Что мы планируем и ищем?

Если подумать о том, какие существуют составляющие интеллекта, то в первую очередь необходимо назвать способность к планированию. Мы планируем путешествие по незнакомой стране, начало нового проекта, написание функции в коде и бесчисленное множество других ситуаций. *Планирование* происходит на разных уровнях детализации и в разных контекстах, что позволяет добиваться наиболее эффективного результата (рис. 2.1).

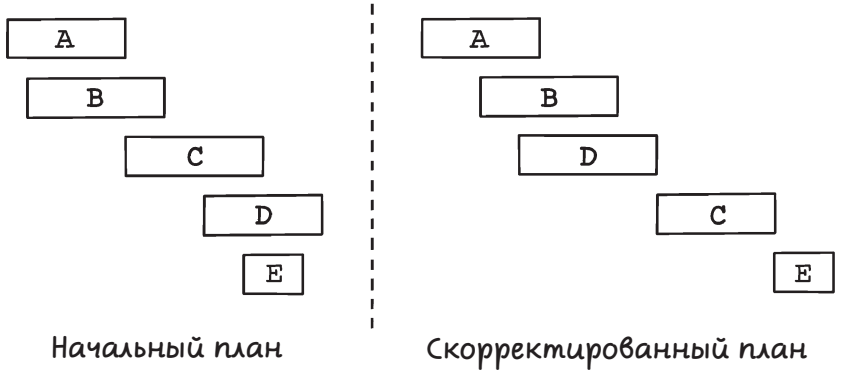


Рис. 2.1. Пример изменения планов проекта

Планы редко реализуются так, как было изначально задумано. Мы живем в постоянно меняющейся среде, поэтому просто невозможно учесть все переменные и неизвестные. Мы практически всегда отклоняемся от начального плана из-за изменений в предметной области. Нам приходится менять планы, и иногда не раз, когда мы уже прошли несколько этапов, но происходят неожиданные события, которые приходится учитывать для достижения цели. В результате фактический план, как правило, отличается от начального.

Поиск — это способ управлять планированием путем разделения плана на этапы. Например, когда мы планируем путешествие, то ищем интересующие нас маршруты, оцениваем остановки на пути и сопутствующие возможности. Мы также ищем, где жить и чем заниматься, исходя из предпочтений и бюджета. В зависимости от результатов всех этих поисков меняется и план.

Предположим, что мы задумали поездку на побережье, расстояние до которого около 500 км, и решили сделать две остановки: одну в зоопарке и вторую в пиццерии. По прибытии мы планируем переночевать в домике на берегу и поучаствовать в трех мероприятиях. Путь до места назначения займет приблизительно 8 часов. Мы также решаем сократить путь по платной автостраде, съезд на которую будет после ресторана. Но эта дорога открыта только до 2:00.

Путешествие начинается, и все идет по плану. Мы приезжаем в зоопарк и знакомимся с чудесными животными. Затем путь продолжается, и уже подходит время перекусить в ресторане, но по приезде оказывается, что он недавно

закрылся. Теперь нужно скорректировать план и найти поблизости другой ресторан или кафе, которые нам подойдут.

Проехав еще какое-то расстояние, мы находим ресторан, с удовольствием съедаем пиццу и возвращаемся на дорогу. При подъезде к частной автостраде мы понимаем, что время уже 2:20 и она закрыта. И здесь снова приходится менять план. Мы ищем объездной путь и выясняем, что он добавит к поездке еще 120 км. Это значит, что нам нужно найти другое место для ночлега еще до прибытия на побережье. Мы находим такое место и меняем маршрут. Из-за потраченного времени мы успеваем поучаствовать только в двух из трех мероприятий. Нам пришлось искать альтернативные решения, и поэтому наш первоначальный план серьезно изменился, но в конечном итоге поездка обернулась очень интересным приключением, и мы прекрасно провели время.

Этот пример показывает, как поиск используется для планирования и влияет на его итог для достижения намеченной цели. Меняются обстоятельства — меняются и цели. При этом путь к ним также неизбежно придется корректировать (рис. 2.2). Изменения в планах практически всегда внезапны и вносятся по мере необходимости.

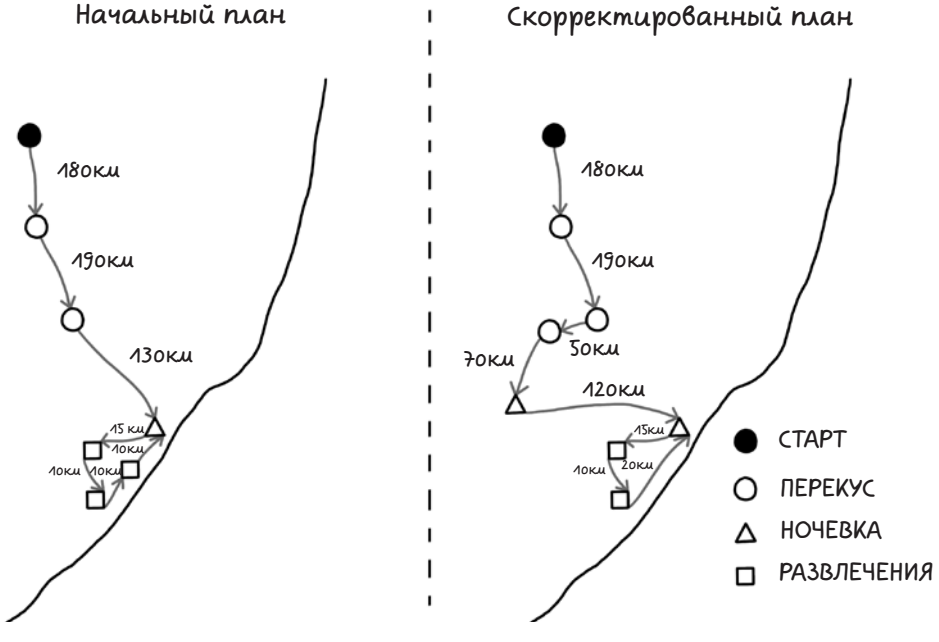


Рис. 2.2. Исходный план поездки и его измененный конечный вариант

Поиск подразумевает оценку будущих состояний на пути к цели для нахождения оптимального пути из этих состояний к намеченной точке. Эта глава посвящена различным подходам к поиску, различающимся по типу решаемых

задач. Поиск — не новый, но мощный инструмент разработки интеллектуальных алгоритмов.

Стоимость вычислений: причина использования умных алгоритмов

В программировании функции состоят из операций, и согласно принципу работы современных компьютеров разные функции требуют разного количества времени на обработку. Чем больше требуется вычислений, тем более дорогостоящей получается функция. Для описания сложности функции или алгоритма используется *нотация «О большое»*. Эта нотация моделирует количество операций, необходимых при возрастающем размере ввода. Вот некоторые примеры и связанные с ними сложности:

- *Операция, выводящая на экран надпись* `Hello World`, — это единичная операция, следовательно, стоимость вычисления равна O(1).
- *Функция, перебирающая список и выводящая каждый его элемент*, — количество операций зависит от количества элементов в списке. Стоимость будет составлять O(n).
- *Функция, сравнивающая каждый элемент списка с каждым элементом другого списка*, — стоимость операции $O(n^2)$.

На рис. 2.3 отражена различная стоимость алгоритмов. Те из них, которые требуют увеличения количества операций по мере увеличения входных данных, выполняются медленнее всех. При этом алгоритмы, для которых требуется постоянное количество операций по мере роста количества входных данных, выполняются быстрее.

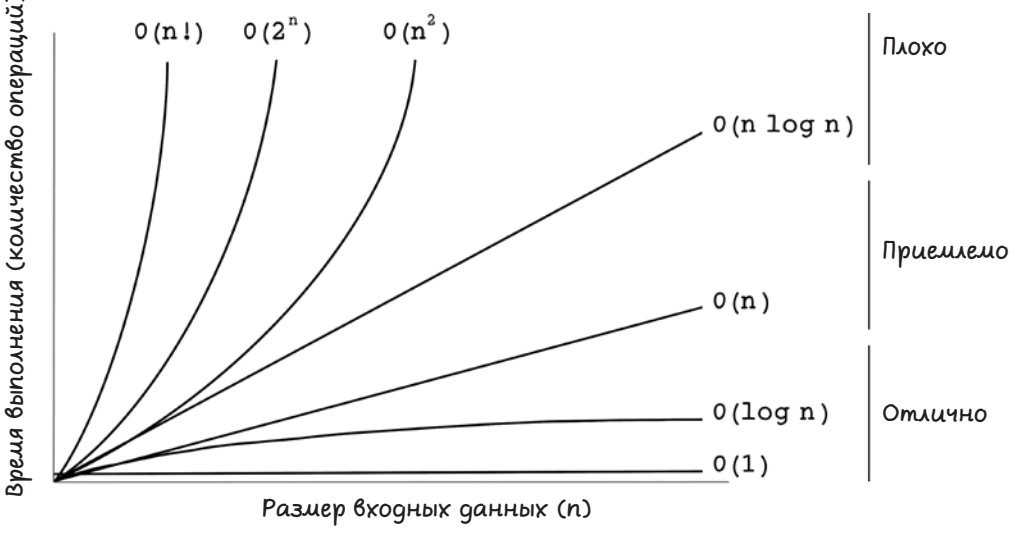


Рис. 2.3. Сложность «О большое»

Понимание того, что различные алгоритмы имеют различную вычислительную стоимость, очень важно, потому что в ее минимизации и состоит сама цель интеллектуальных алгоритмов, способных справляться с задачами быстро и эффективно. Теоретически почти любую задачу можно решить перебором возможных решений, пока не будет найдено лучшее. Но в реальности на подобные вычисления ушли бы часы, недели и даже годы, поэтому они не подходят для практического применения.

Задачи, подходящие для алгоритмов поиска

Практически любую задачу, требующую очередности решений, можно реализовать с помощью поисковых алгоритмов, которые подбираются согласно типу задачи и размеру области поиска. В зависимости от выбранного алгоритма и конфигурации можно найти хорошее или лучшее решение. Другими словами, можно найти хорошее, но не обязательно лучшее решение. Когда мы говорим «хорошее решение» или «лучшее решение», то подразумеваем степень его эффективности в достижении поставленной цели.

Возьмем, к примеру, сценарий игрока в лабиринте, который стремится найти кратчайший путь к цели. Предположим, что он находится в квадратном лабиринте размером 10 × 10 клеток, как показано на рис. 2.4. На карте лабиринта звездой отмечена интересующая нас цель, а также добавлены препятствия. Задача — найти путь к звезде, обойдя препятствия и затратив на это как можно меньше шагов. Перемещаться можно на север, юг, восток и запад, но не по диагонали.

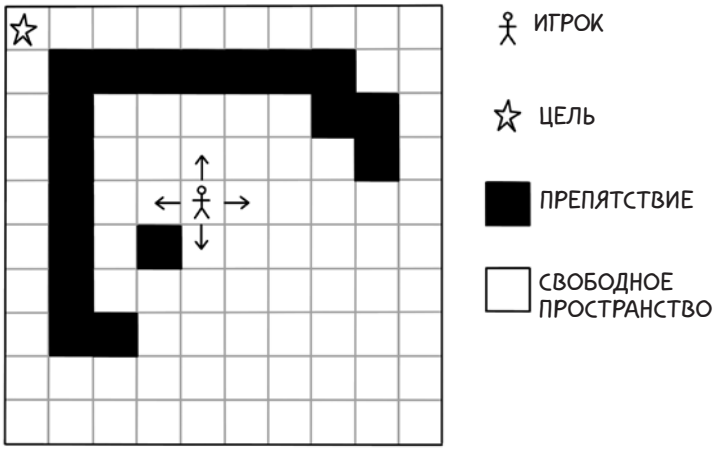


Рис. 2.4. Пример задачи с лабиринтом

Как найти кратчайший путь и обойти препятствия? Оценивая задачу с позиции человека, можно попробовать каждый вариант и подсчитать количество ходов. Таким образом, методом проб и ошибок можно найти кратчайшие пути, учитывая, что лабиринт относительно мал.

На рис. 2.5 показаны некоторые из возможных путей к цели, но обратите внимание, что в первом варианте мы ее не достигаем.

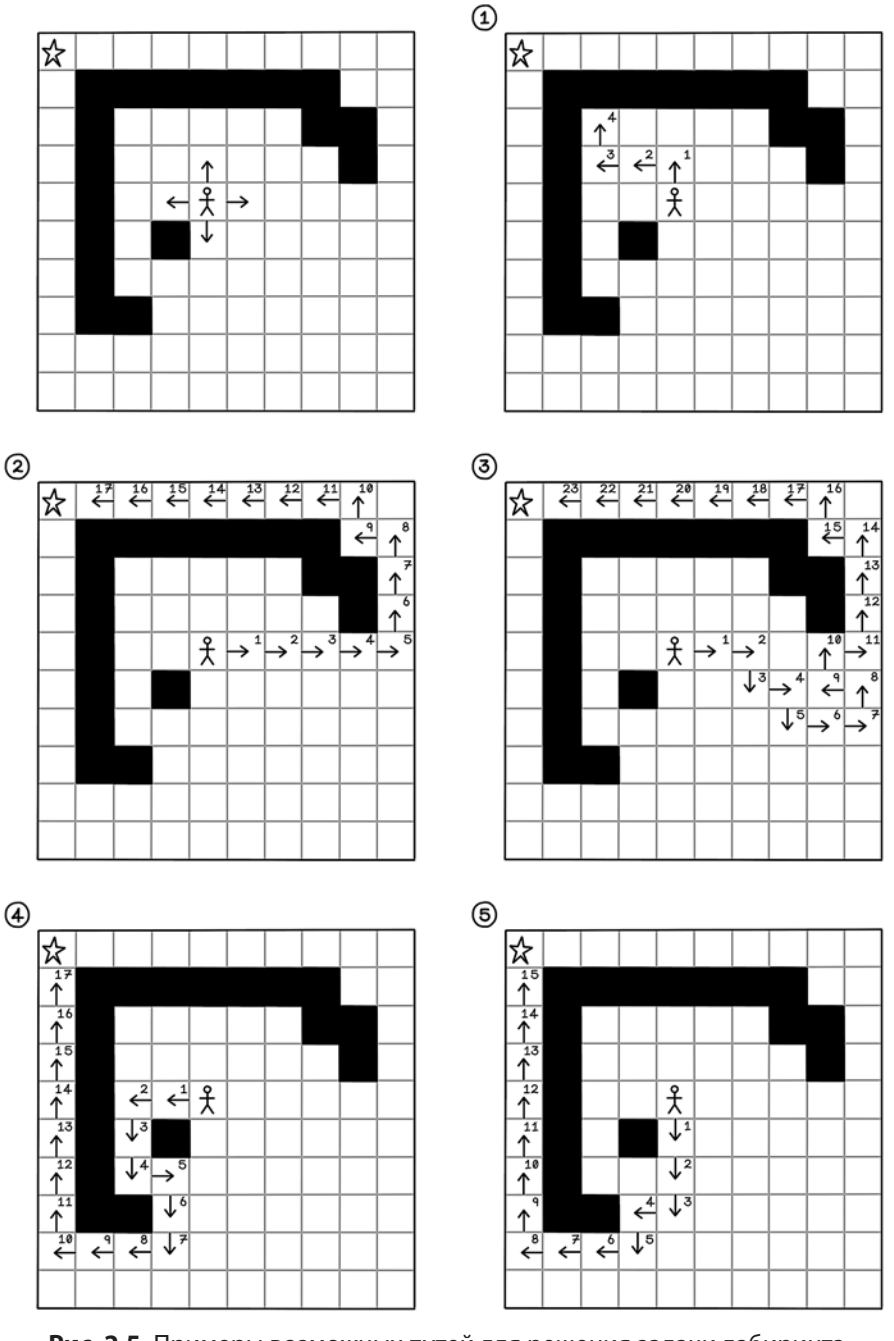


Рис. 2.5. Примеры возможных путей для решения задачи лабиринта

Рассматривая лабиринт и подсчитывая количество клеток в различных направлениях, можно найти несколько решений задачи. Чтобы найти четыре успешных варианта из неизвестного количества возможных, мы предприняли пять попыток. Для вычисления же всех возможных решений потребовалось бы очень много усилий:

- Попытка 1 не является подходящим решением. В ней выполняется 4 хода и цель при этом не достигается.
- Попытка 2 решает задачу, на что уходит 17 ходов.
- Попытка 3 решает задачу и требует совершения 23 ходов.
- Попытка 4 решает задачу снова за 17 ходов.
- Попытка 5 оказывается наилучшим доступным решением, так как требует всего 15 ходов. Несмотря на то что эта попытка оптимальна, она была найдена случайно.

Если бы лабиринт был намного больше, как, например, на рис. 2.6, то, чтобы найти кратчайший путь вручную, потребовалось бы огромное количество времени. Тут-то и пригодятся поисковые алгоритмы.

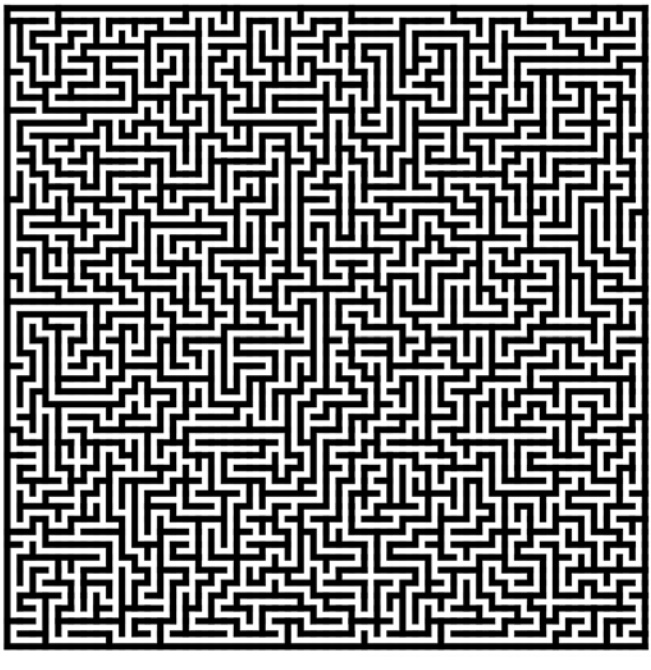


Рис. 2.6. Пример задачи с большим лабиринтом

Наше преимущество как людей состоит в том, что мы способны визуально воспринять задачу, понять ее и найти решение на основе заданных параметров.

Мы понимаем и интерпретируем данные абстрактно. Компьютер пока что не способен понимать обобщенную информацию в столь же естественной форме, как это делаем мы. Поэтому пространство задачи нужно представить в виде, который будет пригоден для вычисления и обработки поисковым алгоритмом.

Представление состояния: создание структуры для представления пространства задачи и решений

При выражении данных и информации в понятном для компьютера виде необходимо кодировать их логически, чтобы это понимание было объективным. Несмотря на то что данные будет субъективно кодировать человек, выполняющий эту задачу, необходимо предусмотреть краткий и согласованный способ их представления.

Проясним разницу между данными и информацией. *Данные* — это сырые факты о чем-либо, а *информация* — это интерпретация этих фактов, их анализ для конкретной области. Чтобы информация стала значимой, требуется контекст и обработка данных. Вот пример: каждое отдельное пройденное в лабиринте расстояние представляет собой данные, а сумма всех расстояний является информацией. В зависимости от точки восприятия, уровня детализации и желаемого результата классификация чего-либо как данных или информации может быть субъективной для контекста и человека либо команды (рис. 2.7).

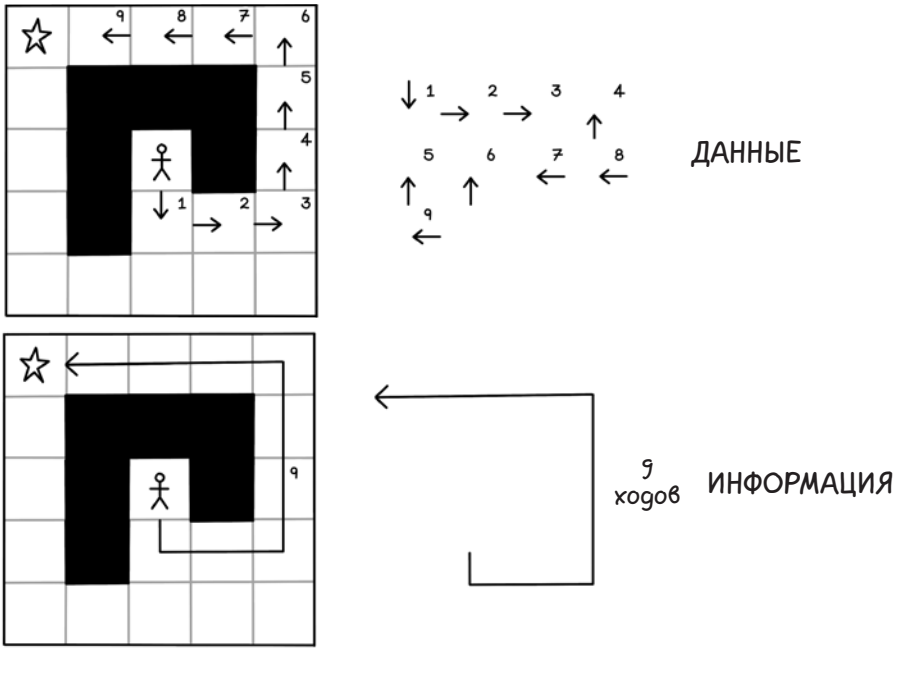


Рис. 2.7. Данные и информация

Структуры данных — это понятие компьютерной науки; с помощью структуры данные представляются в таком виде, чтобы их можно было обрабатывать с применением алгоритмов. Это абстрактный тип данных, состоящий из данных и операций, организованных определенным образом. При этом используемая структура определяется контекстом задачи и целью.

Примером такой структуры является *массив*, который представляет собой просто набор данных. Разные типы массивов обладают различными свойствами, которые делают их более подходящими для тех или иных целей. В зависимости от используемого языка программирования массив может содержать значения разных типов либо только одного типа; кроме того, в нем могут быть запрещены повторяющиеся значения. Различные виды массивов, как правило, называются тоже по-разному. Особенности и ограничения различных структур данных ведут к более эффективным вычислениям (рис. 2.8).

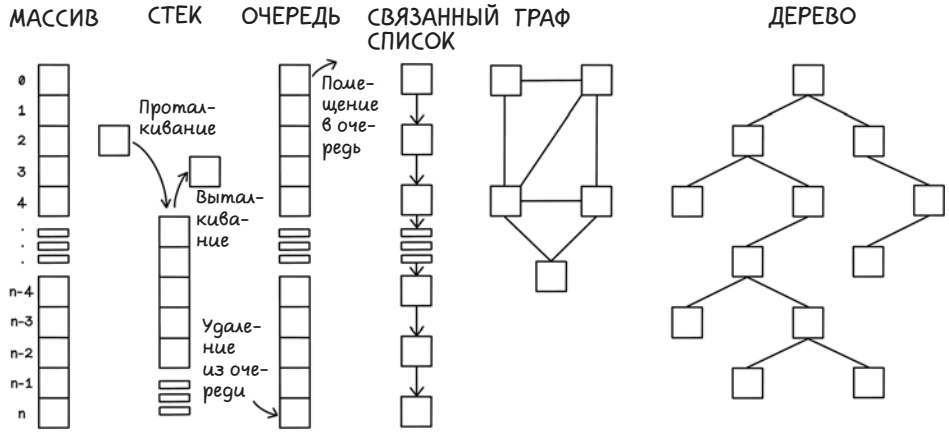
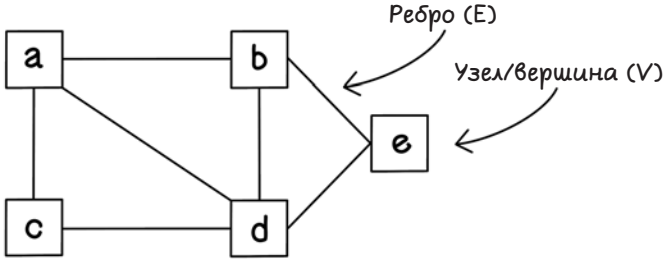


Рис. 2.8. Структуры данных, используемые с алгоритмами

Структуры дерева и графа — идеальная организация данных для поисковых алгоритмов.

Графы: представление задач поиска и решений поиска

Граф — это структура данных, содержащая несколько связанных между собой состояний. Каждое состояние называется *вершиной* (или *узлом*), а связь между двумя состояниями — *ребром*. Понятие графа происходит из теории графов в математике и используется для моделирования связей между объектами. Это полезные структуры данных, которые благодаря простоте визуального представления и логической природе легко понятны человеку, что очень удобно для обработки с помощью различных алгоритмов (рис. 2.9).



V = {a, b, c, d, e}

E = {ab, ac, ad, bd, be, cd, de}

Рис. 2.9. Обозначения, используемые для записи графов

На рис. 2.10 приведен граф поездки на побережье, о которой мы недавно говорили. Каждая остановка представлена в виде вершины, ребра между вершинами отражают точки, между которыми мы перемещались, а веса каждого ребра указывают пройденное расстояние.

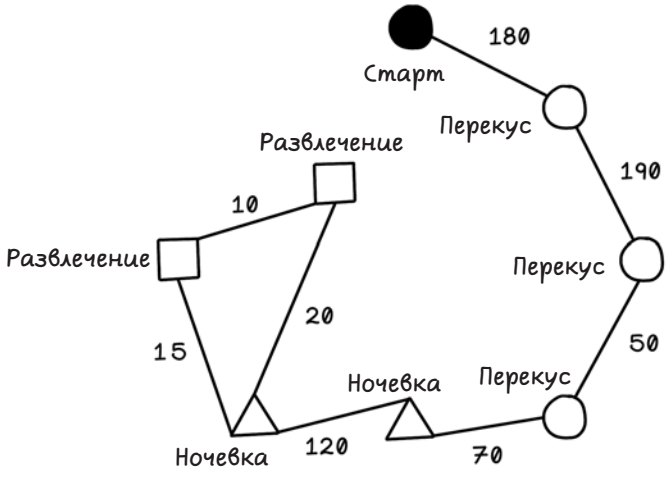


Рис. 2.10. Пример описания поездки в виде графа

Представление графа как конкретной структуры данных

Для эффективной обработки алгоритмами граф можно представить несколькими способами. По сути, его можно отразить с помощью массива массивов, указывающего связи между вершинами (рис. 2.11). Иногда полезно создать еще один массив, просто перечисляющий все вершины графа, чтобы отдельные вершины не приходилось выводить на основе связей.