1 АЛГОРИТМЫ НА ГРАФАХ

1.1 МИНИМАЛЬНОЕ ОСТОВНОЕ ДЕРЕВО

Напомним, что неориентированным *деревом* называется связный (неориентированный) граф без циклов.

Определение 1. Остовом *(неориентированного) графа* $G = (V, E)$ *называется дерево* $S = (V, T)$ *такое, что* $T \subseteq E$.

Пусть задана функция $c : E \to R$, *приписывающая каждому ребру* $e \in E$ *его стоимость (вес, длину)* $c(e) \in R$ (R — *множество вещественных чисел). Тогда стоимость* $c(S)$ *дерева* S *определяется как сумма стоимостей всех его ребер, т.е.* $c(S) = \sum_{e \in T} c(e)$.

Минимальным остовом *называется остов минимальной стоимости.*

Таким образом, минимальный остов — это самая дешевая (короткая) система путей, связывающая все вершины G.

Лемма 1. *Пусть* $S = (V, T)$ *- остовное дерево для* G. *Тогда*
(а) для любых двух вершин v_1 *и* v_2 *из* V *путь между* v_1 *и* v_2 *в* S *единственный;*
(б) если к S *добавить любое ребро из* $E \setminus T$, *то возникнет ровно один цикл.*

Лемма 2. *Пусть* $\{(V_1, T_1), (V_2, T_2), \ldots, (V_k, T_k)\}$ *- произвольный остовной лес для* G, $k > 1$, $T = \cup_{i=1}^{k} T_i$. *Пусть* $e = (v, w)$ — *ребро наименьшей стоимости в* $E \setminus T$ *такое, что его концы принадлежат разным деревьям:* $v \in V_i$, $w \in V_j$ *и* $i \neq j$. *Тогда существует остовное дерево, содержащее все ребра из* T *и* e, *стоимость которого не больше стоимости любого остовного дерева для* G, *содержащего* T.

Доказательство. Пусть $S' = (V, T')$ — остовное дерево минимальной стоимости, содержащее все ребра T, и ребро $e \notin T'$. Тогда в S' имеется некоторый путь p из v в w, не содержащий e. Выберем на нем первое такое ребро $e' = (v', w')$, для которого $v' \in V_i$, а $w' \notin V_i$. Рассмотрим множество ребер $T_1' = (T' \cup \{e\}) \setminus \{e'\}$. Это множество задает остовное дерево, так как добавление ребра e к T' приводит по лемме 1 к образованию единственного цикла, а удаление ребра e' этот цикл разрушает. Так как $e' \notin T$, то выбор e гарантирует, что $c(e) \leq c(e')$. Поэтому $c(T_1') = c(T') + c(e) - c(e') \leq c(T')$. Из выбора T' тогда следует, что $c(T_1') = c(T')$ и, следовательно, $S_1 = (V, T_1')$ является требуемым остовным деревом.
□

Эта лемма является обоснованием следующего "жадного"алгоритма построения минимального остовного дерева, который был предложен Крускалом в 1956г. Для улучшения эффективности в нем используются эффективные алгоритмы для реализации очередей с приоритетами (можно для этого выбрать, например, сортирующие деревья или 2-3-деревья) и структуры данных и алгоритмы для представления системы множеств и выполнения операций ОБЪЕДИНИТЬ-НАЙТИ (для этого можно выбрать, например, деревья со сжатием путей).

АЛГОРИТМ МинОстов

Вход: $G = (V, E)$- связный неориентированный граф, $c(e)$ - функция стоимости ребер.

Выход: T - множество ребер минимального остовного дерева.

Структуры данных: Q - очередь с приоритетами для ребер, VS - набор непересекающихся подмножеств вершин G, элементы VS - множества вершин остовных деревьев в текущем остовном лесу.

1. $T := \emptyset$; $VS := \emptyset$;

2. создать очередь с приоритетами Q из всех ребер из E;
3. **FOR EACH** $v \in V$ **DO** добавить $\{v\}$ к VS;
4. **WHILE** $|VS| > 1$ **DO**
5. { $(v, w) := MIN(Q)$; УДАЛИТЬ$(Q, (v, w))$;
6. $W1 :=$ НАЙТИ(v); $W2 :=$ НАЙТИ(w);
7. **IF** $W1 \neq W2$ **THEN**
8. { ОБЪЕДИНИТЬ$(W1, W2, W1)$; $T := T \cup \{(v, w)\}$}
9. }

Теорема 1.1. *Алгоритм* **МинОстов** *строит минимальное остовное дерево для связного графа* $G = (V, E)$. *Если в цикле в строках 4 - 9 рассматривается* d *ребер, то затрачивается время* $O(d \log_2 m)$, *где* $m = |E|$. *В худшем случае выполнение алгоритма* **МинОстов** *занимает время* $O(m \log_2 m)$.

Доказательство. Нетрудно проверить, что инвариантом цикла в строках 4-9 является следующее свойство:
система множеств $VS = \{V_1, \ldots, V_k\}$ *и набор ребер* T *задают остовной лес* $\{(V_1, T_1), (V_2, T_2), \ldots, (V_k, T_k)\}$ *исходного графа* G, ($T = \cup_{i=1}^{k} T_i$), *который может быть расширен до минимального остовного дерева графа* G.

Это свойство, очевидно выполнено перед началом цикла, т.к. каждое из деревьев в VS содержит одну вершину, а множество ребер T пусто. Пусть оно выполнено перед очередным выполнением тела цикла. Если на этом выполнении для выбранного ребра (v, w) вершины v и w оказываются в разных деревьях, то это ребро добавляется к T, а соответствующие деревья объединяются в стр. 8. Очевидно, циклы при таком объединении не появляются и система VS остается остовным лесом для G. Так как ребро (v, w) имеет минимальный вес, то по лемме 2 этот лес может быть расширен до минимального остовного дерева.

Из связности G следует, что после завершения **МинОстов** лес VS будет состоять из одного дерева и оно и будет минимальным.
□

Рассмотрим следующий нагруженный граф G_1 :

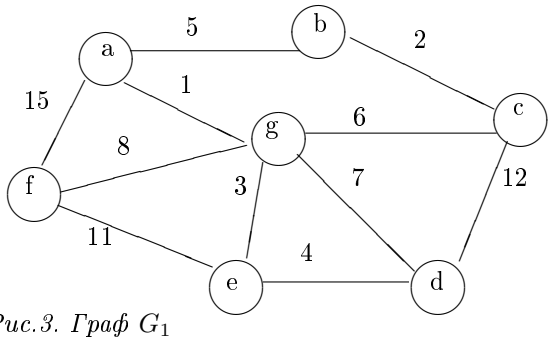

Рис.3. Граф G_1

Применим к нему алгоритм **МинОстов**. Для простоты сразу покажем порядок, в котором будут выдаваться ребра из очереди Q в стр.5 алгоритма.

$Q = \{(a, g), (b, c), (g, e), (e, d), (a, b), (g, c), (d, g), (f, g), (e, f), (c, d), (a, f)\}$.

Выполнение основного цикла представим в виде таблицы, каждая строка которой соответствует рассмотрению очередного ребра (v, w) в стр. 5. В столбце T ребра, включаемые в T отмечены $+$. Вначале система VS состоит из семи множеств V_i $(i = 1, \ldots, 7)$, содержащих по одной вершине.

$(\mathbf{v},\mathbf{w})$	**T**	**W1**	**W2**	$\mathbf{V_1}$	$\mathbf{V_2}$	$\mathbf{V_3}$	$\mathbf{V_4}$	$\mathbf{V_5}$	$\mathbf{V_6}$	$\mathbf{V_7}$
				$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$
(a,g)	+	1	7	$\{a,g\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	
(b,c)	+	2	3	$\{a,g\}$	$\{b,c\}$		$\{d\}$	$\{e\}$	$\{f\}$	
(g,e)	+	1	5	$\{a,e,g\}$	$\{b,c\}$		$\{d\}$		$\{f\}$	
(e,d)	+	1	4	$\{a,d,e,g\}$	$\{b,c\}$				$\{f\}$	
(a,b)	+	1	2	$\{a,b,c,d,e,g\}$					$\{f\}$	
(g,c)	−	1	1	$\{a,b,c,d,e,g\}$					$\{f\}$	
(d,g)	−	1	1	$\{a,b,c,d,e,g\}$					$\{f\}$	
(f,g)	+	1	5	$\{a,b,c,d,e,f,g\}$						

Таким образом, мы построили для G_1 минимальный остов $S=(V,T)$, где $T=\{(a,g),(b,c),(g,e),(e,d),(a,b),(f,g)\}$. Его стоимость $c(S)=23$.

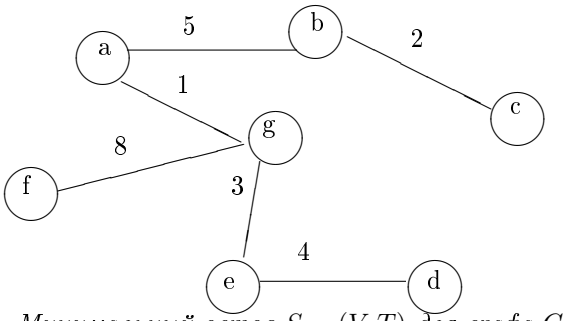

Рис.4. Минимальный остов $S=(V,T)$ для графа G_1

Задача 1. *Найти минимальное остовное дерево для неориентированного графа $G=(V,E)$, где $V=\{v_1,v_2,v_3,v_4,v_5,v_6,v_7,v_8,v_9\}$, $E=\{(v_1,v_2,18),(v_1,v_3,2),(v_3,v_2,4),(v_3,v_4,6),(v_3,v_5,8),(v_4,v_6,5),(v_5,v_4,4),(v_6,v_1,7),(v_6,v_8,4),(v_6,v_7,3),(v_7,v_5,1),(v_7,v_8,7),(v_8,v_1,5),(v_8,v_9,3),(v_9,v_1,1)\}$ (третий параметр в скобках - стоимость ребра).*

1.2 АЛГОРИТМЫ ОБХОДА ГРАФОВ

1.2.1 Поиск в глубину на неориентированном графе и задача о лабиринте

Задачу поиска выхода из лабиринта можно формализовать так: лабиринт — это неориентированный граф, вершины которого представляют "перекрестки"лабиринта, а ребра — дорожки между соседними перекрестками. Одна или несколько вершин отмечены как выходы. Задача состоит в построении пути из некоторой исходной вершины в вершину-выход.

В этом разделе мы рассмотрим метод обхода всех вершин графа, называемый *поиском в глубину*. Его идею кратко можно описать так: находясь в некоторой вершине v, идем из нее в произвольную еще не посещенную смежную вершину w, если такой вершины нет, то возвращаемся в вершину, из которой мы пришли в v.

Алгоритм поиска в глубину

Вход: $G=(V,E)$ – неориентированный граф, представленный списками смежностей: для каждой $v\in V$ список L_v содержит перечень всех смежных с v вершин.

Выход: $NUM[v]$ – массив с номерами вершин в порядке их прохождения и множество (древесных) ребер $T\subseteq E$, по которым осуществляется обход.

Алгоритм ПОГ

1. $T=\{\}$; $NOMER=1$;
2. для каждой вершины $v\in V$ положим $NUM[v]=0$ и пометим v как "новую";

3. **ПОКА** существует "новая"вершина $v \in V$
4. **ВЫПОЛНЯТЬ** ПОИСК(v);

Основную роль в этом алгоритме играет следующая рекурсивная процедура.
procedure ПОИСК(v):

5. пометить v как "старую";
6. $NUM[v] = NOMER$; $NOMER = NOMER + 1$;
7. **FOR EACH** $w \in L_v$ **DO**
8. **IF** вершина w "новая"
9. **THEN**
10. { добавить (v, w) к T;
11. ПОИСК(w);
12. }

Теорема 1.2. *Алгоритм* **ПОГ** *обходит (нумерует) все вершины графа* $G = (V, E)$ *за время* $O(\max(|V|, |E|))$. *Если* G *— связный граф, то* $S = (V, T)$ *— это остов* G, *если граф* G *не является связным, то* $S = (V, T)$ *— это остовной лес для* G, *т.е. объединение остовных деревьев для каждой из компонент связности* G.

Доказательство. Оценка времени следует из того, что процедура ПОИСК может вызываться для каждой вершины не более одного раза, а правильность является непосредственным следствием утверждения о работе процедуры ПОИСК.

Лемма 3. *Пусть* $G_1 = (V_1, E_1)$ *— компонента связности графа* G *и* $v_1 \in V_1$ *— первая вершина, для которой в стр. 4 вызывается процедура ПОИСК*(v_1). *Тогда*
а) в процессе выполнения этого вызова процедура ПОИСК будет вызвана один раз для любой вершины $w \in V_1$;
б) после завершения этого вызова все вершины из V_1 *"старые"и имеют номера от* $NOMER$ *до* $NOMER + |V_1| - 1$;
в) добавленные к T *во время этого вызова ребра* T_1 *образуют остовное дерево для* G_1.

Доказательство. Пункт (а) доказывается индукцией по расстоянию от v_1 до w.

Если это расстояние равно 1, то $w \in L_{v_1}$ и рассматривается в цикле в стр.7. Если она в этот момент "старая", то значит ПОИСК(w) уже вызывался. Если же w "новая", то в стр.11 происходит вызов ПОИСК(w).

Предположим теперь, что утверждение ПОИСК(u) вызывается для всех вершин u, находящихся на расстоянии $k \geq 1$ от v_1, и пусть вершина $w \in L_{v_1}$ находится на рсстоянии $(k + 1)$ от v_1. Тогда имеется путь длины $(k + 1)$ от v_1 до w. Пусть u — это предпоследняя вершина на этом пути. Тогда расстояние от v_1 до u равно k и по нашему предположению в некоторый момент выполняется вызов ПОИСК(u). Так как $w \in L_u$, то в этом вызове вершина w в некоторый момент рассматривается в цикле в стр.7. Как и выше, если она в этот момент "старая", то ПОИСК(w) уже вызывался. Если же w еще "новая", то в стр.11 происходит вызов ПОИСК(w).

Пункт (б) непосредственно следует из (а), а циклы в T отсутствуют, так как каждое добавляемое ребро ведет из "старой"вершины в "новую".
□.

Алгоритм поиска в глубину часто используется как основа для различных алгоритмов обработки графов. Вместо строки 6, в котрой вершина v получает номер $NUM(v)$, можно вставить вызов любой процедуры, обрабатывающей информацию, связанную с этой вершиной (например, для задачи о лабиринте это может быть проверка того, что v является выходом из лабиринта). И тогда полученный вариант алгоритма обеспечит обработку всех вершин графа. Ниже мы рассмотрим несколько применений этого алгоритма.

Применим алгоритм **ПОГ** к графу G_2, изображенному на следующем рисунке.

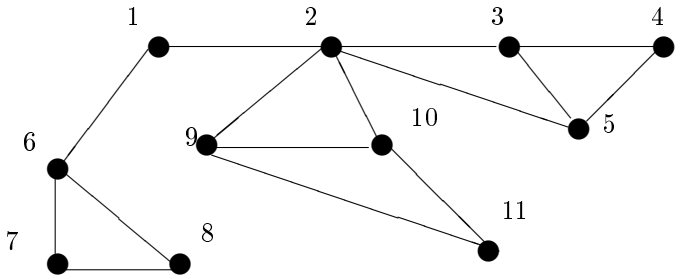

Рис.5. Граф G_2

Его представление в виде списков смежности имеет следующий вид:

$L_1: 6, 2$ $\quad L_7 : 6, 8$
$L_2: 1, 9, 10, 3$ $\quad L_8 : 6, 7$
$L_3: 2, 5, 4$ $\quad L_9 : 2, 10, 11$
$L_4: 3, 5$ $\quad L_{10} : 2, 9, 11$
$L_5: 2, 3, 4$ $\quad L_{11} : 9, 10$
$L_6: 1, 7, 8$

Алгоритм **ПОГ** вызовет процедуру ПОИСК(1). Эта процедура рекурсивно вызовет ПОИСК(6) и т.д. Вот структура всех получающихся вызовов процедуры ПОИСК:
ПОИСК(1) $\Rightarrow$ ПОИСК(6) $\Rightarrow$ ПОИСК(7) $\Rightarrow$ ПОИСК(8)
$\Downarrow$
ПОИСК(2) $\Rightarrow$ ПОИСК(9) $\Rightarrow$ ПОИСК(10) $\Rightarrow$ ПОИСК(11)
$\Downarrow$
ПОИСК(3) $\Rightarrow$ ПОИСК(5) $\Rightarrow$ ПОИСК(4)

Вначале идут "горизонтальные"вызовы, затем возвраты справа налево и вызовы "по вертикали". В результате вершины G_2 получат следующие номера, отражающие порядок их прохождения:

V:	1	2	3	4	5	6	7	8	9	10	11
NUM:	1	5	9	11	10	2	3	4	6	7	8

Ребра остова T, построенные в процессе обхода графа, показаны на следующем рисунке. Стрелки указывают направление обхода.

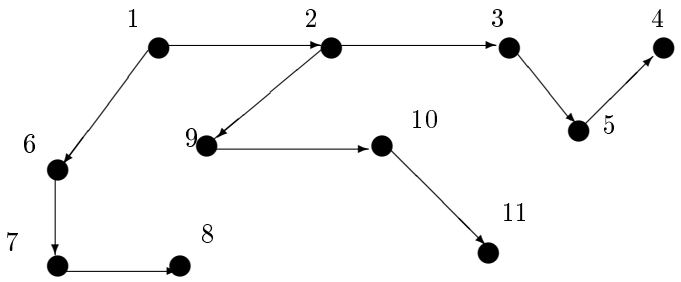

Рис.6. Остовное "глубинное"дерево T графа G_2

1.2.2 Поиск в ширину на неориентированном графе

Алгоритм поиска в ширину ПОШ отличается от алгоритма ПОГ только процедурой ПОИСК.

procedure ПОИСКШ(v):

5. создать пустую очередь Q;
6. ДОБАВИТЬ(Q, v); пометить v как "старую";
7. **WHILE** $Q \neq \emptyset$ **DO**
8. { $w :=$ НАЧАЛО(Q); УДАЛИТЬ(Q, w);
9. $NUM[w] := NOMER$; $NOMER := NOMER + 1$;
10. **FOR EACH** $u \in L_w$ **DO**
11. **IF** вершина u "новая"
12. **THEN** { ДОБАВИТЬ(Q, u); пометить u как "старую" }
13. }

Теорема 1.3. *Алгоритм ПОШ обходит (нумерует) все вершины графа $G = (V, E)$ за время $O(\max(|V|, |E|))$.*

Задача 2. *Модифицировать алгоритм ПОГ (ПОШ) так, чтобы он определял компоненты связности графа G (например, приписывал каждой вершине v номер ее компоненты связности КОМП$[v]$).*

Задача 3. *Доказать, что построенное ПОГ множество ребер T образует остовной лес G.*

Задача 4. *Пусть в результате работы ПОГ ребро (v, w) оказалось обратным, т.е. оно принадлежит $E \backslash T$. Тогда либо v — предок w, либо w — предок v в построенном ПОГ остовном лесу.*

Задача 5. *Рассмотрите модификацию алгоритма ПОГ (ПОШ) для ориентирован- ного графа $G = (V, E)$, в которой $L_v = \{w|\ (v, w) \in E\}$. Покажите, что полученный алгоритм нумерует все вершины и строит остовной лес.*

Задача 6. *Пусть в результате алгоритма ПОГ на ориентированном графе G ребро (v, w) оказалось поперечным, т.е. v и w не являются потомками друг друга в . Доказать, что тогда $NUM[v] > NUM[w]$.*

Задача 7. *Пусть F - это лес, построенный алгоритмом ПОГ для ориентированного графа $G = (V, E)$. Доказать, что G - ациклический граф $\Longleftrightarrow$ $(E \setminus F)$ не содержит дуги (v, w), в которой вершина w является предком вершины v в F.*

Задача 8. *Предложите алгоритм, определяющий имеется ли в заданном неориентированном графе $G = (V, E)$ цикл за время $O(|V|)$.*

Задача 9. Топологическая сортировка.
Для ориентированного ациклического графа $G = (V, E)$ назовем линейный порядок $<$ на множестве вершин V топологическим, если из $v < w$ следует, что в G нет пути из w в v (вообще говоря, этот порядок не единственный). Пусть ПОГ+пост - это вариант алгоритма поиска в глубину, в котором в процедуре ПОИСК(v) вершина v получает номер перед выходом из процедуры (т.е. операторы строки 6 стоят после строки 12). Доказать, что ПОГ+пост нумерует вершины G в порядке, обратном топологическому.

Задача 10. Сильно связные компоненты орграфов.
Ориентированный граф $G = (V, E)$ называется сильно связным, если для любой пары его вершин v и w в G имеется путь из v в w и путь из w в v. Сильно связной компонентой G называется максимальный сильно связный подграф G. Рассмотрим следующий алгоритм ССК:

1. Выполнить ПОГ +пост на G и для каждой вершины v определить ее номер $NUM[v]$;
2. Построить граф $G' = (V, E')$, "перевернув"стрелки: $E' = \{(v, w) | (w, v) \in E\}$;
3. $W := V$;
4. **WHILE** $W \neq \emptyset$ **DO**
5. { выбрать $w \in W$ с максимальным номером $NUM[w]$;
6. выполнить ПОГ(w) на G' и выдать пройденные вершины V_w;
7. $W := W \setminus V_w$}.

Доказать, что алгоритм ССК корректно выдает все сильно связные компоненты G.

Задача 11. *Предположим, что в алгоритме ССК из предыдущей задаче в цикле 4-7 вместо графа $G' = (V, E')$ с перевернутыми стрелками используется исходный граф G, а вершины рассматриваются в порядке увеличения их номеров. Будет ли такой алгоритм работать правильно?*

Задача 12. *Определим для ориентированного графа $G = (V, E)$ его* граф компонент *$G^K = (V^K, E^K)$ следующим образом:*
$V^K = \{C \mid C - $ *компонента сильной связности* $G\}$,
$E^K = \{(C_1, C_2) \mid$ *существует такое ребро* $(v, u) \in E$, *что* $v \in C_1, u \in C_2\}$.
Предложите алгоритм, который по графу G строит его граф компонент G^K за время $O(|V| + |E|)$.

1.2.3 Двусвязные компоненты

Определение 2. *Неориентированный граф $G = (V, E)$ называется двусвязным, если для любых трех его попарно различных вершин w, v и a существует путь между w и v, не проходящий через a.*

Вершина v называется точкой сочленения (точка раздела) графа G, если для некоторой пары различных вершин x и y (не совпадающих с v) всякий путь между x и y проходит через v.

Следствие. *Связный граф является двусвязным тогда и только тогда, когда в нем нет точек сочленения.*

Определение 3. *Максимальный двусвязный подграф графа G называется его двусвязной компонентой.*

Задача 13. *Докажите, что двусвязная компонента - это максимальный набор ребер, любые два ребра которого принадлежат общему простому циклу.*

Лемма 4. *Пусть $G_i = (V_i, E_i)$ $(i = 1, ..., k)$ - двусвязные компоненты графа $G = (V, E)$. Тогда*
1) граф G_i двусвязен $(i = 1, ..., k)$:
2) для любой пары $i \neq j$ $|V_i \cap V_j| \leq 1$;
3) a - точка сочленения $G \iff$ для некоторых i и j $\{a\} = V_i \cap V_j$.

Лемма 5. *Пусть $G = (V, E)$ - связный неориентированный граф, а $S = (V, T)$ - глубинное остовное дерево. Вершина a является точкой сочленения G огда и только тогда, когда*
1) a - корень дерева S и он имеет более одного сына или
2) a - не корень и у него есть сын s такой, что между ним и его потомками и предками a нет обратных ребер.

Пусть $NUM[v]$ - номер, присвоенный v алгоритмом ПОГ. Определим функцию НИЖ$[v] = min\{\{NUM[v]\}, \{NUM[w] \mid$ существует такое обратное ребро (x, w), что x - потомок v, а w - предок v в глубинном остовном дереве $(V, T)\}\}$.

Следствие леммы 5: v - точка сочленения $\iff$ у v есть сын s, для которого НИЖ$[s] \geq NUM[v]$.

Из определения НИЖ следует, что
НИЖ$[v] = min\{\{NUM[v]\} \cup \{$НИЖ$[s] | s-$сын v$\} \cup NUM[w] | (v, w)-$обратное ребро$\}\}$.

Алгоритм ДВУСВЯЗ

Вход: связный неориентированный граф $G = (V, E)$, заданный посредством списков смежности L_v, вершина $v_0 \in V$.

Выход: списки ребер двусвязных компонент G.

1. $i := 0$; СоздатьСтек(S); $T := \emptyset$;
2. **FOR** $v \in V$ **DO** $\{NUM[v := 0$; отметить v как "новую"$\}$;
3. ПОИСКБ(v_0).

Procedure ПОИСКБ(v)

1. $i := i + 1$; $NUM[v] := i$; НИЖ$[v] := i$;
2. Пометить v как "старую";
3. **FOR EACH** $w \in L_v$ **DO**
4. { **IF** $(v, w) \notin S$ **THEN** ВТОЛК$(S, (v, w))$ **END-IF**;
5. **IF** w - "новая" **THEN** /* (v, w) прямое ребро
6. { $T := T \cup (v, w)$;
7. ОТЕЦ$[w] := v$;
8. ПОИСКБ$[w]$;
9. **IF** НИЖ$[w] \geq NUM[v]$ **THEN** /* v -точка сочленения
10. Вытолкнуть из S все ребра до (v, w) включительно
11. **END-IF**;
12. НИЖ$[v] := min\{$НИЖ$[v],$ НИЖ$[w]\}$
13. }

14. **ELSE** /* (v, w) - обратное ребро
15. **IF** ОТЕЦ$[v] \neq w$ **THEN**
16. НИЖ$[v] := min\{$НИЖ$[v], NUM[w]\}$ **END-IF**
17. **END-IF**
18. }

Теорема 1.4. *Алгоритм ДВУСВЯЗ правильно находит двусвязные компоненты графа G за время $O(|E|)$.*

Задача 14. *Ребро неориентированного связного графа называется мостом, если при его удалении граф перестает быть связным.*
А) Докажите, что ребро графа G является мостом тогда и только тогда, когда оно не входит ни в какой простой цикл.
Б) Постройте алгоритм нахождения всех мостов графа за время $O(|E|)$.

1.3 Задачи о путях на графе

1.3.1 Транзитивное замыкание

Определение 4. *Пусть $G = (V, E)$ — ориентированный граф.* **Граф достижимости** $G^* = (V, E^*)$ *для G имеет то же множество вершин V и следующее множество ребер $E^* = \{(u, v) \mid$ в графе G вершина v достижима из вершины $u\}$.*

Иначе говоря, отношение E^* является рефлексивным и транзитивным замыканием отношения E.

Рассмотрим следующий граф G:

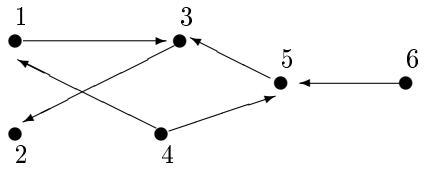

Рис.1 Граф G

Тогда можно проверить, что граф достижимости G^* для G выглядит так (ребра-петли при каждой из вершин не показаны):

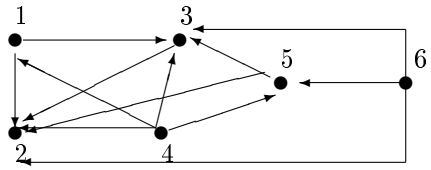

Рис.2 Граф G^*

Каким образом по графу G можно построить граф G^*? Один способ заключается в том, чтобы для каждой вершины графа G определить множество достижимых из нее вершин, последовательно добавляя в него вершины, достижимые из нее путями длины 0, 1, 2 и т.д.

Другой способ, основан на использовании матрицы смежности A_G графа G и булевых операций. Пусть множество вершин $V = \{v_1, \ldots, v_n\}$. Матрица A_G — это булева матрица размера $n \times n$ такая, что

$$a_{ij} = \begin{cases} 1, & (v_i, v_j) \in E \\ 0 & \text{в противном случае} \end{cases}$$

Обозначим через E_n единичную матрицу размера $n \times n$. Положим $\tilde{A} = A_G \vee E_n$. Определим "булевы"степени этой матрицы, используя дизъюнкцию $\vee$ в качестве сложения, а конъюнкцию $\wedge$ в качестве умножения (будем для нее использовать обычную точку: $\cdot$).

Пусть $\tilde{A}^0 = E_n, \tilde{A}^1 = \tilde{A}, \ldots, \tilde{A}^{k+1} = \tilde{A}^k \cdot \tilde{A}$. Наша процедура построения G^* основана на следующем утверждении.

Лемма 6. *Пусть* $\tilde{A}^k = (a_{ij}^{(k)})$. *Тогда*

$$a_{ij}^{(k)} = \begin{cases} 1, & \text{в графе } G \text{ из } v_i \text{ в } v_j \text{ имеется путь длины } \leq k \\ 0 & \text{в противном случае} \end{cases}$$

Доказательство проведем индукцией по k.

Базис. При $k = 0$ и $k = 1$ утверждение справедливо по определению $\tilde{A}^0$ и $\tilde{A}^1$.

Индукционный шаг. Пусть лемма справедлива для k. Покажем, что она остается справедливой и для $k+1$. По определению $\tilde{A}^{k+1}$ имеем:

$$a_{ij}^{(k+1)} = a_{i1}^{(k)} a_{1j}^{(1)} \vee \ldots \vee a_{ir}^{(k)} a_{rj}^{(1)} \vee \ldots a_{in}^{(k)} a_{nj}^{(1)}.$$

Предположим, что в графе G из v_i в v_j имеется путь длины $\leq k+1$. Рассмотрим кратчайший из таких путей. Если его длина $\leq k$, то по предположению индукции $a_{ij}^{(k)} = 1$. Кроме того, $a_{jj}^{(1)} = 1$. Поэтому $a_{ij}^{(k)} a_{jj}^{(1)} = 1$ и $a_{ij}^{(k+1)} = 1$. Если длина кратчайшего пути из из v_i в v_j равна $k+1$, то пусть v_r — его предпоследняя вершина. Тогда из v_i в v_r имеется путь длины k и по предположению индукции $a_{ir}^{(k)} = 1$. Так как $(v_r, v_j) \in E$, то $a_{rj}^{(1)} = 1$. Поэтому $a_{ir}^{(k)} a_{rj}^{(1)} = 1$ $a_{ij}^{(k+1)} = 1$.

Обратно, если $a_{ij}^{(k+1)} = 1$, то хотя бы для одного r слагаемое $a_{ir}^{(k)} a_{rj}^{(1)}$ в сумме равно 1. Если это $r = j$, то $a_{ij}^{(k)} = 1$ и по индуктивному предположению в G имеется путь из v_i в v_j длины $\leq k$. Если же $r \neq j$, то $a_{ir}^{(k)} = 1$ и $a_{rj}^{(1)} = 1$. Это означает, что в G имеется путь из v_i в v_r длины $\leq k$ и ребро $(v_r, v_j) \in E$. Объединив их, получаем путь из v_i в v_j длины $\leq k+1$.

□

Из леммы 6 непосредственно получаем

Следствие 1. *Пусть* $G = (V, E)$ — *ориентированный граф с* n *вершинами, а* G^* — *его граф достижимости. Тогда* $A_{G^*} = \tilde{A}^{n-1}$.

Доказательство. Из определения пути следует, что если в G имеется путь из u в $v \neq u$, то в нем имеется и простой путь из u в v длины $\leq n-1$. А по лемме 6 все такие пути представлены в матрице $\tilde{A}^{n-1}$.

□

Таким образом процедура построения матрицы смежности A_{G^*} графа достижимости для G сводится к возведению матрицы $\tilde{A}$ в степень $n-1$. Для этого достаточно выполнить $]\log n[$ возведений в квадрат:

$$\tilde{A} \Rightarrow \tilde{A}^2 \Rightarrow \tilde{A}^{2^2} \Rightarrow \ldots \Rightarrow \tilde{A}^{2^k},$$

где k — это наименьшее число такое, что $2^k \geq n-1$.

Так как стандартный алгоритм умножения $n \times n$ матриц можно выполнить за время $O(n^3)$, то мы получили алгоритм вычисления графа достижимости за время $O(n^3 \log n)$. Отметим, что это оценка числа битовых операций.

Оказывается, что эту оценку можно улучшить.

Алгоритм Уоршолла

Вход: A_G
Выход: $B = A_{G^*}$

1. $B := A_G$;
2. **for** $k = 1$ **to** n **do**
3. **for** $i = 1$ **to** n **do**
4. **for** $j = 1$ **to** n **do**
5. $b_{ij} := b_{ij} \vee (b_{ik} \wedge b_{kj})$;

Теорема 1.5. *Алгоритм Уоршолла вычисляет матрицу* $B = A_{G^*}$ *за время* $O(n^3)$.

При доказательстве корректности алгоритма используется лемма 7.

Определение 5. l *- путем из* v_i *в* v_j *называется последовательность:* $v_i, v_{i_1}, v_{i_2}, \ldots, v_{i_k}, v_j$, *если* $1 \leq i_r \leq l$, *а* i *и* j *- любые.*

Лемма 7. *Пусть* $B^{(l)}$ *- матрица, полученная после* l *выполнений цикла по* k.
Тогда $b_{ij}^{(l)} = 1 \Leftrightarrow$ *в* G *есть* l *- путь из* v_i *в* v_j.

1.3.2 Кратчайшие пути между всеми парами вершин

$G = (V, E)$ - ориентированный граф.
c_{ij} - длина ребра (v_i, v_j), $c_{ij} \geq 0$ (если его нет, то равна ∞).
$C = (c_{ij})$ - матрица длин.
d_{ij} - длина кратчайшего пути из v_i в v_j.
$D = (d_{ij})$ - матрица длин кратчайших путей.

Алгоритм Уоршолла-Флойда

Вход: C
Выход: D

1. $D := C$;
2. **for** $k = 1$ **to** n **do**
3. **for** $i = 1$ **to** n **do**
4. **for** $j = 1$ **to** n **do**
5. $d_{ij} := min\{d_{ij}, d_{ik} + d_{kj}\}$;

Теорема 1.6. *Алгоритм Уоршалла-Флойда строит матрицу кратчайших расстояний за время* $O(n^3)$.

Лемма 8. $d_{ij}^{(l)}$ *- длина кратчайшего* l*-пути из* v_i *в* v_j.

Задача 15. *Как изменить алгоритм, чтобы находить сами кратчайшие пути?*

1.3.3 Задача о кратчайших путях из одного источника

Пусть $G = (V, E)$ — ориентированный граф, для каждого ребера $e \in E$ которого указана его (неотрицательная) длина: $c(e) \geq 0$. Тогда длина пути $p = v_1, v_2, \ldots, v_{k+1}$ определяется как сумма длин ребер, входящих в этот путь: $c(p) = \sum_{i=1}^{k} c(v_i, v_{i+1})$. Если в G имеется путь из вершины a в вершину b, то имеется и такой путь минимальной длины. Он называется *кратчайшим путем из* a *в* b, а его длина обозначается как $\rho(a, b)$. Конечно, может оказаться несколько различных кратчайших путей. Естественно спросить, как узнать длину кратчайшего пути из a в b и построить его? Лучшие известные на сегодняшний день алгоритмы, отвечающие

на этот вопрос, решают, на самом деле, более общую задачу построения всех кратчайших путей из одного источника: *по вершине a найти длины кратчайших путей из a во все достижимые из нее вершины и построить для каждой из таких вершин некоторый кратчайший путь из a*. Если для каждой вершины $v \in V$, достижимой из a, зафиксировать один кратчайший путь из a в v, то получившийся граф будет представлять ориентированное дерево с корнем a (докажите это!). Это дерево называется *деревом кратчайших путей из a*.

Мы рассмотрим алгоритм построения дерева кратчайших путей и определения их длин, предложенный в 1959г. Е. Дейкстрой. Его идея следующая: перед каждым этапом известно множество *отмеченных вершин* S, для которых кратчайшие пути найдены ранее; тогда на очередном этапе к нему добавляется вершина w, с самым коротким путем из a, проходящим по множеству S; после этого пересчитываются длины кратчайших путей из a в оставшиеся вершины из $V \setminus S$ с учетом новой вершины w. Длина текущего кратчайшего пути из a в v, проходящего по множеству S, заносится в ячейку $D[v]$ массива D. В конце работы в этом массиве находятся длины соответствующих кратчайших путей. Для определения дерева кратчайших путей служит массив ОТЕЦ, его элемент ОТЕЦ[v] содержит ссылку на вершину, из которой кратчайший путь приходит в v.

Алгоритм Дейкстры
Вход: $G = (V, E)$ — ориентированный граф, $c(u, v) \geq 0$ —длина ребра $(u, v) \in E$ (если $(u, v) \notin E$, то считаем, что $c(u, v) = \infty$).

1. $S := \{a\}$; ' отметить a
2. $D[a] := 0$; ' расстояние от a до a
3. **ДЛЯ КАЖДОЙ** $v \in V, v \neq a$ **ВЫПОЛНЯТЬ**
4. $\{D[v] := c(a, v)$; ' расстояние от a до v через a
5. **ЕСЛИ** $c(a, v) < \infty$ **ТО** ОТЕЦ[v] $:= a$ **ИНАЧЕ** ОТЕЦ[v] $:= -\}$;
' === ОСНОВНОЙ ЦИКЛ ===
6. **ПОКА** $V \setminus S \neq \emptyset$ **ВЫПОЛНЯТЬ** ' есть неотмеченные вершины
7. { выбрать неотмеченную вершину w с минимальным $D[w]$;
8. $S := S \cup \{w\}$; ' отметить w
9. **ДЛЯ КАЖДОЙ** (неотмеченной) $u \in V \setminus S$ **ВЫПОЛНЯТЬ**
10. { **ЕСЛИ** $D[u] > D[w] + c(w, u)$
11. **ТО** { $D[u] := D[w] + c(w, u)$;
12. ОТЕЦ[u] $:= w\}$
13. }

Рассмотрим работу этого алгоритма на нагруженном графе $G = (V = \{a, b, c, d, e, f\}, E)$ и выделенной вершине $a \in V$. Зададим длины ребер матрицей
$C = (c_{uv})$, где элемент $c_{uv} = c(u, v)$:

$$\begin{pmatrix} & a & b & c & d & e & f \\ a & 0 & 25 & 5 & 30 & \infty & 75 \\ b & 12 & 0 & \infty & \infty & 120 & 20 \\ c & \infty & 15 & 0 & 20 & 45 & 60 \\ d & \infty & \infty & \infty & 0 & 23 & 20 \\ e & \infty & \infty & 75 & 20 & 0 & 20 \\ f & 40 & 15 & 15 & 26 & \infty & 0 \end{pmatrix}$$

Поэтапную работу алгоритма Дейкстры удобно представлять в виде таблицы, строки которой соответствуют его этапам. Первый столбец — номер этапа, второй показывает изменение

N	S	w	D[w]	**D**						О	Т	Е	Ц
				b	c	d	e	f	b	c	d	e	f
1.	a	c	5	25	5	30	∞	75	a	a	a	-	a
2.	a, c	b	20	20	-	25	50	65	c	a	c	c	c
3.	a, c, b	d	25	-	-	25	50	40	c	a	c	c	b
4.	a, c, b,d	f	45	-	-	-	48	45	c	a	c	d	b
5.	a, c, b,d	e	45	-	-	-	-	45	c	a	c	d	b

Таблица 1: Алгоритм Дейкстры на графе G.

множества отмеченных вершин S, третий — вершину w, добавляемую к S на текущем шаге, четвертый — длину кратчайшего пути из a в w, затем идут столбцы со значениями элементов массивов D и ОТЕЦ.

Дерево кратчайших путей из вершины a задается массивом ОТЕЦ. Оно представлено на следующем рисунке.

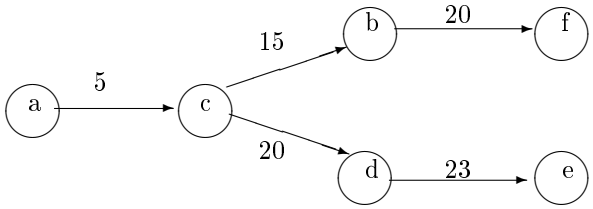

Рис. 7. Дерево кратчайших путей из вершины a в графе G

Теорема 1.7. *Алгоритм Дейкстры строит дерево кратчайших путей из вершины a во все достижимые из нее вершины и для каждой такой вершины v определяет длину $D[v] = \rho(a, v)$ кратчайшего пути в нее из a.*

Время работы алгоритма Дейкстры - $O(n^2)$.

Доказательство. . Докажем по индукции, что после каждого этапа алгоритма выполнены следующие условия:
а) для любой вершины $v \in S$ величина $D[v]$ равна длине $\rho(a, v)$ кратчайшего пути из a в v;
б) для любой вершины $v \in V \setminus S$ величина $D[v]$ равна длине кратчайшего пути из a в v, проходящего по множеству S;
в) для каждой вершины v дерева T, задаваемого массивом ОТЕЦ, длина пути из корня a в v равна $D[v]$.

Эти три условия, очевидно выполняются после инициализации в строках 1- 5.

Предположим теперь что они выполнены перед началом k-го этапа. Пусть w — вершина, добавляемая к S на k-ом этапе. По предположению, $D[w]$ — длина кратчайшего пути из a в w, все вершины которого, кроме w, входят в S. Предположим, что есть другой более короткий путь p из a в w. Зафиксируем на этом пути первую вершину u, не входящую в S. По выбору p $u \neq w$. Поэтому путь p разбивается на две непустые части: путь p_1 из a в u и путь p_2 из u в w. Но по выбору w мы имеем, что длина $p_1 \geq D[u] \geq D[w]$. Так как длина p_2 неотрицательна, то длина $p \geq D[w]$, т.е. этот путь не короче пути, представленного в дереве T. Таким образом, $D[w]$ — это длина кратчайшего пути из a в w. Следовательно, условие (а) выполнено и после k-го этапа.

Рассмотрим теперь произвольную вершину $u \in V \setminus (S \cup \{w\})$. Кратчайший путь p из a в u, проходящий по множеству $S \cup \{w\}$, либо не включает вершину w и в этом случае его длина равна $D[u]$ и он имеется в текущем дереве T, либо он проходит через w и составлен из

кратчайшего пути из a в w через S, продолженного ребром (w, u). В последнем случае длина пути равна $D[w]+c(w, u)$. Но в 10-ой строке алгоритма эти величины сравниваются и, если путь через w короче, то его длина становится новым значениием $D[u]$ (строка 11) и он фиксируется в дереве T (строка 12). Следовательно, условия (б) и (в) также выполнены после k-го этапа.

Так как после завершения алгоритма $S = V$, то в завершающем дереве T представлены кратчайшие пути из a во все достижимые из нее вершины, а массив D содержит длины этих путей. Значение $D[u] = \infty$ указывает на то, что вершина u не достижима из вершины a.

Для оценки времени отметим, что число этапов не превосходит n, так как на каждом этапе в S добавляется новая вершина. На каждом этапе поиск вершины w с минимальным значением $D[w]$ и последующий пересчет значений $D[v]$ требует $O(n)$ шагов. Отсюда, общее время ограничено $O(n^2)$.
□

Задача 16. *Где в доказательстве правильности алгоритма Дейкстры используется неотрицательность весов ребер? Приведите пример графа (с отрицательными весами), для которого алгоритм Дейкстры дает неверный ответ.*

Задача 17. *Докажите, что на каждом шаге алгоритма Дейкстры кратчайший путь из исходной вершины в любую вершину множества S проходит только через вершины множества S.*

Задача 18. *Сколько раз может меняться для одной вершины v значение $D[v]$ в ходе работы алгоритма Дейкстры для графа с 5 вершинами. Привести пример на каждый возможный случай.*

Алгоритм Дейкстры требует, чтобы веса всех ребер были неотрицательны (см. задачу 16). Но бывают ситуации, когда естественно рассматривать и графы с отрицательными весами ребер. Если в них нет циклов отрицательной длины, то понятие кратчайшего пути от одной вершины до другой определено корректно. Следующий алгоритм позволяет решить задачу о кратчайших путей из одного источника для таких графов.

Алгоритм Беллмана-Форда

procedure Relax(u, v)

1. **if** $D[v] > D[u] + c(u, v)$ **then**{
2. $D[v] := D[u] + c(u, v)$;
3. ОТЕЦ$[v] := u$;}

procedure Bellman_Ford(G, c, v_0)

1. **forall** $v \in V$ **do**
2. $\{D[v] := \infty$; ОТЕЦ$[v]$:=nil;$\}$
3. $D[v_0] := 0$;
4. **for** $i = 1$ **to** $n - 1$ **do**
5. **forall** $(u, v) \in E$ **do** Relax(u, v);
6. **forall** $(u, v) \in E$ **do**
7. **if** $D[v] > D[u] + c(u, v)$ **then** return false;
8. return true;

Лемма 9. *Если в графе $G = (V, E)$ нет циклов отрицательной длины, достижимых из v_0, то алгоритм Беллмана-Форда в $D[v]$ возвращает длину кратчайшего пути из v_0 в v.*

Следствие. *Вершина v достижима из $v_0 \Leftrightarrow$ алгоритм Беллмана-Форда выдает $D[v] < \infty$.*

Теорема 1.8.

1. *Если в графе $G = (V, E)$ нет циклов отрицательной длины, достижимых из v_0, то алгоритм Беллмана-Форда возвращает true и для всех $v \in V$ $D[v] = \rho(v_0, v)$ - длина кратчайшего пути из v_0 в v, ОТЕЦ[v] - задает дерево кратчайших путей.*
2. *Если в графе $G = (V, E)$ есть цикл отрицательной длины, достижимый из v_0, то алгоритм Беллмана-Форда возвращает false (некорректная задача)*

Сложность (время работы) алгоритма: $O(|V||E|)$.

Задача 19. *$G = (V, E)$ - ориентированный граф без циклов отрицательной длины. c - весовая функция на ребрах (ребра могут быть отрицательной длины). Построить алгоритм, который для каждой вершины $v \in V$ находит расстояние до ближайшего соседа:*

$$\rho^*(v) = \min_{u \neq v} \rho(u, v)$$

Задача 20. Валютные операции
Пусть на рынке имеется n валют, текущие курсы которых заданы $n \times n$ матрицей R: 1единица i-ой валюты обменивается на $R[i, j]$ единиц j-ой валюты. Разработайте алгоритм, который находит такую последовательность валют (цикл) $i_1, i_2, \ldots, i_k, i_1$, для которой $R[i_1, i_2] \cdot R[i_2, i_3] \cdot \ldots \cdot R[i_k, i_1] > 1$. Оцените время работы этого алгоритма.

Задача 21. Вложенные ящики.
Скажем, что d-мерный ящик размера $(x_1, x_2, \ldots, x_d)$ вкладывается в *ящик размера $(y_1, y_2, \ldots, y_d)$, если существует такая перестановка $i_1, i_2, \ldots, i_d$ измерений $1, 2, \ldots, d$, для которой $x_{i_1} \leq y_1, x_{i_2} \leq y_2, \ldots, x_{i_d} \leq y_d$.*

а) Докажите, что отношение "вкладывается в" *транзитивно.*

б) Предложите алгоритм проверки этого отношения.

в) Пусть дано n d-мерных ящиков. Предложите алгоритм для определения самой большой подпоследовательности вложенных ящиков (матрешки из максимально возможного числа ящиков). Оцените его сложность.

Системы ограничений на разности.
Система ограничений на разности – это система неравенств вида:

$$(*) \quad x_i - x_j \leq b_k \quad 1 \leq i, j \leq n,\ 1 \leq k \leq m$$

(здесь x_i, x_j – переменные, b_k – константы). Свяжем с этой системой граф $G_* = (V_*, E_*)$, у которого
$V_* = \{v_0, v_1, \ldots, v_n\}$,
$E_* = \{(v_0, v_i) \mid 1 \leq i \leq n\} \cup \{(v, v_j) \mid$ неравенство $x_i - x_j \leq b_k$ входит в систему $(*)\}$.
Зададим веса ребер следующим образом:
$c(v_0, v_i) = 0$ $(1 \leq i \leq n)$, $c(v_i, v_j) = b_k$, если $x_i - x_j \leq b_k$ входит в систему $(*)$.

Задача 22. *Докажите следующее утверждение.*

Теорема 1.9.
а) Если в графе G_* нет циклов отрицательной длины, то $x_1 = \rho(v_0, v_1), x_2 = \rho(v_0, v_2), \ldots, x_n = \rho(v_0, v_n)$ является решением системы (*).
б) Если в графе G_* есть циклы отрицательной длины, то система (*) не имеет решений.

Задача 23. *Используя предыдущую задачу, предложите алгоритм, для проверки разрешимости систем вида (*) и нахождения их решений.*