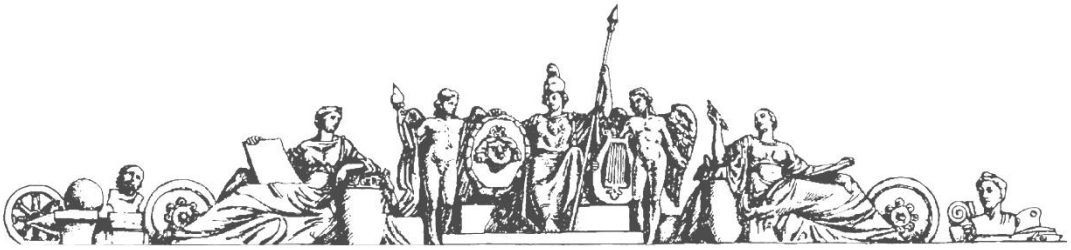

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н. Э. Баумана» (МГТУ им. Н. Э. Баумана)

Факультет Информатики и систем управления (ИУ)
Кафедра «Информационная безопасность» (ИУ8)

ОТЧЕТ ПО ДОМАШНЕЙ РАБОТЕ
по курсу
«Алгоритмы и структуры данных»

Выполнил:
студент группы ИУ8-51
Лашманов Р.Д.

Преподаватели:
Ключарёв П. Г.
Чесноков В. О.

Москва 2016

Содержание

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ ДОМАШНЕГО ЗАДАНИЯ ПО КУРСУ: «АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»

Описание домашнего задания:

Вариант 34.

Вы устроились на работу специалистом по ИБ. Руководство поставило задачу обеспечения ИБ в предприятии. Известно, что при повышении безопасности страдает удобство пользователей. У вас есть набор средств и мер ИБ, каждая из которых защищает от одной из нескольких угроз. При этом каждое средство снижает удобство пользователя. Постройте алгоритм выбора средств, которые с одной стороны защищают от всех известных угроз, а с другой — наименее неудобны для пользователя.

Описание известной задачи

Поставленная задача сводится, на мой взгляд, сводится к нескольким известным задачам: задача о ранце, о вершинном покрытии, о покрытии множества. Но больше всего, данная задача похожа именно на задачу о покрытии множества.

Формализация текстовой задачи.

Представим, что требуемая безопасность представляет собой множество S размерности N (N – количество существующих угроз), состоящее из единиц (единица в данном случае показывает, что угроза устранена). Тогда наши меры – подмножества P ($P_1, P_2 \ldots P_M$) множества S, включающие в себя 0 и 1. Каждое из этих подмножеств имеет свою стоимость c_i(в нашем случае, роль стоимости играет удобство пользователя). Тогда задача выбора средств состоит в нахождении семейства O ($O_1, O_2 \ldots O_M$) подмножеств P, такого, что $\bigcup_1^M O_i$ = S. Таких семейств, представляющих собой комбинации мер защиты, может быть много. Нам нужно семейство, имеющее самую низкую стоимость, которая складывается из отдельных стоимостей подмножеств.

Сравним с формулировкой задачи о покрытии множества:

Исходными данными задачи о покрытии множества является конечное множество U и семейство S его подмножеств. Покрытием называют семейство S' наименьшей мощности, объединением которых является U.

Пример

В качестве примера задачи о покрытии множества можно привести следующую проблему: представим себе, что для выполнения какого-то задания необходим некий набор навыков S. Также есть группа людей, владеющих некоторыми из этих навыков.

Необходимо сформировать минимальную группу для выполнения задания, включающую в себя носителей всех необходимых навыков.

Описание основных подходов к решению задачи о покрытии множества.

Задача о покрытии множества является NP-полной задачей. Четких алгоритмов для решения задач такого типа не существует, поэтому применяются приближенные, эвристические методы и алгоритмы.

1. Жадный алгоритм.

Жадный алгоритм выбирает множества, руководствуясь следующим правилом: на каждом этапе выбирается множество, покрывающее максимальное число ещё не покрытых элементов.

Смысл работы таков: пусть у нас есть множество U – заданное множество всех элементов, F – семейство подмножеств. Создаем множество X = U и C равное пустому множеству.

Пока X != пустому множеству, выбираем одно из подмножеств S, принадлежащих F, с наибольшим объединением X и S. Из множества X удаляем S. К множеству C добавляем выбранный S. В конце алгоритма возвращаем C.

Преимущества данного алгоритма – хорошее время работы.

Однако, данный алгоритм не всегда может выдать оптимальное решение. Подбирая на каждом шагу подмножество с наибольшим объединением, некоторые элементы U могут быть покрыты множество раз, что является избыточным.

2. Генетический алгоритм.

Неформально генетический алгоритм (ГА) может быть представлен
следующим образом:

Шаг 1: Выбор начальной популяции.

Шаг 2: Оценка приспособленности каждого индивида в популяции.

Шаг 3: Выбор родительских индивидов по степени приспособленности.
Применение генетических операторов (кроссовер или мутации)
над родительскими индивидами для получения потомков.

Шаг 4: Оценивание степени приспособленности полученных индивидов-потомков. Заменить некоторых (или всех) индивидов в популяции индивидами-потомки.

Шаг 5: Если достигнуто желаемое решение, то остановиться. Иначе перейти к шагу 3.

Данный алгоритм довольно сложен в исполнении, но, если верить ученому по имени Нгуен Минь Ханг, показывает прекрасные результаты.

3. Частные случаи.

В случае, если каждое из подмножеств имеет ровно по две единицы, задачу можно свести к задаче о вершинном покрытии.

На первом шаге мы берем случайное подмножество и удаляем из списка те, которые пересекаются с выбранным. Таким образом, в конце мы получаем нужный список подмножеств (семейство). Точность такого алгоритма – примерно равна 2.

Однако, использование такого алгоритма на менее удобных входных данных приводит к тому, что зачастую, после окончания работы алгоритма, мы получаем семейство, которое не полностью покрывает заданное множество.

Описание выбранного алгоритма.

Для решения данной задачи существует множество алгоритмов. Я выбрал для решения жадный алгоритм с определенной точностью - размер покрытия, построенного жадным алгоритмом, превосходит минимальное покрытие не более чем в 1 + ln m раз.

Используемые структуры данных

В программе будет использоваться только одна структура данных – массив. Массивом будет «множество безопасности» - массив, состоящий только из единиц, «меры» - массив из 0 и 1, и временные массивы, которые будут использованы во время работы алгоритма.

ПРАКТИЧЕСКАЯ ЧАСТЬ ДОМАШНЕГО ЗАДАНИЯ ПО КУРСУ: «АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»

Инструкция по использования программы

Для запуска программы нужно подать 2 консольных аргумента – входной и выходной файл. Выходной файл создаться сам, если такого файла не существует в директории.

Строки входного файла состоят из целых чисел. Этот файл должен быть заполнен следующим образом:

Строка первая – число n, отображающее количество угроз, которое нужно устранить . В конце строки – Enter.

Строка вторая – предпоследняя – числа от 1 до n, записанные через пробел. Каждая строка – мера, каждое число в строке показывает, закрывает ли мера угрозу, номер которой совпадает с числом. В конце каждой строки – Enter.

Строка последняя – числа от 0 до 9, записанные через пробел, отражающие неудобство пользователя при использовании той или иной меры. Количество таких чисел – количество мер (вторая – предпоследняя строка). В конце строки – Enter.

На выходе из программы файл в формате txt. Файл отражает ответ задачи: меры, использованные для защиты от угроз и неудобство пользователя.

Логика работы программы

После заполнения исходных данных (функции Zapolnenie, Meri,Comfortable), начинает свою работу основной алгоритм программы, основанный на жадном алгоритме.

Основной цикл отвечает за то, чтобы работа программы не продолжалась, после выполнения задачи – покрытия всех угроз (массив Odin_array). Дальше мы выбираем меры. Выбор происходит по следующим критериям:

1. Сколько угроз покрывается мерой
2. Удобство меры

Если количество покрываемых угроз совпадает, программа делает выбор в пользу той меры, которая приносит меньше неудобств.

Далее, если лучше мер программа не смогла найти, мера добавляется к ответу. Вместе с мерой добавляется и неудобство, соответствующее этой мере.

Когда основной цикл завершает свою работу – массив Odin_array полностью обнуляется – алгоритм завершает свою работу. На выход подается 2 массива – ответа - Answer_comfort_array, Answer_meri_array. После записи в файл (Zapis_v_fail) программа завершает свою работу.

Тестирование основных методов

Тестами покрыта вся программа, за исключением функций Odinochnii и Pustoi – логика их работы слишком очевидна.

Описание тестов:

Тест 1. Correct_test. Тест проверяет, запустится ли программа, если входной файл будет заведомо неверно заполнен. Успешное прохождение теста – не запуск в случае «плохого» файла и работа в случае «хорошего». Переменная oshibka показывает тесту, есть ли ошибка.

Тест 2. Input_test. Тест проверяет, правильно ли программа считывает входные данные. Эталон входных данных вводится в коде тестов заранее.

Тест 3. Algoritm_test. Тест проверяет, правильно ли сработал алгоритм. Правильно в данной интерпретации – на сколько решение, предоставленное алгоритмом отличается от эталонного решения. Эталон вводится заранее.

Тест 4. Output_test. Тест проверяет, правильно ли совершена запись в файл. Передается 3 аргумента – то, что требуется записать (обычно, те же аргументы, что и в тесте алгоритма) и выходной файл. Выходной файл задается в программе раньше.

Сложность основного алгоритма

Основной цикл, отвечающий за остановку алгоритма, имеет в худшем случае O(n) итераций (где n – количество угроз) – в случае, если в каждой мере по 1 закрытой угрозе. Придется перебрать все, чтобы закончить работу.

Вложенный в него цикл (while (counter_1 < len(Meri_ok))) работает m раз, где m – количество мер (в том же, худшем, случае; иначе, меньше m, ведь некоторые меры могут быть не обработаны, по причине решения задачи).

Следующий вложенный цикл (counter_2 < len(mera)) работает меньше чем k раз, где k – размер покрытия множества. Будет работать ровно k раз в том случае, если придется использовать все меры защиты, для устранения всех угроз.

Цикл, вложенный в основной цикл - counter_2 < len(mera) отвечает за обнуление основного массива (Odin_array). Он будет работать n раз, где n – количество угроз. Ведь если массив полностью обнулится, алгоритм остановит свою работу.

Итак, подводя итоги, вычислим общую сложность алгоритма:

O(2n + m + k) = O(n + m + k), где m - количество мер, n - количество угроз, k - размер покрытия множества.