Решения отборочного тура Открытой Олимпиады СПбГУ среди студентов и молодых специалистов «Petropolitan Science (Re)Search» в 2023/24 учебном году по предмету «Вычислительные технологии» для обучающихся и выпускников бакалавриата

Задание 1.

Постановка задачи

Задача состоит в том, чтобы решить головоломку Судоку в общем случае (размер поля произволен), используя язык программирования C++ без сторонних библиотек.

Описание алгоритма

Судоку считается решенным, если в каждой из n строк, в каждом из n столбцов и в каждом из n квадрантов (поле делится на n квадрантов размерами $\sqrt{n}$). Если из n не является квадратом целого числа, то такое поле не является корректным (выводим Invalid field). Иначе, согласно Условию, решение головоломки всегда найдется.

Существует множество алгоритмов решения этой головоломки, однако все они так или иначе сводятся к методу перебора. Чтобы хоть как-то оптимизировать алгоритм, требуется хотя бы сузить множество перебираемых решений. Одним из способов подобной оптимизации является алгоритм перебора с возвратом (backtracking).

Первым шагом данного алгоритма является поиск еще не заполненной ячейки. Как только такая ячейка нашлась, генерируем список корректных чисел для этой ячейки (корректными являются числа, которые еще не встречались в строке, столбце и квадранте, в которые входит текущая ячейка).

После этого вставляем число из списка в ячейку и сразу же вызываем рекурсивно функцию — она снова найдет пустую ячейку, вставит туда число и т. д.

Выход из рекурсии предусмотрен в случае, когда в поле не осталось пустых ячеек — решение найдено.

Ссылка на программный код решения

https://github.com/DVPDVPDVP/Petropolitan-Science-Re-Search/tree/main/1

Инструкция по запуску программы

1. Зайдите в директорию с кодом: cd Petropolitan-Science-Re-Search/1
2. Установите библиотеку mpi.h для работы с параллельными вычислениями (это дополнительно, нужна только для параллельной версии программы, обычная версия не использует сторонние библиотеки, что соответствует условию задания).
3. Соберите программу: make

4. Запустите обычную программу ./simple или параллельную ./parallel. Обычная версия программы считывает данные из стандартного потока ввода, в то время как параллельная из файла test1.txt.

Возможности распараллеливания алгоритма

Задачу поиска верного решения судоку можно представить в виде дерева, узлы которого являются решениями. В листьях будут храниться либо полностью заполненные поля (верные решения), либо неудачные решения, в процессе которых возникли ошибки. Каждый нелистовой узел в дереве соответствует частичному решению. В корне дерева находится исходное поле.

Алгоритм backtracking подразумевает разбиение задачи на несколько независимых подзадач — проверку решения для одного из корректных значений ячейки. В худшем случае для проверки допустимости решения придется тестировать каждую перестановку поля.

Задача, которая соответствует узлу дерева, зависит от вычислений, выполненных его родителем (списка значений ячеек поля). По завершении ее выполнения потоку следует сообщить другим потоком найденное решение или невозможность решения.

Вычисление на каждом из путей от корня до листа (зависимые подзадачи) выполняются на одном кластерном узле, чтобы упростить механизм коммуникации между потоками. Однако, если несколько потоков на одном узле простаивают, можно перенести задачи с занятого узла на этот узел.

Каждая задача, соответствующая существующему узлу дерева, у которого еще нет занятых потомков, может быть назначена простаивающему узлу в режиме реального времени.

Для поддержания набора задач, выделенных потоку, будут использоваться стеки, которые заменят рекурсию при обходе дерева в глубину.

Изначально одному потоку назначается корень дерева, который находится на вершине локального стека потока. Поток считается занятым или работающим, когда ему назначен хотя бы один узел задачи.

На каждом шаге занятый поток выполняет следующие действия:

1. Смотрит узел на вершине стека, если это не листовой узел. Если это листовой узел, который дает допустимое решение, задача решена, все потоки уведомляются и прекращают работу. Если это недопустимый листовой узел, он отбрасывается.

2. Передает работу простаивающему потоку по запросу. Простаивающий поток — это поток, которому не назначены узлы задач, и который случайным образом выбирает поток и запрашивает делиться работой. При передаче работы поток назначает половину

узлов верхнего уровня, которые у него есть (в его стеке), только простаивающему потоку, если поступил запрос о передаче.

3. Узлы верхнего уровня - это узлы задач с минимальным уровнем, выделенным потоку, которые необходимо завершить.

4. Только простаивающий поток может запрашивать работу.

5. Только занятый поток может передавать работу.

6. Получающий поток может получать работу только от одного потока.

7. Процесс парного соответствия, то есть, когда занятые потоки передают работу простаивающим потокам, является случайным в том смысле, что простаивающие потоки случайным образом запрашивают работу у других потоков.

Простаивающему потоку необходимо получить половину частичных решений, вычисленных на данный момент занятым потоком, с которым простаивающий поток будет делить работу. Занятый поток выбирается простаивающим потоком случайным образом. Таким образом, предположим, что в занятом потоке есть m верхнеуровневых узлов, и размер каждого частичного решения будет равен размеру поля. Тогда размер сообщения составит (m/2)*n.

Данный алгоритм предполагает, что все потоки могут взаимодействовать со всеми другими потоками с одинаковыми накладными расходами. Но общий пул потоков распределен по отдельным узлам, и узлы взаимодействуют с помощью MPI. Чтобы уменьшить накладные расходы на коммуникацию, можно следовать следующей структуре при назначении задач простаивающим потокам:

1. В каждом узле есть главный поток, который управляет всей коммуникацией с другими узлами.

2. Уровень любого потока - это минимум из уровней всех узлов задач. Уровень 0 начинается с корня.

3. Узел считается простаивающим, если все потоки этого узла кластера простаивают, то есть им не назначены задачи. Узел считается занятым, если хотя бы один поток в этом узле занят.

4. Если занятый узел получает запрос от простаивающего узла, главный поток занятого узла равномерно делит верхнеуровневые задачи потока с минимальным уровнем.

5. Внутри занятого узла каждый занятый поток делит свои верхнеуровневые задачи с простаивающим потоком, если таковой имеется.

6. Главный поток простаивающего узла случайным образом выбирает узел из кластера и запрашивает работу у главного потока этого узла. Если был запрошен занятый узел, его главный поток поделится работой, как в пункте 1. Если был запрошен другой

простаивающий узел, простаивающему узлу будет сообщено об отсутствии работы, и он снова случайным образом выберет другой узел.

Задание 2.

Постановка задачи

Задача состоит в том, чтобы найти угол поворота изображения документа, содержащего печатный текст и фото и скорректировать изображение путем поворота, используя найденный угол.

Описание алгоритма

Программа написана на языке C++ с использованием библиотеки opencv для работы с изображениями. Изображение представляет собой матрицу размерами, соответствующими размерам изображения. В ее ячейках хранится цветовая гамма BGR пикселей.

Заметим, что в тексте всегда присутствует фото — прямоугольное изображение, цветовая гамма которого отличается от фонового (в документах фон всегда белый) цвета.

Первым шагом предложенного мной алгоритма будет поиск фото. Для этого будем итерироваться окном размера 10x10 пикселей по матрице, и как только внутри окна будут отсутствовать белые пиксели — фото найдено.

Далее требуется найти границы фото. Так как мы можем смотреть цвет каждого пикселя картинки и итерироваться по ним, то возможно найти такую окрестность найденной на первом шаге точки, которая соответствует фото, содержащемуся в документе. Воспользуемся алгоритмом поиска в ширину (BFS), начиная поиск от этой точки и добавляя в множество только пиксели, цвет которых отличается от фонового. В результате поиска мы нашли множество пикселей, из которых состоит фото.

Полученное на втором шаге множество является прямоугольником, повернутым на искомый угол. Путем итераций по множеству найдем его 4 угла. Для этого найдем в множестве точки, минимальные и максимальные по двум координатам x и y. Соединив точки с минимальной координатами x и y получим сторону прямоугольника. Сравним эту сторону с соседней: если в изначальном фото у прямоугольника ширина была больше высоты, то становится возможным найти направление поворота. Назовем сторону, вершины которой являются точками с минимальными координатами x и y AB, а соседнюю с ней — BC.

Пусть точки $A(x_a, y_a)$ и $B(x_b, y_b)$. Если AB > BC, то прямая AB наклонена под углом $\alpha = \arctan(\frac{y_b - y_a}{x_b - x_a})$

Если же AB < BC, то $\alpha = \frac{\pi}{2} + \arctan(\frac{y_b - y_a}{x_b - x_a})$ (изображение повернуто в другую сторону).

Далее с помощью библиотеки opencv строим матрицу поворота, и поворачиваем изображение на $-\alpha$, сохраняя результат.

Обоснование предложенного решения с точки зрения работы алгоритмов, занимаемой памяти и используемых ресурсов

В Условии сказано, что в тестовых документах всегда присутствует именно фото, а не векторный рисунок. Из этого можно сделать вывод, что цвет большинства пикселей картинки отличается от фонового. Фактически, все что мы знаем о входном изображении - это его размеры и цветовая гамма пикселей. Фото всегда представляет собой ровный прямоугольник, наклоненный на искомый угол. Именно поэтому угол удобнее всего искать, вычислив координаты его сторон.

Следует заметить, что цвет букв также всегда отличается от фона и он всегда черный. Поэтому можно было искать пиксели, цвет которых не белый и не черный. Однако в таком случае мы теряем в точности: BFS не добавил бы в множество потенциально возможные черные пиксели фото. Поэтому я предпочел итерироваться окном фиксированного размера, чтобы отличить буквы от фото: в буквах очень малая окрестность пикселей не фонового цвета. При таком подходе программа потенциально может дать неправильный ответ при наличии слишком жирной буквы, однако зная стандарт, которому соответствует документ, можно подобрать такой размер окна, чтобы исключить возможность распознания буквы как фото.

Не следует забывать, что пиксели фонового цвета могут содержаться в фото. Однако, алгоритм BFS их просто проигнорирует и добавит все остальные пиксели в окрестности белой точки, все равно образуя прямоугольник. Неточность возникает только в том случае, если углы фото является пикселями белого цвета, и, если объект является именно фотографией, а не рисунком на белом фоне, это не столь критично.

Также программа выдает неправильный ответ, если в изначальном документе ширина фото была меньше высоты: программа повернет его боком. Это невозможно решить, поскольку возникает неопределенность в направлении поворота картинки.

Что касается производительности алгоритма, то здесь мажорирует временная сложность алгоритма BFS + добавление в set ($O(nmlog(mn))$, где n – ширина фото, m – высота фото). Такая сложность возникает потому, что приходится обойти все фото, и добавлять пиксели в set за логарифмическую сложность. Потенциально можно использовать unordered_set, избавившись от логарифма, но зато тратя больше памяти. Сам BFS, хоть и находится в реализации внутри цикла поиска внутренней точки фото, но запускается лишь тогда, когда эта точка уже найдена, и программа завершается сразу после поиска.

Кроме того, функции opencv, такие как getRotationMatrix2D и warpAffine могут потребовать даже больше времени, чем поиск в ширину, находясь с ним на одном уровне вложенности. Однако это стандартные функции и их использование вполне оправдано.

По объему занимаемой памяти программа также не слишком требовательна. Приходится хранить в оперативной памяти исходное изображение, результат, множество пикселей фото (в виде двух координат) и матрицу поворота. При обычных размерах изображения программа не будет использовать критически много памяти.

Ссылка на программный код решения

https://github.com/DVPDVPDVP/Petropolitan-Science-Re-Search/tree/main/2

Инструкция по запуску программы

1. Установите библиотеку opencv для C++
2. Зайдите в директорию с кодом: cd Petropolitan-Science-Re-Search/2
3. Соберите программу: cmake .
4. Запустите решение: ./solve
5. В консоль введите путь до исходного изображения и результата (в какой файл он должен быть записан)

Экспериментальная оценка точности работы алгоритма

Требуется провести оценку точности работы алгоритма. Будет проведено две серии тестов. В первой тестовой выборке будет 5 изображений с фото в разных местах (4 угла и центр), повернутых на 50 градусов. Во второй выборке будет 5 тех же изображений, повернутых на -50 градусов. Оценивать будем среднюю абсолютную ошибку (MAE):

$$MAE = \frac{1}{n} \cdot \sum_{i=1}^{n} \quad [|y_i - \hat{y}_i|],$$

Где y_i- точное значение, $\hat{y}_i$- результат работы программы.

Для первого теста:

$$MAE_1 = \frac{1}{5}(|48.9863 + 49.9863 + 48.276 + 49.9863 + 42.1659 - 50 * 5|) = 1.91984$$

Для второго теста:

$$MAE_2 = \frac{1}{5}(|-50.1194 - 50.1194 - 50.1455 - 50.1194 - 49.7845 + 50 * 5|) = 0.05764$$

В среднем за два теста программа показала MAE равное 0.63. Это значит, что на случайных подходящих изображениях она будет показывать относительно точные результаты.

Задание 3.

Постановка задачи

Задача состоит в том, чтобы определить алгоритм шифрования указанной в Условии выборки данных и предоставить реализацию дешифратора на языке Python. Согласно Условию, что для защиты персональных данных хватит и обычных шифров.

Описание алгоритма

Рассмотрим первый столбец выборки (адреса электронной почты). Адреса состоят из маленьких латинских букв, цифр, точки и «собаки». Сравнив зашифрованные данные с реальными образцами электронной почты, можно заметить, что шифры соответствуют паттерну ^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$. Из этого следует, что шифровке подверглись только маленькие латинские буквы в адресах.

Исходя из Условия, сразу отбросим сложные шифры по типу AES или RCA. Зная, что первый по популярности DNS-суффикс .com, сравним посимвольно суффикс длины 3 шифра с com. Предположив, что использовался шифр Цезаря, найдем сдвиг по каждой букве суффикса и сравним их между собой. Если сдвиги совпали, то сообщение может быть зашифровано шифром Цезаря.

Вычислив сдвиг и расшифровав только латинские буквы из первого паттерна vaeqbt52@symux.oay, получили строку joseph52@gmail.com, что совпадает с реальным адресом электронной почты. Из этого можно сделать вывод, что при шифровке использовался шифр Цезаря.

Для успешной расшифровки необходимо вычислить длину сдвига путем вычитания из номера закодированного символа в кодировке номера буквы “c”. Чтобы сдвиг не превышал размер алфавита, возьмем остаток от деления сдвига на размер алфавита (в данном случае размер алфавита равен 26 — маленькие латинские буквы). Если сдвиг получился отрицательный, то прибавляем к нему размер алфавита.

После этого создадим строку с расшифрованным адресом электронной почты. Вычислив новый номер буквы по формуле: <старый номер буквы> - <сдвиг>, сравним его с номером буквы “a” в кодировке Python. Если полученное значение больше буквы ”a”, то в ответ пишем букву, номер которой соответствует сумме номера буквы ”a” и нового номера буквы, иначе — вычтем из номера буквы ”z” разность между сдвигом и номером новой буквы в кодировке. Результат добавим в конец новой строки.

При тестировании решения также выяснилось, что один из паттернов имеет исходный доменный суффикс .org. Поэтому для него в коде реализована отдельная логика.

Второй столбец исходного файла — домашние адреса, состоящие из больших и маленьких русских букв и знаков препинания. Последнее слово в каждой строке — 2 буквы, точка и цифра. Предположим, что это номер квартиры, состоящий из сокращения ”кв.x”.

Так как в шифре используются большие и маленькие русские буквы, то размер алфавита будет 64 (32 большие русские буквы и 33 маленькие, за исключением буквы ”ё”). В кодировке Python после идущих подряд 32 больших русских букв находятся 32 подряд идущие маленькие русские буквы. Аналогично расшифровав первую строку, можно прийти к выводу, что при кодировке также использовался шифр Цезаря.

При расшифровке домашних адресов важно учитывать: сокращение ”кв” может быть написано большими и маленькими буквами. Поэтому следует вычислить 2 сдвига — по маленьким буквам и по большим и выбрать среди них правильный.

Ссылка на решение

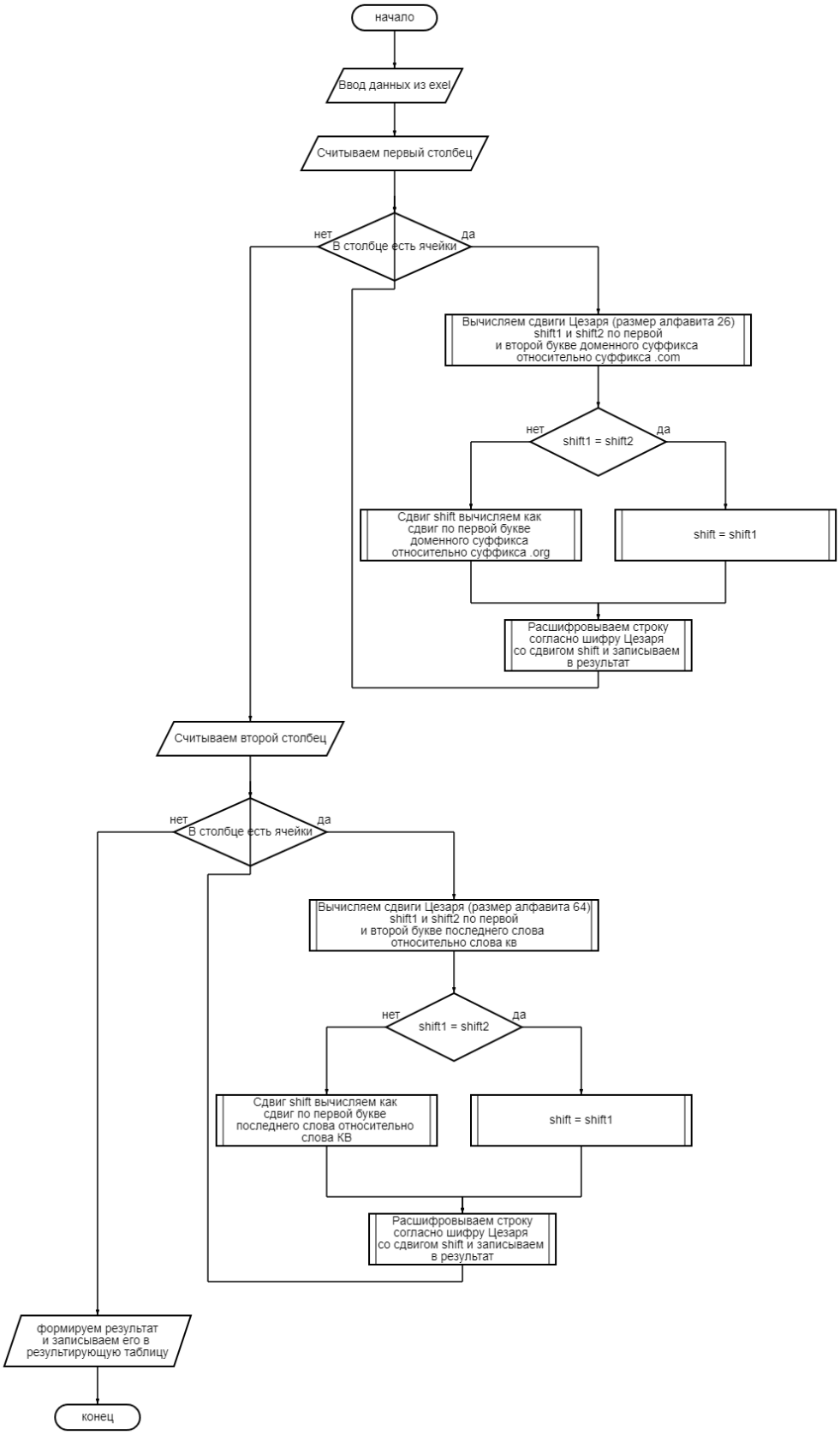
https://github.com/DVPDVPDVP/Petropolitan-Science-Re-Search/tree/main/3

Инструкция по запуску программы

1. Установите модуль pandas: pip install pandas
2. Установите модуль openpyxl: pip install openpyxl
3. Зайдите в директорию с кодом: cd ./Petropolitan-Science-Re-Search/3
4. Запустите программу: python solve.py

Программа работает с файлом sample.xslx, в котором содержится выборка, предоставленная в Условии. Результат работы записывается в файл result.xlsx.

Блок-схема алгоритма



Ссылка на блок-схему:

https://github.com/DVPDVPDVP/Petropolitan-Science-Re-Search/blob/main/3/block-scheme.png

Результат работы программы

Деобезличенный датасет с добавлением столбцов с ключом шифрования:

Почта	Адрес	Ключ
joseph52@gmail.com	РаХмановский пер.д.37 кв.399	12
kiehn.sincere@gmail.com	пр. № 4914д.14 кв.284	15
electa.hill@torphy.com	2-Я пОкРОвСкаЯ УЛ.д.73 кв.70	21
rhayes@glover.com	87-й км МКАДд.16 кв.158	7
ford.lebsack@hand.com	1-й сПаСоналивковСкий ПеР.д.35 кв.167	17
mikayla56@hotmail.com	ул. ЗаболотЬед.32 кв.476	5
jazmyne.boyle@goodwin.org	сХодненСкий ТУп.д.37 кв.368	15
mkling@wisozk.com	ЧоботовскаЯ ул.д.57 кв.41	5
xcrooks@hotmail.com	пОлеваЯ Ул.д.68 кв.257	18
eriberto.lind@gmail.com	АлЫмова ул.д.76 кв.49	8

Заметим, что ключи шифрования почты и адреса совпадают, а сокращение «кв» всегда было с маленькой буквы, поэтому конкретно для этой выборки можно было только 1 раз вычислить сдвиг (ключ) относительно «кв» и однозначно расшифровать всю строку, используя этот сдвиг.