Кратчайшие пути I.

Скрытая угроза

Простой путь

Найти кратчайший путь от start до finish в неориентированном графе.

Длина пути — количество рёбер.

Динамическое программирование?

$$D(\texttt{start}) = 0$$

$$D(u) = 1 + \min_{v \in \texttt{NB}(u)} D(v)$$

- Точно знаем ответ для вершины start
- Можем найти множество вершин на расстоянии 1

BFS

Breadth first search (поиск в ширину)

S[i] — множество вершин на расстоянии i.

$$S[0] = \{\texttt{start}\}$$

$$S[i+1] = \texttt{NB}(S[i]) - \cup_{j \le i} S[j]$$

```
def bfs(graph, start):
    work = {start}
    dist = {}

    current_dist = 0
    while work:
        for u in work:
            dist[u] = current_dist

        work = {nb for u in work
                   for nb in graph[u]
                   if v not in dist}
```

Детали релизации

- Вместо двух массивов можно использовать одну очередь.
- Какого максимального размера может быть очередь?
- Массив и два указателя — ring buffer.
- Не бывает stack overflow, в отличие от dfs => если надо обойти граф как-то на практике, используйте bfs.
- Как восстановить путь?
- Какая сложность?

Взвешенные рёбра

У каждого ребра есть неотрицательный вес (длина дороги). Найти кратчайший путь в смысле суммы весов рёбер.

$$D[\texttt{start}] = 0$$

$$D[u] = \min_{v \in \texttt{NB}(u)} D[v] + d[v, u]$$

Алгоритм Дейкстры (Dijkstra's algorithm)

- Точно знаем расстояние до вершины start.
- Не знаем расстояние до её соседей (путь из двух рёбер может быть короче, чем путь из одного ребра)
- Но! Знаем расстояние до одной из вершин.

Идея: если S это вершины, для которых мы точно знаем ответ, то мы точно знаем ответ и для ближайшей вершины из соседей S

```python
def dijkstra(graph, start):
    dist = {start: 0}
    border = {u: weight for (u, weight) in graph[start]}

    while border:
        u = min(border, key=lambda u: border[u])
        dist[u] = border.pop(u)
        for (v, weight) in graph[u]:
            if v in dist:
                assert dist[u] + weight >= dist[v]
                continue

            border[v] = min(
                dist[u] + weight,
                border.get(v, float("inf"))
            )
```

Трудоемкость

- На каждой итерации увеличиваем S на 1 => V итераций.
- На каждой итерации извлекаем минимум из границы => в худшем случае V на итерацию
- На каждой итерации рассматриваем какие-то рёбра, и на каждое ребро смотрим ровно один раз => E за все итерации.

Итого: O(V^2 + E)

Напоминание: в дереве E = V, в полном графе E = V^2

Посмотрим на границу

- Хранит вершины и веса
- Можно добавить вершину с весом/уменьшать весь (E раз)
- Можно доставать минимальную вершину из границы (V раз)

Как хранить границу

- Массив весов (V^2 + E) всегда
- Хэш мап весов (V*(размер границы) + E)
- Очередь с приоритетом (V + E) * log(V)

Напоминание про очередь с приоритетом

Хорошая структура данных — куча.

Стандартные кучи не умеют изменять значения ключа.

Можно написать свою кучу, можно использовать дерево поиска.

Когда не работает Дейкстра?

Отрицательные веса

Решим всё:

Найти расстояние между всеми парами вершин, веса могут быть отрицательными.

Отрицательные веса

Решим всё:

Найти расстояние между всеми парами вершин, веса могут быть отрицательными, но нет отрицательных циклов

$$D[u, v] = d[u, v]$$
$$D[u, v] = \min_{m \in \mathbf{V}} D[u, m] + D[m, v]$$

Как навести порядок?

- Перейдём в третье измерение!

D[u, v, m] — длина кратчайшего пути от u до v по вершинам с номерами меньше m.

D[u, v, 0] — путь, где все промежуточные вершины имеют номер меньше 0 =>

нет промежуточных вершин => D[u, v, 0] = d[u, v]

D[u, v, m] =

// нет вершины m - 1 вообще

D[u, v, m - 1]

// есть ровно одна вершина m - 1

D[u, m - 1, m -1] + D[m - 1, v, m - 1]

Как программировать?

Трёхмерная табличка, заполняем в порядке увеличения третьей координаты.

Нулевой уровень — матрица смежности.

Можно сэкономить на памяти, и помнить только один предыдущий уровень.

Сложно…

Floyd–Warshall algorithm O(n^3)

```
for k in range(n):
  for i in range(n):
    for j in range(n):
      d[i, j] = min(d[i, j],
                    d[i, k] + d[k, j])
```

А если есть отрицательный цикл?

Дополнительное время

Регулярное выражение

R =
  'a', 'b', 'c' — буквы
  R1 R2 — конкатенация двух регулярных выражений
  R1 | R2 — альтернация, либо R1, либо R2
  R* — замыкание Клини: конкатенация произвольного числа R.

  "" — пустое слово
  0 — пустой язык

Конечный автомат

Ориентированный мультиграф состояний.

На каждом ребре написана буква.

Есть стартовое и некоторое количество финальных состояний.

Посмотрим на язык

$$(a \mid b)^* aab$$

Слова из букв a, b, которые оканчиваются на aab.

Флойд наносит ответный удар

Задача: Построить по данному конечному автомату эквивалентное регулярное выражение.