Алгоритмы на графах

Модуль 2. Кратчайшие расстояния.

Лекция 9.

Кратчайшие пути (часть 2).

Адигеев Михаил Георгиевич

2022

План лекции

1. Задача о кратчайшем пути при отрицательных весах. Алгоритм Беллмана-Форда.
2. Алгоритм A*.
3. Задание 5.

Задачи о кратчайшем пути

Возможный вариант: искать только пути, не содержащие контуров.

К сожалению, такая постановка задачи является NP-трудной (к ней можно свести незамкнутую ориентированную задачу коммивояжёра).

Поэтому поставим себе задачу в таком виде: для заданного графа $G(V, E)$, $w: E \to R$, с отрицательными весами дуг, и выбранной вершины $s \in V$ выдать один из ответов:

a) «В графе есть контур отрицательного веса» (достижимый из s).

b) Если контуров отрицательного веса нет, то построить кратчайшие пути из s во все вершины графа.

Эту задачу эффективно решает **алгоритм Беллмана-Форда**.

Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда реализует метод *динамического программирования*.

Принципы динамического программирования:

- ✓ Исходная задача сводится к последовательности подзадач меньшего размера.
- ✓ Задачи решаются от меньшей к большей. Результаты сохраняются в таблицу.

В качестве параметра, характеризующего размер задачи, выберем ***длину*** пути: L =количество дуг в пути. То есть, мы инициализируем таблицу весами путей длины $L = 0$ (такой путь только один: $s \to s$, вес=0), потом рассчитываем кратчайшие пути длины $L = 1$ и т.д.

Алгоритм Беллмана-Форда

Если уже рассчитаны кратчайшие пути длин $1, \ldots, L$, то как найти кратчайшие пути длины $\leq L + 1$?

Каждый такой путь (с концевой вершиной v) либо имеет длину $\leq L$, либо может быть получен из некоторого пути $s \rightsquigarrow u$ (кратчайший путь из s в u длины $\leq L$) добавлением дуги (u, v). Это следует из следующей леммы, описывающей принцип оптимальности (ключевое условие для применимости динамического программирования) для задачи о кратчайших путях.

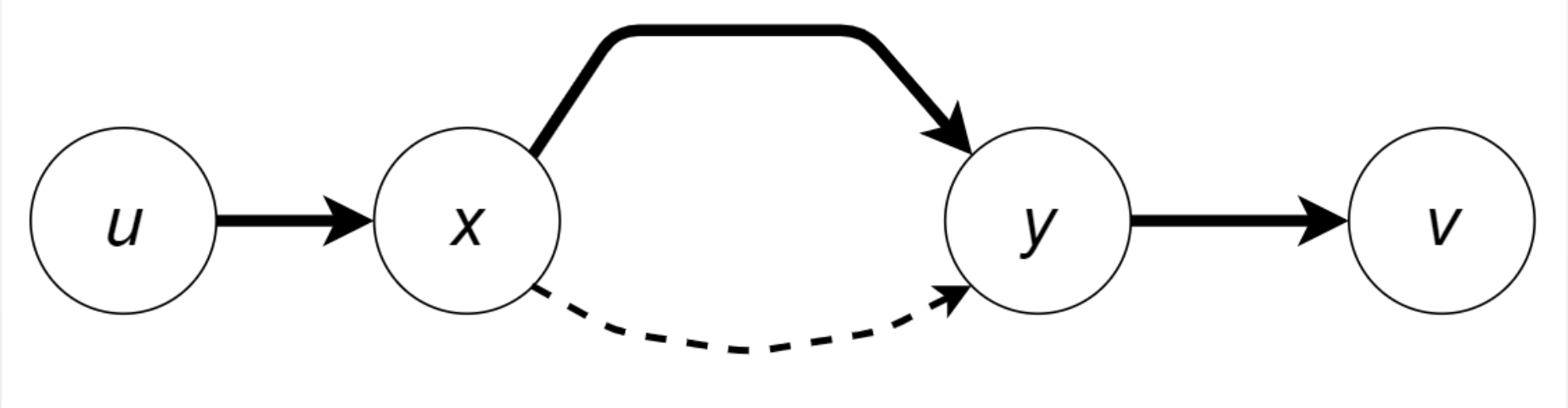
Алгоритм Беллмана-Форда

Лемма (принцип оптимальности для кратчайших путей). Любой фрагмент кратчайшего пути является кратчайшим путём между своими концевыми вершинами.

Доказательство.

Рассмотрим путь *P* из вершины u в вершину v и проходящий через вершины x и y (возможно, $x = u$ и/или $y = v$).

Предположим, что фрагмент пути *P* между x и y не является кратчайшим путём из x в y.

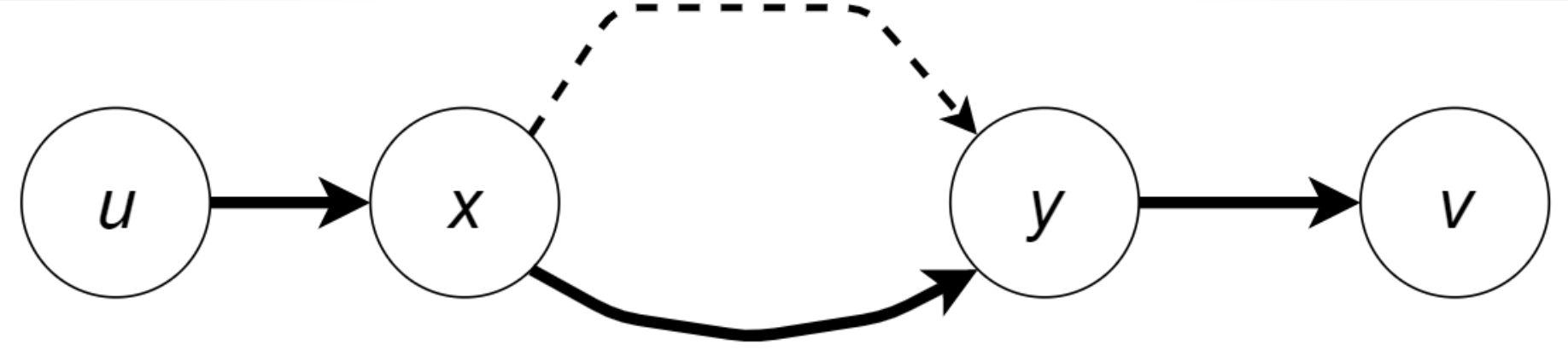


Алгоритм Беллмана-Форда

Заменим этот фрагмент на кратчайший путь из x в y – получим путь P'.

Очевидно, что $w(P') < w(P)$, поэтому P – не кратчайший путь из u в v.

Полученное противоречие доказывает лемму.



Алгоритм Беллмана-Форда

Итак, пусть уже рассчитаны кратчайшие пути длин $1, \dots, L$. Обозначим такие пути $P_L(v)$, где v – конечная вершина пути, а через $d_L(v)$ - веса этих кратчайших путей.

Тогда веса кратчайших пути длины $\leq L + 1$ можно вычислить с помощью рекуррентных соотношений:

$$d_{L+1}(v) = \min \begin{cases} d_L(v), \\ \min\{d_L(u) + w(u, v) : (u, v) \in E\} \end{cases}$$

Сами кратчайшие пути найдём с помощью ссылок на «предка» в кратчайшем пути.

Алгоритм Беллмана-Форда

Лемма 1.

Если для некоторого $k \geq 0$ и для всех $v \in V$ выполняется

$$d_{k+1}(v) = d_k(v),$$

то

1) Для всех $L > k$ и всех $v \in V$ $d_L(v) = d_k(v)$.
2) Для всех $v \in V$ величина $d_k(v)$ равна расстоянию от s до v.

Лемма 2.

Если для какой-то вершины $d_n(v) \neq d_{n-1}(v)$, то на графе есть контур отрицательного веса.

Алгоритм Беллмана-Форда

```
// Инициализация
For each v ∈ V:
        d[i] := +∞;
        p[i] := NULL;
d[s] := 0;
// Расчёт кратчайших путей
Для i от 1 до |V|-1:
        Для всех (u,v) ∈ E:
                Если d[v] > d[u]+w[u,v], то
                        d[v] := d[u]+w[u,v];
                        p[v] := u;
// Проверка
NegativeCycle := False;
Для всех (u,v) ∈ E:
        Если d[v] > d[u]+w[u,v], то
                NegativeCycle := True;

Вернуть NegativeCycle.
```

Алгоритм Беллмана-Форда

Теорема. Алгоритм Беллмана-Форда корректно решает поставленную задачу, то есть

a) Если в графе есть контур отрицательного веса (достижимый из s), то возвращает True.

b) Если контуров отрицательного веса нет, то рассчитывает веса кратчайших путей (массив d) и данные для построения кратчайших путей (массив p) из s во все вершины графа.

Корректность следует из лемм 1 и 2.

Временная сложность алгоритма: $O(nm) \sim O(n^3)$. То есть, он работает дольше, чем алгоритм Дейкстры.

Как можно для некоторых случаев сократить время работы алгоритма?

Алгоритм Беллмана-Форда

```
// Инициализация
For each v ∈ V:
        d[i] := +∞;
        p[i] := NULL;
d[s] := 0;
// Расчёт кратчайших путей
Для i от 1 до |V|-1:                    n-1 итерация
        Для всех (u,v) ∈ E:             m дуг
                Если d[v] > d[u]+w[u,v], то
                        d[v] := d[u]+w[u,v];
                        p[v] := u;
// Проверка
NegativeCycle := False;
Для всех (u,v) ∈ E:                     m итераций
        Если d[v] > d[u]+w[u,v], то
                NegativeCycle := True;

Вернуть NegativeCycle.
```

Алгоритм Беллмана-Форда

Достоинства алгоритма Беллмана-Форда:

- Корректно работает для задач с отрицательными весами дуг.
- Корректно работает даже для задач с отрицательными контурами.
- Подходит для распределённых вычислений.

Алгоритм A*

«Эвристический» алгоритм информированного поиска кратчайшего пути из стартовой вершины s в целевую вершину t.

Алгоритм A* можно рассматривать как модификацию алгоритмов поиска в ширину и алгоритма Дейкстры.

Идея: для каждой вершины $v \in V$ рассчитывается оценка $f(v) = g(v) + h(v)$, где

- $g(v)$ – вес уже найденного кратчайшего пути из s в v;
- $h(v)$ – *оценка* веса кратчайшего пути из v в t.

Анализируем вершины в порядке увеличения $f(v)$.

При $h(v) \equiv 0$ алгоритм A* совпадает с алгоритмом Дейкстры.

Алгоритм A*

Алгоритм A*

$V' \coloneqq \{s\}$
For each $v \in V$:
    d[i] := $+\infty$;
    p[i] := NULL;
    f[i] := $+\infty$;
d[s] := 0;
f[s] := H(s, t);
Пока существует дуга (u,v): $u \in V', v \notin V'$:
    Выбрать дугу (u*,v*): **f(v*,t)** минимально;
    $V' \coloneqq V' \cup \{v^*\}$;
    d[v*] := d[u*]+w[u*,v*];
    p[v*] := u*;
    Если v*=t, то прекратить;
    Update_F&P(v*);

Алгоритм A*

```
Update F&P(v)
For each (v,u)∈E:
      if u∈V\V'   &   d[u] > d[v]+w[v,u]:
            d[u] := d[v]+w[v,u];
            f[u] := d[u]+H(u,t);
```

Алгоритм A*

$f(v) = g(v) + h(v)$.

Как выбирать h() и как от выбора h() зависит поведение алгоритма?

- При $h(v) \equiv 0$ алгоритм A* совпадает с алгоритмом Дейкстры.
- При $h \gg g$ алгоритм A* превращается в жадный эвристический (неточный, но быстрый) алгоритм.
- Если функция $h()$ является *допустимой*, то есть во всех случаях выполняется $h(v) \leq dist(v, t)$, то A* является точным, то есть находит кратчайший путь из s в t.

Временная сложность A* зависит от используемой эвристики: в худшем случае экспоненциальна, но полиномиальна, если выполняется условие $|h(v) - dist(v, t)| \leq O(\log(dist(v, t))$.

Ёмкостная сложность – в худшем случае экспоненциальна ☹

Алгоритм A*

$f(v) = g(v) + h(v)$.

Как выбирать h() и как от выбора h() зависит поведение алгоритма?

- При $h(v) \equiv 0$ алгоритм A* совпадает с алгоритмом Дейкстры.
- При $h \gg g$ алгоритм A* превращается в жадный эвристический (неточный, но быстрый) алгоритм.
- Если функция $h()$ является ***допустимой***, то есть во всех случаях выполняется $h(v) \leq dist(v, t)$, то A* является точным, то есть находит кратчайший путь из s в t.

Временная сложность A* зависит от используемой эвристики: в худшем случае экспоненциальна, но полиномиальна, если выполняется условие $|h(v) - dist(v, t)| \leq O(\log(dist(v, t))$.

Ёмкостная сложность – в худшем случае экспоненциальна ☹

Алгоритм A*

Как для конкретной прикладной задачи выбирать h()?

- (В редких случаях) можно заранее рассчитать dist(v,t) и использовать эти значения в качестве h(v).
- Если вершины графа – точки на плоскости/в пространстве/на решётке, то в качестве h() можно выбрать евклидову/манхеттенскую метрику.
- Правило расчёта h(v) можно изменять динамически по ходу выполнения алгоритма.
- Если выбрать *недопустимую* h(), то A* может найти не самый оптимальный путь, но а) намного быстрее, и б) учесть дополнительные важные для практики характеристики (отсутствие резких поворотов и т.п.).

Задание 5

Задание 5

Задача «Лягушка».

Лягушка мечтает добраться к мухомору. Чтобы это сделать ей надо прыгать по болоту с кочки на кочку. Одним прыжком лягушка может переместиться только до ближайших кочек, которые находятся на расстоянии не более R от текущего местоположения лягушки. Всего на болоте N кочек.

Помогите лягушке добраться до мухомора, так чтобы суммарное расстояние, которое она пропрыгает, было минимально.

Задание 5

Формат входных данных

В первой строке два числа N и R разделенные одним пробелом. Далее следуют N строк с координатами кочек. Каждая строка представляет собой два целых числа (абсцисса и ордината кочки) в диапазоне от 0 до 1000, разделенных одним пробелом. Лягушка находится всегда на первой в списке кочке, а мухомор на последней.

Формат выходных данных

Если решение не существует, то вывести -1. Если решение существует - вывести суммарное расстояние, которое придется напрыгать лягушке, и последовательность координат кочек (координаты каждой кочки на отдельной строке, разделены пробелом), в том порядке, в котором по ним должна прыгать лягушка. Вещественные числа будут сравниваться с точностью до второго знака после запятой.

Задание 5

Пример

input.txt	output.txt
4 3 1 1 1 3 4 1 4 4	6.00 1 1 4 1 4 4