1

Обзор алгоритмов

Эта книга содержит информацию, необходимую для понимания, классификации, выбора и реализации важных алгоритмов. Помимо объяснения логики алгоритмов, в ней рассматриваются структуры данных, среды разработки и производственные среды, подходящие для тех или иных классов алгоритмов. Мы сфокусируемся на современных алгоритмах машинного обучения, которые обретают все большее значение. Наряду с теорией мы рассмотрим и практические примеры использования алгоритмов для решения актуальных повседневных задач.

В первой главе дается представление об основах алгоритмов. Она начинается с раздела, посвященного основным концепциям, необходимым для понимания работы разных алгоритмов. В этом разделе кратко рассказывается о том, как люди начали использовать алгоритмы для математической формулировки определенного класса задач. Речь также пойдет об ограничениях различных алгоритмов. В следующем разделе будут объяснены способы построения логики алгоритма. Поскольку в этой книге для написания алгоритмов используется Python, мы объясним, как настроить среду для запуска примеров. Затем обсудим различные способы количественной оценки работы алгоритма и сравнения с другими алгоритмами. В конце главы познакомимся с различными способами проверки конкретной реализации алгоритма.

Если кратко, то в этой главе рассматриваются следующие основные моменты:

- Что такое алгоритм.
- Определение логики алгоритма.

- Введение в пакеты Python.
- Методы разработки алгоритмов.
- Анализ производительности.
- Проверка алгоритма.

ЧТО ТАКОЕ АЛГОРИТМ

Говоря простым языком, алгоритм — это набор правил для выполнения определенных вычислений, направленных на решение определенной задачи. Он предназначен для получения результатов при использовании любых допустимых входных данных в соответствии с точно прописанными командами. Словарь American Heritage дает такое определение:

> *«Алгоритм — это конечный набор однозначных инструкций, которые при заданном наборе начальных условий могут выполняться в заданной последовательности для достижения определенной цели и имеют определимый набор конечных условий».*

Разработка алгоритма — это создание набора математических правил для эффективного решения реальной практической задачи. На основе такого набора правил можно создать более общее математическое решение, многократно применимое к широкому спектру аналогичных задач.

Этапы алгоритма

Различные этапы разработки, развертывания и, наконец, использования алгоритма проиллюстрированы на следующей диаграмме (рис. 1.1).

Как показано на диаграмме, прежде всего нужно сформулировать задачу и подробно описать необходимый результат. Как только задача четко поставлена, мы подходим к этапу разработки.

Разработка состоит из двух этапов.

- *Проектирование*. На этом этапе разрабатываются и документируются архитектура, логика и детали реализации алгоритма. При разработке алгоритма мы учитываем как точность, так и производительность. Часто для решения одной и той же задачи можно использовать два или несколько разных алгоритмов. Этап проектирования — это итеративный процесс,

который включает в себя сравнение различных потенциально пригодных алгоритмов. Некоторые алгоритмы могут давать простые и быстрые решения, но делают это в ущерб точности. Другие могут быть очень точными, но их выполнение занимает значительное время из-за их сложности. Одни сложные алгоритмы эффективнее других. Прежде чем сделать выбор, следует тщательно изучить все компромиссы. Разработка наиболее эффективного алгоритма особенно важна при решении сложной задачи. Правильно разработанный алгоритм приведет к эффективному решению, способному одновременно обеспечить как надлежащую производительность, так и достаточную степень точности.

- *Кодирование*. Разработанный алгоритм превращается в компьютерную программу. Важно, чтобы программа учитывала всю логику и архитектуру, предусмотренную на этапе проектирования.

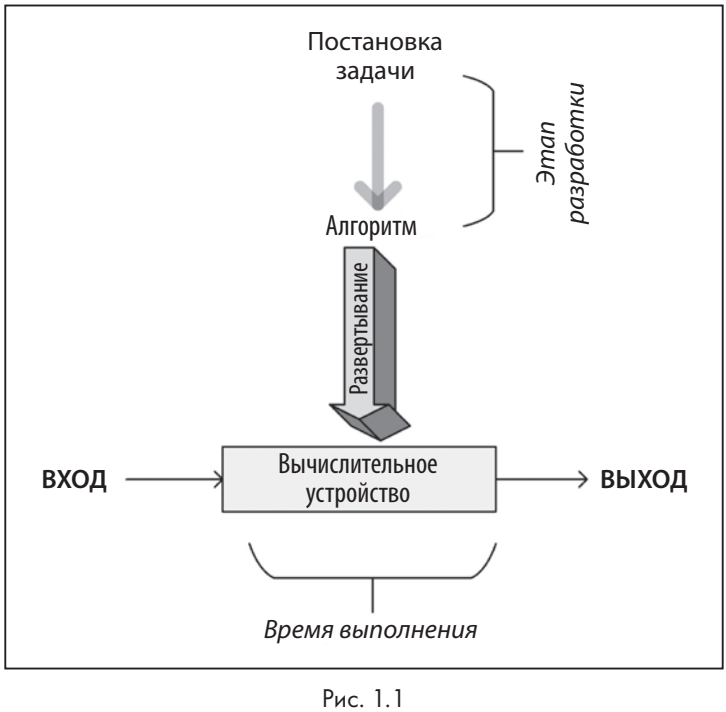

Рис. 1.1

Этапы проектирования и кодирования алгоритма носят итеративный характер. Разработка алгоритма, отвечающего как функциональным, так и нефункциональным требованиям, занимает массу времени и усилий. Функциональные

требования определяют то, какими должны быть правильные выходные данные для конкретного набора входных данных. Нефункциональные требования алгоритма в основном касаются производительности для заданного объема данных. Проверку алгоритма и анализ его производительности мы обсудим далее в этой главе. Проверка должна подтвердить, соответствует ли алгоритм его функциональным требованиям. Анализ производительности — подтвердить, соответствует ли алгоритм своему основному нефункциональному требованию: производительности.

После разработки и реализации на выбранном языке программирования код алгоритма готов к развертыванию. Развертывание алгоритма включает в себя разработку реальной производственной среды, в которой будет выполняться код. Производственная среда должна быть спроектирована в соответствии с потребностями алгоритма в данных и обработке. Например, для эффективной работы распараллеливаемых алгоритмов потребуется кластер с соответствующим количеством компьютерных узлов. Для алгоритмов, требующих больших объемов данных, возможно, потребуется разработать конвейер ввода данных и стратегию их кэширования и хранения. Более подробно проектирование производственной среды обсуждается в главах 13 и 14. После разработки и реализации производственной среды происходит развертывание алгоритма, который принимает входные данные, обрабатывает их и генерирует выходные данные в соответствии с требованиями.

ОПРЕДЕЛЕНИЕ ЛОГИКИ АЛГОРИТМА

При разработке алгоритма важно найти различные способы его детализировать. Нам необходимо разработать как его логику, так и архитектуру. Этот процесс можно сравнить с постройкой дома: нужно спроектировать структуру, прежде чем фактически начинать реализацию. Для эффективности итеративного процесса проектирования более сложных распределенных алгоритмов важно предварительно планировать, как их логика будет распределена по нодам кластера во время выполнения алгоритма. Этого можно добиться с помощью псевдокода и планов выполнения, о которых мы поговорим в следующем разделе.

Псевдокод

Самый простой способ задать логику алгоритма — составить высокоуровневое неформальное описание алгоритма, которое называется *псевдокодом*. Прежде

чем описывать логику в псевдокоде, полезно сформулировать основные шаги на простом человеческом языке. Затем это словесное описание структурируется и преобразуется в псевдокод, точно отражающий логику и последовательность алгоритма. Хорошо написанный псевдокод должен достаточно подробно описывать алгоритм, даже если такая детализация не обязательна для описания основного хода работы и структуры алгоритма. На рис. 1.2 показана последовательность шагов.

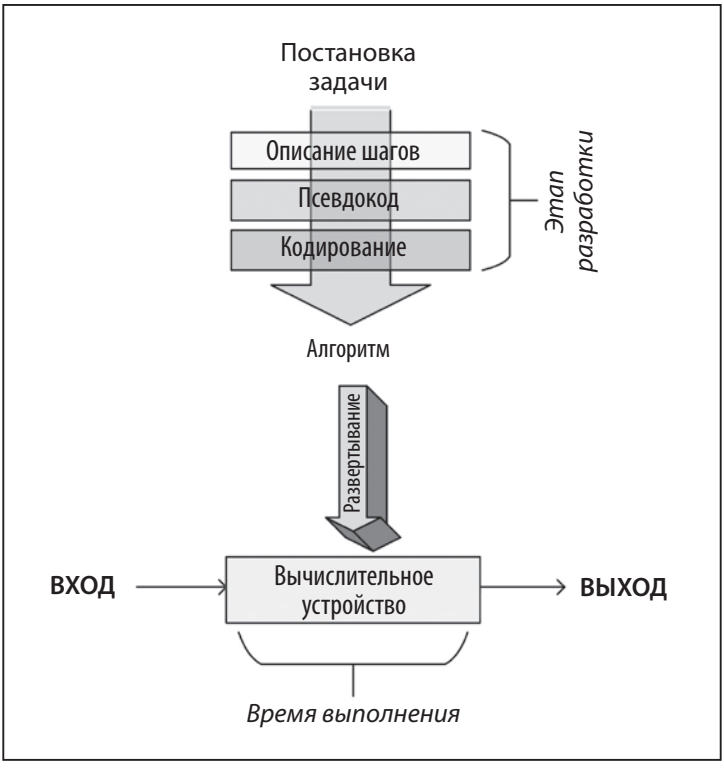

Рис. 1.2

Как только псевдокод готов (см. следующий раздел), мы можем написать код алгоритма, используя выбранный язык программирования.

Практический пример псевдокода

Ниже представлен псевдокод алгоритма распределения ресурсов, называемого *SRPMP*. В кластерных вычислениях возможно множество ситуаций, когда па-

раллельные задачи необходимо выполнить на наборе доступных ресурсов, в совокупности называемых *пулом ресурсов*. Данный алгоритм назначает задачи ресурсу и создает маппинг-сет (mapping set), называемый Ω (омега). Представленный псевдокод отражает логику и последовательность алгоритма, что более подробно объясняется в следующем разделе.

```
1: BEGIN Mapping_Phase
2: Ω = { }
3: k = 1
4: FOREACH Ti∈T
5:     ωi = A(Δk,Ti)
6:     add {ωi,Ti} to Ω
7:     state_changeTi [STATE 0: Idle/Unmapped] → [STATE 1: Idle/Mapped]
8:     k=k+1
9:     IF (k>q)
10:       k=1
11:    ENDIF
12: END FOREACH
13: END Mapping_Phase
```

Давайте построчно разберем этот алгоритм.

1. Мы начинаем маппинг с выполнения алгоритма. Маппинг-сет Ω пуст.
2. Первый раздел выбирается в качестве пула ресурсов для задачи T_1 (см. строку 3 кода). *Целевой телевизионный рейтинг* (Television Rating Point, TRPs) итеративно вызывает алгоритм *ревматоидного артрита* (Rheumatoid Arthritis, RA) для каждой задачи T_i с одним из разделов, выбранным в качестве пула ресурсов.
3. Алгоритм RA возвращает набор ресурсов, выбранных для задачи T_i, представленный посредством ω_i (см. строку 5 кода).
4. T_i и ω_i добавляются в маппинг-сет (см. строку 6 кода).
5. Состояние T_i меняется со `STATE 0:Idle/Mapping` на `STATE 1:Idle/Mapped` (см. строку 7 кода).
6. Обратите внимание, что для первой итерации выбран первый раздел и `k=1`. Для каждой последующей итерации значение `k` увеличивается до тех пор, пока не достигнуто `k>q`.
7. Если переменная `k` становится больше `q`, она сбрасывается в `1` (см. строки 9 и 10 кода).
8. Этот процесс повторяется до тех пор, пока каждой задаче не будет присвоен набор используемых ресурсов, что будет отражено в маппинг-сете, называемом Ω.

9. Как только задаче присваивается набор ресурсов на этапе маппинга, она запускается на выполнение.

Использование сниппетов

С ростом популярности такого простого, но мощного языка программирования, как Python, приобретает популярность и альтернативный подход, который заключается в представлении логики алгоритма непосредственно на языке программирования, но в несколько упрощенной версии. Как и псевдокод, такой фрагмент кода отражает основную логику и структуру предлагаемого алгоритма без указания подробностей. Иногда его называют *сниппетом*. В этой книге сниппеты используются вместо псевдокода везде, где это возможно, поскольку они экономят один дополнительный шаг. Например, давайте рассмотрим простой сниппет для функции Python, которая меняет местами значения двух переменных:

```
define swap(x, y)
    buffer = x
    x = y
    y = buffer
```

Обратите внимание, что сниппеты не всегда могут заменить псевдокод. В псевдокоде мы иногда резюмируем много строк кода в виде одной строки псевдокода, отражая логику алгоритма, при этом не отвлекаясь на излишние детали.

Создание плана выполнения

С помощью псевдокода или сниппетов не всегда возможно задать логику более сложных распределенных алгоритмов. Разработку таких алгоритмов можно разбить на этапы, выполняющиеся в определенном порядке. Верная стратегия разделения крупной задачи на оптимальное количество этапов с должной очередностью имеет решающее значение для эффективного исполнения алгоритма.

Нам необходимо и применить эту стратегию, и одновременно полностью отразить логику и структуру алгоритма. План выполнения — один из способов детализации того, как алгоритм будет разделен на множество задач. Задача может представлять собой сопоставления или преобразования, которые сгруппированы в блоки, называемые *этапами*. На следующей диаграмме (рис. 1.3) показан план, который создается средой выполнения Apache Spark перед исполнением

алгоритма. В нем подробно описываются задачи времени выполнения, на которые будет разделено задание, созданное для выполнения нашего алгоритма.

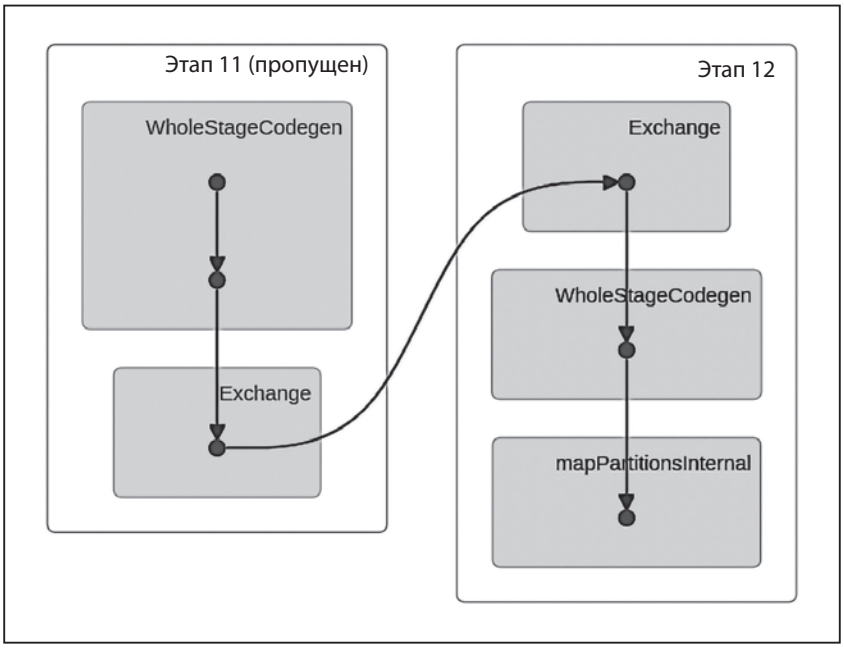

Рис. 1.3

Обратите внимание, что на данной диаграмме представлены пять задач, которые были разделены на два разных этапа: *этап 11* и *этап 12*.

ВВЕДЕНИЕ В БИБЛИОТЕКИ PYTHON

После разработки алгоритм должен быть реализован на языке программирования в соответствии с проектом. Для этой книги я выбрал Python, так как это гибкий язык программирования с открытым исходным кодом. Python также является базовым языком для инфраструктур облачных вычислений, приобретающих все более важное значение, таких как *Amazon Web Services* (AWS), *Microsoft Azure* и *Google Cloud Platform* (GCP).

Официальная домашняя страница Python доступна по адресу https://www.python.org/. Здесь вы найдете инструкции по установке и полезное руководство для начинающих.

Если вы раньше не использовали Python, рекомендуем ознакомиться с этим руководством в целях самообразования. Базовое представление о языке Python поможет лучше понять концепции, представленные в этой книге.

Установите последнюю версию Python 3. На момент написания это версия 3.7.3, которую мы и будем использовать для выполнения упражнений в книге.

Библиотеки Python

Python — это язык общего назначения. Он разработан таким образом, чтобы обеспечить минимальную функциональность. В зависимости от цели использования Python вам придется установить дополнительные библиотеки. Самый простой способ — программа установки pip. Следующая команда `pip` устанавливает дополнительную библиотеку:

```
pip install a_package
```

Установленные ранее библиотеки необходимо периодически обновлять, чтобы иметь возможность использовать новейшие функциональные возможности. Обновление запускается с помощью флага `upgrade`:

```
pip install a_package --upgrade
```

Другим дистрибутивом Python для научных вычислений является Anaconda. Его можно загрузить по адресу https://www.anaconda.com.

В дистрибутиве Anaconda применяется следующая команда для установки новых библиотек:

```
conda install a_package
```

Для обновления существующих библиотек дистрибутива Anaconda:

```
conda update a_package
```

Существует множество доступных библиотек Python. Некоторые важные библиотеки, имеющие отношение к алгоритмам, описаны далее.

Экосистема SciPy

Scientific Python (SciPy) — произносится как *сай пай* — это группа библиотек Python, созданных для научного сообщества. Она содержит множество функций, включая широкий спектр генераторов случайных чисел, программ линейной

алгебры и оптимизаторов. SciPy — это комплексная библиотека, и со временем специалисты разработали множество расширений для ее настройки и дополнения в соответствии со своими потребностями.

Ниже приведены основные библиотеки, которые являются частью этой экосистемы:

- *NumPy.* При работе с алгоритмами требуется создание многомерных структур данных, таких как массивы и матрицы. NumPy предлагает набор типов данных для массивов и матриц, которые необходимы для статистики и анализа данных. Подробную информацию о NumPy можно найти по адресу http://www.numpy.org/.
- *scikit-learn*. Это расширение для машинного обучения является одним из самых популярных расширений SciPy. Scikit-learn предоставляет широкий спектр важных алгоритмов машинного обучения, включая классификацию, регрессию, кластеризацию и проверку моделей. Вы можете найти более подробную информацию о scikit-learn по адресу http://scikit-learn.org/.
- *pandas.* Библиотека программного обеспечения с открытым исходным кодом. Она содержит сложную табличную структуру данных, которая широко используется для ввода, вывода и обработки табличных данных в различных алгоритмах. Библиотека pandas содержит множество полезных функций, а также обеспечивает высокую производительность с хорошим уровнем оптимизации. Более подробную информацию о pandas можно найти по адресу http://pandas.pydata.org/.
- *Matplotlib*. Содержит инструменты для создания впечатляющих визуализаций. Данные могут быть представлены в виде линейных графиков, точечных диаграмм, гистограмм, круговых диаграмм и т. д. Более подробную информацию можно найти по адресу https://matplotlib.org/.
- *Seaborn*. Считается аналогом популярной библиотеки ggplot2 в R. Она основана на Matplotlib и предлагает великолепный расширенный интерфейс для графического отображения статистики. Подробную информацию можно найти по адресу https://seaborn.pydata.org/.
- *IPython*. Усовершенствованная интерактивная консоль, предназначенная для того, чтобы упростить написание, тестирование и отладку кода Python.
- *Запуск Python в интерактивном режиме*. Этот режим программирования полезен для обучения и экспериментов с кодом. Программы на Python могут быть сохранены в текстовом файле с расширением `.py`, и этот файл можно запустить из консоли.

Реализация Python с помощью Jupyter Notebook

Еще один способ запуска программ на Python — через Jupyter Notebook. Jupyter Notebook представляет собой пользовательский интерфейс для разработки на основе браузера. Именно Jupyter Notebook используется для демонстрации примеров кода в этой книге. Возможность комментировать и описывать код с помощью текста и графики делает Jupyter Notebook идеальным инструментом для демонстрации и объяснения любого алгоритма, а также отличным инструментом для обучения.

Чтобы запустить программу, вам нужно запустить процесс `Jupyter-notebook`, а затем открыть свой любимый браузер и перейти по ссылке http://localhost:8888 (рис. 1.4).

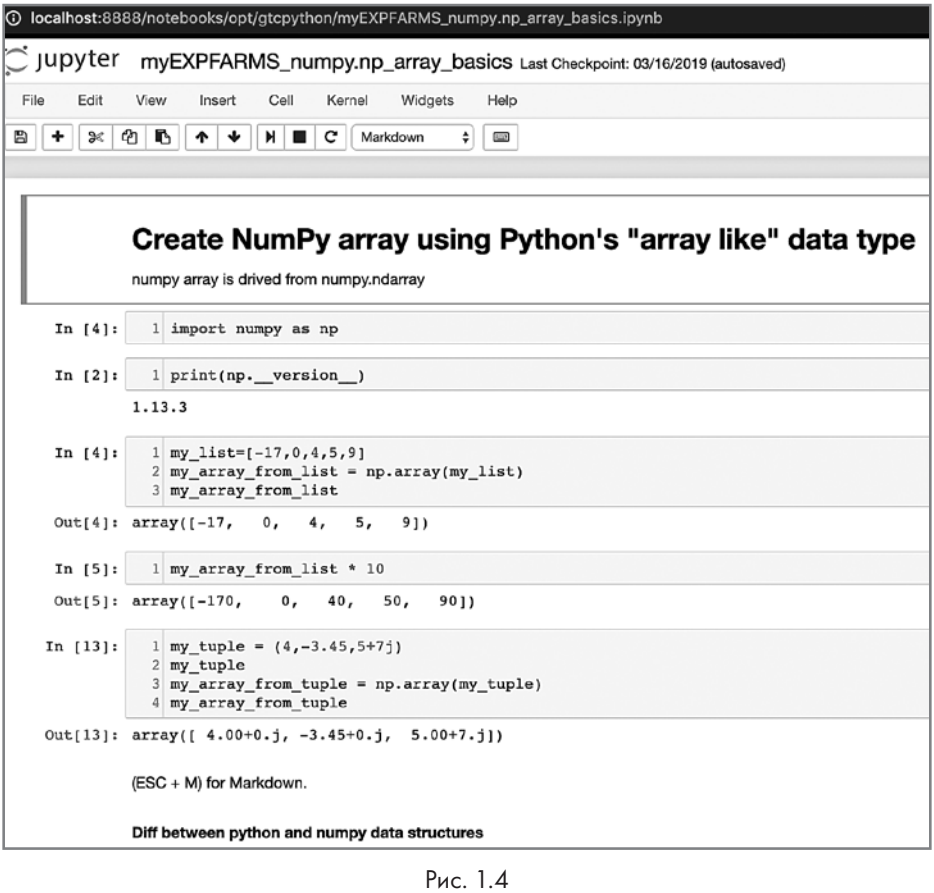

Рис. 1.4

Обратите внимание, что Jupyter Notebook состоит из ряда блоков, называемых *ячейками*.

МЕТОДЫ РАЗРАБОТКИ АЛГОРИТМОВ

Алгоритм — это математическое решение реальной проблемы. При разработке и настройке алгоритма мы должны задавать себе следующие вопросы:

- *Вопрос 1*. Даст ли алгоритм тот результат, который мы ожидаем?
- *Вопрос 2*. Является ли данный алгоритм оптимальным способом получения этого результата?
- *Вопрос 3*. Как алгоритм будет работать с большими наборами данных?

Важно оценить сложность задачи, прежде чем искать для нее решение. При разработке подходящего решения полезно охарактеризовать задачу с точки зрения ее требований и сложности. Как правило, алгоритмы можно разделить на следующие типы в зависимости от характеристик задачи:

- *Алгоритмы с интенсивным использованием данных*. Такие алгоритмы предъявляют относительно простые требования к обработке. Примером является алгоритм сжатия, применяемый к огромному файлу. В таких случаях размер данных обычно намного больше, чем объем памяти процессора (одной ноды или кластера), поэтому для эффективной обработки данных в соответствии с требованиями может потребоваться итеративный подход.
- *Вычислительноемкие алгоритмы*. Такие алгоритмы предъявляют значительные требования к обработке, но не задействуют больших объемов данных. Пример — алгоритм поиска очень большого простого числа. Чтобы добиться максимальной производительности, нужно найти способ разделить алгоритм на фазы так, чтобы хотя бы некоторые из них были распараллелены.
- *Вычислительноемкие алгоритмы с интенсивным использованием данных*. Хорошим примером здесь служат алгоритмы, используемые для анализа эмоций в видеотрансляциях. Такие алгоритмы являются наиболее ресурсоемкими алгоритмами и требуют тщательной разработки и разумного распределения доступных ресурсов.

Чтобы определить сложность и ресурсоемкость задачи, необходимо изучить параметры данных и вычислений, чем мы и займемся в следующем разделе.