Комбинаторные алгоритмы

Пути в сетях

Гальперин Александр Леонидович

2018 г.

Пути в сетях

Постановка задачи

Определение

Взвешенный ориентированный граф $G(V, E, c)$ называется *сетью*.

Пути в сетях

Постановка задачи

Определение

Взвешенный ориентированный граф $G(V, E, c)$ называется *сетью*.

Сеть можт быть представлена матрицей весов дуг или списками смежности $\overrightarrow{list}[v]$ или $\overleftarrow{list}[v]$.

Пути в сетях

Постановка задачи

Определение

Взвешенный ориентированный граф $G(V, E, c)$ называется *сетью*.

Сеть можт быть представлена матрицей весов дуг или списками смежности $\overrightarrow{list}[v]$ или $\overleftarrow{list}[v]$.

В этом разделе, следуя традиции, нам удобнее ормаршрут называть *путем*.

Пути в сетях

Постановка задачи

Пусть P — некоторый (v, w)–путь:

$$v = v_0 \xrightarrow{e_1} v_1 \xrightarrow{e_2} \dots \xrightarrow{e_k} v_k = w.$$

Пути в сетях

Постановка задачи

Пусть P — некоторый (v, w)–путь:

$$v = v_0 \xrightarrow{e_1} v_1 \xrightarrow{e_2} \ldots \xrightarrow{e_k} v_k = w.$$

Положим

$$c(P) = c(e_1) + c(e_2) + \ldots + c(e_k).$$

Пути в сетях

Постановка задачи

Пусть P — некоторый (v, w)–путь:

$$v = v_0 \xrightarrow{e_1} v_1 \xrightarrow{e_2} \ldots \xrightarrow{e_k} v_k = w.$$

Положим

$$c(P) = c(e_1) + c(e_2) + \ldots + c(e_k).$$

Величину $c(P)$ назовем **длиной пути** P.

Пути в сетях

Постановка задачи

Наименьшую из длин (v, w)–путей назовем **расстоянием от** v **до** w, ...

Пути в сетях

Постановка задачи

Наименьшую из длин (v, w)–путей назовем **расстоянием от** v **до** w, ...
...а тот (v, w)–путь, длина которого равна расстоянию от v до w, будем называть **кратчайшим** (v, w)**–путем**.

Пути в сетях

Постановка задачи

Наименьшую из длин (v, w)–путей назовем **расстоянием от** v **до** w, ...

...а тот (v, w)–путь, длина которого равна расстоянию от v до w, будем называть **кратчайшим** (v, w)**–путем**.

$\mathcal{NB}$

Ясно, что расстояние от v до w может отличаться от расстояния от w до v.

Задача о кратчайшем пути (ЗКП) между фиксированными вершинами формулируется следующим образом:

Задача о кратчайшем пути

В заданной сети G с двумя выделенными вершинами s и t найти кратчайший (s, t)–путь.

Пути в сетях

Постановка задачи

$\mathcal{NB}1$

Задачу о кратчайшем пути можно рассматривать и в неориентированных взвешенных графах, заменив каждое ребро vw двумя дугами vw и wv, считая, что веса обеих дуг равны весу ребра vw.

Пути в сетях

Постановка задачи

$\mathcal{NB}$1

Задачу о кратчайшем пути можно рассматривать и в неориентированных взвешенных графах, заменив каждое ребро vw двумя дугами vw и wv, считая, что веса обеих дуг равны весу ребра vw.

$\mathcal{NB}$2

Если положить вес каждого ребра равным единице, то получим данное ранее определение длины пути как числа входящих в него ребер.

Пути в сетях

Общий случай

Алгоритм форда–Беллмана

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

- Всюду в дальнейшем будем предполагать, что если вершины v и w не являются смежными в сети $G = (V, E, c)$, то $c(v, w) = \infty$.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

- Всюду в дальнейшем будем предполагать, что если вершины v и w не являются смежными в сети $G = (V, E, c)$, то $c(v, w) = \infty$.
- Для удобства изложения и во избежание вырожденных случаев при оценке сложности алгоритмов будем считать, что $n \leqslant m$. Это исключает ситуации, когда большинство вершин изолированы.
- Будем рассматривать только такие сети, в которых нет контуров отрицательной длины. Понятно, что если есть контур отрицательной длины, то расстояние между некоторыми парами вершин становится неопределенным: обходя этот контур, можно построить путь между этими вершинами меньше любой наперед заданной величины.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Идея метода решегия ЗКП:

1. вначале размечаем все вершины данной сети (прямой ход алгоритма), вычисляя расстояние от s до всех вершин;
2. затем, используя специальные метки, обратным ходом строим требуемый путь.

$\mathcal{NB}$

Интересно отметить, что для вычисления расстояния от s до заданной вершины t мы вынуждены вычислять расстояния от s до всех вершин сети.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

$\mathcal{NB}$

В настоящее время неизвестен ни один алгоритм нахождения расстояния между фиксированными вершинами, который был бы существенно более эффективным, чем известные алгоритмы вычисления расстояний от одной из фиксированных вершин до всех остальных.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Начнем с первого этапа: вычисления расстояний от фиксированной вершины s до всех вершин.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Начнем с первого этапа: вычисления расстояний от фиксированной вершины s до всех вершин.

Метод, который мы здесь используем, часто называют *динамическим программированием*, а алгоритм вычисления расстояний *алгоритмом Форда–Беллмана*.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Начнем с первого этапа: вычисления расстояний от фиксированной вершины s до всех вершин.

Метод, который мы здесь используем, часто называют *динамическим программированием*, а алгоритм вычисления расстояний *алгоритмом Форда–Беллмана*.

Появился этот алгоритм в работе Форда 1956 года и в работе Беллмана 1958 года.

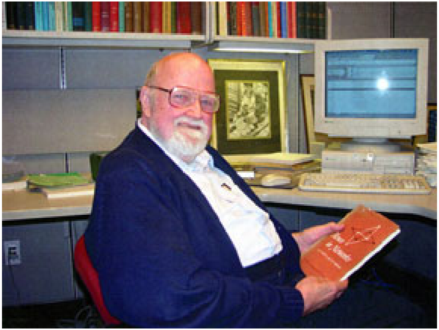
Пути в сетях

Алгоритм

Форда–Беллмана

Лестер Рендольф Форд младший

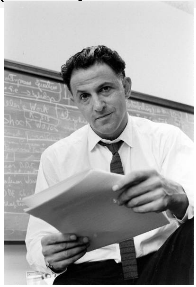
(23 сентября 1927 — 26 февраля 2017)



Американский математик.
В 1956 году предложил алгоритм построения кратчайшего пути в сетях.

Ричард Беллман

(26 августа 1920 — 19 марта 1984)



Американский математик — специалист по прикладной математике. Окончил Бруклинский колледж в 1941 году (B.A.), университет Висконсин–Мэдисон (M.A.). Работал в подразделении теоретической физики лаборатории в Лос–Аламос, где в 1946 году получил степень PhD. Награжден почетной медалью IEEE (Institute of Electrical and Electronics Engineers) за вклад в развитие теории принятия решений, теории управления и, в особенности, за создание и развития теории динамического программирования. Был действительным членом Американской академии наук и искусств (с 1975 года), и Национальной инженерной академии США (с 1977 года).

Предложил алгоритм построения кратчайшего пути в сетях **в 1958 году**.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

- Основная идея алгоритма — поэтапное вычисление кратчайших расстояний.
- Пусть $d_k(v)$ — длина кратчайшего среди всех (s, v)– путей, содержащих не более k дуг.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

- Основная идея алгоритма — поэтапное вычисление кратчайших расстояний.
- Пусть $d_k(v)$ — длина кратчайшего среди всех (s, v)– путей, содержащих не более k дуг.
- Легко понять, что

$$d_1(v) \geqslant d_2(v) \geqslant \ldots, \geqslant d_{n-1}(v).$$

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

- Основная идея алгоритма — поэтапное вычисление кратчайших расстояний.
- Пусть $d_k(v)$ — длина кратчайшего среди всех (s, v)– путей, содержащих не более k дуг.
- Легко понять, что

$$d_1(v) \geqslant d_2(v) \geqslant \ldots, \geqslant d_{n-1}(v).$$

- Поскольку в графе нет контуров отрицательной длины, кратчайший (s, v)–путь не может содержать более, чем $n-1$ дугу. Поэтому $d_{n-1}(v)$ дает искомое расстояние.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

- Значения $d_1(v)$ вычисляются просто:

$$d_1(v) = c(s, v), \quad \text{для всех } v \in V.$$

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

- Значения $d_1(v)$ вычисляются просто:

$$d_1(v) = c(s, v), \quad \text{для всех } v \in V.$$

- Пусть значения $d_k(v)$ вычислены для всех $v \in V$. Легко видеть, что

$$d_{k+1}(v) = \min\left\{d_k(v), d_k(w) + c(w, v) | w \in V\right\}.$$

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Организовать все эти вычисления можно с помощью всего одного одномерного массива D длины n.

- Положив $D[v] = c(s, v)$, будем иметь равенство

$$D[v] = d_1(v).$$

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Организовать все эти вычисления можно с помощью всего одного одномерного массива D длины n.

- Положив $D[v] = c(s, v)$, будем иметь равенство

$$D[v] = d_1(v).$$

- Просматривая после этого все вершины v, произведем пересчет значений $D[v]$ по формуле:

$$D[v] = \min\{D[v], D[w] + c(w, v) | w \in V\}. \tag{1}$$

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

После завершения первого пересчета значений $D[v]$ для всех v, будем иметь неравенства $D[v] \leqslant d_2(v)$.

Почему именно неравенства?

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

- Пусть пересчет начался из вершины v_1. Ясно, что тогда

$$D[v_1] = d_2(v_1).$$

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

- Пусть пересчет начался из вершины v_1. Ясно, что тогда

$$D[v_1] = d_2(v_1).$$

- Пусть следующей вершиной, для которой был сделан пересчет, была вершина v_2. Тогда

$$D[v_2] \leqslant D[v_1] + c(v_1, v_2)$$

(это вытекает из (1)).

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

- Отсюда следует , что возможна ситуация, в которой $D[v_2] = D[v_1] + c(v_1, v_2)$, и, кроме того, значения $D[v_1]$ могли быть получены на пути , состоящем из двух дуг.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

- Отсюда следует , что возможна ситуация, в которой $D[v_2] = D[v_1] + c(v_1, v_2)$, и, кроме того, значения $D[v_1]$ могли быть получены на пути , состоящем из двух дуг.
- Следовательно, значение $D[v_2]$ может быть получено по некоторому пути из трех дуг, т.е. $D[v_2] \leqslant d_2(v_2)$.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

- Отсюда следует , что возможна ситуация, в которой $D[v_2] = D[v_1] + c(v_1, v_2)$, и, кроме того, значения $D[v_1]$ могли быть получены на пути , состоящем из двух дуг.
- Следовательно, значение $D[v_2]$ может быть получено по некоторому пути из трех дуг, т.е. $D[v_2] \leqslant d_2(v_2)$.
- Повторив $n-2$ раза процесс пересчета D[v], будем иметь равенства

$$D[v] = d_{n-1}(v),$$

т.е. $D[v]$ дает расстояние от s до v.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Прежде, чем дать формальное описание равенства $D[v] = d_{n-1}(v)$, приведем формальное описание алгоритма Форда–Беллмана.

Построение кратчайших путей удобно вести с помощью одномерного массива *Previous* длины n, где *Previous*$[v]$ дает имя вершины, предшествующей вершине v в кратчайшем (s, v)–пути.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Общий случай. Алгоритм Форда–Беллмана

Вход: сеть $G = (V, E, c)$, заданная матрицей весов A порядка n; вершины s и t.

Выход: расстояния $D[v]$ от s до всех вершин $v \in V$, стек S, содержащий кратчайший (s, t)-путь, или сообщение, что искомого пути в сети не существует.

```
procedure Distance
begin
  D[s] := 0; Previous[s] := 0;
  for v ∈ V \ {s} do
    begin D[v] := A[s, v]; Previous[v] := s end;
  for k := 1 to n − 2 do
    for v ∈ V \ {s} do
      for w ∈ V do
        if D[w] + A[w, v] < D[v] then
          begin
            D[v] := D[w] + A[w, v];
            Previous[v] := w
          end
end;
begin
  Distance;
  if D[t] < ∞ then
    begin
      S := nil; S ⇐ t; v := t;
      while Previous[v] ≠ 0 do
        begin v := Previous[v]; S ⇐ v end
    end
  else writeln («Not exists»);
end.
```

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Напомним, что по определению матрицы весов справедливы равенства:

$$A[v,w] = \begin{cases} c(v,w), & \text{если } (v,w) \in E; \\ \infty, & \text{если } (v,w) \notin E. \end{cases}$$

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

```
1.  procedure Distance
2.  begin
3.    D[s] := 0; Previous[s] := 0;
4.    for v ∈ V \ {s}  do
5.      begin D[v] := A[s, v]; Previous[v] := s end;
6.    for k := 1 to n − 2 do
7.      for v ∈ V \ {s}  do
8.        for w ∈ V  do
9.          if D[w] + A[w, v] < D[v] then
10.           begin
11.             D[v] := D[w] + A[w, v];
12.             Previous[v] := w
13.           end
14. end;
```

Вернемся к обоснованию равенства $Dv = d_{n-1}(v)$. Заметим, что при первом входе в цикл, начинающийся в строке 6, справедливы равенства

$$D[v] = d_1(v)$$

для всех $v \in V$.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Предположим, что при входе в k–ую итерацию цикла 6 справедливы неравенства

$$D[v] \leqslant d_k(v)$$

для всех $v \in V$.

```
1.  procedure Distance
2.  begin
3.    D[s] := 0; Previous[s] := 0;
4.    for v ∈ V \ {s}  do
5.      begin D[v] := A[s, v]; Previous[v] := s end;
6.    for k := 1 to n − 2 do
7.      for v ∈ V \ {s}  do
8.        for w ∈ V  do
9.          if D[w] + A[w, v] < D[v] then
10.           begin
11.             D[v] := D[w] + A[w, v];
12.             Previous[v] := w
13.           end
14. end;
```

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Предположим, что при входе в k–ую итерацию цикла 6 справедливы неравенства

$$D[v] \leqslant d_k(v)$$

для всех $v \in V$.

```
1.  procedure Distance
2.  begin
3.    D[s] := 0; Previous[s] := 0;
4.    for v ∈ V \ {s}  do
5.      begin D[v] := A[s, v]; Previous[v] := s end;
6.    for k := 1 to n − 2 do
7.      for v ∈ V \ {s}  do
8.        for w ∈ V  do
9.          if D[w] + A[w, v] < D[v] then
10.           begin
11.             D[v] := D[w] + A[w, v];
12.             Previous[v] := w
13.           end
14. end;
```

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Докажем по индукции по k, что по завершении этой итерации для любой вершины $v \in V$ будут справедливы соотношения

$$D[v] \leqslant d_{k+1}(v).$$

```
procedure Distance
begin
  D[s] := 0; Previous[s] := 0;
  for v ∈ V \ {s}  do
    begin D[v] := A[s, v]; Previous[v] := s end;
  for k := 1 to n − 2 do
    for v ∈ V \ {s}  do
      for w ∈ V  do
        if D[w] + A[w, v] < D[v] then
          begin
            D[v] := D[w] + A[w, v];
            Previous[v] := w
          end
end;
```

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Действительно, после выполнения цикла в строке 6 для произвольной вершины v выполнено соотношение

$$D[v] \leqslant D[w] + A[w, v]$$

для всех $w \in V$.

```
1.  procedure Distance
2.  begin
3.    D[s] := 0; Previous[s] := 0;
4.    for v ∈ V \ {s}  do
5.      begin D[v] := A[s, v]; Previous[v] := s end;
6.    for k := 1 to n − 2 do
7.      for v ∈ V \ {s}  do
8.        for w ∈ V  do
9.          if D[w] + A[w, v] < D[v] then
10.           begin
11.             D[v] := D[w] + A[w, v];
12.             Previous[v] := w
13.           end
14. end;
```

По предположению индукции $D[w] \leqslant d_k(v)$. Учитывая, что

$$A[w, v] = c(w, v),$$

получаем неравенство

$$D[v] \leqslant d_k(w) + c(w, v).$$

По предположению индукции $D[w] \leqslant d_k(v)$. Учитывая, что

$$A[w, v] = c(w, v),$$

получаем неравенство

$$D[v] \leqslant d_k(w) + c(w, v).$$

В частности, это неравенство справедливо и для той вершины w, которая является предпоследней в кратчайшем (s, v)–пути, состоящем из $k + 1$ дуги, откуда и следует требуемое неравенство

$$D[v] \leqslant d_{k+1}(v).$$

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Таким образом, по завершении $n-2$ итерации цикла в строке 6 справедливы неравенства

$$D[v] \leqslant d_{n-1}(v)$$

для всех вершин $v \in V$.

```
1.  procedure Distance
2.  begin
3.    D[s] := 0; Previous[s] := 0;
4.    for v ∈ V \ {s}  do
5.      begin D[v] := A[s, v]; Previous[v] := s end;
6.    for k := 1 to n − 2 do
7.      for v ∈ V \ {s}  do
8.        for w ∈ V  do
9.          if D[w] + A[w, v] < D[v] then
10.           begin
11.             D[v] := D[w] + A[w, v];
12.             Previous[v] := w
13.           end
14. end;
```

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Выше уже отмечалось, что поскольку сеть не содержит контуров отрицательной длины, то $d_{n-1}(v)$ дает длину кратчайшего (s, v)–пути. Следовательно, $D[v] = d_{n-1}[v]$.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Выше уже отмечалось, что поскольку сеть не содержит контуров отрицательной длины, то $d_{n-1}(v)$ дает длину кратчайшего (s, v)–пути. Следовательно, $D[v] = d_{n-1}[v]$.

Тем самым обоснование этого равенства, а вместе с ним и корректности алгоритма Форда–Беллмана, завершено.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Теорема 1

Алгоритм Форда–Беллмана имеет сложность $O(n^3)$.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Доказательство. Ясно, что сложность алгоритма определяется сложностью процедуры *Distance*, т.к. основная программа требует числа операций, пропорционального n.

```
begin
  Distance;
  if D[t] < ∞ then
    begin
      S := nil; S ⇐ t; v := t;
      while Previous[v] ≠ 0 do
        begin v := Previous[v]; S ⇐ v end
    end
  else writeln («Not exists»);
end.
```

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Ясно, что количество операций в цикле 4 пропорционально n.

```
1.  procedure Distance
2.  begin
3.    D[s] := 0; Previous[s] := 0;
4.    for v ∈ V \ {s} do
5.      begin D[v] := A[s, v]; Previous[v] := s end;
6.    for k := 1 to n − 2 do
7.      for v ∈ V \ {s} do
8.        for w ∈ V do
9.          if D[w] + A[w, v] < D[v] then
10.           begin
11.             D[v] := D[w] + A[w, v];
12.             Previous[v] := w
13.           end
14. end;
```

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

Поскольку цикл в строке 6 выполняется $n-2$ раза, в строке 7 — $n-1$ раз, а в строке 8 — n раз, и число выполнений оператора присваивания при каждом входе в цикл 8 ограничено константой,

```
1.  procedure Distance
2.  begin
3.    D[s] := 0; Previous[s] := 0;
4.    for v ∈ V \ {s}  do
5.      begin D[v] := A[s, v]; Previous[v] := s end;
6.    for k := 1 to n − 2 do
7.      for v ∈ V \ {s}  do
8.        for w ∈ V  do
9.          if D[w] + A[w, v] < D[v] then
10.           begin
11.             D[v] := D[w] + A[w, v];
12.             Previous[v] := w
13.           end
14. end;
```

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

то сложность выполнения цикла, начинающегося в строке 6, есть

$$O\left((n-2)\cdot(n-1)\cdot n\right) = \\ = O(n^3).$$

Что и требовалось доказать.■

```
1.  procedure Distance
2.  begin
3.    D[s] := 0; Previous[s] := 0;
4.    for v ∈ V \ {s}  do
5.      begin D[v] := A[s, v]; Previous[v] := s end;
6.    for k := 1 to n − 2 do
7.      for v ∈ V \ {s}  do
8.        for w ∈ V  do
9.          if D[w] + A[w, v] < D[v] then
10.           begin
11.             D[v] := D[w] + A[w, v];
12.             Previous[v] := w
13.           end
14. end;
```

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

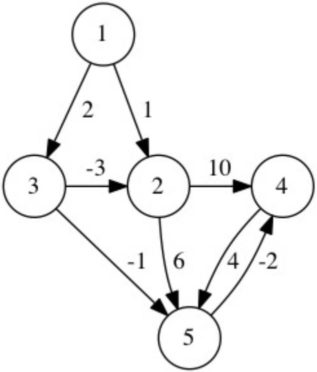
$\mathcal{NB}$

Вычисления по процедуре Distance можно завершить при таком выходе из цикла по k, при котором не происходит изменения ни одного значения $D[v]$.

Это может произойти при $k < n - 2$, однако такая модификация алгоритма не изменяет существенным образом его сложности, поскольку в худшем случае придется осуществить $n - 2$ итерации цикла по k.

Пути в сетях

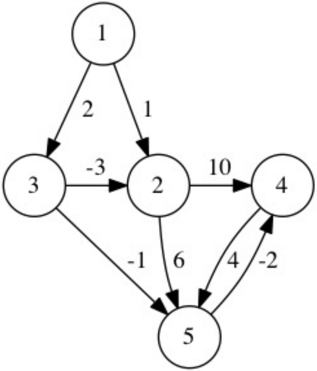
Общий случай. Алгоритм Форда–Беллмана



Рассмотрим работу алгоритма Форда–Беллмана на примере следующего связного ориентированного графа.

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

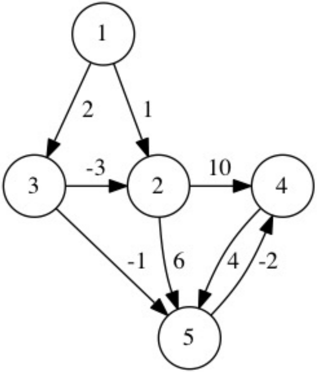


Проведем инициализацию.

	[1]	[2]	[3]	[4]	[5]
D	0	1	2	∞	∞
ПРЕДШ[v]	0	1	1	0	0

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

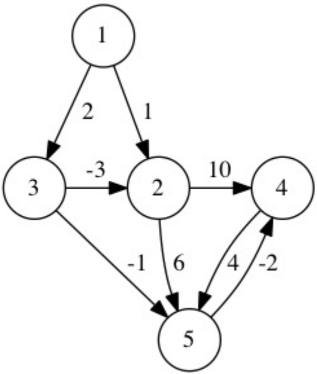


Произведем перерасчет значений $D[v]$ первый раз.

	[1]	[2]	[3]	[4]	[5]
$D_{\text{стар}}$	0	1	2	∞	∞
ПРЕДШ$[v]_{\text{стар}}$	0	1	1	0	0
D	0	-1	2	9	1
ПРЕДШ$[v]$	0	3	1	2	3

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

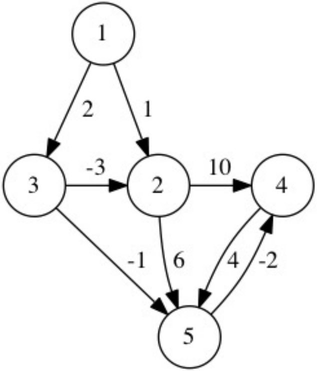


Произведем перерасчет значений $D[v]$ второй раз.

	[1]	[2]	[3]	[4]	[5]
$D_{\text{стар}}$	0	-1	2	9	1
ПРЕДШ$[v]_{\text{стар}}$	0	3	1	2	3
D	0	-1	2	-1	1
ПРЕДШ$[v]$	0	3	1	5	3

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана



Произведем перерасчет значений $D[v]$ третий и последний раз. Работа алгоритма закончена.

	[1]	[2]	[3]	[4]	[5]
$D_{\text{стар}}$	0	-1	2	-1	1
ПРЕДШ$[v]_{\text{стар}}$	0	3	1	5	3
D	0	-1	2	-1	1
ПРЕДШ$[v]$	0	3	1	5	3

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

В случае, когда сеть задана списками смежности $\overrightarrow{list}\,[v]$, для решения ЗКП в алгоритма Форда–Беллмана достаточно цикл в строке 8 процедуры *Distance* записать следующим образом:

8. **for** $w \in \overrightarrow{list}[v]$ **do**
9. **if** $D[w] + A[w, v] < D[v]$ **then**
10. **begin** $D[v] := D[w] + c[v, w]$; $Previous[v] := w$ **end**

Предыдущая версия:

8. **for** $w \in V$ **do**
9. **if** $D[w] + A[w, v] < D[v]$ **then**
10. **begin**

Пути в сетях

Общий случай. Алгоритм Форда–Беллмана

В таком случае алгоритм Форда–Беллмана будет иметь сложность $O(m \cdot n)$.