Алгоритм A*

Функция “предсказания” $f(v)=g(v)+h(v)$

Используем эвристики:

- Манхэттенское расстояние
- Расстояние Чебышева
- Евклидово

Ограничение на эвристики

Допустимость h(v) -- для любой вершины эвристика не превосходит реальное кратчайшее расстояние

Монотонность h(v) -- для любых двух вершин на пути разница эвристик не превышает реального расстояния между ними

Монотонность -> допустимость

Монотонность -> На любом пути f(v) не убывает

Остовные деревья

Остовное дерево

Ациклический связный подграф, включающий все вершины графа.

Как найти?

Минимальное остовное дерево

Остовное дерево, обладающее минимальным суммарным весом рёбер

Лемма о безопасном ребре

Пусть G’ -- подграф миностова G

Ребро не из G’ -- безопасное, если при добавлении его в G’ он не прекратит быть подграфом миностова

Разрез -- разделение множества вершин на S и T, <S,T>

Ребро пересекает разрез <S,T>, если один его конец принадлежит S, а второй -- T

Лемма о безопасном ребре

Рассмотрим граф G -- взвешенный неориентированный. G’ -- подграф его миноста, <S,T> -- разрез G такой, что его не пересекает ни одно ребро G’. Тогда ребро минимального веса, пересекающее разрез -- безопасное.

Алгоритм Прима

Похож на Дейкстру

Пытаемся построить миностов, начиная с любой вершины. Когда есть несколько вершин -- получаем, по факту, разрез. Ищем минимальное ребро в этом разрезе, включая новую вершину.

Включив новую вершину -- релаксируем пути до невключённых, если нужно

Асимптотика алгоритма Прима

Структура данных для приоритетной очереди	Асимптотика времени работы
Наивная реализация	$O(V^2 + E)$
Двоичная куча	$O(E \log V)$
Фибоначчиева куча	$O(V \log V + E)$

Алгоритм Крускала

Отсортируем все рёбра в порядке возрастания и будем последовательно добавлять в остов.

Если ребро соединяет вершины из разных компонент связности текущего остова -- берём, иначе -- не берём

Система непересекающихся множеств

Каждое множество -- дерево, корень -- его “представитель”. В каждой вершине -- ссылка на родителя

Объединение -- подвесим корень одного к корню другого. Понять, в каком множестве вершина -- пройтись до родителя

Эвристики СНМ

Объединение по рангу.

Подвешиваем дерево с меньшей высотой к дереву с большей. Чтобы не считать высоты -- используем ранги. Ранг одной вершины -- 0. При объединении -- максимум из двух. Если одинаковые -- +1

Сжатие пути

Храним указатель не на родителя, а на корень. Для этого при получении родителя обновляем все указатели, чтобы указывали на корень

Асимптотика -- обратная функция Аккермана (сложно, считать константной)

Асимптотика алгоритма Крускала

Сортировка рёбер -- O(ElogE)

CHM -- O(E*alpha(V))

Общая асимптотика -- O(ElogE)

Алгоритм Борувки

1. Каждая вершина графа -- дерево
2. Для каждого дерева найдём минимальное инцидентное ему ребро. Добавим в миностов все рёбра
3. Повторяем шаг 2, пока не получим одно дерево

(В пункте 2 лучше брать ребро минимального номера при равных весах)

Алгоритм Борувки. Асимптотика

1. На каждом шаге количество деревьев сокращается как минимум вдвое. То есть шагов -- logV
2. Каждый раз надо просмотреть все рёбра, ведущие от дерева.

Итоговая асимптотика -- O(ElogV)

Максимальный поток в сети. RMQ & LCA

Определение сети и потока

Сеть -- орграф, c:E->R+, c(e) -- пропускная способность. В сети есть исток s и сток t

Поток в Сети -- f:(VxV)->R, где:

1)f(u,v) = -f(v,u)

2)f(u,v) <= c(u,v)

3) Сумма f(x,y) = 0 для всех вершин, кроме s и t

Величина потока -- сумма f(s,v) для всех v

Разрез. Поток через разрез

Разрез <S,T> -- знаем

Пропускная способность разреза -- сумма всех рёбер, пересекающих разрез

Минимальный разрез -- разрез с минимально возможной пропускной способностью

Поток в разрезе -- сумма по f(u,v), (u,v) пересекает разрез

Лемма о величине потока

f(S,T) = |f|

▷

$$f(S,T) = f(S,V) - f(S,S) = f(S,V) = f(S \setminus s, V) + f(s,V) = f(s,V) = |f|$$

- 1-е равенство выполняется, так как суммы не пересекаются: $f(S,V) = f(S,S) + f(S,T)$
- 2-е равенство выполняется из-за антисимметричности: $f(S,S) = -f(S,S) = 0$
- 3-е равенство выполняется, как и 1-е, из-за непересекающихся сумм
- 4-е равенство выполняется из-за сохранения потока

◁

Лемма о минимальном разрезе

Если f(S,T) = c(S,T), то поток f максимален, а разрез -- минимален

Ещё определения

Определение:

Остаточной пропускной способностью (англ. *residual capacity*) ребра (u, v) называется величина дополнительного потока, который мы можем направить из u в v, не превысив пропускную способность $c(u, v)$. Иными словами $c_f(u, v) = c(u, v) - f(u, v)$.

Определение:

Для заданной транспортной сети $G = (V, E)$ и потока f, **остаточной сетью**, (**дополняющая сеть**, англ. *residual network*) в G, порожденной потоком f, является сеть $G_f = (V, E_f)$, где $E_f = \{(u, v) \in V \times V \mid c_f(u, v) > 0\}$

Определение:

Для заданной транспортной сети $G = (V, E)$ и потока f **дополняющим путем** (англ. *augmenting path*) p является простой путь из истока в сток в остаточной сети $G_f = (V, E_f)$.

Теорема Форда-Фалкерсона

Если f -- поток в сети G = (V,E), то следующие утверждения эквивалентны:

1. Поток f максимален
2. В Gf не существует пути из s в t
3. |f| = c(S,T) для некоторого разреза сети

Алгоритм Форда-Фалкерсона

Положим f(u,v) = 0
Далее поток увеличивается итеративно через поиск увеличивающего пути
Поиск можно осуществлять с помощью dfs

Повторяем, пока можем найти увеличивающий путь

O(|E|f)

Алгоритм Эдмонса-Карпа

Улучшение алгоритма Форда-Фалкерсона, в качестве дополняющего пути берём кратчайший по рёбрам

Асимптотика O(VE^2)

Найти новый кратчайший по рёбрам путь можем только VE раз
Во-первых, длина пути -- от 1 до V
Сколько различных путей длины i? Ну где-то E

Алгоритм Диница

Слоистая сеть

Определим для каждой вершины v длину кратчайшего s->v.

В слоистую сеть включаем только те рёбра (u,v), что d[u] + 1 = d[v]

Слоистая сеть -- вспомогательная

Найдём блокирующий поток -- такой, что любой путь из s в t содержит насыщенное этим потоком ребро. Увеличить не получается.

Поиск блокирующего потока

1) Можно просто искать все пути s->t по одному и наполнять(по крайней мере одно ребро удалится
2) Оптимизация -- удалять заполненные рёбра и все лишние (из которых нельзя дойти до стока) O(VE)

Алгоритм Диница

1) Инициализируем f(u,v) = 0
2) Построим вспомогательную сеть для остаточной данного графа. Если пустая -- останавливаемся
3) Найдём блокирующий поток f’ в Gl
4) Дополним поток f найденным потоком f’ и повторим с 2

Поиск блок.потока в слоистой сети -- O(VE). Почему итераций -- не более |V|? После пропускания блок.потока по слоистой сети кратчайший дополняющий путь будет длиннее хотя бы на 1. max(length) = |V|

Асимптотика -- O(V^2E)

Паросочетания

Паросочетание -- набор попарно несмежных рёбер в двудольном графе. Максимальное паросочетание -- паросочетание с наибольшим количеством рёбер

Чередующаяся цепь -- рёбра поочерёдно принадлежат/не принадлежат паросочетанию

Увеличивающая цепь -- чередующаяся цель, у которой начальная и конечная вершины не принадлежат паросочетанию

Теорема Бержа

Паросочетание максимально тогда и только тогда, когда не существует увеличивающих относительно него цепей.

Пусть существует увеличивающая цепь. Тогда “сдвинем” рёбра на один (раз первое и последнее не принадлежат -> увеличим паросочетание -> оно не максимально

Теорема Бержа <=

1. Пусть парсоч M не содержит ув. пути, но есть парсоч M’, который больше чем M
2. Рассмотрим граф G -- симметрич. отрицание M и M’ -- все рёбра, которые лежат либо в M, либо в M’, но не одновременно. Рассмотрим как граф
3. У каждого ребра из G есть смежное (если нет, то просто добавим его в M или M’, получим противоречение)
4. Циклы -- чётные + чередуются из M и M’. Значит, в циклах равное количество
5. Пути -- чётные. Почему? Если неч, значит начинается с M’ и заканчивается M’, то есть для M это увеличивающий путь.
6. Если всё чётное -- их размер одинаковые и условие не выполнено

Алгоритм Куна поиска паросочетаний

Возьмём пустое паросочетание, пока в графе находим увеличивающую цепь -- выполняем “чередование” и повторяем процесс.

Ищем любую цепь, запуская обход в глубину или ширину вдоль каждой вершины из первой доли. Найдя -- “насыщаем”, то есть вычёркиваем из оставшихся, а обратную считаем созданной. Повторяем, пока не кончатся увеличивающие цепи

Алгоритм Куна

1. Возьмём пустое паросочетание
2. Пока удаётся найти увеличивающую цепь -- выполняем чередование

Массив matching[v] -- вторая вершина из парсоча или -1

Массив used -- посещена вершина или нет

Функция -- просматриваем все вершины, смежные с v. Если to ненасыщенная или удаётся найти ненасыщенную через рекурсивный запуск от to -- нашли увеличивающую цепь и чередуем.

Функция применяется последовательно ко всем вершинам, перед каждым запуском обнуляем used

O(VE)

Алгоритм Куна

```
bool dfs(v: int):
    if (used[v])
        return false
    used[v] = true
    for to in g[v]
        if (matching[to] == -1 or dfs(matching[to])):
            matching[to] = v
            return true
    return false
```

```
function main():
    fill(matching, -1)
    for i = 1..n
         fill(used, false)
         dfs(i)
    for i = 1..n
         if (matching[i] != -1)
              print(i, " ", matching[i])
```