Про реализацию у нас есть несколько советов, которыми стоит воспользоваться.

Решение этой задачи состоит из двух частей: бинарный поиск по ответу и проверка того, подходит ли текущее значение с помощью алгоритма поиска максимального потока.
Код же должен состоять их трех частей:

1. Простая библиотека для поиска максимального потока в сети.
2. Общая функция для бинарного поиска
3. Функция решения, которая пользуется двумя предыдущими частями.

Библиотека для поиска максимального потока в сети

На алгоритм поиска потока мы в этой задаче накладываем ограничение, что алгоритм должен быть strongly polynomial.
Ниже приведены советы для алгоритма Edmonds–Karp, но они применимы и для большинства алгоритмов, которые вы можете выбрать для поиска максимального потока.

Алгоритм Edmonds–Karp состоит из двух частей: поиск дополняющего пути и проталкивание потока вдоль этого пути.

Первая часть может быть сформулирована в терминах поиска (кратчайшего) пути в графе, который составлен из ребер с ненулевой residual capacity. Поэтому алгоритм поиска пути (bfs) не должен знать ничего о потоках.
Хорошая новость! Вы уже писали Breadth-first search для задачи “минимальный размер компании”. А значит, что и его, и структуру для графа можно смело копировать. Правда напрямую воспользоваться графом из предыдущей задачи не получится, так как нам нужно игнорировать ребра с нулевой residual capacity.

Для того, чтобы хранить flow network, стоит сделать одноименный класс. Этот класс и должен хранить все ребра в этой сети, как объект типа граф, и текущие значения flow и capacity для ребер. Теперь пора ответить на вопрос, как запустить bfs на графе, в котором есть только ребра с ненулевой residual capacity. Для этого удобно воспользоваться design pattern delegate. То есть реализовать class FilteredGraph, который можно создать из двух объектов: графа и предиката на множестве ребер. FilteredGraph отвечает графу, в котором остались только те ребра, для которых предикат истинен. Публичный интерфейс этого класса должен совпадать с публичным интерфейсом графа. (Кстати, это хороший тест на то, что ваш интерфейс графа действительно прост). Для delegates типично, что почти все методы имеют реализацию

```
T Delegate::f() {
  return underlying.f();
```

}
Так и будет для всех методов, кроме того, который предоставляет доступ к списку смежных вершине ребер. Этот метод должен вернуть список ребер underlying графа, отфильтрованный в соответствии с предикатом. Для этой подзадачи удобно реализовать простой аналог класса FilterIterator, который можно создать из пары итераторов и предиката. В нашем случае предикат будет проверкой, что residual capacity положительна. Не забывайте, что предикаты в c++ принято передавать по значению (Effective STL Item 38. Design functor classes for pass-by-value).
Отметим, что создавать какой-либо базовый класс для Graph и FilteredGraph в этом случае избыточно, так как, например, BreadthFirstSearch должен работать с любым шаблонным графом, если для него определена функция получения смежных вершине ребер.
Edmonds–Karp (как и многие другие алгоритмы) выполняет поиски путей в residual network, поэтому удобно предоставить метод ResidualNetworkView(), который возвращает текущий residual network в виде графа (FilteredGraph), на котором мы и будем запускать bfs.

Вопрос о том, как получить кратчайший путь (то есть, augmenting path) из запуска bfs, является простым упражнением на написание соответствующего визитора. Обратите внимание, что по сравнению с предыдущей задачей вам могут понадобиться новые события визитора.

Как хранить flow и residual capacity?

Для того, чтобы хранить для каждого ребра какую-то информацию, удобно сопоставить каждому ребру графа id. Так как наша реализация графа не поддерживает удаление ребер, то очень просто назначить ребрам id от 0 до |E| - 1 (|E| -- это количество ребер в графе). После этого flow network может просто хранить необходимые величины в векторе vector<EdgeProperties>.

Flow network – это сложный для построения объект: требуется добавить ребра с их пропускными способностями, назначить исток и сток. Для таких объектов (как мы уже видели в задаче “минимальный размер компании”) удобно использовать builder design pattern. Напомним, что в c++ существует практика делать BuilderX friend’ом класса X. Таким образом, flow network может содержать только один non-const метод, который меняет величину потока для данного ребра (при этом обновляет и соответствующие величины для обратного ребра). Чем меньше non-const методов в классе, тем проще гарантировать непротиворечивость внутреннего состояния (идеальные классы в этом смысле те, которые не содержат non-const методов).

Общая функция для бинарного поиска

Несмотря на то, что о бинарном поиске знают все, написать удобную и гибкую функцию для него не так просто.
Для нашей задачи функция бинарного поиска должна принимать два числа (полуинтервал) и бинарный предикат. Вопрос о том, что эта функция возвращает, мы оставляем на ваше усмотрение.
Однако к этой функции **необходимо** написать комментарий, который описывает контракт, предоставляемый этой функций. Контракт здесь означает то, какие требования функция налагает на аргументы, и что является возвращаемым значением. Контракт должен быть простым, но покрывать **все** крайние случаи. После чтения контракта у читателя не должно быть вопросов о том, как пользоваться этой функцией.