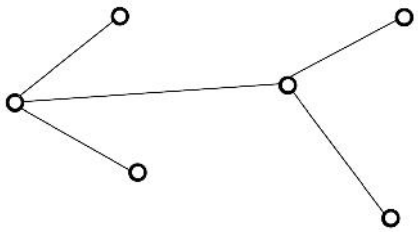
Лекция 12

План лекции

1. Определение дерева, свойства деревьев
2. Теорема Кэли
3. Процедура построения остовного леса
4. Свойства циклического ранга
5. Фундаментальная система циклов графа
6. Обход графов
6.1. Обход в глубину
6.2. Обход в ширину
6.3. Программа обхода графов в глубину
7. Поиск путей в графе
7.1. Алгоритм Терри
7.1.1. Описание последовательности действий в алгоритме Терри
7.1.2. Решение задачи вручную
7.1.3. Программа по алгоритму Терри
8. Волновой алгоритм
9. Поиск кратчайшего пути
9.1. Алгоритм Дейкстры
9.1.1. Пример ручного решения задачи
9.1.2. Описание машинного алгоритма
9.1.3. Программа по алгоритму Дейкстры
9.2. Алгоритм Форда – Беллмана нахождения минимального пути
9.2.1. Пример ручного решения
9.2.2. Программа по алгоритму Форда-Беллмана
9.2.3. Алгоритм Флойда-Уоршелла
9.2.4. Программа по алгоритму Флойда-Уоршелла

Определение дерева. Свойства деревьев

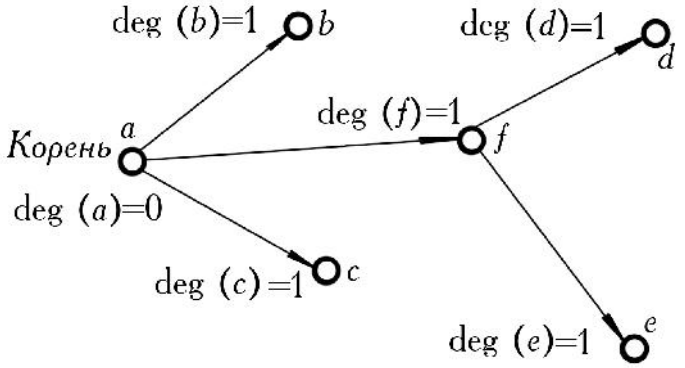
Неориентированным деревом называется связный неориентированный граф без циклов.



Корневым деревом называется такое дерево, в котором выделена вершина, называемая *корнем.*

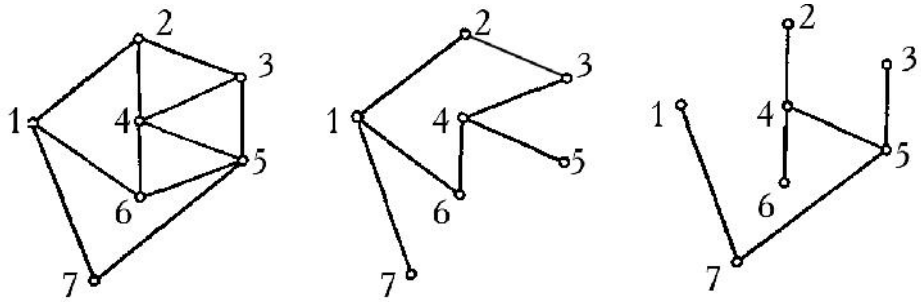
Корень в неориентированном графе – это одна из вершин, выбранная по желанию наблюдателя.

Ориентированным деревом называется связный ориентированный граф без циклов, в котором полустепень захода каждой вершины, за исключением корневой, равна единице, а полустепень захода корневой вершины равна 0.



Остовным деревом графа *G* называется остовной подграф графа *G*, являющийся деревом.

Остовным подграфом называется такой подграф, в котором множество его вершин совпадает с множеством вершин самого графа.

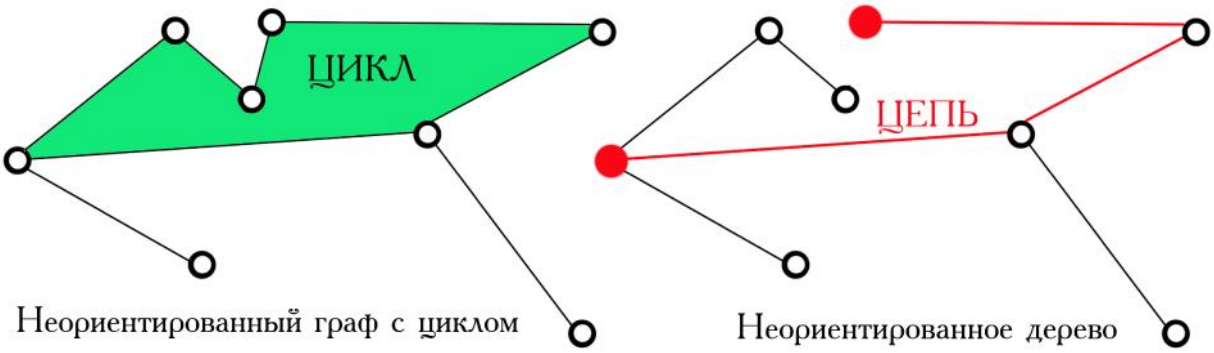


Исходный граф | Остовной подграф | Остовное дерево

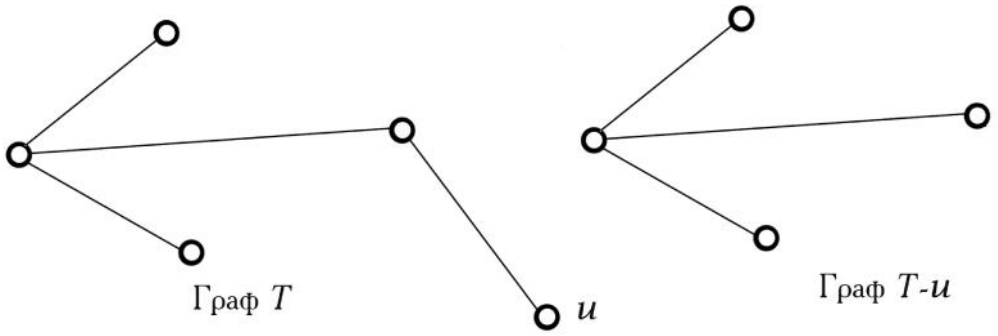
Теорема 1. Граф является деревом тогда и только тогда, когда любые две его вершины связаны единственной цепью.

Доказательство. Пусть граф является деревом. Если допустить существование более одной цепи, связывающей любые две его вершины, то в таком графе существует цикл, то есть граф не может быть деревом.

Наоборот, поскольку любые две вершины графа соединены цепью, то граф связный, а в силу того, что эта цепь единственная, он не имеет циклов. Значит, граф является деревом. Теорема доказана.



Следствие 1. Если T – дерево и u – его концевая вершина, то граф $T-u$ *(T минус u)* – дерево.
Действительно, граф $T-u$ – подграф дерева T, для которого выполняются все условия теоремы.

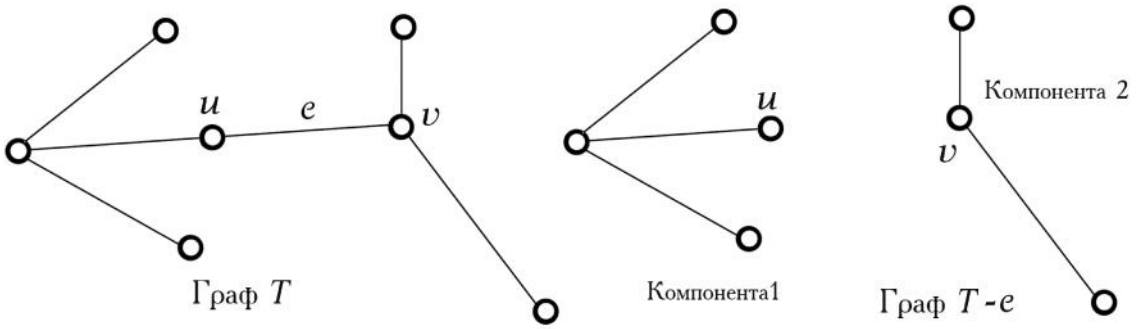


Следствие 2. Всякое непустое дерево имеет по крайней мере две висячие вершины и одно висячее ребро.
1. **Висячая вершина в неориентированном граф**е – это вершина степени 1.
2. **Висячая вершина в орграфе** – вершина с полустепенью захода, равной 1, и полустепенью исхода, равной 0.
3. **Висячее ребро** – это ребро, инцидентное вершине со степенью 1.

Определение. Ребро связного графа называется *существенным*, если его удаление ведет к нарушению связности этого графа.

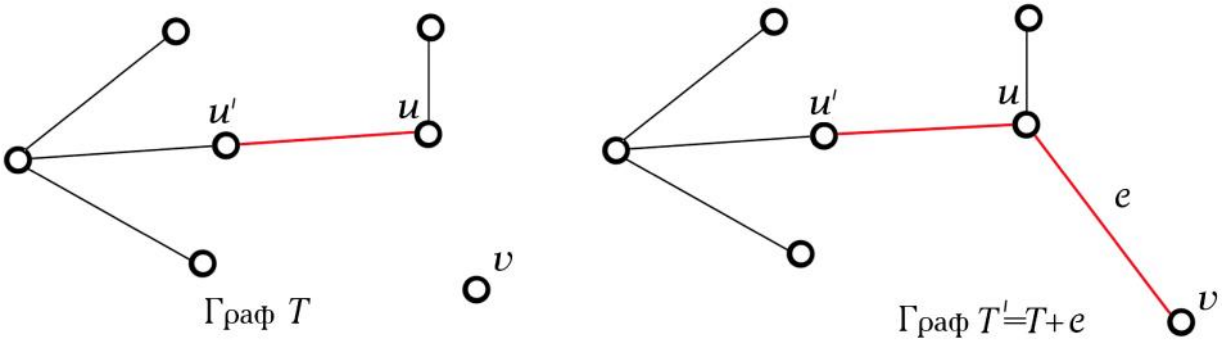
Следствие. В неориентированном графе существенным ребром является мост.
Теорема. В дереве каждое ребро существенное.
Доказательство. Доказательство вытекает из того, что удаление ребра $e=(u,v)$ в дереве T в силу единственности цепи, соединяющей вершины u и v, ведет к появлению двух компонент связности: одна содержит вершину u и другая – вершину v.

Следовательно, граф $T-e$ не является связным.



Теорема 2. *Если* $T=(V,E)$ *– дерево и вершина* $v\notin V$, *то граф* $T'=\left(V\cup\{v\},E\cup\{(u,v)\}\right)$, *где* u *– произвольная вершина из* V, *тоже является деревом.*

Доказательство. Поскольку T – дерево, то существует единственная цепь, соединяющая любую вершину u' с вершиной u. Поскольку вершина $v\notin V$, то добавление одного концевого ребра (u,v) приводит к тому, что из каждой вершины u' имеем лишь единственную цепь, которая соединяет вершины u' и v. В силу теоремы 1 граф T' является деревом.



Теорема 3. *Пусть граф* T *имеет* n *вершин. Тогда эквивалентными являются такие утверждения:*

1. Т является деревом.
2. Т не имеет циклов и имеет $n-1$ ребро.
3. Т – связный граф и имеет $n-1$ ребро.
4. Т – связный граф и каждое его ребро является мостом.
5. Любые две вершины графа T соединены ровно одной простой цепью.
6. T не имеет циклов, но добавление любого нового ребра в T способствует возникновению ровно одного цикла.

Определение. Лес – несвязный n-граф без циклов;

1. Связные компоненты леса являются деревьями.
2. Любая часть леса или дерева также не имеет циклов.
3. Любая часть леса является лесом или деревом.

Теорема. Пусть лес G содержит n вершин и k компонент. Тогда лес G имеет $n-k$ ребер.

Доказательство. Из условия 2 теоремы 3 вытекает, что каждая компонента G_i имеет (n_i-1) ребро. Но тогда число ребер в G равно

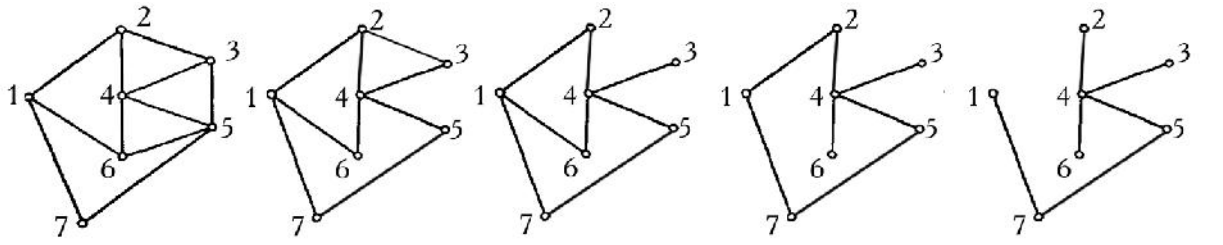
$$(n_1-1)+(n_2-1)+...+(n_k-1)=n_1+n_2+...+n_k-k=n-k,$$

что и следовало доказать.

Теорема Кэли. Число разных деревьев, которые можно построить на n вершинах, равно n^{n-2}.

Процедура построения остовного дерева

1. Удаление из связного графа G одного ребра, принадлежащего некоторому циклу, не нарушает связности графа G.
2. Применим процедуру удаления ребра к одному из циклов в графе G.
3. Будем повторять удаление до тех пор, пока в G не останется ни одного цикла.
4. В результате получим дерево, содержащее все вершины графа G. Это дерево называется *остовным деревом графа* G.



T $T1=T-(3,5)$ $T2=T1-(2,3)$ $T3=T2-(1,6)$ $T4=T3-(1,2)$

Процедура построения остовного леса

Пусть G представляет собой граф с n вершинами, m ребрами и k компонентами.

1. Применим процедуру удаления ребра к одному из циклов в каждой компоненте связности графа G.

2. Будем повторять удаление до тех пор, пока в каждой компоненте G не останется ни одного цикла.

3. В результате получим граф, который называется *остовным лесом.*

4. Число ребер, которые при этом удаляются, называется *цикломатическим числом* или *циклическим рангом графа* G и обозначается $C(G)$. Таким образом, цикломатическое число является мерой связности графа.

Свойства циклического ранга

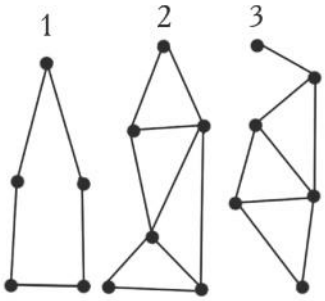
1. **Циклический ранг дерева** равен нулю.
2. **Циклический ранг циклического графа** равен единице.

Циклический граф – связный регулярный граф степени 2, единственная компонента связности циклического графа является простым циклом.

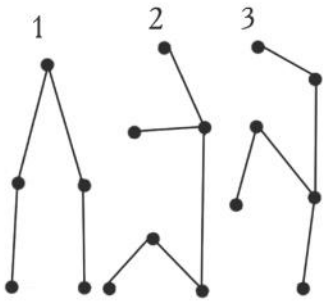
Остовное дерево графа – это дерево, содержащее все вершины графа.

Остовным лесом называется несвязный граф, состоящий из компонент, являющихся остовными деревьями**.**

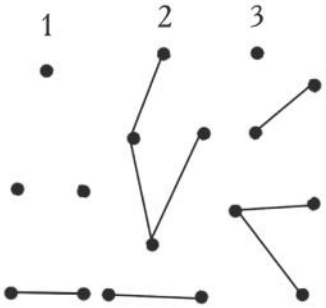
Теорема. Пусть T – остовной лес графа G . Граф G', полученный из графа G путем удаления всех ребер графа T, называется *дополнением* остовного леса T графа G.



Несвязный граф
G



Остовной лес
T

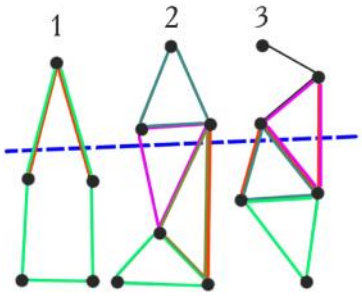


Дополнение
остовного леса G'

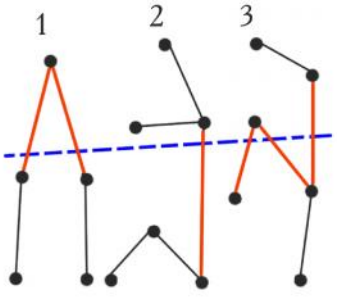
Теорема 4. Если T – остовной лес графа G , то

а) всякое сечение в G имеет общее ребро с T ;

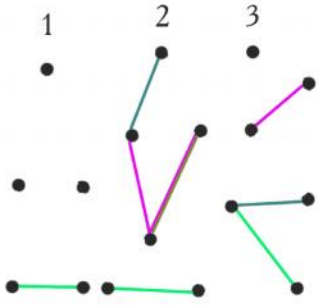
б) всякий цикл в G имеет общее ребро с дополнением T **.**



Несвязный граф
G



Остовной лес
T



Дополнение
остовного леса G'

Фундаментальная система циклов графа

Пусть T – остовной лес графа G.

Если добавить к T любое не содержащееся в нем ребро графа G, то по предложению п. 6 теоремы 3 получим единственный цикл.

Определение. Множество всех циклов, получаемых путем добавления по отдельности каждого ребра из G, не содержащегося в T, называется фундаментальной системой циклов, ассоциированной с Т.

Циклы данной фундаментальной системы будут различными, и их число равняется циклическому рангу графа G.

На рисунке показаны графы, демонстрирующие фундаментальную систему циклов.

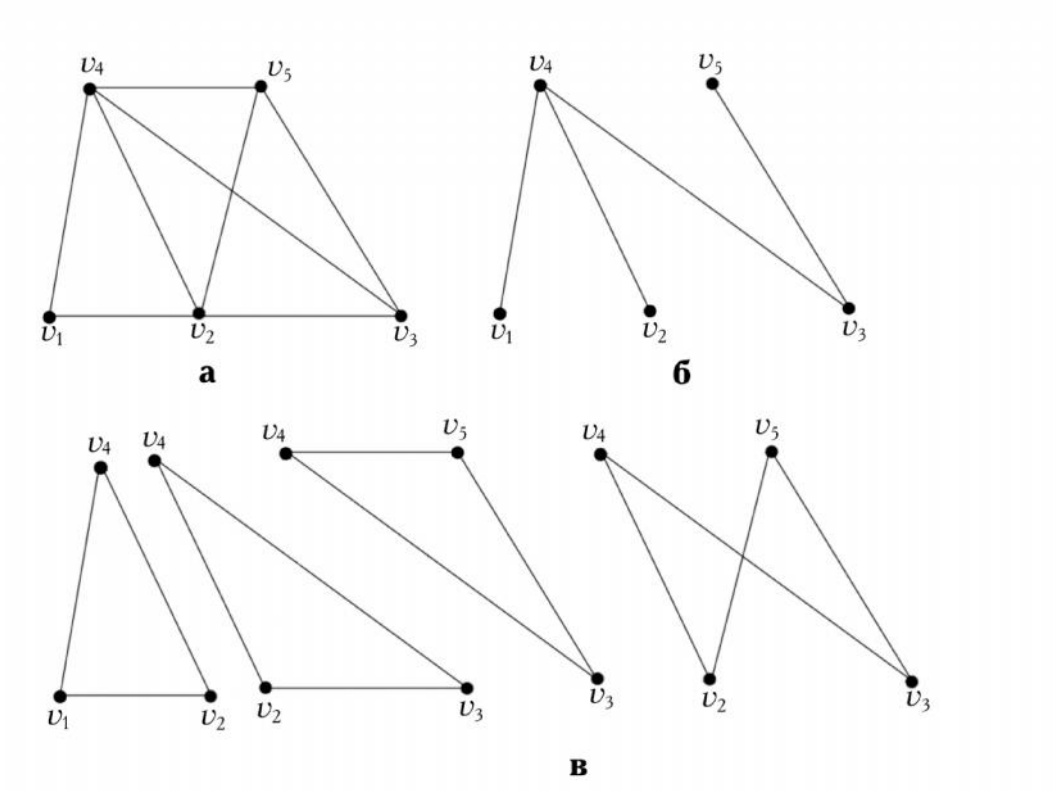


Рисунок: а – граф G; **б** – остовное дерево графа G; **в** – фундаментальная система циклов, ассоциированная с остовным деревом графа G.

Теорема 5. Пусть $G=(V,E)$ – произвольный конечный граф. Число ребер графа *G*, которые необходимо удалить для получения остовного леса *T*, не зависит от порядка их удаления и равно $C(G)=|E|-|V|+k$, где k – число компонентов связности графа *G*.

Следствие. Граф G представляет собой остовной лес тогда и только тогда, когда $C(G)=0$.

Следствие. Граф G имеет единственный цикл тогда и только тогда, когда $C(G)=1$.

Следствие. Граф G, у которого число ребер превышает число вершин, имеет цикл.

Следствие. Всякое дерево порядка $n \geq 2$ имеет по крайней мере две концевые вершины.

Остов наименьшего веса

Постановка задачи

Пусть каждому ребру графа $G=(V,E)$ поставлено в соответствие некоторое число

$$d_i = d(e_i), \quad e_i \in E, \quad i = 1,2,...,|E|.$$

Необходимо в графе G найти остовное дерево, сумма чисел d_i в котором наименьшая.

$$S = \min \sum_{e_i \in E} \left(d(e_i) \right)$$

Число d_i в этом случае называется *весом ребра* e_i, а сам граф G – таким, что имеет *вес*.

Следовательно, задача состоит в том, чтобы найти остовное дерево с наименьшим весом.

Практические приложения задачи поиска остовного дерева с минимальным весом

Эта задача возникает при проектировании дорог, линий электропередач, трубопроводов и пр., когда необходимо соединить заданные точки некоторой системой связи так, чтобы общая длина линий связи была минимальной.

Рассмотрим несколько методов решения этой задачи.

Решение, следующее из теоремы Кэли

1. Рассмотрим все возможные остовные деревья полного графа G с n вершинами.

Согласно теореме Кэли ч*исло разных деревьев, которые можно построить на* n *вершинах, равно* n^{n-2}.

2. При выборе каждого дерева определим сумму его весов.

3. Применив сортировку по величине суммы весов, в начале списка получим те остовные деревья, которые имеют минимальную сумму весов.

Но поскольку число n^{n-2} *даже при небольшом* n *будет большим, то такой путь решения данной задачи является достаточно трудоемким.*

Поэтому актуальным является использование более эффективных алгоритмов.

Предварительные замечания

1. **Порядок графа** – число, соответствующее количеству вершин графа.
2. **Пустой граф** – граф, не содержащий ребер или регулярный граф степени 0.

Алгоритм Краскала

Алгоритм Краскала состоит в выполнении такой последовательности действий:

1. Берем пустой граф O и строим граф $T_1 = O + e_1$,

где e_1 – ребро графа $G = (V, E)$ минимального веса.

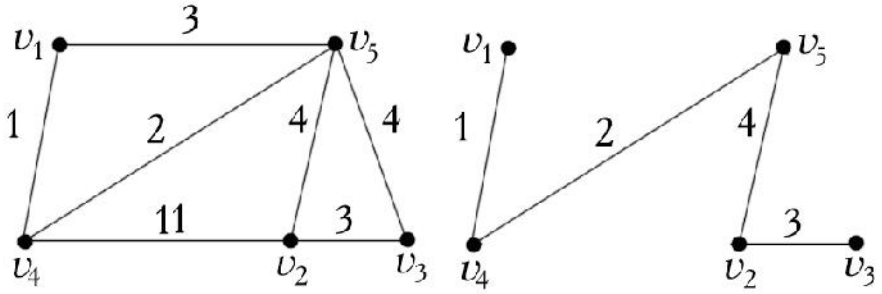
2. Если граф T_k уже построен и $k < n-1$, то строим граф $T_{k+1} = T_k + e_{k+1}$, где e_{k+1} – ребро минимального веса среди ребер графа G, не вошедших в граф T_k, которое не составляет цикла с ребрами графа T_k.

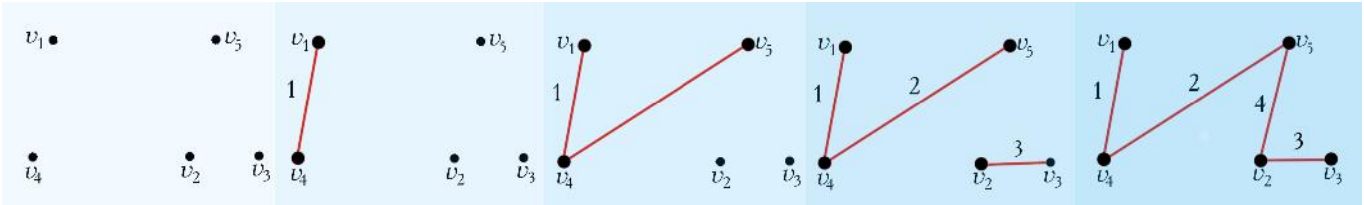
Этот алгоритм базируется на специальной теореме.

Теорема 7. Пусть G – связный граф порядка n и T – подграф графа G, полученный в результате выполнения шагов 1 и 2 алгоритма Краскала. Тогда подграф T – остовное дерево графа G минимального веса.

Пример. Пусть дан граф G, показанный на рисунке слева. Необходимо найти остовное дерево минимального веса этого графа с помощью алгоритма Краскала.

Решение.





Свойства алгоритма Краскала

1. Алгоритм используется для взвешенных неориентированных графов.

2. Алгоритм может быть использован для построения остовного леса в многокомпонентных несвязных графах.

В этом случае **структуры данных**, описывающие каждую из компонент связности, **должны быть отдельными**.

3. Вычислительная сложность алгоритма Краскала $O(e \cdot \log e)$, где e – количество ребер в данном графе.

Алгоритм Прима

Алгоритм Прима состоит в выполнении такой последовательности действий:

1. В пустом графе O выбираем произвольную **вершину**.

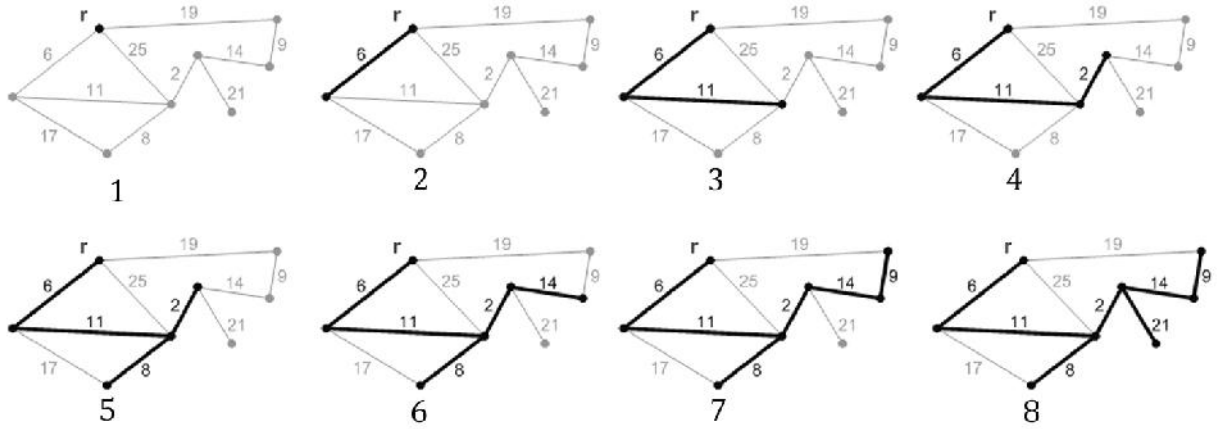
2. Выбираем ребро e_1 минимального веса к смежной вершине и строим подграф $T_1 = O + e_1,$.

3. Если граф T_k уже построен и $k < n-1$, то строим граф $T_{k+1} = T_k + e_{k+1}$, где e_{k+1} – ребро минимального веса, соединяющее концевую вершину графа T_k со смежной вершиной, не включенной в множество вершин графа T_k.

Пример. Пусть дан граф G. Необходимо найти остовное дерево минимального веса этого графа с помощью алгоритма Прима.

Решение. 1. Выбираем произвольную вершину r и проводим инцидентное ребро минимального веса.

2. Затем, просматривая инцидентные ребра на каждой из концевых вершин, находим ребро минимального веса.



Свойства алгоритма Прима

1. Алгоритм используется для взвешенных неориентированных связных графов.

2. Вычислительная сложность алгоритма Прима $O\left(n^2\right)$, где n – количество вершин графа. Если значение n достаточно большое, то использование этого алгоритма не рационально.

3. Если e значительно меньше n^2, то алгоритм Краскала предпочтительнее, но если e близко к n^2, рекомендуется применять алгоритм Прима.

Структура алгоритма Прима

Дан граф $G(V,E)$, где множество вершин $V=\{1,2,...,i,...,n\}$
$U=\varnothing$ – множество вершин остовного дерева.
$T=\varnothing$ – множество ребер остовного дерева.

procedure Prim (*G*: граф; ***var*** *T*: множество ребер;
var *U*: множество вершин;
u ,*v*: вершина;
begin
T := ∅; *U* := {i};
while *U* ≠ *V* ***do***
begin
Нахождение ребра (*u* ,*v*) наименьшей стоимости и такого, что *u* ∈*U* и *v*∈*V**U*
T:= *T* ∪ {(*u* ,*v*)};
U:= *U* ∪ {*v*}
end
end.

Обход графов

При решении многих задач с использованием графов необходимо уметь обходить все вершины и ребра (дуги) графа. Обойти граф — это побывать во всех вершинах точно по одному разу. Существует два классических алгоритма обхода графов: обход графа в глубину и обход графа в ширину.

Работа всякого алгоритма обхода состоит в последовательном посещении вершин и исследовании ребер. Какие именно действия выполняются при посещении вершины и исследовании ребра – зависит от конкретной задачи, для решения которой производится обход. В любом случае факт посещения вершины запоминается, так что с момента посещения и до конца работы алгоритма она считается посещенной.

Вершину, которая ещё не посещена, называют новой. В результате посещения вершина становится открытой и остается такой, пока не будут исследованы все инцидентные ей ребра. После этого она превращается в закрытую.

Поиск в глубину

Обход в глубину – это обход графа по следующим правилам:

(1) находясь в вершине x, нужно двигаться в любую другую, ранее не посещенную вершину (если таковая найдется), одновременно запоминая дугу, по которой мы впервые попали в данную вершину;

(2) если из вершины x мы не можем попасть в ранее не посещенную вершину или таковой нет, то мы возвращаемся в вершину z, из которой впервые попали в x, и продолжаем обход в глубину из вершины z.

При выполнении обхода графа по этим правилам мы стремимся проникнуть "вглубь" графа так далеко, как только это возможно, затем отступаем на шаг назад и снова стремимся пройти вперед, и так далее.

Таким образом, допустим, что мы находимся в вершине графа v. Далее, выбираем произвольную, но еще не просмотренную вершину u, смежную с вершиной v. Эту новую вершину рассматриваем теперь уже как v и, начиная с нее, продолжаем обход. Если не существует ни одной новой (еще не просмотренной) вершины, смежной с v, то говорят, что вершина v просмотрена, и возвращаются в вершину, из которой мы попали в v.

Главное отличие от поиска в ширину состоит в том, что в качестве активной выбирается та из открытых вершин, которая была посещена последней. При обходе в глубину чем позднее будет посещена вершина, тем раньше она будет использована.

Поиск в глубину ищет любой возможный путь – то есть первый попавшийся.

Поиск в ширину

Основная особенность поиска в ширину, отличающая его от других способов обхода графов, состоит в том, что в качестве активной вершины выбирается та из открытых, которая была посещена раньше других. Именно этим обеспечивается главное свойство поиска в ширину: чем ближе вершина к старту, тем раньше она будет посещена.

Использование вершины происходит с помощью просмотра сразу всех еще не просмотренных вершин, смежных этой вершине. Таким образом, поиск ведется во всех возможных направлениях одновременно.

Сначала посещается сама вершина a, затем все вершины, смежные с a, находящиеся от нее на расстоянии 1, затем вершины, находящиеся от a на

расстоянии 2, и т. д. Чем ближе вершина к стартовой вершине, тем раньше она будет посещена. Поиск в ширину ищет наиболее короткий возможный путь.

Программа, описывающая обход графов в глубину (Pascal).

```
Program graff;
Var n, v, u: integer;
       fr: text;
       gr: array[1..30, 1..30] of integer;
       nov: array[1..15] of boolean;

  procedure dfs (v: integer);
  Var u: integer;
  begin
    readln;
    write (v, ‘ ‘);
    nov [v]:=false;
    for u:=1 to n do
    if (gr[v,u]>0) and (nov[u]) then dfs (u);
  end;

begin
    n:=3; (*Задаем количество вершин графа*)
   for v:=1 to n do
   begin
      nov [v]:=true;
      writeln;
      for u:=1 to n do
      begin
        nov[u]:=true;
         (*Ввод инциденции для вершин u и v*)
        write (‘ gr[‘,v,u,’]=’);
        read (gr [v, u] );
      end;
  end;
  for v:=1 to n do
  begin
     (*Вызов процедуры обхода*)
     if nov[v] then dfs (v);
  end;
  readln;
end.
```

Поиск путей в графе

Алгоритм Терри

Исходя из начальной вершины и осуществляя последовательный переход от каждой достигнутой вершины к смежной вершине, следовать следующим правилам.

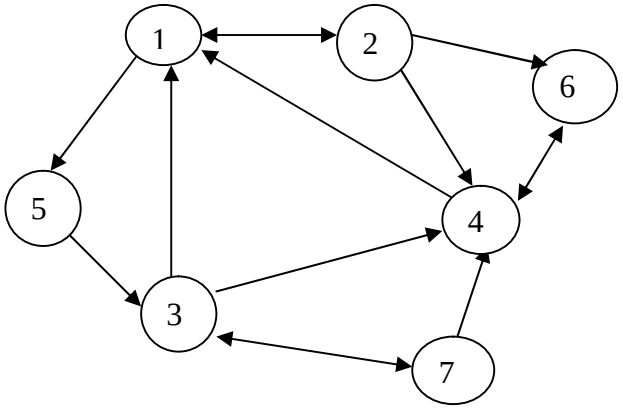
1. Отмечать направление, в котором проходим ребро, как в прямом, так и в обратном направлениях.
2. Исходя из любой вершины, всегда следовать только по тому ребру, которое еще не было пройдено или было пройдено в противоположном направлении.
3. Для всякой вершины отмечать первое заходящее ребро, если вершина встречается впервые.
4. Из всякой вершины идти по ребру, по которому пришли в эту вершину (в обратном направлении) лишь тогда, когда нет другой возможности.

Описание последовательности действий в алгоритме Терри

1. Составить матрицу смежности MG[i,j].
2. Задать начальную и конечную вершины vS и vF.
3. Задать две переменные:
4. row – текущая строка MG[i,j];
5. column – текущий столбец MG[i,j].
6. row:= vS; column := 1.
7. Если (i,j) элемент матрицы MG[i,j] единичный, то обнуляем этот элемент и элемент (j,i) (если он единичный).
8. Заносим номер i-ой строки в стек. Переходим в строку с номером j.
9. Если (i,j) элемент нулевой, то переходим к анализу следующего столбца i-ой строки.
10. Если оказалось, что все элементы анализируемой строки нулевые, то достаем из стека номер вершины и переходим в строку с этим номером.
11. Если мы переходим из одной строки в другую, то номер столбца делаем единичным.
12. Анализ матрицы заканчивается, когда попадаем в строку с vF -м номером.
13. Содержимое стека – маршрут из vS в vF.

Решение задачи вручную

Зададимся графом.
Ниже приведена схема графа.



ЗАДАНИЕ:

v6 – начальная вершина;
v7 – конечная вершина;

РЕШЕНИЕ:

0)

	1	2	3	4	5	6	7
1		1			1		
2	1			1		1	
3	1			1			1
4	1					1	
5			1				
6				1			
7			1	1			

1) Переходим к строке 6 столбцу 1.
   Элемент (6,1) нулевой, переходим к (6,2) элементу.
2) Элемент (6,2) нулевой, переходим к (6,3).
3) Элемент (6,3) нулевой, переходим к (6,4).
4) Элемент (6,4) единичный. Заполняем стек: {6}. Переписываем матрицу.

	1	2	3	4	5	6	7
1		1			1		
2	1			1		1	
3	1			1			1
4	1					0	
5			1				
6				0			
7			1	1			

5) Переходим к анализу строки 4 и столбца 1. Содержимое стека: {6}.
6) Элемент (4,1) единичный. Заполняем стек: {6,4}. Переписываем матрицу.

	1	2	3	4	5	6	7
1		1			1		
2	1			1		1	
3	1			1			1
4	0					0	
5			1				
6				0			
7			1	1			

7) Переходим к анализу строки 1 и столбца 1. Содержимое стека: {6,4}. Элемент (1,1) нулевой, переходим к элементу (1,2).
8) Элемент (1,2) единичный. Заполняем стек: {6,4,1}. Переписываем матрицу.

	1	2	3	4	5	6	7
1		0			1		
2	0			1		1	
3	1			1			1
4	0					0	
5			1				
6				0			
7			1	1			

9) Переходим к анализу строки 2 и столбца 1. Содержимое стека: {6,4,1}. Элемент (2,1) нулевой, переходим к элементу (2,2).
10) Элемент (2,2) нулевой, переходим к элементу (2,3).
11) Элемент (2,3) нулевой, переходим к элементу (2,4).
12) Элемент (2,4) единичный. Заполняем стек: {6,4,1,2}. Переписываем матрицу.

	1	2	3	4	5	6	7
1		0			1		
2	0			0		1	
3	1			1			1
4	0					0	
5			1				
6				0			
7			1	1			

13) Переходим к анализу строки 4 . Содержимое стека: {6,4,1,2}.

Строка 4 нулевая, вынимаем из стека элемент. Содержимое стека: {6,4,1}.

14) Переходим к анализу строки 2 и столбца 1 . Содержимое стека: {6,4,1}. Элемент (2,1) нулевой, переходим к элементу (2,2).
15) Элемент (2,2) нулевой, переходим к элементу (2,3).
16) Элемент (2,3) нулевой, переходим к элементу (2,4).
17) Элемент (2,4) нулевой, переходим к элементу (2,5).
18) Элемент (2,5) нулевой, переходим к элементу (2,6).
19) Элемент (2,6) единичный. Заполняем стек: {6,4,1,2}. Переписываем матрицу.

	1	2	3	4	5	6	7
1		0			1		
2	0			0		0	
3	1			1			1
4	0					0	
5			1				
6				0			
7			1	1			

20) Переходим к анализу строки 6. Содержимое стека: {6,4,1,2}.
Строка 6 нулевая, вынимаем из стека элемент. Содержимое стека: {6,4,1}.
21) Переходим к анализу строки 2 Содержимое стека: {6,4,1}.
Строка 2 нулевая, вынимаем из стека элемент. Содержимое стека: {6,4}.
22) Переходим к анализу строки 1 и столбца 1. Содержимое стека: {6,4}. Элемент (1,1) нулевой, переходим к элементу (1,2).
23) Элемент (1,2) нулевой, переходим к элементу (1,3).
24) Элемент (1,3) нулевой, переходим к элементу (1,4).
25) Элемент (1,4) нулевой, переходим к элементу (1,5).
26) Элемент (1,5) единичный. Заполняем стек: {6,4,1}. Переписываем матрицу.

	1	2	3	4	5	6	7
1		0			0		
2	0			0		0	
3	1			1			1
4	0					0	
5			1				
6				0			
7			1	1			

27) Переходим к анализу строки 5 и столбца 1. Содержимое стека: {6,4,1,2}.

Элемент (5,1) нулевой, переходим к элементу (5,2).

28) Элемент (5,2) нулевой, переходим к элементу (5,3).

29) Элемент (5,3) единичный. Заполняем стек: {6,4,1,5}. Переписываем матрицу.

	1	2	3	4	5	6	7
1		0			0		
2	0			0		0	
3	1			1			1
4	0					0	
5			0				
6				0			
7			1	1			

30) Переходим к анализу строки 3 и столбца 1. Содержимое стека: {6,4,1,5}.

Элемент (3,1) единичный. Заполняем стек: {6,4,1,5,3}. Переписываем матрицу.

	1	2	3	4	5	6	7
1		0			0		
2	0			0		0	
3	0			1			1
4	0					0	
5			0				
6				0			
7			1	1			

31) Переходим к анализу строки 1. Содержимое стека: {6,4,1,5,3}.

Строка 1 нулевая, вынимаем из стека элемент. Содержимое стека: {6,4,1,5}.

32) Переходим к анализу строки 3 и столбца 1. Содержимое стека: {6,4,1,5}.

Элемент (3,1) нулевой, переходим к элементу (3,2).

33) Элемент (3,2) нулевой, переходим к элементу (3,3).

34) Элемент (3,3) нулевой, переходим к элементу (3,4)

35) Элемент (3,4) единичный. Заполняем стек: {6,4,1,5,3}. Переписываем матрицу.

	1	2	3	4	5	6	7
1		0			0		
2	0			0		0	
3	0			0			1
4	0					0	
5			0				
6				0			
7			1	1			

35) Переходим к анализу строки 4. Содержимое стека: {6,4,1,5,3}.
Строка 4 нулевая, вынимаем из стека элемент. Содержимое стека: {6,4,1,5}.
36) Переходим к анализу строки 3 и столбца 1. Содержимое стека: {6,4,1,5}.
Элемент (3,1) нулевой, переходим к элементу (3,2).
37) Элемент (3,2) нулевой, переходим к элементу (3,3).
38) Элемент (3,3) нулевой, переходим к элементу (3,4).
39) Элемент (3,4) нулевой, переходим к элементу (3,5).
40) Элемент (3,5) нулевой, переходим к элементу (3,6).
41) Элемент (3,6) нулевой, переходим к элементу (3,7).
42) Элемент (3,7) единичный. Заполняем стек: {6,4,1,5,3}.

Ответ: {6,4,1,5,3,7}.

```
program Terri;
uses crt;
const m=100;   // Максимальный размер массива
var
  mg:array[1..m,1..m] of byte;    // Матрица смежности
  stack:array[1..m-1] of word;    // Стек
  i,j:word;
  column,row:word;
  sp:word;   // номер последнего элемента в стеке
  vs,vf:word;
  n: word;   // Реальный размер матрицы смежности
begin
  clrscr;
  write('Input size matrix: ');
  read(n);
  clrscr;
  for i:=1 to n do
  for j:=1 to n do
  if i=j then mg[i,j]:=0 else // Диагональным элементам присваивается ноль
  begin
    write('mg(',i,',',j,') = ');
    read(mg[i,j]);
    clrscr;
  end;
  write('Input Start: ');       // Вводим начальную вершину
  read(vs);
  write('Input Finish: ');   // Вводим конечную вершину
  read(vf);
  row:=vs;                    // Начальное значение строки
```

```
 column:=1;
 sp:=0;                    // Стек пустой
 while row<>vf do
 if mg[row,column]=1 then   // Если элемент единичный,
 begin
  mg[row,column]:=0;     // то заменить его,
  if mg[column,row]=1 then   // а, если надо, то и симметричный ему
  mg[column,row]:=0;
  sp:=sp+1;        // указатель на вершину стека сместить
  stack[sp]:=row;  // положить в стек номер строки
  row:=column;   // строка принимает значение столбца
  column:=1;   // столбец сбрасывается
 end else
 if column=n then   // Если все элементы нулевые
 begin
  row:=stack[sp];   // вытащить из стека
  sp:=sp-1;   // сместить указатель
  column:=1;   // сбросить столбец
 end else
 column:=column+1;   // перейти к другому столбцу
 sp:=sp+1;
 stack[sp]:=row;    //  положить в стек конечный элемент
 writeln;
 write('Path : ');
 for i:=1 to sp do write(' ',stack[i]);   // Вывести на экран содержимое стека
 readln;
end.
```

Волновой алгоритм

Дано: непустой граф $G = (V, E)$. Требуется найти путь между вершинами s и t графа (s не совпадает с t), содержащий минимальное количество промежуточных вершин (ребер).

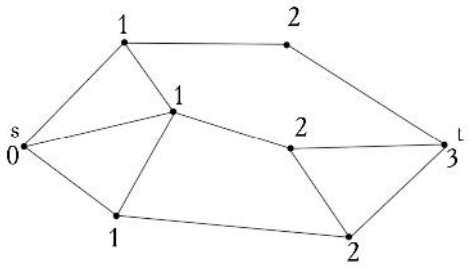
Алгоритм состоит в следующем:

1. Каждой вершине v_i приписывается целое число $T(v_i)$ – волновая метка (начальное значение $T(v_i) = -1$).
2. Заводятся два списка OldFront и NewFront (старый и новый "фронт волны"), а также переменная T (текущее время);
3. OldFront:={s}; NewFront:={}; T(s):=0; T:=0;
4. Для каждой из вершин, входящих в OldFront, просматриваются инцидентные (смежные) ей вершины u_j, и если $T(u_j) = -1$, то $T(u_j):=T+1$, NewFront:=NewFront + $\{u_j\}$;

5. Если NewFront = {}, то ВЫХОД("нет решения");
6. Если t – NewFront (т. е. одна из вершин u_j совпадает t), то найден кратчайший путь между s и t с T(t)=T+1 промежуточными ребрами; ВЫХОД ("решение найдено");
7. OldFront:=NewFront; NewFront:={}; T:=T+1; goto (4).

Замечание: на шаге (4) "соседними" вершинами для неориентированных графов считаются все смежные вершины, а для орграфов – вершины, в которые из данной вершины ведут дуги.

Если на шаге (6) была достигнута вершина t, то восстановить кратчайший путь можно следующим образом: среди соседей вершины t найдем любую вершину с волновой меткой $T\left(t\right)-1$, среди соседей последней – вершину с меткой $T\left(t\right)-2$, и т. д., пока не достигнем s. Найденная последовательность вершин определяет один из кратчайших путей из s в t. На практике выгодно сохранять на шаге (4) информацию о том, из какой вершины "волна" пришла в вершину u_j – тогда восстановление пути осуществляется быстрее.



На рисунке показана нумерация вершин, полученная в результате работы волнового алгоритма.

Поиск кратчайшего пути

Пусть дан граф G**,** дугам которого приписаны веса (стоимости), задаваемые матрицей $C=\left\|c_{i,j}\right\|$. *Задача о кратчайшем пути* состоит в нахождении кратчайшего пути от заданной начальной вершины $s\in V$ до заданной конечной вершины $t\in V$ при условии, что такой путь существует. Ниже приводится простой и эффективный алгоритм решения этой задачи для случая $c_{ij}\geq 0$, известный как алгоритм Дейкстры. Этот алгоритм основан на приписывании вершинам временных пометок, причем пометка вершины дает верхнюю границу длины пути от *s* к этой вершине. Величины этих пометок постепенно уменьшаются при помощи итерационной процедуры, и на каждом шаге итерации одна из временных пометок становится постоянной. Величина этой пометки является точной длиной кратчайшего пути от *s* к рассматриваемой вершине.

Алгоритм Дейкстры

Пусть $l(v_i)$ – пометка вершины v_i

Присвоение начальных значений

Шаг 1. Положить $l(s) = 0$ и считать эту пометку постоянной. Положить $l(v_i) = \infty$ для всех $v_i \neq s$ и считать эти пометки временными. Положить $p = s$.

Обновление пометок

Шаг 2. Для всех $v_i \in \Gamma(p)$, пометки которых временные, изменить пометки в соответствии со следующем выражением:

$l(v_i) \leftarrow \min[l(v_i), l(p) + c(p, v_i)]$

Превращение пометки в постоянную

Шаг 3. Среди вершин с временными пометками найти такую, для которой

$l(v_i^*) = \min l(v_i),\ v_i \in \Gamma(p)$

Шаг 4. Считать пометку $l(v_i^*)$ постоянной и положить $p = v_i^*$.

Шаг 5. Если надо найти путь от s к t. Если $p = t$, то $l(p)$ является длиной кратчайшего пути от вершины s к вершине t. Останов.

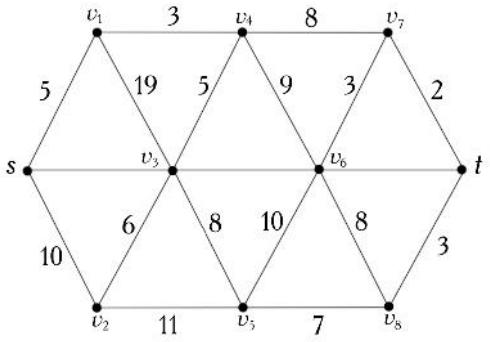
Если $p \neq t$, перейти к шагу 2.

В случае, когда требуется найти пути от s ко всем остальным вершинам. Если все вершины отмечены как постоянные, то эти пометки дают длины кратчайших путей. Останов.

Если некоторые пометки являются переменными, перейти к шагу 2.

Пример ручного решения задачи

Рассмотрим граф, изображенный на рисунке, где каждое неориентированное ребро рассматривается как пара противоположно ориентированных дуг равного веса. Этот вес выписан на рисунке, изображающем граф, рядом с ребром.



Требуется найти кратчайшие пути от вершины s ко всем остальным вершинам. Для этого используем алгоритм Дейкстры. Постоянные пометки отмечаются знаком +, остальные пометки являются временными.

Схема применения алгоритма следующая.

Шаг 1. $l\left(s\right)=0^{+}, l\left(v_i\right)=\infty,\ i=1,...,8,\ p=s.$

Первая итерация.

Шаг 2. $\Gamma\left(s\right)=\left\{v_1,v_2,v_3\right\}-$ все пометки временные.

$l\left(v_1\right)=\min\left[\infty,0^{+}+5\right]=5,$

$l\left(v_2\right)=\min\left[\infty,0^{+}+14\right]=14,$

$l\left(v_3\right)=\min\left[\infty,0^{+}+10\right]=10.$

Шаг 3. $l\left(v_1\right)=\min\limits_{i=1,2,3} l\left(v_i\right)=5.$

Шаг 4. $l\left(v_1\right)=5^{+}-v_1$ получает постоянную пометку; $p=v_1$.

Шаг 5. Не все вершины имеют постоянные пометки, поэтому переходим к шагу 2.

Вторая итерация.

Шаг 2. $\Gamma\left(p\right)=\Gamma\left(v_1\right)=\left\{s,v_3,v_4\right\}$. Вершина s имеет постоянную пометку;

$l\left(v_3\right)=\min\left[10,5^{+}+19\right]=10,$

$l\left(v_4\right)=\min\left[\infty,5^{+}+3\right]=8.$

Шаг 3. $l\left(v_4\right)=\min\limits_{i=3,4} l\left(v_i\right).$

Шаг 4. Вершина v_4 получает постоянную метку: $l\left(v_4\right)=8^{+};\ p=v_4$.

Шаг 5. Не все вершины имеют постоянные пометки, поэтому переходим к шагу 2.

Третья итерация.

Шаг 2. $\Gamma\left(p\right)=\Gamma\left(v_4\right)=\left\{v_1,v_3,v_6,v_7\right\}$. Вершина v_1 имеет постоянную пометку;

$l\left(v_3\right)=\min\left[10,8^{+}+5\right]=10,$

$l\left(v_7\right)=\min\left[\infty,8^{+}+8\right]=16,$

$l\left(v_6\right)=\min\left[\infty,8^{+}+9\right]=17.$

Шаг 3. $l\left(v_3\right)=\min\limits_{i=3,6,7} l\left(v_i\right)=10.$

Шаг 4. Вершина v_3 получает постоянную метку: $l\left(v_3\right)=10^{+};\ p=v_3$.

Шаг 5. Не все вершины имеют постоянные пометки, поэтому переходим к шагу 2.

Четвертая итерация.

Шаг 2. $\Gamma\left(p\right)=\Gamma\left(v_3\right)=\left\{s,v_1,v_2,v_4,v_5,v_6\right\}$. Вершины s,v_1,v_4 имеют постоянные пометки;

$l\left(v_2\right)=\min\left[14,10^{+}+6\right]=14,$

$l\left(v_5\right)=\min\left[\infty,10^{+}+8\right]=18,$

$l\left(v_6\right)=\min\left[17,10^{+}+8\right]=17.$

Шаг 3. $l\left(v_2\right)=\min\limits_{i=2,5,6} l\left(v_i\right)=14.$

Шаг 4. Вершина v_2 получает постоянную метку: $l\left(v_2\right)=14^{+};\ p=v_2.$

Шаг 5. Не все вершины имеют постоянные пометки, поэтому переходим к шагу 2.

Пятая итерация.

Шаг 2. $\Gamma\left(p\right)=\Gamma\left(v_2\right)=\left\{s,v_3,v_5\right\}.$ Вершины s,v_3 имеет постоянные пометки;

$l\left(v_5\right)=\min\left[18,14^{+}+11\right]=18.$

Шаг 3. $l\left(v_5\right)=\min\limits_{i=5} l\left(v_i\right)=18.$

Шаг 4. Вершина v_5 получает постоянную метку: $l\left(v_5\right)=18^{+};\ p=v_5.$

Шаг 5. Не все вершины имеют постоянные пометки, поэтому переходим к шагу 2.

Шестая итерация.

Шаг 2. $\Gamma\left(p\right)=\Gamma\left(v_5\right)=\left\{v_2,v_3,v_6,v_8\right\}.$ Вершины v_2,v_3 имеет постоянные пометки;

$l\left(v_6\right)=\min\left[17,18^{+}+10\right]=17,$

$l\left(v_8\right)=\min\left[\infty,18^{+}+7\right]=25.$

Шаг 3. $l\left(v_6\right)=\min\limits_{i=6,8} l\left(v_i\right)=17.$

Шаг 4. Вершина v_6 получает постоянную метку: $l\left(v_6\right)=17^{+};\ p=v_6.$

Шаг 5. Не все вершины имеют постоянные пометки, поэтому переходим к шагу 2.

Седьмая итерация.

Шаг 2. $\Gamma\left(p\right)=\Gamma\left(v_6\right)=\left\{v_3,v_4,v_4,v_7,v_8,t\right\}.$ Вершины v_3,v_4,v_5 имеют постоянные пометки;

$l\left(v_7\right)=\min\left[16,17^{+}+3\right]=16,$

$l\left(v_6\right)=\min\left[25,17^{+}+8\right]=25,$

$l\left(t\right)=\min\left[\infty,17^{+}+14\right]=31.$

Шаг 3. $l\left(v_7\right)=\min\limits_{i=7,8,t} l\left(v_i\right)=16.$

Шаг 4. Вершина v_7 получает постоянную метку: $l\left(v_7\right)=16^{+};\ p=v_7.$

Шаг 5. Не все вершины имеют постоянные пометки, поэтому переходим к шагу 2.

Восьмая итерация.

Шаг 2. $\Gamma\left(p\right)=\Gamma\left(v_7\right)=\left\{v_4,v_6,t\right\}$. Вершины v_4,v_6 имеют постоянные пометки;

$l\left(t\right)=\min\left[31,16^{+}+2\right]=18.$

Шаг 3. $l\left(v_t\right)=\min\limits_{i=t} l\left(v_i\right)=18$.

Шаг 4. Вершина t получает постоянную метку: $l\left(t\right)=18^{+};\ p=t$.

Шаг 5. Не все вершины имеют постоянные пометки, поэтому переходим к шагу 2.

Девятая итерация.

Шаг 2. $\Gamma\left(p\right)=\Gamma\left(t\right)=\left\{v_6,v_7,v_8\right\}$. Вершины v_6,v_7 имеют постоянные пометки;

$l\left(v_8\right)=\min\left[25,18^{+}+3\right]=21.$

Шаг 3. $l\left(v_8\right)=\min\limits_{i=8} l\left(v_i\right)=21$.

Шаг 4. Вершина v_8 получает постоянную метку: $l\left(v_8\right)=21^{+};\ p=v_8$.

Шаг 5. Пометки всех вершин постоянные. Останов.

Замечание 1.

Как только все пометки вершин графа становятся постоянными, т. е. найдены длины кратчайших путей от вершины *s* до любой другой вершины, сами пути можно получить, используя равенство

$$l\left(v_i'\right)+c\left(v_i',v_i\right)=l\left(v_i\right).$$

Здесь $c\left(v_i',v_i\right)$ – вес дуги, соединяющей вершину v_i' с вершиной v_i. Это равенство дает возможность для каждой вершины v_i найти предшествующую ей вершину v_i' и восстановить весь путь от *s* до v_i .

В рассмотренном примере кратчайший путь от вершины *s* до вершины *t* можно восстановить из равенств:

$l\left(t\right)=l\left(v_6\right)+c\left(v_6,t\right),$

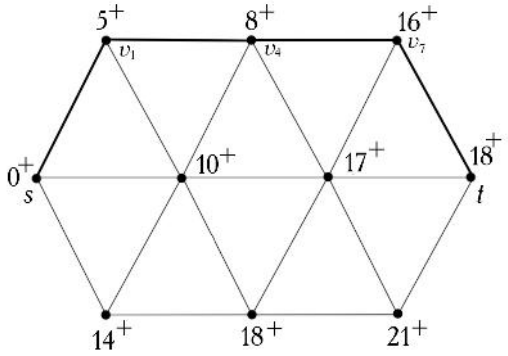
$l\left(v_6\right)=l\left(v_4\right)+c\left(v_4,v_7\right),$

$l\left(v_4\right)=l\left(v_1\right)+c\left(v_1,v_4\right),$

$l\left(v_1\right)=l\left(s\right)+c\left(s,v_1\right),$

полученных в 7-й, 2-й и 1-й итерациях.

Ему соответствует последовательность вершин s,v_1,v_4,v_7,t.



Описание машинного алгоритма

Алгоритм использует три массива из n чисел каждый. Первый массив Visited содержит метки с двумя значениями: False (вершина еще не рассмотрена) и True (вершина уже рассмотрена); второй массив Len содержит текущие кратчайшие расстояния от начальной до соответствующей вершины; третий массив *C* содержит номера вершин – k-ый элемент C есть номер предпоследней вершины на текущем кратчайшем пути из начальной вершины в k-ю. Используется также Matrix – матрица расстояний.

Опишем алгоритм Дейкстры:

1 (инициализация). В цикле от 1 до n заполнить значением False массив Visited; заполнить числом i массив C (i – номер стартовой вершины); перенести i-ю строку матрицы Matrix в массив Len;
Visited[i]:=True; C[i]:=0;

2 (общий шаг). Найти минимум среди неотмеченных (т.е. тех к, для которых Visitid[k]=False); пусть минимум достигается на индексе j, т.е. Len[j]≤Len[k];

Затем выполнять следующие операции:
Visited[i]:=True;
если Len[k]>Len[j]+Matrix[j, k], то (Len[k]:=Len[j]+Matrix[j, k]; C[k]:=j)
{Если все Visited[k] отмечены, то длина пути от v_i до v_k равна C[k]. Теперь надо перечислить вершины, входящие в кратчайший путь}.

3 (выдача ответа). {Путь от v_i до v_k выдается в обратном порядке следующей процедурой:}
3.1 z:=C[k];
3.2 Выдать z
3.3 z:=C[z]. Если z =0, то конец,
иначе перейти к 3.2.

```
Program Deikstra;
Uses Crt;
Const MaxSize=10;
        Infinity=1000;
Var    Mattr: array [1..MaxSize, 1..MaxSize] of integer;
       Visited: array [1..MaxSize] of boolean;
       Len,Path: array [1..MaxSize] of integer;
       n, Start, Finish, k, i: integer;

Procedure Init;
Var  f: text;
       i, j: integer;
begin
  Assign(f, 'INPUT.MTR');
  Reset(f);
  Readln(f, n);
  For i:=1 to n do
  begin
    For j:=1 to n do Read(f, mattr[i,j]);
    Readln(f)
  end;
  Write('Начальная вершина: '); Readln(Start);
  For i:=1 to n do
  begin
    Visited[i]:=False;
    Len[i]:=Mattr[Start, i];
    Path[i]:=Start;
  end;
  Path[Start]:=0;
  Visited[Start]:=True;
end;

Function Possible: Boolean;
Var i: integer;
begin
  Possible:=True;
  For i:=1 to n do If not Visited[i] then Exit;
  Possible:=False;
end;

Function Min: Integer;
Var i, minvalue, currentmin: integer;
begin
  Minvalue:=Infinity;
  For i:=1 to n do
```

```
  If not Visited[i] then
  If Len[i]<minvalue then
  begin
    currentmin:=i;
    minvalue:=Len[i]
  end;
  min:=currentmin;
end;



begin
  ClrScr;
  Init;
  While Possible do
  begin
    k:=min;
    Visited[k]:=True;
    For i:=1 to n do
    If Len[i]>Len[k]+Mattr[i, k] then
    begin
      Len[i]:=Len[k]+Mattr[i, k];
      Path[i]:=k;
    end;
  end;
  Write('Конечная вершина: '); Readln(Finish);
  Write(Finish);
  Finish:=Path[Finish];
  While Finish<>0 do
  begin
    Write('<-', Finish);
    Finish:=Path[Finish];
  end;
  ReadKey;
end.
```

Алгоритм Форда – Беллмана нахождения минимального пути

Предполагается, что ориентированный граф не содержит контуров отрицательной длины.

Алгоритм 1. (*Алгоритм Форда – Беллмана*)

Основными вычисляемыми величинами этого алгоритма являются величины $\lambda_i\left(k\right)$, где $i = 1,2,\ldots,n$ (n – число вершин графа); $k = 1,2,\ldots,n-1$.

Для фиксированных i и k величина $\lambda_i\left(k\right)$ равна длине минимального пути, ведущего из заданной начальной вершины v_1 в вершину v_i и содержащего не более k дуг.

Шаг 1. Установка начальных условий. Ввести число вершин графа n и матрицу весов $C = \left|c_{ij}\right|$.

Шаг 2. Положить $k = 0$. Положить $\lambda_i\left(0\right) = \infty$ для всех вершин, кроме v_1; положить $\lambda_1\left(0\right) = 0$.

Шаг 3. В цикле по k , $k = 1,2,\ldots,n-1$, каждой вершине v_i на k-м шаге приписать индекс $\lambda_i\left(k\right)$ по следующему правилу:

$$\lambda_i\left(k\right) = \min_{1\le j\le n}\left\{\lambda_j\left(k-1\right) + c_{ji}\right\} \qquad (1)$$

для всех вершин, кроме v_1, положить $\lambda_1\left(k\right) = 0$.

В результате работы алгоритма формируется таблица индексов

$$\lambda_i\left(k\right),\ i = 1,2,\ldots,n;\ k = 0,1,2,\ldots,n-1.$$

При этом $\lambda_i\left(k\right)$ определяет длину минимального пути из первой вершины в i-ю, содержащего не более k дуг.

Шаг 5. Восстановление минимального пути. Для любой вершины v_s предшествующая ей вершина v_r определяется из соотношения:

$$\lambda_r\left(n-2\right) + c_{rs} = \lambda_s\left(n-1\right),\ v_r \in G^{-1}\left(v_s\right), \qquad (2)$$

где $G^{-1}\left(v_s\right)$ – прообраз вершины v_s.

Для найденной вершины v_r предшествующая ей вершина v_q определяется из соотношения:

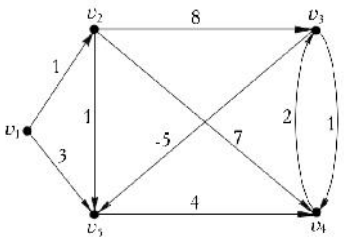
$$\lambda_q\left(n-3\right) + c_{qr} = \lambda_r\left(n-2\right),\ v_q \in G^{-1}\left(v_r\right),$$

где $G^{-1}\left(v_r\right)$ – прообраз вершины v_r, и т. д.

Последовательно применяя это соотношение, начиная от последней вершины v_i, найдем минимальный путь.

Пример ручного решения

С помощью *алгоритма Форда-Беллмана* найдем минимальный путь из вершины v_1 в вершину v_3 в графе, изображенном на рисунке.



Значения индексов $\lambda_i(k)$ будем заносить в таблицу индексов (табл. 1).

Шаг 1. Введем число вершин графа *n* = 5. Матрица весов этого графа имеет вид:

$$C = \begin{Vmatrix} \infty & 1 & \infty & \infty & 3 \\ \infty & \infty & 8 & 7 & 1 \\ \infty & \infty & \infty & 1 & -5 \\ \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & 4 & \infty \end{Vmatrix}$$

Шаг 2. Положим $k = 0$,

$\lambda_1(0) = 0,\ \lambda_2(0) = \lambda_3(0) = \lambda_4(0) = \lambda_5(0) = \infty$

Эти значения занесем в первый столбец табл. 1.

Шаг 3. $k = 1.\ \lambda_1(0) = 0.$

λ1 (1) = 0.

Равенство (1) для *k* = 1 имеет вид:

$$\lambda_i(1) = \min_{1 \le j \le 5} \{\lambda_j(0) + c_{ji}\}$$

$$\lambda_2(1) = \min\{\lambda_1(0) + c_{12};\ \lambda_2(0) + c_{22};\ \lambda_3(0) + c_{32};\ \lambda_4(0) + c_{42};\ \lambda_5(0) + c_{52}\} =$$

$$= \min\{0+1; \infty+\infty; \infty+\infty; \infty+\infty; \infty+\infty\} = 1.$$

$$\lambda_3(1) = \min\{\lambda_1(0) + c_{13}; \lambda_2(0) + c_{23}; \lambda_3(0) + c_{33}; \lambda_4(0) + c_{43}; \lambda_5(0) + c_{53};\} =$$

$$= \min\{0+\infty; \infty+8; \infty+\infty; \infty+2; \infty+\infty\} = \infty.$$

$$\lambda_4(1) = \min\{\lambda_1(0) + c_{14}; \lambda_2(0) + c_{24}; \lambda_3(0) + c_{34}; \lambda_4(0) + c_{44}; \lambda_5(0) + c_{54}\} =$$

$$= \min\{0+\infty; \infty+7; \infty+1; \infty+\infty; \infty+4\} = \infty.$$

$$\lambda_5(1) = \min\{\lambda_1(0) + c_{15}; \lambda_2(0) + c_{25}; \lambda_3(0) + c_{35}; \lambda_4(0) + c_{45}; \lambda_5(0) + c_{55}\} =$$

$$= \min\{0+3; \infty+1; \infty-5; \infty+\infty; \infty+\infty\} = 3.$$

Полученные значения $\lambda_i(1)$ занесем во второй столбец таблицы. Убеждаемся, что второй столбец, начиная со второго элемента, совпадает с первой строкой матрицы весов, что легко объясняется смыслом величин $\lambda_i(1)$, которые равны длине минимального пути из первой вершины в i-ю, содержащего не более одной дуги.

$k = 2.\ \lambda_1(2) = 0.$

Равенство (1) для *k* = 2 имеет вид:

$$\lambda_i(2) = \min_{1 \le j \le 5} \{\lambda_j(1) + c_{ji}\}$$

$$\lambda_2(2) = \min\{0+1; 1+\infty; \infty+\infty; \infty+\infty; 3+\infty\} = 1.$$

$$\lambda_3(2) = \min\{0+\infty; 1+8; \infty+\infty; \infty+2; 3+\infty\} = 9.$$

$\lambda_4\left(2\right) = \min\left\{0+\infty;1+7;\infty+1;\infty+\infty;3+4\right\} = 7.$

$\lambda_5\left(2\right) = \min\left\{0+3;1+1;\infty-5;\infty+\infty;3+\infty\right\} = 2.$

Полученные значения $\lambda_i\left(2\right)$ занесем в третий столбец таблицы. Величины $\lambda_i\left(2\right)$ равны длине минимального пути из первой вершины в *i*-ю, содержащего не более двух дуг.

$k = 3$. $\lambda_1\left(3\right) = 0$.

Равенство (1) для $k = 3$ имеет вид:

$$\lambda_i\left(3\right) = \min_{1\le j\le 5}\left\{\lambda_j\left(2\right) + c_{ji}\right\}$$

$\lambda_2\left(3\right) = \min\left\{0+1;1+\infty;9+\infty;7+\infty;2+\infty\right\} = 1.$

$\lambda_3\left(3\right) = \min\left\{0+\infty;1+8;9+\infty;7+2;2+\infty\right\} = 9.$

$\lambda_4\left(3\right) = \min\left\{0+\infty;1+7;9+1;7+\infty;2+4\right\} = 6.$

$\lambda_5\left(3\right) = \min\left\{0+3;1+1;9-5;7+\infty;2+\infty\right\} = 2.$

Полученные значения $\lambda_i\left(3\right)$ занесем в четвертый столбец таблицы. Величины λ*i* (3) равны длине минимального пути из первой вершины в *i*-ю, содержащего не более трех дуг.

$k = 4$. $\lambda_1\left(4\right) = 0$.

Равенство (1) для $k = 4$ имеет вид:

$$\lambda_i\left(4\right) = \min_{1\le j\le 5}\left\{\lambda_j\left(3\right) + c_{ji}\right\}$$

$\lambda_2\left(4\right) = \min\left\{0+1;1+\infty;9+\infty;6+\infty;2+\infty\right\} = 1.$

$\lambda_3\left(4\right) = \min\left\{0+\infty;1+8;9+\infty;6+2;2+\infty\right\} = 8.$

$\lambda_4\left(4\right) = \min\left\{0+\infty;1+7;9+1;6+\infty;2+4\right\} = 6.$

$\lambda_5\left(4\right) = \min\left\{0+3;1+1;9-5;6+\infty;2+\infty\right\} = 2$

Полученные значения $\lambda_i\left(4\right)$ занесем в пятый столбец таблицы. Величины $\lambda_i\left(4\right)$ равны длине минимального пути из первой вершины в *i*-ю, содержащего не более четырех дуг.

i (номер вершины)	$\lambda_i\left(0\right)$	$\lambda_i\left(1\right)$	$\lambda_i\left(2\right)$	$\lambda_i\left(3\right)$	$\lambda_i\left(4\right)$
1	0	0	0	0	0
2	∞	1	1	1	1
3	∞	∞	9	9	8
4	∞	∞	7	6	6
5	∞	3	2	2	2

Шаг 5. Восстановление минимального пути.

Для последней вершины v_3 предшествующая ей вершина v_r определяется из соотношения (2), полученного при $s = 3$:

$$\lambda_r\left(3\right) + c_{r3} = \lambda_3\left(4\right),\ v_r \in G^{-1}\left(v_3\right), \tag{3}$$

где $G^{-1}\left(v_3\right)$ – прообраз вершины v_3.

$$G^{-1}\left(v_3\right) = \left\{v_2, v_4\right\}.$$

Подставим в (3) последовательно $r = 2$ и $r = 4$, чтобы определить, для какого r это равенство выполняется:

$\lambda_2\left(3\right) + c_{23} = 1 + 8 \neq \lambda_3\left(4\right) = 8$,

$\lambda_4\left(3\right) + c_{43} = 6 + 2 = \lambda_3\left(4\right) = 8$,

Таким образом, вершиной, предшествующей вершине v_3, является вершина v_4

Для вершины v_4 предшествующая ей вершина v_r определяется из соотношения (2) полученного при $s = 4$:

$$\lambda_r\left(2\right) + c_{r4} = \lambda_4\left(3\right),\ v_r \in G^{-1}\left(v_4\right), \tag{4}$$

где $G^{-1}\left(v_4\right)$ – прообраз вершины v_4.

G –1(x4) = {x2, x3, x5}.

Подставим в (4) последовательно $r = 2,\ r = 3$ и $r = 5$, чтобы определить, для какого r это равенство выполняется:

$\lambda_2\left(2\right) + c_{24} = 1 + 7 \neq \lambda_4\left(3\right) = 6$,

$\lambda_3\left(2\right) + c_{34} = 1 + 1 \neq \lambda_4\left(3\right) = 6$,

$\lambda_5\left(2\right) + c_{54} = 2 + 4 = \lambda_4\left(3\right) = 6$

Таким образом, вершиной, предшествующей вершине v_4, является вершина v_5.

Для вершины v_5 предшествующая ей вершина v_r определяется из соотношения (2), полученного при s = 5:

$$\lambda_r\left(1\right) + c_{r5} = \lambda_5\left(2\right),\ v_r \in G^{-1}\left(v_5\right), \tag{5}$$

где $G^{-1}\left(v_5\right)$ – прообраз вершины v_5.

$$G^{-1}\left(v_5\right) = \left\{v_1, v_2\right\}.$$

Подставим в (5) последовательно $r = 1$ и $r = 2$, чтобы определить, для какого r это равенство выполняется:

$\lambda_1\left(1\right) + c_{15} = 0 + 3 \neq \lambda_5\left(2\right) = 2$,

$\lambda_2\left(1\right) + c_{25} = 1 + 1 = \lambda_5\left(2\right) = 2$.

Таким образом, вершиной, предшествующей вершине v_5, является вершина v_2.

Для вершины v_2 предшествующая ей вершина v_r определяется из соотношения (2), полученного при $s = 2$.

$$\lambda_r\left(0\right) + c_{r2} = \lambda_2\left(1\right),\ v_r \in G^{-1}\left(v_2\right), \qquad (6)$$

Где $G^{-1}\left(v_2\right)$ – прообраз вершины v_2.

$$G^{-1}\left(v_2\right) = \left\{v_1\right\}.$$

Подставим в (6) *r* = 1, чтобы определить, выполняется ли это равенство:

$$\lambda_1\left(0\right) + c_{12} = 0 + 1 = \lambda_2\left(1\right) = 1$$

Таким образом, вершиной, предшествующей вершине v_2, является вершина v_1.

Итак, найден минимальный путь – v_1, v_2, v_5, v_4, v_3, его длина равна 8.

Программа по алгоритму Форда-Беллмана

```
(*Алгоритм  Форда-Беллмана*)
Program Ford;
var a : array [1..20,1..20] of word;(*матрица смежности*)
c, pred, fl, d : array [1..20] of word;
(*c – массив кратчайших расстояний
pred – массив предыдущих вершин
fl – массив флагов
d – массив для записи пути*)

i, j, k, n, first, last : byte;
f : text;(* переменная для открытия файла in.txt*)
(*процедура обхода графа вглубь – для поиска всех путей*)
Procedure Dfs(x : word);
var i : byte;  (*локальная переменная*)
begin
  if x=last then (*если конечная вершина то выводим путь*)
  begin
    write(first,' ');
    for i:=1 to j do (*выводим путь*)
    write(d[i],' ');
    writeln;
    exit; (*выходим из процедуры*)
  end;
  fl[x]:=1; (*помечаем, что были в вершине*)
  for i:=1 to n do
  if (fl[i]=0)and(a[x,i]<>32767) then
  begin
    inc(j);
    d[j]:=i; (*записываем в путь вершины*)
    dfs(i); (*вызываемся от i-й вершины*)
```

```
    dec(j);
  end;
  fl[x]:=0; (*помечаем, что вершина свободна*)
end;
(*Основная программа*)
begin
  assign(f,'in.txt'); (*Открываем файл для чтения*)
  reset(f);
  readln(f, n); (*считываем количество вершин*)
  for i := 1 to n do
  for j := 1 to n do
  read(f, a[i,j]); (*считываем матрицу смежности*)
  writeln('Matrix:');
  for i:=1 to n do  (*выводим матрицу на экран*)
  for j:=1 to n do
  if j=n then writeln(a[i,j]) else write(a[i,j],' ');
  for i:=1 to n do (*заменяем нули бесконечностью*)
  for j:=1 to n do
  if a[i,j]=0 then a[i,j]:=32767;
  writeln('Введите вершину 1');
  readln(first);
  writeln('Введите вершину 2');
  readln(last);
  close(f); (* закрываем file in.txt*)
  for j := 1 to n do
  begin
    c[j] := a[first,j]; (* записываем начальные значения*)
    if a[first,j] < 32767 then
    pred[j] := first;
  end;
  for i := 3 to n do
  for j := 1 to n do
  if j <> first then
  for k := 1 to n do  (*если не бесконечность и путь более выгодный*)
  if (c[k] < 32767) and (c[k] + a[k,j] < c[j]) then
  begin
    c[j] := c[k] + a[k,j];(*записывем новое значение*)
    pred[j] := k;{записываем предвершины}
  end;
  if c[last] = 32767 then writeln('Нет путей') else
  begin
    writeln;
    writeln('Кратчайший путь:');
    write(first,' ');
    i := last;
```

```
    k := 1;
    while i <> first do (*в обратном порядке обходим путь*)
    begin
      d[k] := i;(*записываем путь в массив*)
       k := k + 1;
       i := pred[i];
    end;
    for i:= k-1 downto 1 do  (*выводим кратчайший путь*)
    write(d[i],' ');
    writeln;
    writeln('Все пути:');
    j:=0;
    Dfs(first);(*вызываем процедуру поиска всех путей*)
  end;
  readln; readln; (*ждем нажатия клавиши*)
end.
```

Алгоритм Флойда-Уоршелла

Рассматриваемый алгоритм иногда называют алгоритмом Флойда-Уоршелла. Алгоритм Флойда-Уоршелла является алгоритмом на графах, который разработан в 1962 году Робертом Флойдом и Стивеном Уоршеллом. Он служит для нахождения кратчайших путей между всеми парами вершин графа.

Метод Флойда непосредственно основывается на том факте, что в графе с положительными весами ребер всякий неэлементарный (содержащий более 1 ребра) кратчайший путь состоит из других кратчайших путей.

Этот алгоритм более общий по сравнению с алгоритмом Дейкстры, так как он находит кратчайшие пути между любыми двумя вершинами графа.

В алгоритме Флойда используется матрица A размера $n \times n$, в которой вычисляются длины кратчайших путей. Элемент $A\left[i, j\right]$ равен расстоянию от вершины i к вершине j, которое имеет конечное значение, если существует ребро $\left(i, j\right)$, и равен бесконечности в противном случае.

Алгоритм Флойда

Основная идея алгоритма. Пусть есть три вершины i, j, k и заданы расстояния между ними. Если выполняется неравенство $A\left[i,k\right] + A\left[k,j\right] < A\left[i,j\right]$, то целесообразно заменить путь $i \to j$ путем $i \to k \to j$. Такая замена выполняется систематически в процессе выполнения данного алгоритма.

Шаг 0. Определяем начальную матрицу расстояния A_0 и матрицу последовательности вершин S_0. Каждый диагональный элемент обеих матриц равен 0, таким образом, показывая, что эти элементы в вычислениях не участвуют. Полагаем $k = 1$.

Основной шаг k. Задаем строку k и столбец k как ведущую строку и ведущий столбец. Рассматриваем возможность применения замены описанной выше, ко всем элементам $A\left[i, j\right]$ матрицы A_{k-1}. Если выполняется неравенство

$A\left[i,k\right] + A\left[k,j\right] < A\left[i,j\right], \left(i \neq k, j \neq k, i \neq j\right),$ тогда выполняем следующие действия:

1. создаем матрицу A_k путем замены в матрице A_{k-1} элемента $A\left[i, j\right]$ на сумму $A\left[i,k\right] + A\left[k,j\right]$;
2. создаем матрицу S_k путем замены в матрице S_{k-1} элемента $S\left[j, j\right]$ на k. Полагаем $k = k + 1$ и повторяем шаг k.

Таким образом, алгоритм Флойда делает n итераций, после i-й итерации матрица A будет содержать длины кратчайших путей между любыми двумя парами вершин при условии, что эти пути проходят через вершины от первой до i-й. На каждой итерации перебираются все пары вершин и путь между ними сокращается при помощи i-й вершины.

```
Program Floid_Uorshel;
Uses Crt;
Const
PP=50;
Type
Graph = array[1..pp,1..pp] of integer;
Var
p:integer;
t,c,h:graph;
i,j: integer;

Procedure Floyd (var t:graph; c:graph; var h:graph);
var i,j,k:integer;
GM:real;
begin
  GM:=10000;
  for i:=1 to p do
  for j:=1 to p do t[i,j]:=c[i,j];
  if c[i,j]=GM then H[i,j]:=0  else
```

```
  begin
    H[i,j]:=j;
  end;
  for i:=1 to p do
  for j:=1 to p do
  for k:=1 to p do
  if (i<>j)and(T[j,i]<>GM)and(i<>k)and(T[i,k]<>GM)and(T[j,k]=GM) or
(T[j,k]>T[j,i]+T[i,k]) then
  begin
    H[j,k]:=H[j,i];
    T[j,k]:=T[j,i]+T[i,k]
  end;
end;

Procedure ReadFileGraph (var T:graph);
var
 i,j:integer;
 f: text;
begin
  Writeln ('Reading from the text file');
  Assign (f,'nell.txt');
  reset(f);
  Readln(f,P);
  for i:=1 to p do for j:=1 to p do
  read(f,t[i,j]); close(f);
end;

begin
  ClrScr;
  ReadFileGraph(c);
  floyd(t,c,h);
  writeln('----------------------------------------');
  for i:=1 to p do
  begin
    for j:=1 to p do write (t[i,j]:3);
    writeln
  end;
  writeln('----------------------------------------');
  for i:=1 to p do
  begin
    for j:=1 to p do write (h[i,j]:3);
    writeln
  end;
  readln;
end.
```