
[$graph[$_]->[0],$graph[$_]->[1],\@s];
}(0..@graph-1);
```

А с использованием преобразования Шварца — немного сложнее

```
@out_graph=map{
[$_->[0],$_->[1],[map{$_->[0]}@{$_->[2]}]]
                }map{
          [$_->[0],$_->[1],[sort{$a->[1]<=>$b->[1]} @{$_->[2]}]];
               }map add_phi($_),(0..@graph-1);
```

Приведем результаты измерений. Сортировка производилась для полного графа K_n с n вершинами.

n	Обычная сортировка (секунд)	Сортировка с преобразованием Шварца (с)
10	0.125	0.037
15	0.775	0.235
20	2.269	0.688
25	5.484	1.751
30	12.359	3.734
40	41.672	12.142

СПИСОК ЛИТЕРАТУРЫ

1. Препарата, Ф. Вычислительная геометрия / Ф. Препарата, М. Шеймос. — М. : Мир, 1985. — 478 с.

2. Уолл, Л. Программирование на Perl / Л. Уолл, Т. Кристиансен, Дж. Орвант. — СПб. : Символ-плюс, 2006. — 1152 с.

3. Шварц, Р. Л. Изучаем глубже Perl / Р. Л. Шварц, Б. Д. Фой, Т. Феникс. — СПб. : Символ-плюс, 2007. — 320 с.

4. Schwartzian transform. — Electronic text data. — Mode of access: http://en.wikipedia.org/wiki/Schwartzian_transform. — Title from screen.

REFERENCES

1. Preparata F., Shamos M. *Vychislitel'naya geometriya* [Computational Geometry: An Introduction]. Moscow, Mir Publ., 1985. 478 p.

2. Wall L., Christiansen T., Orwant J. *Programmirovanie na Perl* [Programming Perl]. SPb., Simvol-plyus Publ., 2006. 1152 p.

3. Schwartz R.L., Foy B.D., Phoenix T. *Izuchaem glubzhe Perl* [Intermediate Perl]. SPb., Simvol-plyus Publ., 2007. 320 p.

4. Schwartzian transform. Available at: http://en.wikipedia.org/wiki/Schwartzian_transform.

APPLICATION OF THE METHOD OF SORTING SCHWARTZIAN TRANSFORM IN THE COMPUTATIONAL GEOMETRY

Klyachin Vladimir Aleksandrovich

Doctor of Physical and Mathematical Sciences,
Head of Department of Computer Science and Experimental Mathematics
Volgograd State University
klchnv@mail.ru, kiem@volsu.ru
Prosp. Universitetsky, 100, 400062 Volgograd, Russian Federation

Abstract. The article discusses the use of Schwartzian transform sorting applied to computational geometry objects. In computer science, the Schwartzian transform is a Perl programming idiom used to improve the efficiency of sorting a list of items. This idiom is appropriate for comparison-based sorting when the ordering is actually based on the ordering of a certain value of the elements of array, where computing this value is an intensive operation that should be performed a minimal number of times. The Schwartzian Transform for Perl language is notable in that it does not use named temporary arrays. We consider the question how to use Schwartzian transform for some computational geometry problems such that construct convex hull and orientation planar graph. The results is illustrated by tables with fulfilment time of sorting. The gain in time is approximately 3.3 times. Also, we give implementation of Schwarzian transform for Python programming language.

Key words: sorting, sorting methods, Schwartz transformation, sorting edges of graph, Perl and Python programming language.