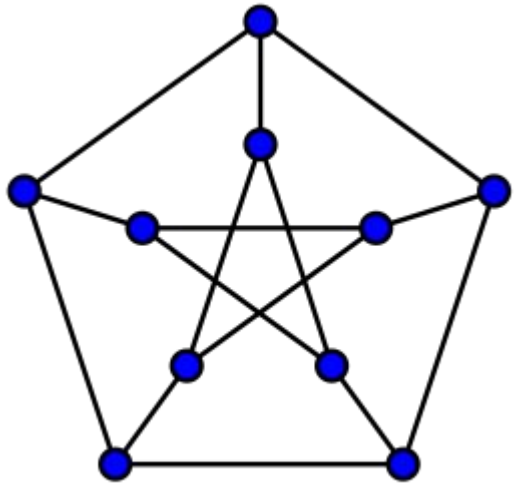
Графы

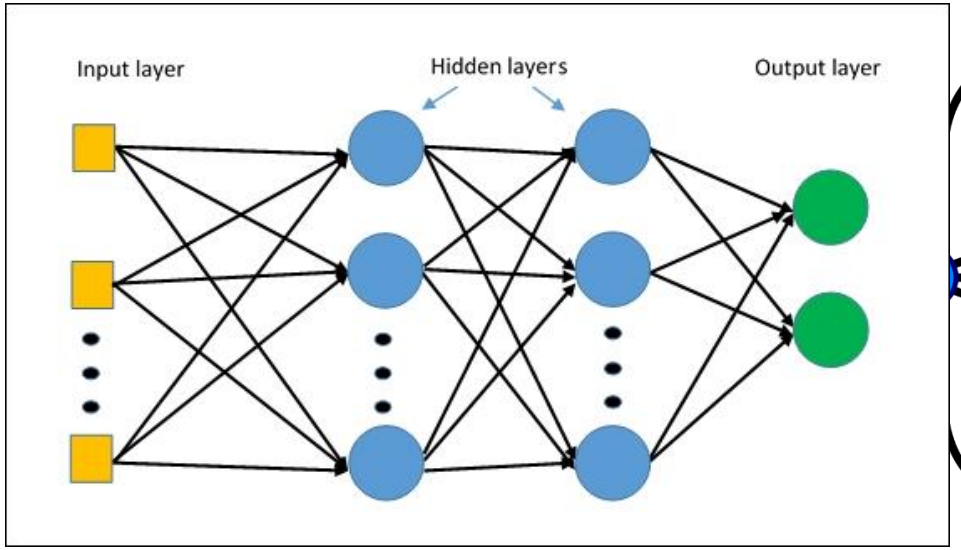
ЛЕКЦИЯ №9

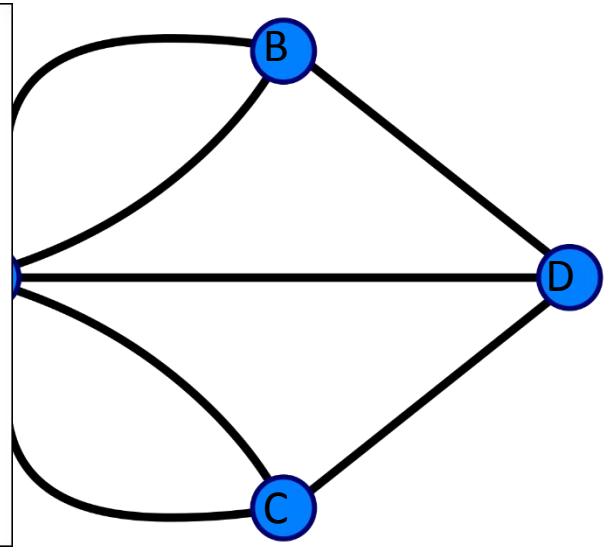
Графы

Во многих жизненных ситуациях старая привычка толкает нас рисовать на бумаге точки, обозначающие людей, города, химические вещества, и показывать линиями (возможно со стрелками) связи между ними. Описанная картинка называется графом.

Граф - это совокупность узлов (вершин) и соединяющих их ребер (дуг).







Графы

Если дуги имеют **направление** (вспомните улицы с односторонним движением), то такой граф называется направленным или **ориентированным графом** (**орграфом**).

Цепью называется последовательность ребер, соединяющих **две** (возможно не соседние) вершины **u** и **v**.

В **направленном** графе такая последовательность ребер называется «**путь**».

Граф называется **связным**, если существует цепь между любой **парой** вершин.

Если граф **не связный**, то его можно разбить на k связных компонент – он **называется k-связным**.

В практических задачах часто рассматриваются взвешенные графы, в которых каждому ребру приписывается **вес** (или **длина**). Такой граф называют **сетью**.

Циклом называется **цепь** из какой-нибудь вершины v в **нее** саму.

Деревом называется граф **без циклов**.

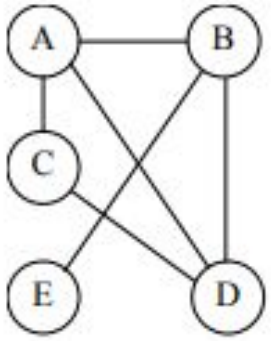
Полным называется граф, в котором проведены **все возможные ребра** (для графа, имеющего n вершин таких ребер будет **n(n-1)/2**).

Описание графов

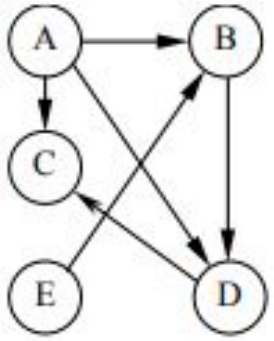
Для описания графов часто используют два типа матриц – **матрицу смежности** (для **невзвешенных графов**) и **весовую матрицу** (для **взвешенных**).

Матрица смежности графа с **N** вершинами – это матрица размером N на N, где каждый элемент с индексами **(i,j)** является логическим значением и показывает, есть ли дуга из вершины **i** в вершину **j**.

Часто вместо логических значений (**истина/ложь**) используют целые числа (**1/0**). Для **неориентированных** графов матрица смежности всегда **симметрична** относительно главной диагонали. Для ориентированных графов это не всегда так, потому что может существовать путь из вершины **i** в вершину **j** и не существовать обратного пути.



	A	B	C	D	E
A	0	1	1	1	0
B	1	0	0	1	1
C	1	0	0	1	0
D	1	1	1	0	0
E	0	1	0	0	0

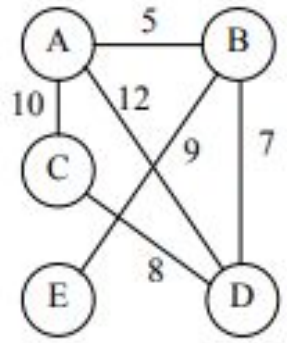


	A	B	C	D	E
A	0	1	1	1	0
B	0	0	0	1	0
C	0	0	0	0	0
D	0	0	1	0	0
E	0	1	0	0	0

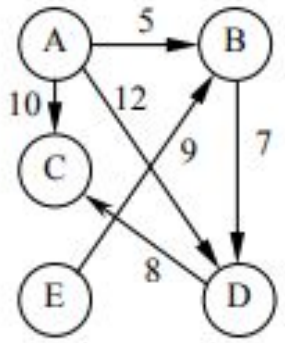
Описание графов

Для взвешенных графов недостаточно просто указать, есть ли связь между вершинами. Требуется еще хранить в памяти «**вес**» каждого ребра, например, стоимость проезда или длину пути. Для этого используется **весовая матрица**.

Весовая матрица графа с **N** вершинами – это матрица размером N на N, где каждый элемент с индексами (i,j) равен «**весу**» ребра из вершины i в вершину j.



	A	B	C	D	E
A	0	5	10	12	0
B	5	0	0	7	9
C	10	0	0	8	0
D	12	7	8	0	0
E	0	9	0	0	0



	A	B	C	D	E
A	0	5	10	12	0
B	0	0	0	7	0
C	0	0	0	0	0
D	0	0	8	0	0
E	0	9	0	0	0

Задача Прима-Краскала

Весовая матрица графа с **N** вершинами – это матрица размером N на N, где каждый элемент с индексами (i,j) равен «**весу**» ребра из вершины i в вершину j.

Город будем изображать **узлом** (**вершиной**). Телефонные линии могут разветвляться только на телефонных станциях, а не в чистом поле. Поскольку требуется линия минимальной общей длины, в ней не будет циклов, потому что иначе можно было бы убрать одно звено цикла и станции по-прежнему были бы связаны. В терминах теории графов эта задача звучит так:

Дан граф с n вершинами; длины ребер заданы матрицей $\{d_{ij}\}$, $i, j = 1..n$. Найти набор ребер, соединяющий все вершины графа (он называется **остовным деревом** (**остовом**)) и имеющий минимальную длину.

Жадные алгоритмы

Представим себе зимовщика, которому предоставили **некоторый запас продуктов на всю зиму**. Конечно, он может сначала съесть все самое вкусное – шоколад, мясо и т.п., но за такой подход придется жестоко расплачиваться в конце зимовки, когда останется только соль и маргарин.

Подобным образом, если оптимальное решение строится по шагам, обычно **нельзя выбирать на каждом этапе «самое вкусное»** – за это придется расплачиваться на последних шагах. Такие алгоритмы называют **жадными**.

Для решения задачи Прима-Краскала используется **жадный** алгоритм:

В цикле **n-1 раз** выбрать из оставшихся ребер самое **короткое** ребро, которое **не образует цикла** с уже выбранными.

Требуется проверить, чтобы циклов не было. Сделать это можно так: в самом начале покрасим все вершины в **разные цвета** и затем, выбрав очередное ребро между вершинами i и j, где i и j имеют разные цвета, **перекрасим вершину j и все соединенные с ней** (то есть имеющие ее цвет) в цвет i. Таким образом, при выборе ребер, соединяющих вершины разного цвета, цикл не возникнет никогда, а после n-1 шагов все вершины будут иметь один цвет.

Жадные алгоритмы, код

Для записи информации о ребрах введем структуру:

```
struct rebro { int i, j; }; // ребро соединяет вершины i и j
```

Причем будем считать, что в паре номер первой вершины i меньше, чем номер второй j.
Ниже приведён алгоритм программы:

1. Покрасить все вершины в разные цвета.
2. Сделать n-1 раз в цикле:
    1. выбрать ребро (i,j) минимальной длины, соединяющее вершины разного цвета;
    2. запомнить его в массиве ребер;
    3. перекрасить все вершины, имеющие цвет j, в цвет i.
3. Вывести результат.

Работа с деревьями

```
for ( i = 0; i < N; i ++ ) // перекрасить все вершины, цвет
if ( Col[i] == col_j ) // которых совпал с цветом
Col[i] = Col[iMin]; // вершины jMin
}
```

```
Dmin = 65535;
for ( i = 0; i < N-1; i ++ )
for ( j = i+1; j < N; j ++ )
   // ищем самое короткое ребро, не образующее цикла
   if ( Col[i] != Col[j] && D[i][j] < Dmin ) {
   Dmin = D[i][j];
   iMin = i;
   jMin = j;
   }
Reb[k].i = iMin; // запомнить найденное ребро
Reb[k].j = jMin;
col_j = Col[jMin];
```

Кратчайший путь

Задача. Задана сеть дорог между населенными пунктами (часть из них могут иметь одностороннее движение). Требуется найти кратчайший путь между двумя заданными пунктами.

Обычно задачу несколько расширяют и находят сразу кратчайшие пути от заданной вершины ко всем остальным. В терминах теории графов задача выглядит так:

В сети, где часть дорог имеет одностороннее движение, найти кратчайшие пути от заданной вершины ко всем остальным.

Алгоритм Дейкстры

Среди нерассмотренных вершин (для которых $\boldsymbol{a[i] = 0}$) найти такую вершину $\boldsymbol{j}$, что расстояние b_j – минимальное. Рассмотреть ее:

1) установить $a[j] = 1$ (вершина рассмотрена);
2) сделать для всех вершин k: если путь от вершины x к вершине k через j короче, чем уже найденный кратчайший путь (то есть $b_j + d_{jk} < b_k$), то запомнить его: $c[k] = j$ и $b_k = b_j + d_j$

2. Массив $\{b_i\}$, в котором b_i – текущее кратчайшее расстояние от выбранной стартовой вершины *x* до вершины *i*.
3. Массив $\{c_i\}$, в котором c_i – номер предпоследней вершины в текущем кратчайшем пути из выбранной стартовой вершины *x* до вершины *i*.

Пусть *x* – номер выбранной стартовой вершины.

1. Заполним весь массив *a* значением 0 (пока ни одна вершина не рассмотрена).
2. В *i*-ый элемент массива *b* запишем расстояние от вершины *x* до вершины *i* (если пути нет, то оно равно ∞, в программе укажем очень большое число).
3. Заполним весь массив c значением *x* (пока рассматриваем только прямые пути от *x* к *i*).
4. Рассмотрим стартовую вершину: присвоим *a*[*x*]=1 и *c*[*x*]=0 (начало всех путей).

Вывод результата

Можно доказать, что в конце такой процедуры массив b, будет содержать кратчайшие расстояния от стартовой вершины x до вершины i (для всех i). Однако хотелось бы еще получить сам оптимальный путь.

Оказывается, массив c содержит всю необходимую информацию для решения этой задачи. Предположим, что надо вывести оптимальный путь из вершины x в вершину i. По построению предпоследняя вершина в этой цепи имеет номер $z = c[i]$. Теперь надо найти оптимальный путь в

1) установить z=i;
2) пока c[z] не равно нулю,
   - z=c[z];
   - вывести z.

Алгоритм Дейкстры. Пример

Пусть дана сеть дорог, показанная на рисунке справа. Найдем кратчайшие расстояния от вершины 3 до всех остальных вершин. Ход выполнения алгоритма Дейкстры по шагам.

Инициализация

	1	2	3	4	5	6	7	8
a	1	1	1	1	1	1	0	1
b	12	25	0	18	47	38	61	60
c	3	3	0	3	2	4	5	2