Акимушкин Василий Александрович,
Поздняков Сергей Николаевич

ПРОСТЫЕ СОРТИРОВКИ И АЛГОРИТМ КРАСКАЛА

Сегодня мы познакомимся с тремя важными идеями информатики:

1) простыми сортировками;

2) методом раскраски вершин для обнаружения циклов в графе;

3) ещё одним жадным алгоритмом построения минимального остовного дерева.

Начнем с последнего пункта.

Алгоритм Краскала гарантирует минимальность строящегося остовного дерева тем, что просматривает все ребра, от самого маленького до самого большого по возрастанию их весов, и если добавление этого ребра к строящемуся дереву не приводит к образованию циклов (то есть, сохраняет свойство графа быть деревом), то ребро добавляется в строящееся дерево.

Таким образом, первое, что нужно сделать, это отсортировать ребра графа по возрастанию весов.

Умеете ли вы быстро сортировать числа?

Если чисел мало, то их сортировка проблемой не является. А вот если чисел 1000

или больше, то нам не удастся «охватить взглядом» все числа сразу и нужно придумать метод упорядочивания, не зависящий от количества чисел. Например, даже колоду карт непросто упорядочить. Обсудим различные методы сортировки на примере сортировки карт младших значений независимо от их масти (рис. 1).

Первый способ называется «сортировка выбором»: мы просматриваем все карты

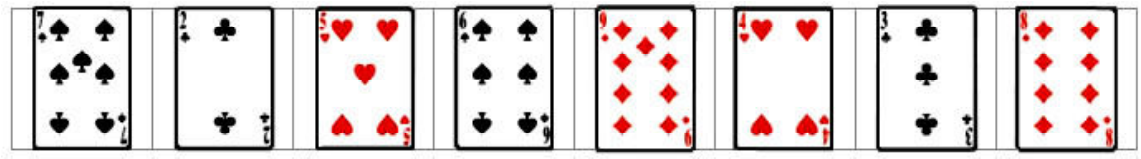

Рис. 1

[1] В журнале № 3 за 2013 год был описан другой жадный алгоритм – алгоритм Прима.

(слева направо) выбираем минимальную и меняем её с первой картой, чтобы на первое место попала самая младшая карта. Потом делаем то же самое, просматривая карты, начиная со 2-ой, потом с 3-ей (чем дальше, тем меньше будет карт, из которых выбираем наименьшую).

Результат применения этого метода показан на рис. 2.

Второй метод – сортировку включениями – мы покажем более экономно, записывая, как это обычно делается, только значения чисел.

Метод состоит в следующем: начиная слева направо, «вынимаем» очередной элемент и сдвигаем элементы, стоящие слева от него, вправо, пока не дойдем до элемента меньше или равного «вынутому». На освободившееся место ставим «вынутый» элемент (см. рис. 3).

Самым известным методом из простых сортировок является метод сортировки об-

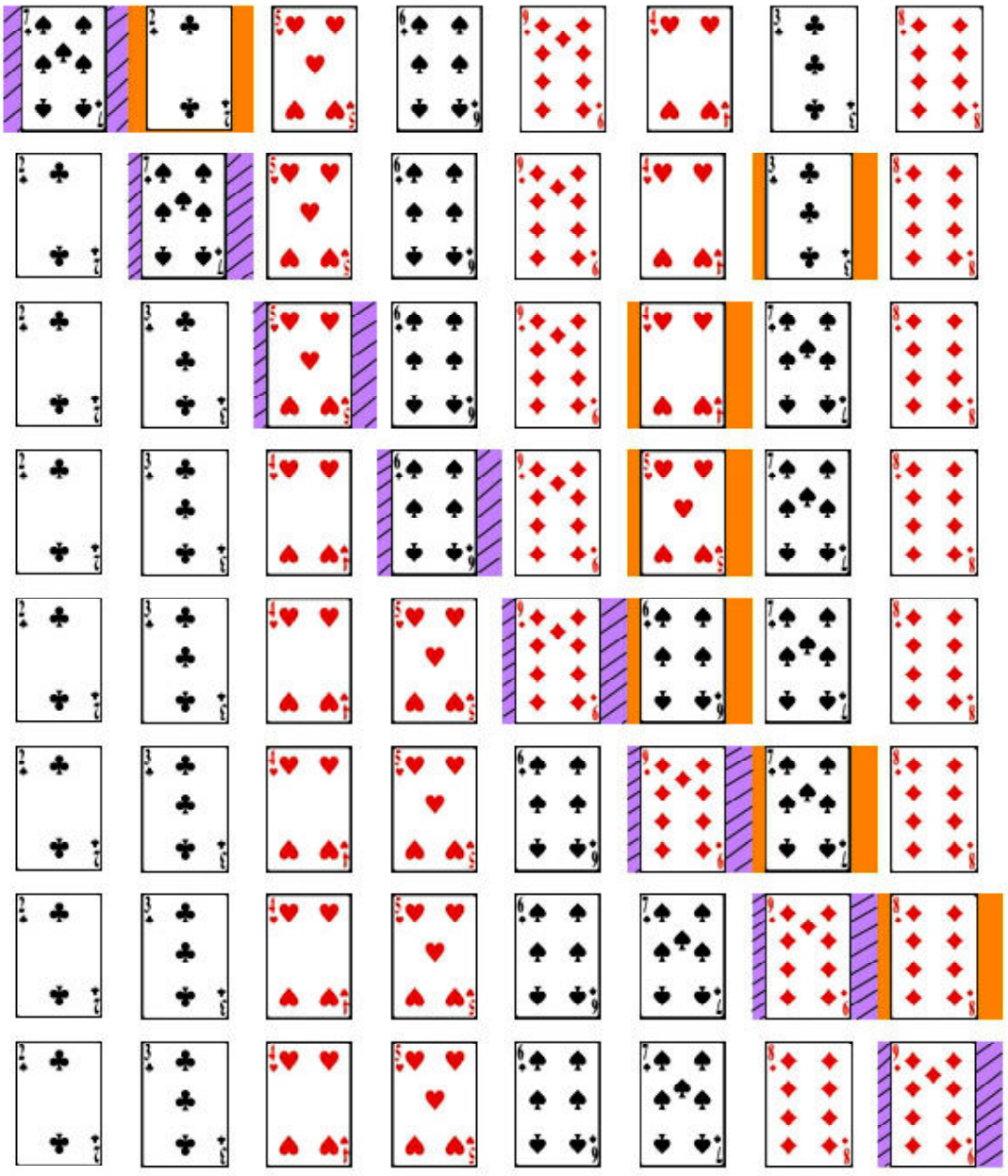

Рис. 2

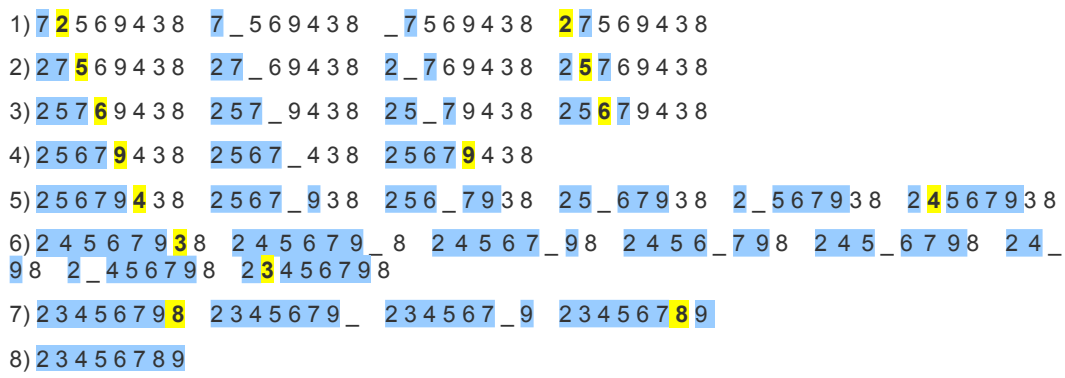

Рис. 3

менами или, как его ещё называют, метод пузырька. Мы покажем его на вертикальном столбике пузырьков (именно вертикальные столбики из отрезков вам нужно будет сортировать при изучении алгоритма Краскала) (см. рис. 4).

Начиная с нижнего элемента и двигаясь вверх, мы меняем соседние элементы местами, если меньший стоит ниже большего. Это создает впечатление всплывающего пузырька. Например, на рис. 4 таким элементом сначала был элемент 3, однако он не «всплыл наверх», потому что «по дороге» ему встретился более маленький элемент 2, который продолжил движение и всплыл наверх. На рисунке показан только один цикл алгоритма сортировки пузырька. Далее цикл повторяется снизу, однако пузырек всплывает уже не до самого верху, а до предпоследнего элемента и т.д., каждый следующий пузырек проходит путь на 1 меньше предыдущего.

Замечание. Приведенный цикл можно расширить другим циклом – опусканием «тяжелого» элемента вниз (обменами, начиная от второго сверху и кончая последним снизу). Далее эти два цикла чередуются, причем число сравнений соседних элементов уменьшается с каждым новым циклом на 1 (как уже говорилось выше).

Мы так подробно остановились на сортировках, поскольку встретились с ними в первый раз, а алгоритмы сортировки играют в программировании важную роль.

В предлагаемом манипуляторе сортировка осуществляется за один шаг, но мы надеемся, что вам будет интересно в режиме тренажера перепробовать все приведенные алгоритмы.

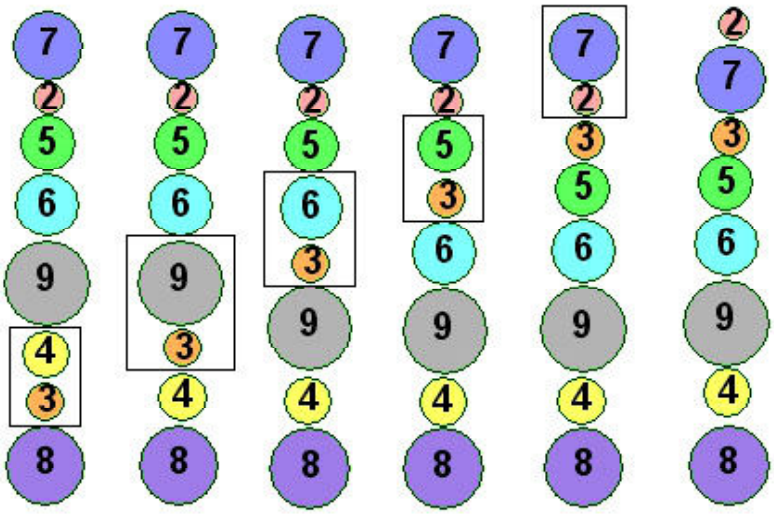

Рис. 4

Вернемся теперь к алгоритму Краскала, который был коротко описан выше. В нем ребро добавляется в остовное дерево, если при этом в графе не образуется цикл. Пока граф небольшой, это можно просто увидеть на рисунке. Но если в графе тысяча вершин и того больше ребер – это ненадежный способ. Тем более, что на все алгоритмы мы должны смотреть с точки зрения, как им

научить компьютер или, иными словами, как их запрограммировать.

Для проверки наличия циклов используется метод раскраски вершин графа.

Сначала все вершины раскрашиваются в разные цвета, но как только две вершины соединяются ребром, они перекрашиваются в один цвет (обычно в цвет одной из этих вершин). При этом главное, что в этот же цвет перекрашиваются все другие вершины графа, которые до этого имели тот же цвет, что и перекрашиваемая вершина.

Это приводит к тому, что все части остовного дерева, между любыми вершинами которых есть путь по ребрам дерева (такие части называют компонентами связности графа), окрашены в один цвет, Поэтому, если попытаться добавить ребро, которое соединяет вершины уже построенного дерева (что привело бы к образованию цикла), они оказываются окрашенными в один цвет, что является сигналом для того чтобы не включать это ребро в остовное дерево (в манипуляторе это осуществляется двумя кнопками: зеленая – означает включение очередного ребра в дерево, красная – отказ от включения).

Приведенные описания алгоритмов весьма кратки, однако мы надеемся, что все уточнения вам станут понятны, как только вы начнете работать с манипулятором.

В заключение приведем выдержку из учебника «Дисретная математика» (авторов С.Н. Позднякова и С.В. Рыбина), относящуюся к алгоритму Краскала.

АЛГОРИТМ КРАСКАЛА

Упорядочим все ребра по возрастанию (неубыванию) весов.

```
ЦИКЛ-ПОКА не кончились ребра
   Рассмотреть очередное ребро
   ЕСЛИ добавление ребра к предыдущим не приводит к появлению цикла
   ТО добавить ребро
КЦ
```

Теорема. Применение алгоритма Краскала к любому взвешенному неориентированному связному графу приводит к построению остовного дерева.

Доказательство.

От противного. Пусть существует дерево T_1 с весом меньше чем дерево T_0, построенное по алгоритму Краскала. Сравним эти деревья по совпадению ребер в том порядке, как ребра вставлялись в дерево T_0. Найдем первое из ребер e, не вошедших в T_1. Добавим это ребро в дерево T_0. В полученном графе появится цикл, содержащий ребро e. В этом цикле должно найтись ребро w, вес которого больше либо равен весу e. Действительно, ребра меньшего веса вместе с e не могут образовать цикла, так как входят в дерево T_0. Заменив ребро w на e, мы получим новое остовное дерево с весом либо меньшим веса T_1 (что сразу приводит к противоречию), либо с равным весом. В последнем случае после замены ребер мы можем продолжить процесс сравнения и использовать те же рассуждения. Так как по предположению вес T_1 меньше веса T_0, то рано или поздно вес нового остовного дерева должен стать меньше веса T_1, что и даст искомое противоречие.

Построенный алгоритм включает определение наличия цикла в графе, что само по себе – трудная задача. Поэтому алгоритм Краскала можно переписать в другой форме, используя идею раскраски вершин.

МОДИФИЦИРОВАННЫЙ АЛГОРИТМ КРАСКАЛА

Упорядочим все ребра по возрастанию (неубыванию) весов.

Присвоим вершинам графа различные номера (раскрасим вершины в разные цвета):

```
ЦИКЛ-ПОКА не кончились ребра
   Рассмотреть очередное ребро
   ЕСЛИ концы ребра окрашены в разные цвета
     ТО добавить ребро в остовное дерево и перекрасить все вершины, номер которых совпадает
        с номером одного конца добавленного ребра в цвет второго конца этого ребра
        (для определенности можно перекрашивать в цвет вершины с большим номером)
КЦ
```

Пример.

Рассмотрим протокол перекраски вершин графа на рис. 5.

Последовательность ребер, упорядоченных по возрастанию (неубыванию) весов, ребра одного веса упорядочены в лексикографическом порядке (можно было взять любой другой порядок): AB, AD, AE, BD, EH, DE, DG, DH, BE, BF, BC, CE, CF, EF, EG, FK, GH, HK, EK, FH (табл. 1)

Замечание. В данном примере видно, что последние 4 шага лишние: дерево построено, и все вершины графа раскрашены в один цвет. Оба этих признака можно использовать для проверки окончания цикла (например, при числе вершин n цикл можно закончить, когда число выбранных ребер равно $n - 1$).

Табл. 1

Номер вершины / Текущее ребро	A	B	C	D	E	F	G	H	K	Выбранное ребро
AB	**1**	**2**	3	4	5	6	7	8	9	AB
AD	**2**	2	3	**4**	5	6	7	8	9	AD
AE	**4**	4	3	4	**5**	6	7	8	9	AE
BD	5	**5**	3	**5**	5	6	7	8	9	
EH	5	5	3	5	**5**	6	7	**8**	9	EH
DE	8	8	3	**8**	**8**	6	7	8	9	
DG	8	8	3	**8**	8	6	**7**	8	9	DG
DH	8	8	3	**8**	8	6	8	**8**	9	
BE	8	**8**	3	8	**8**	6	8	8	9	
BF	8	**8**	3	8	8	**6**	8	8	9	BF
BC	8	**8**	**3**	8	8	8	8	8	9	BC
CE	8	8	**8**	8	**8**	8	8	8	9	
CF	8	8	**8**	8	8	**8**	8	8	9	
EF	8	8	8	8	**8**	**8**	8	8	9	
EG	8	8	8	8	**8**	8	**8**	8	9	
FK	8	8	8	8	8	**8**	8	8	**9**	FK
GH	9	9	9	9	9	9	**9**	**9**	9	
HK	9	9	9	9	9	9	9	**9**	**9**	
EK	9	9	9	9	**9**	9	9	9	**9**	
FH	9	9	9	9	9	**9**	9	**9**	9	

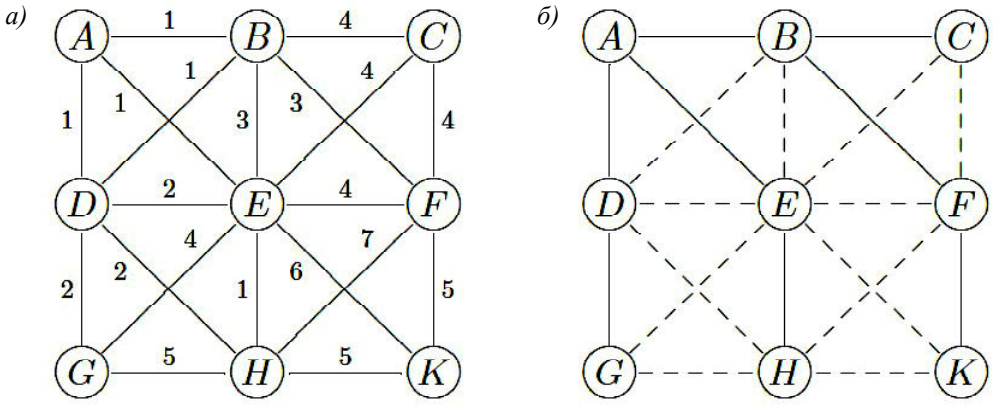

Рис. 5

От редакции: На сайте Интернет-школы современной информатики и дискретной математики http://kioschool.eltech.ru/, открытой Факультетом компьютерных технологий и информатики Санкт-Петербургского электротехнического университета (ЛЭТИ) совместно с Центром информатизации образования «КИО», учредителем одноименного конкурса КИО («Конструируй, исследуй, оптимизируй»), можно зарегистрироваться на обучение в Школе и получить доступ к тренажёрам и модулям контроля.

Акимушкин Василий Александрович,
аспирант математико-механического факультета СПбГУ,
программист АНО «КИО»,

Поздняков Сергей Николаевич,
доктор педагогических наук,
профессор кафедры ВМ-2 СПбГЭТУ «ЛЭТИ»,
научный руководитель Интернет-школы современной информатики и дискретной математики.