Проекты для курса «Программирование на C++ и Python»

В.С. Воробьев

14 декабря 2019 г.

Аннотация

В этом документе представлены описания возможных проектов — заданий на пятёрку. Выполнение проекта предполагает этап проектирования программы. Если вы пишете на `C++`, подумайте заранее о наборе необходимых классов и их интерфейсов, обсудите свой план с семинаристом. Код на `python` также должен быть хорошо структурирован.

Тема проекта не ограничена предложенными здесь вариантами. Студент может предложить свою идею проекта, например, связанную с выполняемой курсовой работой. Тему проекта, особенно если она предложена студентом, необходимо обсудить с семинаристом.

1 Просачивание

Каждая клетка двумерной прямоугольной сетки размера $N \times N$ может находиться в открытом или закрытом состоянии. Назовём «условием просачивания» состояние, при котором верхний и нижний края сетки могут быть соединены непрерывным путём через открытые клетки (Рис. 1).

Ваша задача состоит в моделировании такой системы. Необходим быстрый алгоритм для определения условия просачивания. Замечательным свойством такой системы является существование «фазового перехода»: существует пороговое значение p_0 вероятности p открытости клетки. При $p > p_0 + \varepsilon$, $\varepsilon \ll p_0$, просачивание происходит почти наверняка, а при $p < p_0 + \varepsilon$ просачивание почти наверняка не происходит. Аналитическое изучение системы предсказывает существование порога, но не даёт его значение.

Исследуйте систему и определите величину порога.

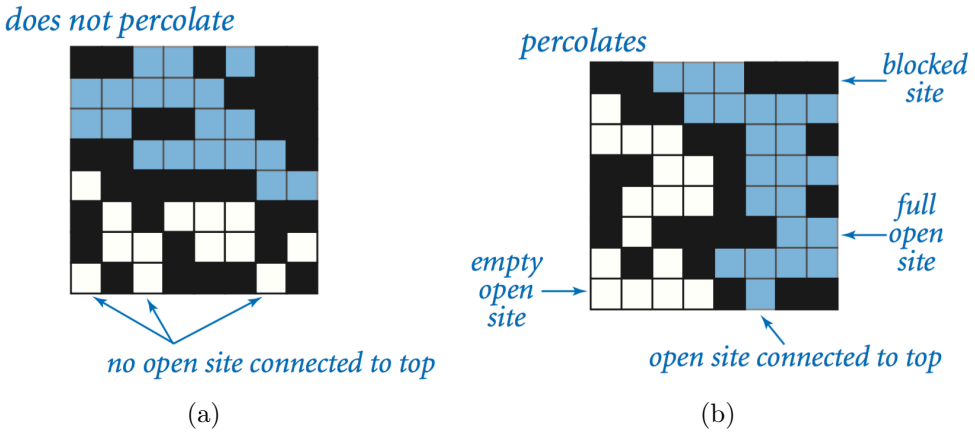


Рис. 1: Поле без просачивания (a) и поле с просачиванием (b) [Источник]

Для детектирования просачивания создайте структуру данных union-find (она же disjoint-set, или «система не пересекающихся множеств»).

При работе в паре, один человек может заниматься алгоритмами и логикой работы, а второй работать над визуализацией сетки и работы алгоритма поиска просачивания.

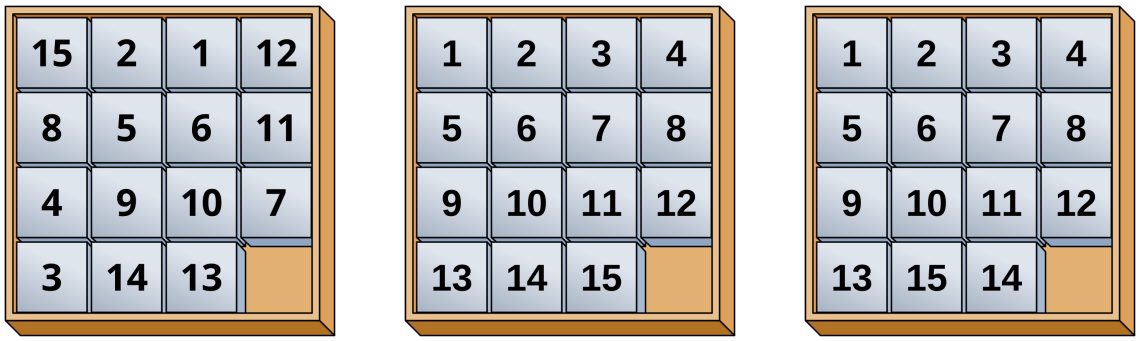
2 Пятнашки

В этом проекте предлагается реализовать алгоритм решения игры-головоломки «Пятнашки». Результатом работы алгоритма должна быть последовательность *оптимальных* шагов, приводящих к собранной позиции, либо вывод о том, что позиция неразрешима (смотрите рисунок 2).

Интересный факт состоит в том, что ровно половина позиций являются неразрешимыми. Чтобы перевести разрешимую позицию в неразрешимую (и наоборот) достаточно поменять местами любые две соседние костяшки.

Для поиска оптимального решения можно использовать алгоритм A^*. Идея алгоритма состоит в выборе хода, который максимально приближает позицию к собранной. «Расстояние» от текущей позиции до собранной можно определять по-разному. Например, с помощью Манхэттенского расстояния:

$$D = \sum_{i=1}^{N} \left(|x_i - \tilde{x}_i| + |y_i - \tilde{y}_i| \right), \tag{1}$$



(a) (b) (c)

Рис. 2: Поле игры «Пятнашки»: (a) не собранная позиция, (b) собранная позиция, (c) неразрешимая позиция.

где суммирование идёт по всем костяшкам, (x_i, y_i) — текущее положение i-й костяшки, и $(\tilde{x}_i, \tilde{y}_i)$ — положение i-й костяшки в собранной позиции.

Алгоритм работы с позицией:

- Проверяем не является ли позиция собранной (если является, то печатаем решение)
- Находим все позиции, в которые можно перейти за один ход, кроме позиции, из которой была получена данная
- Выбираем из этих позиций позицию с наименьшей дистанцией до собранной позиции (если таких позиций несколько, выбираем любую) и переходим к пункту 1

Модифицируйте алгоритм, добавив в него механизм детектирования неразрешимых позиций.

Перед написанием кода подумайте над необходимыми классами и из интерфейсами. Обсудите это с семинаристом.

Проект можно выполнять в паре: один человек занимается описанием доски (включая визуализацию), а второй — алгоритмом решения.

Указание. Достаточно реализовать алгоритм для доски 3×3.

3 Игра Гекс

Гекс (Hex) — это игра, действие которой происходит на гексагональной доске, как показано на рисунке 3. Доска может быть любого размера и различных форм, но традиционно используют доску в форме ромба размера $n \times n$. Правила игры (взято из Wikipedia):

- Один из игроков играет красными фишками, другой — синими. Каждый игрок поочерёдно ставит фишку своего цвета на любое свободное поле. Начинают синие.
- Две противоположные стороны доски окрашены в красный и синий цвета, и называются соответственно красной и синей сторонами. Поля на углах доски относятся к обеим сторонам. Для того чтобы выиграть, игрок должен выстроить цепь из своих фишек, соединив ею стороны своего цвета, то есть красные стремятся построить цепь из красных фишек между двумя красными сторонами доски, а синие — цепь из синих фишек между синими сторонами.

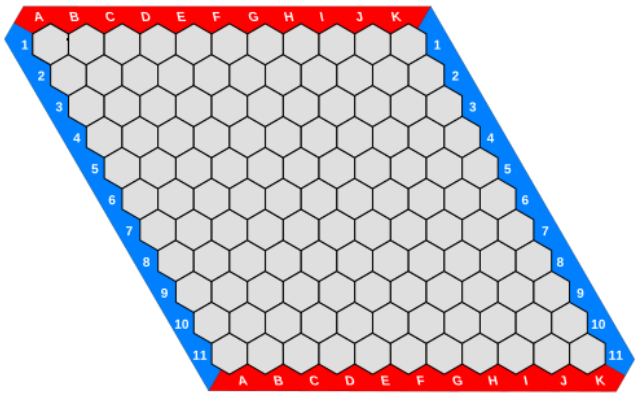


Рис. 3: Поле игры «Hex».

У первого игрока всегда есть победная стратегия, так как лишний ход не может помешать построить цепь. Для уравнивания шансов используется «Правило пирога», которое позволяет второму игроку сменить цвет сразу после того, как первый игрок делает свой первый ход. При игре с «правилом пирога» выигрышная стратегия есть у второго игрока.

Для выполнении этого проекта необходимо:

1. Описать поле и логику игры, в том числе необходим эффективный алгоритм детектирования выигрышной позиции.

2. Написать «движок» — алгоритм, выбирающий следующий ход для данной позиции (детали описаны ниже).

3. *Опционально*: реализовать визуализацию доски и игрового процесса с помощью графической библиотеки.

Последний пункт не является обязательным. Игровой процесс можно реализовать и с помощью консольной графики.

Достаточно сильный движок для Гекс можно написать следующим образом:

1. Для данной позиции составляется список всех возможных ходов.

2. Для каждого возможного хода происходит N *случайных* игр. Под случайной игрой мы понимаем игру, при которой каждый игрок ставит фишку на любую свободную клетку. При этом подсчитывается доля η_i игр, закончившихся победой игрока, который который сейчас делает ход, i — индекс возможного хода.

3. В качестве следующего хода выбирается ход с максимальным η_i.

При $N = 10^4$ уровень игры уже достаточно высокий. Чтобы алгоритм работал за разумное время, необходим эффективный алгоритм проведения случайных игр и определения победителя в каждой из игр.

Придумайте сами как включить в предложенный алгоритм «правило пирога».

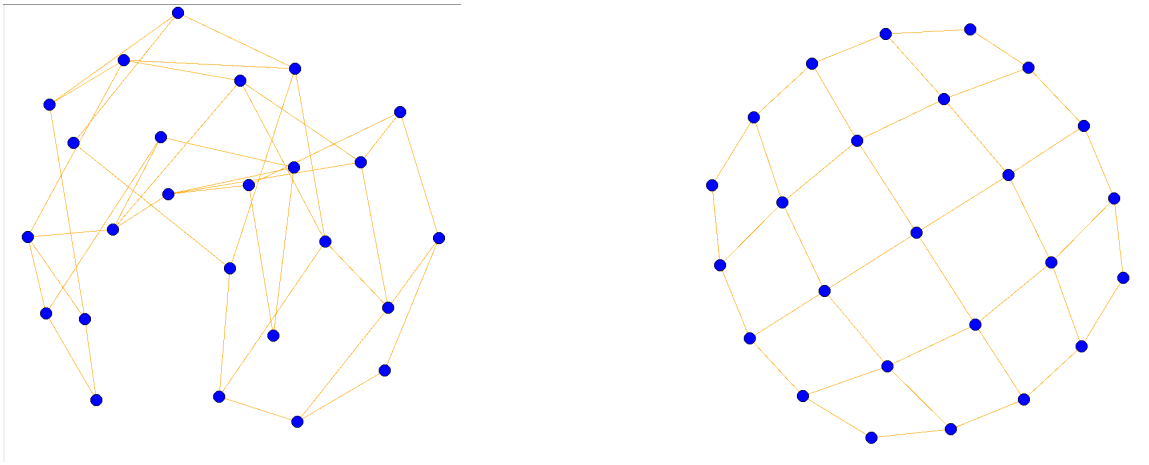
4 Визуализация графа

Визуализация графов на плоскости — классическая задача топологии. Напомним, что граф — это объект, состоящий из множества *вершин* и соединяющих их отрезков (*рёбер*). Существуют различные виды и классификации графов. Мы будем рассматривать графы с *не* направленными вершинами и предполагать, что две вершины могут быть либо не соединены, либо соединены ровно одним ребром.

Предлагаем реализовать динамическую модель визуализации графа. Пусть каждая пара вершин графа отталкиваются друг от друга с силой, зависящей от расстояния между ними, например,

$$\mathbf{F}_{12}^{\text{rep}}(\mathbf{r}_1, \mathbf{r}_2) = k_{\text{rep}} \left|\mathbf{r}_1 - \mathbf{r}_2\right|^{\alpha_{\text{rep}}} \mathbf{n_{12}}, \quad k_{\text{rep}} > 0, \quad \alpha_{\text{rep}} < 0, \tag{2}$$

где $\mathbf{n_{12}}$ — единичный вектор, направленный вдоль вектора $\mathbf{r}_1 - \mathbf{r}_2$.



(a) (b)

Рис. 4: Отрисовка графа: (a) случайное расположение вершин (начальное положение) и (b) расположение после выполнения алгоритма.

Далее, пусть каждому ребру графа соответствует сила притяжения, зависящая от длины ребра

$$\mathbf{F}_{12}^{\text{att}}(\mathbf{r}_1, \mathbf{r}_2) = k_{\text{att}} \left|\mathbf{r}_1 - \mathbf{r}_2\right|^{\alpha_{\text{att}}} \mathbf{n_{12}}, \quad k_{\text{att}} < 0, \quad \alpha_{\text{att}} > 0. \tag{3}$$

Алгоритм состоит в выполнении эволюции такой системы. Начальное расположение вершин может быть случайным, после чего система эволюционирует до

тех пор, пока не окажется в минимуме своей потенциальной энергии. Результат работы подобного алгоритма показан на рисунке 4.

Самая простая реализация алгоритма состоит в вычислении смещения вершины, пропорциональному действующей на неё силы. Более продвинутая версия алгоритма может включать «массу» вершины, её текущую скорость и смещение, которое определяется вторым законом Ньютона.

Исследуйте поведение алгоритма в зависимости от значений параметров k_{att}, k_{rep}, α_{att}, α_{rep} и величины смещения на каждом шаге эволюции системы.

Алгоритм можно улучшить, введя «штраф» на малые углы между смежными рёбрами. Для этого в модель следует добавить отталкивающую силу между центрами смежных рёбер (рёбер, имеющих общую вершину).

5 Логистическая регрессия

Задача классификации объектов — одна из основных задач машинного обучения. Решение этой задачи состоит в построении статистической модели, которая берёт на вход признаки объекта x_k, $k \in \{1, r\}$, и возвращает индекс класса y, которому принадлежит этот объект. В случае бинарной классификации модель может возвращать 0, если модель предсказывает, что объект принадлежит одному классу и 1 — если другому.

Рассмотрим следующую модель для бинарной классификации:

1. Составим линейную комбинацию признаков объекта

$$z = \sum_{j=1}^{r} w_j x_j + b. \tag{4}$$

2. Для классификации будем использовать *нелинейную* функцию, принимающую значения от 0 до 1

$$y = \begin{cases} 1, & f(z) > 0.5 \\ 0, & f(z) < 0.5 \end{cases} \tag{5}$$

В качестве функции $f(z)$ чаще всего использую «сигмоиду»

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{6}$$

Чтобы модель могла классифицировать объекты, необходимо найти подходящие значения *весовых* коэффициентов w_j и *смещения* b. Для нахождения значений коэффициентов будем использовать N *тренировочный набор объектов* — множество объектов с известным типом. Оптимальные параметры модели соответствуют минимуму *целевой функции* (objective function)

$$J(w_j, b) = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \ln f(z_i) + (1 - y_i) \ln \left(1 - f(z_i)\right) \right]. \tag{7}$$

Функция (7) называется *перекрёстной энтропией* (cross entropy) и имеет происхождение в теории информации. Обоснование использования этой функции для оптимизации параметров модели выходит за рамки этого текста.

Метод градиентного спуска

В предыдущем разделе мы описали логистическую регрессию, но не показали способа минимизации целевой функции (7). Минимизация нелинейной функций f, зависящей от k параметров x_i, $i \in \{1, k\}$, можно выполнить методом *градиентного спуска*. В базовом варианте этот метод сводится к следующему алгоритму:

1. Задаём начальное значение вектора параметров $\vec{x}$.
2. Находим градиент функции $\vec{\nabla} f$. Напомним, что
$$\vec{\nabla} f \equiv \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_k} \right). \tag{8}$$
3. Изменяем значения параметров следующим образом:
$$\vec{x} \to \vec{x} - \eta \vec{\nabla} f\left(\vec{x}\right), \tag{9}$$
где η — параметр, определяющий скорость градиентного спуска.
4. Если не выполнено условие остановки, возвращаемся к пункту 2, иначе выходим из алгоритма.

Значение параметра η определяет поведение алгоритма. При небольших значениях η алгоритм работает стабильнее, но медленнее, а при слишком больших значениях происходят слишком большие скачки в пространстве параметров и алгоритм может работать неверно.

Условие остановки алгоритма можно задавать по-разному. Например, можно выполнять фиксированное количество итераций. Другим вариантом может быть отслеживание значения функции f на каждой итерации. Выходить из алгоритма в этом случае можно, если значение функции перестало уменьшаться или оно изменяется слишком медленно (напомню, что если параметры соответствуют минимуму, то градиент функции равен нулю).

Описанный алгоритм предполагает некоторые свойства функции f, в частности, *существование* минимума. Подробное обсуждение вопросов применимости метода градиентного спуска выходит за рамки нашего обсуждения.

Выпишем явные формулы для поиска минимума целевой функции логистической регрессии (7), предполагая, что $f(z)$ задана в виде сигмоиды (6):

$$\frac{\partial J}{\partial w_j} = \frac{1}{N} \sum_{i=1}^{N} \left[\left(\sigma(z_i) - y_i \right) x_j^{(i)} \right], \quad \frac{\partial J}{\partial b} = \frac{1}{N} \sum_{i=1}^{N} \left[\sigma(z_i) - y_i \right]. \tag{10}$$

При выводе этих выражений мы использовали соотношение

$$\frac{d\sigma(z)}{dz} = \sigma(z)\,(1 - \sigma(z))\,. \tag{11}$$

Упражнения:

1. Реализуйте статистическую модель для бинарной классификации и процедуру тренировки модели методом градиентного спуска. Натренеруйте вашу модель на следующих данных:

   (a) Одномерная модель ($r = 1$). Признак x равномерно распределён от -1 до 1. Если $x < 0$, то объект принадлежит классу A, в противном случае он принадлежит классу B.

   (b) Двумерная модель ($r = 2$). Признаки x_1 и x_2 равномерно распределены в единичном квадрате. Классы A и B заданы следующим условием:

   $$\begin{cases} A: & x_1 \sin\alpha + x_2 \cos\alpha \geq 0 \\ B: & x_1 \sin\alpha + x_2 \cos\alpha < 0, \end{cases} \tag{12}$$

   где α — некоторый угол. При $\alpha = 0$ принадлежность к классу определяется знаком признака x_2.

   (c) Нелинейная двумерная модель ($r = 2$). Признаки x_1 и x_2 равномерно распределены в единичном квадрате. Классы A и B заданы следующим условием:

   $$\begin{cases} A: & (x_1 \cos\alpha - x_2 \sin\alpha) \cdot (x_1 \sin\alpha + x_2 \cos\alpha) \geq 0 \\ B: & (x_1 \cos\alpha - x_2 \sin\alpha) \cdot (x_1 \sin\alpha + x_2 \cos\alpha) < 0 \end{cases} \tag{13}$$

   где α — некоторый угол. При $\alpha = 0$ классу A принадлежат объекты, расположенные в первом и третьем квадрантах.