Алгоритм Дейкстры

Denis Bakin

Задача

Дан: взвешенный граф без рёбер отрицательного веса, стартовая вершина s

Найти: кратчайшие расстояния от s до всех остальных вершин

Задача

Дан: взвешенный граф без рёбер отрицательного веса, стартовая вершина s

Найти: кратчайшие расстояния от s до всех остальных вершин

BFS решает эту задачу только для невзвешенных графов

Задача

Дан: взвешенный граф без рёбер отрицательного веса, стартовая вершина s

Найти: кратчайшие расстояния от s до всех остальных вершин

BFS решает эту задачу только для невзвешенных графов

Алгоритм Дейкстры — обобщение BFS на графы с неотрицательными весами

Основная идея

Заведём массив d, где d_v — текущая длина кратчайшего найденного пути из s в v

- Изначально $d_s = 0$, для остальных $d_v = \infty$

Основная идея

Заведём массив d, где d_v — текущая длина кратчайшего найденного пути из s в v

- Изначально $d_s = 0$, для остальных $d_v = \infty$

Массив a: $a_v = 1$, если кратчайший путь до v уже найден (вершина «помечена»)

Основная идея

Заведём массив d, где d_v — текущая длина кратчайшего найденного пути из s в v

- Изначально $d_s = 0$, для остальных $d_v = \infty$

Массив a: $a_v = 1$, если кратчайший путь до v уже найден (вершина «помечена»)

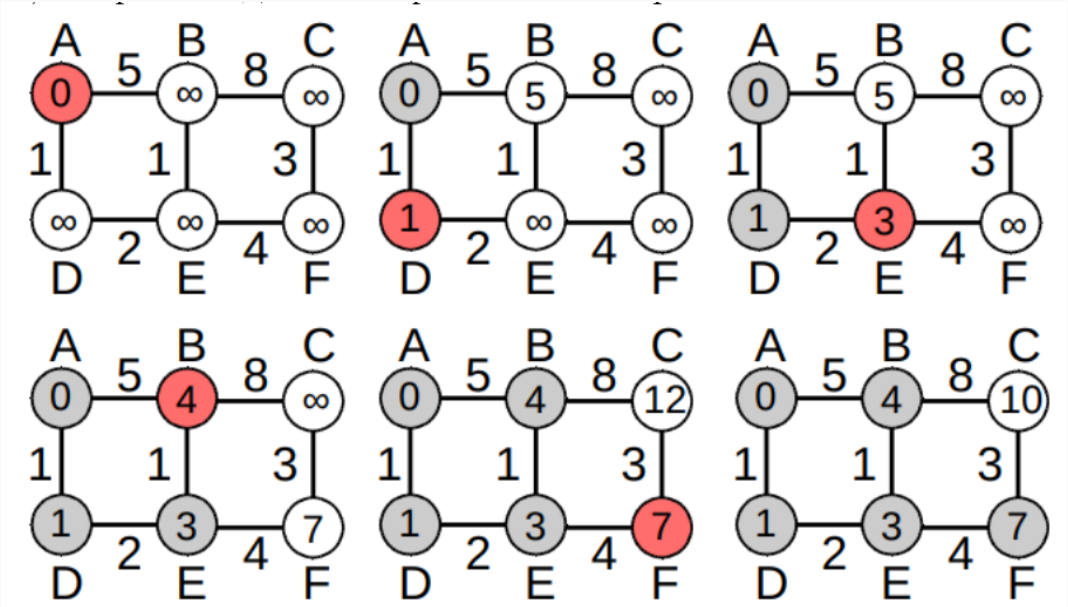
На каждой итерации:

1. Выбираем непомеченную вершину с минимальным d_v: $v = \arg\min_{u : a_u = 0} d_u$
2. Помечаем v
3. **Релаксируем** все рёбра из v: $d_u = \min(d_u,\ d_v + w_{vu})$

Аналогия с BFS

	BFS	Дейкстра
Граф	невзвешенный	неотрицательные веса
Структура	очередь (FIFO)	очередь с приоритетом
Релаксация	$d_u = d_v + 1$	$d_u = \min(d_u, d_v + w)$
Сложность	$O(n + m)$	$O(n^2)$ или $O(m \log n)$

Пошаговый пример



Корректность: утверждение

Утверждение. Когда вершина v помечается, d_v уже равно кратчайшему расстоянию l_v

Корректность: утверждение

Утверждение. Когда вершина v помечается, d_v уже равно кратчайшему расстоянию l_v

Наблюдение: $d_v \geq l_v$ всегда (алгоритм не может найти путь короче кратчайшего)

Корректность: утверждение

Утверждение. Когда вершина v помечается, d_v уже равно кратчайшему расстоянию l_v

Наблюдение: $d_v \geq l_v$ всегда (алгоритм не может найти путь короче кратчайшего)

Осталось показать, что в момент пометки $d_v \leq l_v$, т.е. $d_v = l_v$

Корректность: доказательство

Индукция по номеру итерации

Корректность: доказательство

Индукция по номеру итерации

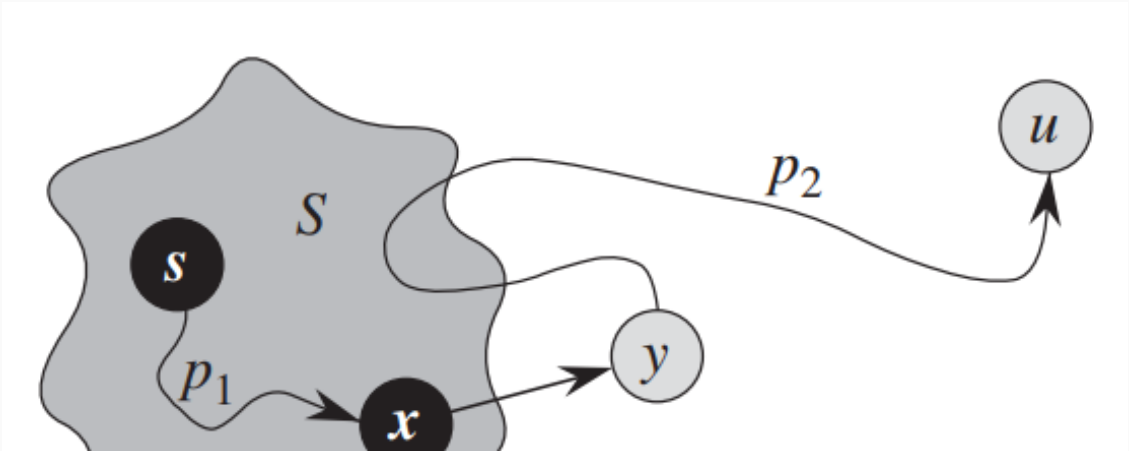
База: $d_s = 0 = l_s$ ✓

Корректность: доказательство

Индукция по номеру итерации

База: $d_s = 0 = l_s$ ✓

Шаг: рассмотрим кратчайший путь до выбранной вершины v



Корректность: доказательство (продолжение)

Обозначим x — последнюю помеченную вершину на кратчайшем пути до v, y — следующую за ней

- По предположению индукции: $d_x = l_x$

Корректность: доказательство (продолжение)

Обозначим x — последнюю помеченную вершину на кратчайшем пути до v, y — следующую за ней

- По предположению индукции: $d_x = l_x$
- Ребро (x, y) на кратчайшем пути $\Rightarrow l_y = l_x + w_{xy}$

Корректность: доказательство (продолжение)

Обозначим x — последнюю помеченную вершину на кратчайшем пути до v, y — следующую за ней

- По предположению индукции: $d_x = l_x$
- Ребро (x, y) на кратчайшем пути $\Rightarrow l_y = l_x + w_{xy}$
- При пометке x была релаксация $\Rightarrow d_y \leq d_x + w_{xy} = l_y$

Корректность: доказательство (продолжение)

Обозначим x — последнюю помеченную вершину на кратчайшем пути до v, y — следующую за ней

- По предположению индукции: $d_x = l_x$
- Ребро (x, y) на кратчайшем пути $\Rightarrow l_y = l_x + w_{xy}$
- При пометке x была релаксация $\Rightarrow d_y \leq d_x + w_{xy} = l_y$
- Но $d_y \geq l_y$ всегда $\Rightarrow d_y = l_y$

Корректность: доказательство (продолжение)

Обозначим x — последнюю помеченную вершину на кратчайшем пути до v, y — следующую за ней

- По предположению индукции: $d_x = l_x$
- Ребро (x, y) на кратчайшем пути $\Rightarrow l_y = l_x + w_{xy}$
- При пометке x была релаксация $\Rightarrow d_y \leq d_x + w_{xy} = l_y$
- Но $d_y \geq l_y$ всегда $\Rightarrow d_y = l_y$

Корректность: доказательство (продолжение)

Обозначим x — последнюю помеченную вершину на кратчайшем пути до v, y — следующую за ней

- По предположению индукции: $d_x = l_x$
- Ребро (x, y) на кратчайшем пути $\Rightarrow l_y = l_x + w_{xy}$
- При пометке x была релаксация $\Rightarrow d_y \leq d_x + w_{xy} = l_y$
- Но $d_y \geq l_y$ всегда $\Rightarrow d_y = l_y$

Может ли $y \neq v$? Нет: $d_v \leq d_y = l_y \leq l_v \leq d_v$

$\Rightarrow v = y$ и $d_v = l_v$ $\square$

Почему не работает с отрицательными рёбрами?

Почему не работает с отрицательными рёбрами?

- Доказательство использует: путь дальше y **не короче** l_y

Почему не работает с отрицательными рёбрами?

- Доказательство использует: путь дальше y **не короче** l_y
- С отрицательными рёбрами это неверно: ребро с $w < 0$ может уменьшить расстояние

Почему не работает с отрицательными рёбрами?

- Доказательство использует: путь дальше y **не короче** l_y
- С отрицательными рёбрами это неверно: ребро с $w < 0$ может уменьшить расстояние
- Помеченная вершина может получить более короткий путь позже

Реализация за $O(n^2)$

```
// ...
std::vector<int> dist(n, INF);
std::vector<bool> used(n, false);
dist[s] = 0;

for (int i = 0; i < n; ++i) {
    int v = -1;
    for (int u = 0; u < n; ++u) {
        if (!used[u] && (v == -1 || dist[u] < dist[v])) {
            v = u;
        }
    }
    if (dist[v] == INF) break;
    used[v] = true;

    for (auto [u, w] : adj[v]) {
        if (dist[v] + w < dist[u]) {
            dist[u] = dist[v] + w;
```

Сложность $O(n^2)$

Сложность $O(n^2)$

- n итераций
- На каждой: поиск минимума за $O(n)$ + релаксации за $O(\deg(v))$
- Итого: $O(n^2 + m) = O(n^2)$

Сложность $O(n^2)$

- n итераций
- На каждой: поиск минимума за $O(n)$ + релаксации за $O(\deg(v))$
- Итого: $O(n^2 + m) = O(n^2)$

Оптимально для **плотных** графов ($m \approx n^2$)

Для **разреженных** графов ($m \approx n$) — можно лучше

Ускорение: очередь с приоритетом

Вместо линейного поиска минимума — используем структуру с операциями:

- извлечь минимум: $O(\log n)$
- вставить / обновить элемент: $O(\log n)$

Ускорение: очередь с приоритетом

Вместо линейного поиска минимума — используем структуру с операциями:

- извлечь минимум: $O(\log n)$
- вставить / обновить элемент: $O(\log n)$

Подходит `std::set<std::pair<int, int>>` — хранит пары (d_v, v)

Ускорение: очередь с приоритетом

Вместо линейного поиска минимума — используем структуру с операциями:

- извлечь минимум: $O(\log n)$
- вставить / обновить элемент: $O(\log n)$

Подходит `std::set<std::pair<int, int>>` — хранит пары (d_v, v)

При релаксации: удаляем старую пару (d_u, u), вставляем новую $(d_v + w, u)$

Массив `used` больше не нужен — множество само хранит необработанные вершины

Реализация за $O(m \log n)$

```
std::vector<int> dist(n, INF);
dist[s] = 0;
std::set<std::pair<int, int>> q;
q.insert({0, s});

while (!q.empty()) {
    auto [d, v] = *q.begin();
    q.erase(q.begin());

    for (auto [u, w] : adj[v]) {
        if (dist[v] + w < dist[u]) {
            q.erase({dist[u], u});
            dist[u] = dist[v] + w;
            q.insert({dist[u], u});
        }
    }
}
```

Сложность $O(m \log n)$

Сложность $O(m \log n)$

- Каждое ребро — не более одной вставки и удаления из `set`
- Каждая операция с `set` — $O(\log n)$
- Итого: $O(m \log n)$

Сложность $O(m \log n)$

- Каждое ребро — не более одной вставки и удаления из `set`
- Каждая операция с `set` — $O(\log n)$
- Итого: $O(m \log n)$

Граф	$O(n^2)$	$O(m \log n)$
Плотный ($m \approx n^2$)	$O(n^2)$	$O(n^2 \log n)$ — хуже!
Разреженный ($m \approx n$)	$O(n^2)$	$O(n \log n)$ — лучше!

Восстановление пути

Заведём массив предков p: p_v — вершина, из которой произошла последняя успешная релаксация в v

Восстановление пути

Заведём массив предков p: p_v — вершина, из которой произошла последняя успешная релаксация в v

При релаксации обновляем p_v параллельно с d_v:

```
if (dist[v] + w < dist[u]) {
    dist[u] = dist[v] + w;
    parent[u] = v;
    // ... обновление очереди
}
```

Восстановление пути

Заведём массив предков p: p_v — вершина, из которой произошла последняя успешная релаксация в v

При релаксации обновляем p_v параллельно с d_v:

```
if (dist[v] + w < dist[u]) {
    dist[u] = dist[v] + w;
    parent[u] = v;
    // ... обновление очереди
}
```

Восстановление — как в BFS: идём от t к s по массиву `parent`

Восстановление пути: реализация

```cpp
std::vector<int> get_path(int s, int t) {
    if (dist[t] == INF) {
        return {};
    }

    std::vector<int> path;
    for (int v = t; v != -1; v = parent[v]) {
        path.push_back(v);
    }

    std::reverse(path.begin(), path.end());
    return path;
}
```

Итоги

- Алгоритм Дейкстры находит кратчайшие пути от s до всех вершин в графе с **неотрицательными** весами
- **Жадный** алгоритм: на каждом шаге помечаем ближайшую непомеченную вершину
- Две реализации: $O(n^2)$ для плотных и $O(m \log n)$ для разреженных графов
- **Не работает** с отрицательными рёбрами
- Восстановление пути — через массив предков (как в BFS)