Теоретическое решение задачи B – Учения

Алгоритм решения и доказательство его правильности

Представим условие задачи, как сеть с n вершинами, m ребрами, истоком s и стоком t. Где вершины – это острова, а ребра – мосты между островами. Для описания ребра графа, создадим структуру rib с такими атрибутами: to(в какую вершину ведет ребро), max_flow(максимальный поток, который можно пустить через данное ребро), c(стоимость единицы потока по данному ребру), f(текущий поток по данному ребру), number(порядковый номер ребра). Для описания графа, создадим массив g размером n, в каждой g_i будем хранить список ребер, которые выходят из i-той вершины.

Для каждого ребра (i, j) добавим в сеть обратное ребро (j, i), с максимальной пропускной способностью 0, потоком $f_{ji} = -f_{ij}$ и стоимостью $c_{ji} = -c_{ij}$. Это позволит нам использовать уже использованные ребра в остаточной сети при поиске нового пути, так как, если по некоторому ребру (i, j) уже течет поток f_{ij} = max $\_flow_{ij}$, в остаточной сети мы сможем рассматривать обратное ребро (j, i), так как поток $f_{ji} = -f_{ij}$, а $-f_{ij} < 0$, то мы имеем возможность дополнить данное ребро на f_{ij}, а по определению остаточной сети, в ней находятся только ребра, которые имеют остаточную пропускную способность (по которым можно дополнить поток).

Так как по условию задачи по мостам можно двигаться в любом направлении, то каждое ребро мы будем записывать в сеть, как два ребра: (i, j) и (j, i), а так как мы также создаем для каждого такого ребра обратное, то каждое ребро из условия будет формировать 4 ребра в исходной сети.

По условию задачи, солдаты начинают с первого моста и должны дойти до последнего, то примем исток s = 1 и сток t = n. Также, так как по условию задачу по одному мосту может проходить только 1 пехотинец, то установим max_flow каждого ребра равным единице.

Далее схему алгоритма можно описать так: пока существует путь из s в t в остаточной сети максимально дополняем поток по самому дешевому пути,

когда такие пути заканчиваются, то восстанавливаем маршруты каждого пехотинца из s в t.

Для реализации алгоритма поиска кратчайшего пути в остаточной сети, нам требуется двусторонняя очередь, куда мы будем записывать вершины для рассмотрения. Так как известно, что очередь не будет больше n, то создадим просто массив q размера n, и целочисленные указатели: qh – указатель на номер первого элемента и qt – указатель на номер последнего элемента очереди. Для добавления в конец очереди записываем значение в q[qt] и увеличиваем qt на 1, если qt вышел за пределы очереди (qt == n), то устанавливаем qt = 0 (зацикливаем очередь). Для добавления значения в начало очереди уменьшаем qh на 1, если qh == -1, то устанавливаем qh = n – 1, и записываем значение в q[qh].

Также введем понятие статуса вершины:

1. 0, если вершина еще не была обработана алгоритмом
2. 1, если вершина находится в очереди
3. 2, если вершина уже была обработана алгоритмом

Создадим массив id размера n, где будем хранить состояние каждой вершины. Также создадим массивы:

1. d, для хранения расстояния от истока к данной вершине (по стоимости)
2. p, где p_i будет хранить номер предыдущей вершины в пути
3. p_rib, где p_rib_i будет хранить номер ребра, которое ведет в вершину i

Добавим в очередь исток, и запустим алгоритм, пока не закончатся элементы в очереди (qt == qh). Сам алгоритм поиска пути: берем вершину из начала очереди, обрабатываем все ребра, которые выходят из нее у которых есть остаточная пропускная способность. Если использования данного ребра уменьшит расстояние до следующей вершины, то используем это ребро и добавляем следующую вершину в очередь.

После того, как найден путь из s в t в остаточной сети, находим максимальный поток, который можно дополнить по данному пути, после чего

дополняем поток, повторяем этот алгоритм пока не закончатся пути в остаточной сети, либо не будет найден поток нужной величины k.

После нахождения потока есть два варианта:

1. найденный поток flow < k
2. найденный поток flow = k

Если максимальный поток данной сети меньше нужного, то по условию задачи выводим -1, если же был найден поток нужной величины, то дальше требуется восстановить пути пехотинцев по мостам. Для этого начинаем с истока i = s, перебираем все ребра, которые выходят из вершины i, если находим ребро с потоком f > 0, то обнуляем его и переходим на следующую вершину. Делаем это до тех пор, пока не дойдем до стока t.

Правильность данного алгоритма доказывается по теореме Форда-Фалкерсона поток f максимален, когда в остаточной сети G_f не существует пути из s в t, так как: предположим, что поток f максимален, но в остаточной сети G_f существует путь p из s в t. Тогда рассмотрим поток f + f_p, по лемме о сумме потоков, поток f + f_p также существует в сети G, причем f + f_p > f, так как f – максимальный поток, что вызывает противоречие. Так мы находим максимальный поток, так как при поиске максимального потока мы ищем кратчайшие пути по стоимости, то данный поток будет и минимальной стоимости. Так как по условию задачи только 1 пехотинец может передвигаться по мосту и нужно переправить k пехотинцев, то если у нас получилось найти поток равный k, то сеть имеет k разных потока, каждый размера 1, поэтому мы всегда можем найти k потоков для каждого солдата и вывести номера ребер, которые формируют этот путь.

Временная сложность

Для поиска пути в остаточной сети, мы перебираем каждую вершину и каждое ребро сети, что дает сложность O(n*m). Таких путей нам требуется найти k. После нахождения каждого пути, требуется два еще два раза перебрать этот путь, первый раз, для того, чтобы вычислить, на сколько можно дополнить поток вдоль него, второй раз для дополнения потока для каждого

из ребер. Пути перебираются по вершинам, поэтому максимальная их длина – n. Также нам требуется восстановить k путей, максимальная длина которых – m. Конечная сложность O(n*m*k).

Затраты памяти

Мы храним m ребер и создаем массив g размера n в ходе работы программы. Также в ходе алгоритма поиска кратчайших путей в остаточном графе, мы создаем пять вспомогательных массива размера n. Итоговые затраты памяти – O(n + m).