18.11.2024

Графы (основы). Ассемблер <-> С

Филиппов Михаил Витальевич

m.filippov@g.nsu.ru

89232283872

Императивное программирование, 2024-2025

Новосибирский государственный университет
*НАСТОЯЩАЯ НАУКА

Давайте познакомимся

Филиппов Михаил Витальевич

- Окончил магистратуру ФФ НГУ
- Окончил аспирантуру ИТ СО РАН
- Являюсь м.н.с. ИТ СО РАН
- 7+ лет опыт в программировании C/C++

Новосибирский государственный университет
*НАСТОЯЩАЯ НАУКА

План лекции

Графы (основы)

60 минут

Ассемблер <-> C

(первые слова)

30 минут

План лекции

Графы (основы)

60 минут

Ассемблер <-> С

(первые слова)

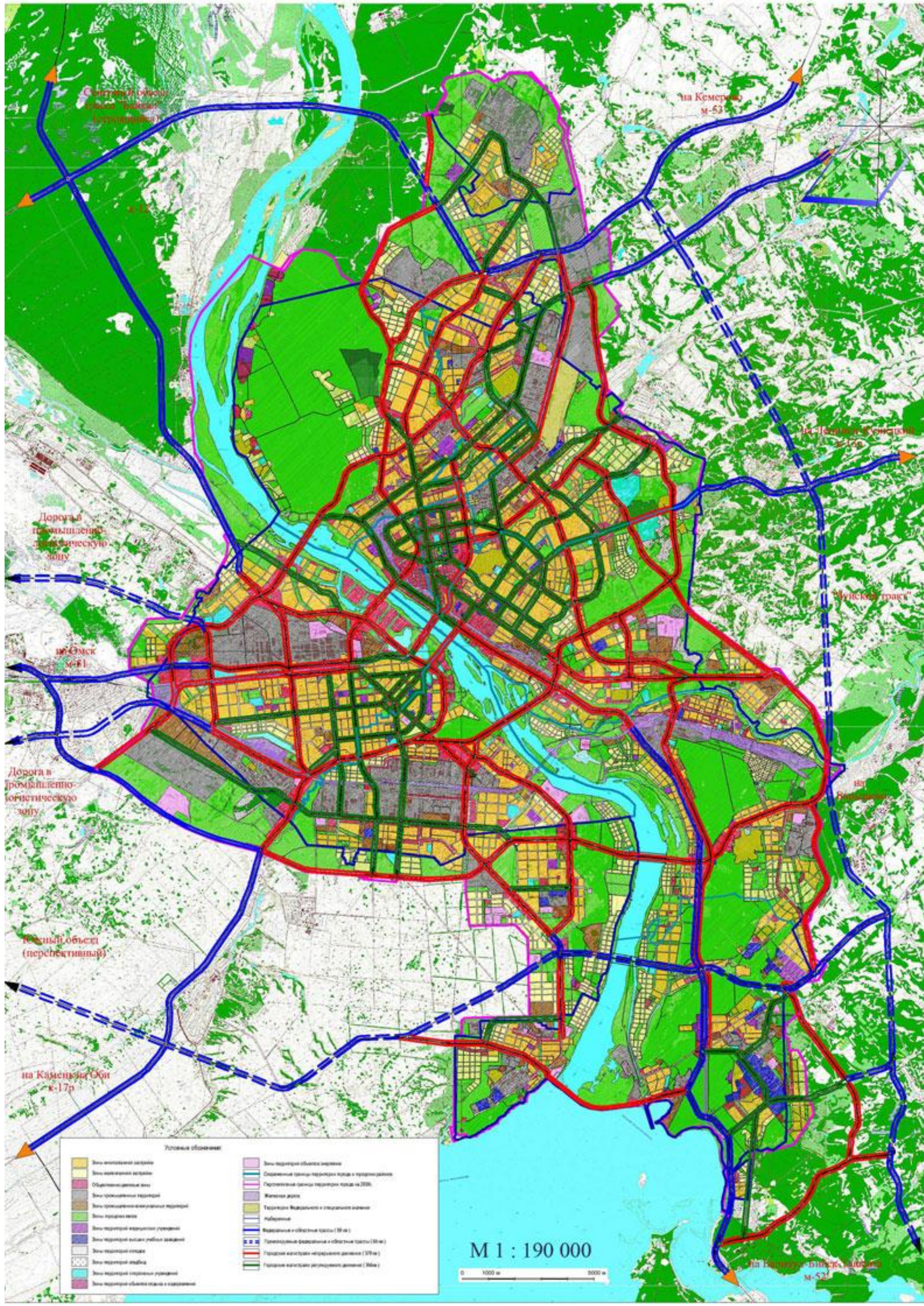
30 минут

Введение

Граф — это абстрактная структура данных, которая используется для реализации математической концепции графов. По сути, это набор вершин (также называемых узлами) и ребер, которые соединяют эти вершины. Граф часто рассматривается как обобщение древовидной структуры, где вместо чисто родительско-дочерних отношений между узлами дерева может существовать любой вид сложных отношений.

Почему графы полезны? Графы широко используются для моделирования любой ситуации, когда сущности или вещи связаны друг с другом парами. Например, следующая информация может быть представлена графами:

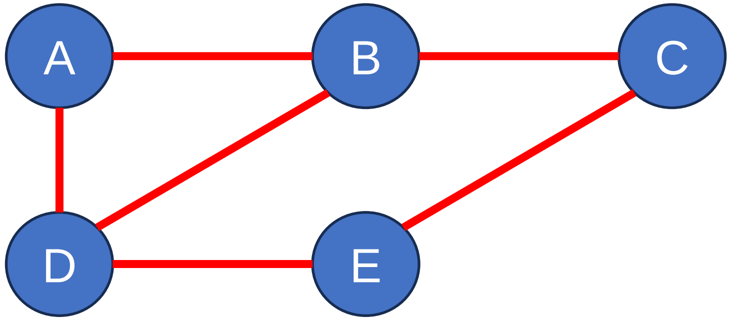
- Семейные деревья, в которых узлы-члены имеют ребро от родителя к каждому из своих дочерних узлов.
- Транспортные сети, в которых узлами являются аэропорты, перекрестки, порты и т. д. Ребра могут быть авиарейсами, односторонними дорогами, судоходными маршрутами и т. д.



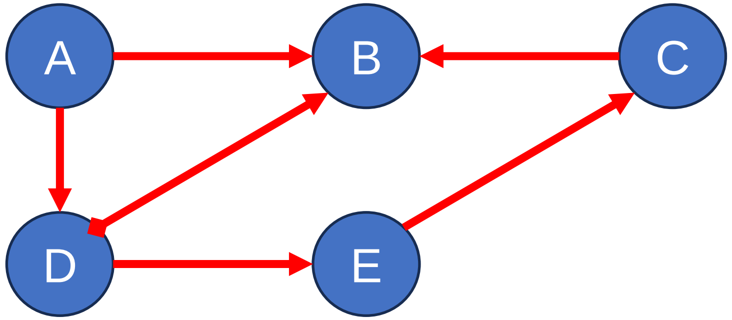
Как определяется граф

Граф G определяется как упорядоченное множество (V, E), где V(G) представляет множество вершин, а E(G) представляет ребра, соединяющие эти вершины. На рисунке показан граф с V(G) = {A, B, C, D и E} и E(G) = {(A, B), (B, C), (A, D), (B, D), (D, E), (C, E)}. Обратите внимание, что в графе пять вершин или узлов и шесть ребер.

Граф бывает неориентированным и ориентированным



Неориентированный граф



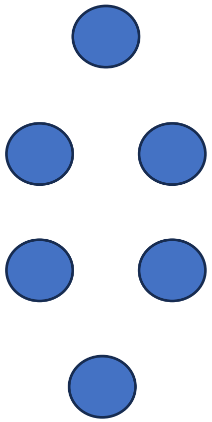
Направленный граф

Терминология графа

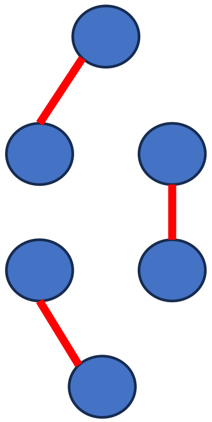
Смежные узлы или соседи Для каждого ребра e = (u, v), соединяющего узлы u и v, узлы u и v являются конечными точками и называются смежными узлами или соседями.

Степень узла Степень узла u, deg(u), — это общее количество ребер, содержащих узел u. Если deg(u) = 0, это означает, что u не принадлежит ни одному ребру, и такой узел называется изолированным узлом.

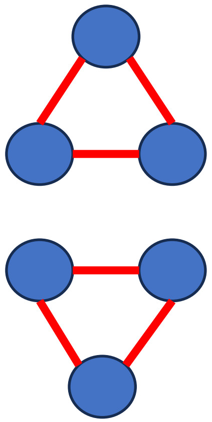
Регулярный граф Это граф, в котором каждая вершина имеет одинаковое количество соседей. То есть каждый узел имеет одинаковую степень. Регулярный граф с вершинами степени k называется k–регулярным графом или регулярным графом степени k.



(0-регулярный граф)



(1-регулярный граф)



(2-регулярный граф)

Терминология графа

Путь Путь P, записанный как $P = \{v_0, v_1, v_2, ..., v_n\}$, длины n от узла u до v, определяется как последовательность из (n+1) узлов. Здесь $u = v_0$, $v = v_n$ и v_{i-1} смежно с v_i для i = 1, 2, 3, ..., n.

Замкнутый путь Путь P называется замкнутым путем, если ребро имеет одинаковые конечные точки. То есть, если $v_0 = v_n$.

Простой путь Путь P называется простым путем, если все узлы в пути различны, за исключением того, что v0 может быть равен v_n. Если $v_0 = v_n$, то путь называется замкнутым простым путем.

Цикл Путь, в котором первая и последняя вершины одинаковы. Простой цикл не имеет повторяющихся ребер или вершин (кроме первой и последней вершин).

Кликовый граф В неориентированном графе G = (V, E) клики — это подмножество множества вершин $C \subseteq V$, такое, что для каждых двух вершин в C существует ребро, соединяющее две вершины.

Петля Ребро, которое имеет идентичные конечные точки, называется петлей. То есть e = (u, u).

Размер графа Размер графа — это общее количество ребер в нем.

Терминология графа

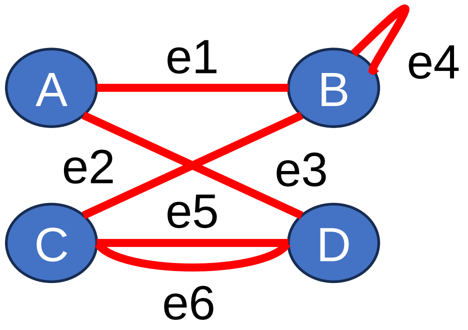
Связный граф Граф называется связным, если для любых двух вершин (u, v) в V существует путь от u до v. То есть в связном графе нет изолированных узлов. Связный граф, не имеющий цикла, называется деревом. Поэтому дерево рассматривается как особый граф.

Полный граф Граф G называется полным, если все его узлы полностью связаны. То есть существует путь от одного узла до каждого другого узла в графе. Полный граф имеет n(n–1)/2 ребер, где n — число узлов в G.

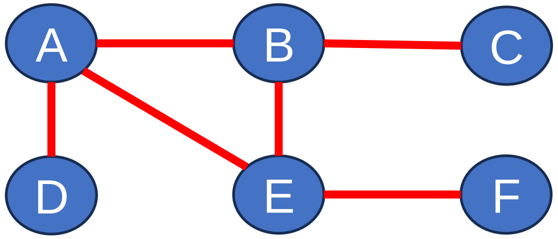
Помеченный граф или взвешенный граф Граф называется помеченным, если каждому ребру в графе назначены некоторые данные. В взвешенном графе ребрам графа назначен некоторый вес или длина. Вес ребра, обозначенный как w(e), является положительным значением, которое указывает стоимость прохождения ребра.

Множественные ребра Различные ребра, которые соединяют одни и те же конечные точки, называются множественными ребрами. То есть e = (u, v) и e' = (u, v) известны как множественные ребра G.

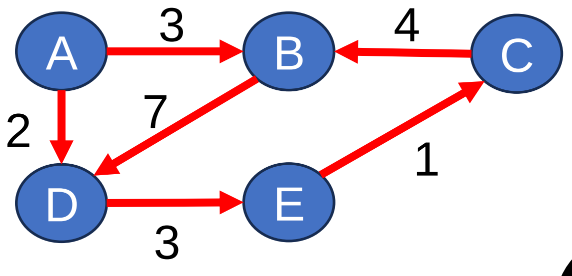
Мультиграф Граф с множественными ребрами и/или петлями называется мультиграфом.



(a) Мультиграф



(b) Дерево



(c) Взвешенный граф

Терминология направленного графа

Исходная степень узла Исходная степень узла u, записанная как outdeg(u), — это количество ребер, которые начинаются в u.

Входящая степень узла Входящая степень узла u, записанная как indeg(u), — это количество ребер, которые заканчиваются в u.

Степень узла Степень узла, записанная как deg(u), равна сумме входящей и исходящей степени этого узла. Следовательно, deg(u) = indeg(u) + outdeg(u).

Изолированная вершина Вершина с нулевой степенью. Такая вершина не является конечной точкой ни одного ребра.

Висячая вершина (также известная как вершина листа) Вершина с единичной степенью.

Срезанная вершина Вершина, удаление которой разорвет оставшийся граф.

Источник Узел u называется источником, если он имеет положительную исходящую степень, но нулевую входящую степень.

Сток Узел u называется стоком, если он имеет положительную входящую степень, но нулевую исходящую степень.

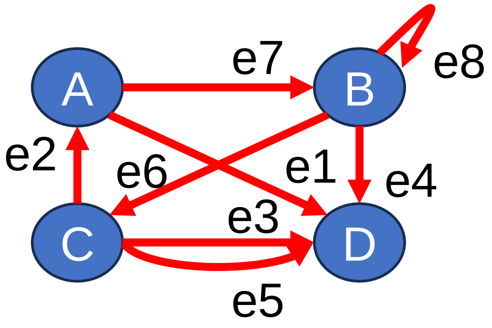
Сильно связанный ориентированный граф Орграф называется сильно связанным, если и только если существует путь между каждой парой узлов в G. То есть, если существует путь от узла u до v, то должен быть путь от узла v до u.

Односторонне связанный граф Орграф называется односторонне связанным, если существует путь между любой парой узлов u, v в G такой, что существует путь от u до v или путь от v до u, но не оба.

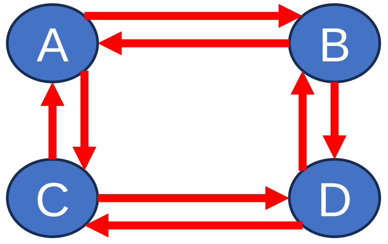
Терминология направленного графа

Параллельные/кратные ребра Различные ребра, которые соединяют одни и те же конечные точки, называются кратными ребрами. То есть e = (u, v) и e' = (u, v) известны как кратные ребра G.

Простой направленный граф Направленный граф G называется простым направленным графом тогда и только тогда, когда он не имеет параллельных ребер. Однако простой направленный граф может содержать циклы, за исключением того, что он не может иметь более одной петли в данном узле.



(a) Направленный ациклический граф



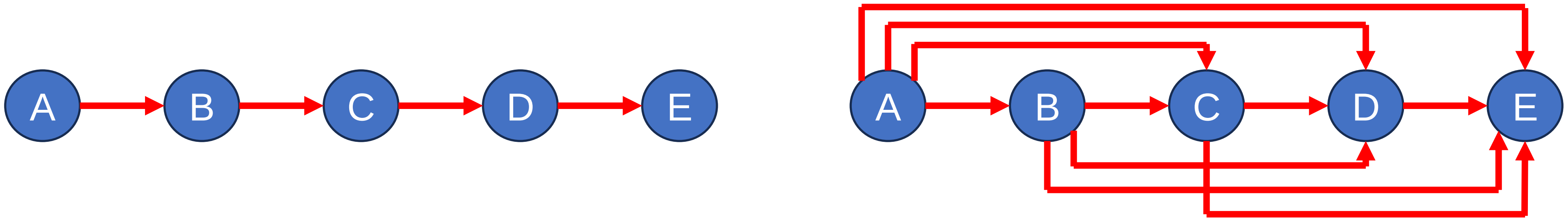
(b) сильно связанный направленный ациклический граф

Транзитивное замыкание направленного графа

Определение Для направленного графа G = (V,E), где V — множество вершин, а E — множество ребер, транзитивное замыкание G — это граф G* = (V,E*). В G* для каждой пары вершин v, w в V существует ребро (v, w) в E* тогда и только тогда, когда существует допустимый путь от v до w в G.

Где и зачем это нужно? Нахождение транзитивного замыкания направленного графа является важной проблемой в следующих вычислительных задачах:

- Транзитивное замыкание используется для поиска анализа достижимости сетей переходов, представляющих распределенные и параллельные системы.
- Оно используется при построении автоматов синтаксического анализа при построении компилятора.
- В последнее время вычисление транзитивного замыкания используется для оценки рекурсивных запросов к базе данных (потому что почти все практические рекурсивные запросы являются транзитивными по своей природе).



(a) Граф G

(b) транзитивное замыкание G*

Транзитивное замыкание направленного графа – алгоритм

```
void transitiveClosure(int **A, int **t, int n)
{
    int i = 0, j = 0, k = 0;
    for (; i < n; i++)
        for (; j < n; j++)
        {
            if (A[i][j] == 1)
                t[i][j] = 1;
            else
                t[i][j] = 0;
        }
    i = 0, j = 0;
    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                t[i][j] = t[i][j] || (t[i][k] && t[k][j]);
}
```

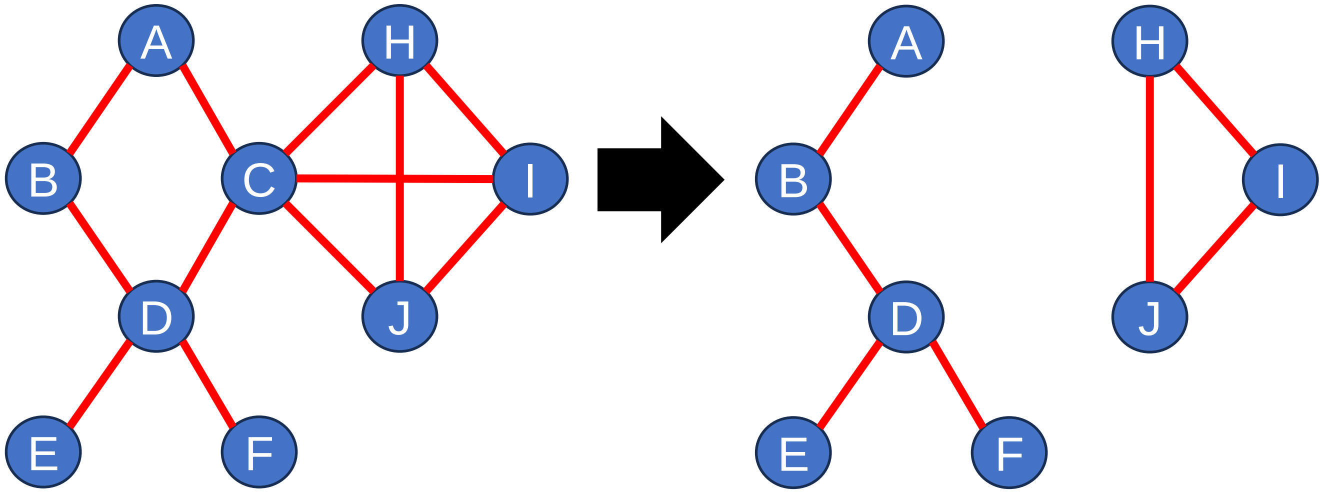
$$\left[\begin{array}{l} \text{если } k = 0\text{: } T_{ij}^{0} = \begin{cases} 0 \text{ если } (i,j) \text{ не в } E \\ 1 \text{ если } (i,j) \text{ в } E \end{cases} \\ \text{если } k \geq 1\text{: } T_{ij}^{k} = T_{ij}^{k-1} \cup \left(T_{ij}^{k-1} \cap T_{ij}^{k-1}\right) \end{array}\right.$$

Двусвязные компоненты

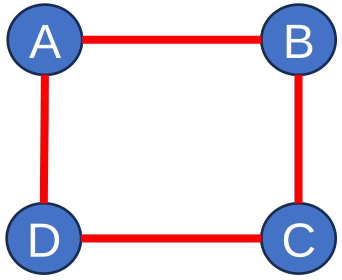
Вершина v графа G называется ***точкой сочленения***, если удаление v вместе с ребрами, инцидентными v, приводит к графу, который имеет по крайней мере два связных компонента. ***Двусвязный граф*** определяется как связный граф, который не имеет вершин сочленения.

Двусвязный неориентированный граф — это связный граф, который нельзя разбить на несвязные части путем удаления любой одной вершины.

В двусвязном ориентированном графе для любых двух вершин v и w существуют два направленных пути из v в w, которые не имеют общих вершин, кроме v и w.



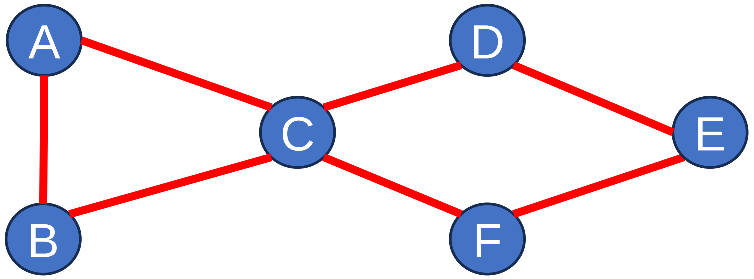
Не двусвязный граф



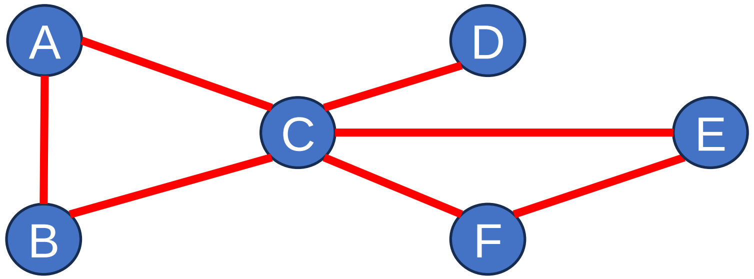
Двусвязный граф

Двусвязные компоненты

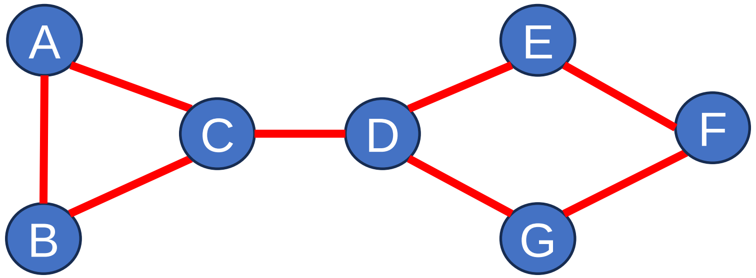
Что касается вершин, то для ребер существует связанное понятие. Ребро в графе называется мостом, если удаление этого ребра приводит к несвязному графу. Кроме того, ребро в графе, которое не лежит на цикле, является мостом. Это означает, что мост имеет по крайней мере одну точку сочленения на своем конце, хотя не обязательно, чтобы точка сочленения была связана с мостом.



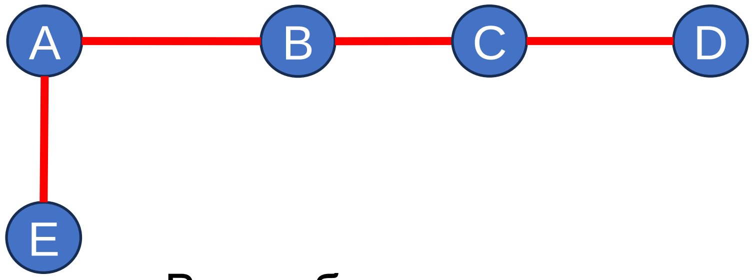
Нет мостов



CD — мост



CD — мост



Все ребра — мосты

Представление графов

Существует три распространенных способа хранения графов в памяти компьютера. Это:

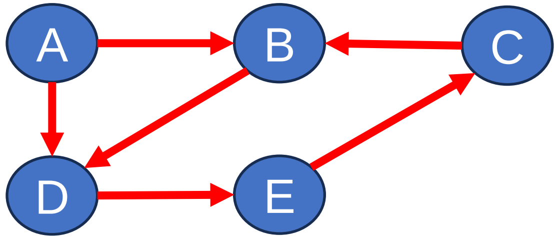
- Последовательное представление с использованием матрицы смежности
- Связанное представление с использованием списка смежности, который хранит соседей узла с помощью связанного списка.
- Мультисписок смежности, который является расширением связанного представления.

Матрица смежности

Матрица смежности используется для представления того, какие узлы смежны друг с другом. По определению, два узла считаются смежными, если их соединяет ребро.

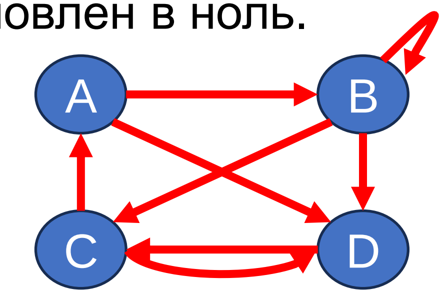
В направленном графе G, если узел v смежный с узлом u, то определенно есть ребро от u к v. То есть, если v смежный с u, мы можем добраться от u к v, пройдя по одному ребру. Для любого графа G, имеющего n узлов, матрица смежности будет иметь размерность $n \times n$.

В матрице смежности строки и столбцы помечены вершинами графа. Элемент a_{ij} в матрице смежности будет содержать 1, если вершины v_i и v_j смежны друг с другом. Однако, если узлы не смежны, a_{ij} будет установлен в ноль.



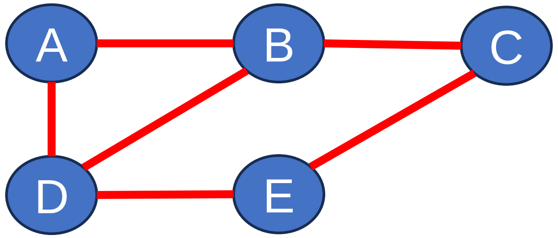
$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

(a) Направленный граф



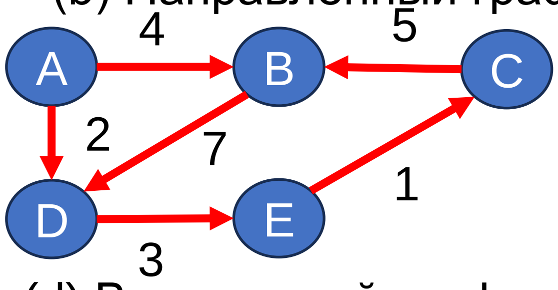
$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

(b) Направленный граф с петлей



$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

(c) Ненаправленный граф



$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{bmatrix} 0 & 4 & 0 & 2 & 0 \\ 0 & 0 & 0 & 7 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

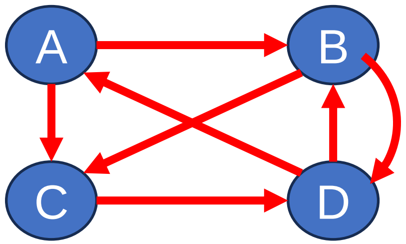
(d) Взвешенный граф

Матрица смежности

Матрица смежности используется для представления того, какие узлы смежны друг с другом. По определению, два узла считаются смежными, если их соединяет ребро.

В направленном графе G, если узел v смежный с узлом u, то определенно есть ребро от u к v. То есть, если v смежный с u, мы можем добраться от u к v, пройдя по одному ребру. Для любого графа G, имеющего n узлов, матрица смежности будет иметь размерность $n \times n$.

В матрице смежности строки и столбцы помечены вершинами графа. Элемент a_{ij} в матрице смежности будет содержать 1, если вершины v_i и v_j смежны друг с другом. Однако, если узлы не смежны, a_{ij} будет установлен в ноль.



$$
\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}
\begin{array}{c}
\begin{array}{cccc} A & B & C & D \end{array} \\
\begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}
\end{array}
$$

Направленный граф с его матрицей смежности

Матрица смежности

Мощности матрицы смежности. Из матрицы смежности A^1 можно сделать вывод, что запись 1 в i-й строке и j-м столбце означает, что существует путь длины 1 от v_i до v_j. Теперь рассмотрим A^2, A^3 и A^4.

$$a_{ij}^2 = \sum a_{ik} a_{kj}$$

$$A^2 = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 2 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 2 & 1 \end{bmatrix}$$

$$A^3 = \begin{bmatrix} 0 & 0 & 1 & 2 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 2 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 0 & 1 \\ 1 & 2 & 2 & 1 \\ 0 & 1 & 2 & 1 \\ 1 & 2 & 2 & 3 \end{bmatrix}$$

$$A^4 = \begin{bmatrix} 2 & 2 & 0 & 1 \\ 1 & 2 & 2 & 1 \\ 0 & 1 & 2 & 1 \\ 1 & 2 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 4 & 2 \\ 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 3 \\ 3 & 4 & 3 & 4 \end{bmatrix}$$

Теперь, определяем матрицу B как: $B^r = A^1 + A^2 + A^3 + \ldots + A^r$

$$\mathrm{B} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 & 2 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 2 & 1 \end{bmatrix} + \begin{bmatrix} 2 & 2 & 0 & 1 \\ 1 & 2 & 2 & 1 \\ 0 & 1 & 2 & 1 \\ 1 & 2 & 2 & 3 \end{bmatrix} + \begin{bmatrix} 1 & 3 & 4 & 2 \\ 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 3 \\ 3 & 4 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 6 & 5 \\ 3 & 5 & 6 & 7 \\ 2 & 3 & 3 & 5 \\ 6 & 8 & 7 & 8 \end{bmatrix}$$

P_{ij}

если есть путь от до v_i до v_j

иначе

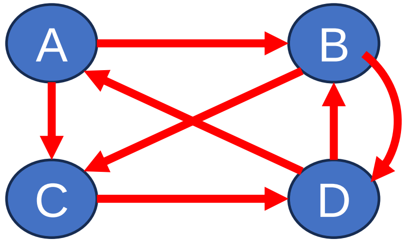
$$P = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

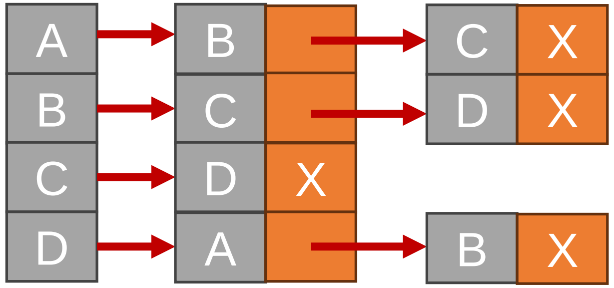
Список смежности

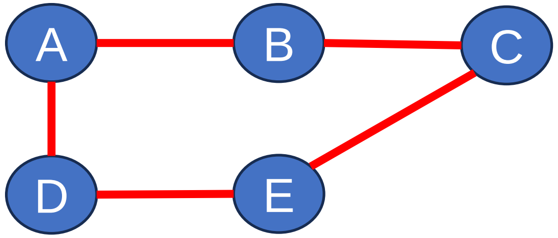
Список смежности — это еще один способ представления графов в памяти компьютера. Эта структура состоит из списка всех узлов в G. Кроме того, каждый узел, в свою очередь, связан со своим собственным списком, который содержит имена всех других узлов, смежных с ним. Основные преимущества использования списка смежности:

- Он прост для понимания и четко показывает смежные узлы конкретного узла.
- Он часто используется для хранения графов с небольшим или средним количеством ребер. То есть список смежности предпочтителен для представления разреженных графов в памяти компьютера; в противном случае хорошим выбором будет матрица смежности.
- Добавление новых узлов в G легко и просто, когда G представлен с помощью списка смежности. Добавление новых узлов в матрицу смежности — сложная задача, так как необходимо изменить размер матрицы и, возможно, придется переупорядочить существующие узлы.

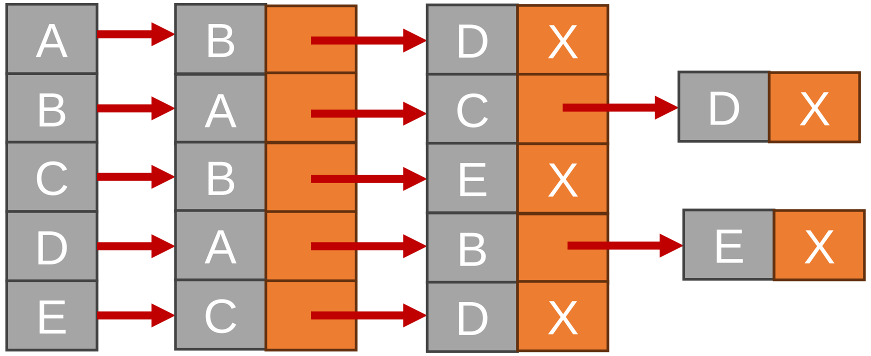
Список смежности

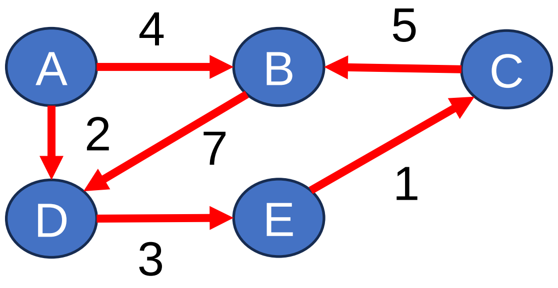




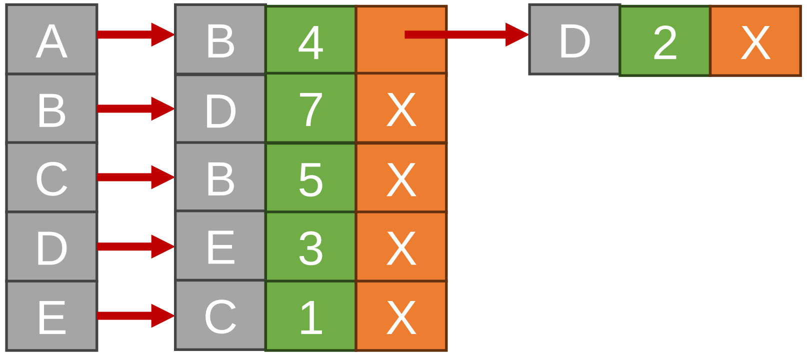


Неориентированный граф





Взвешенный граф



Мультисписок смежности

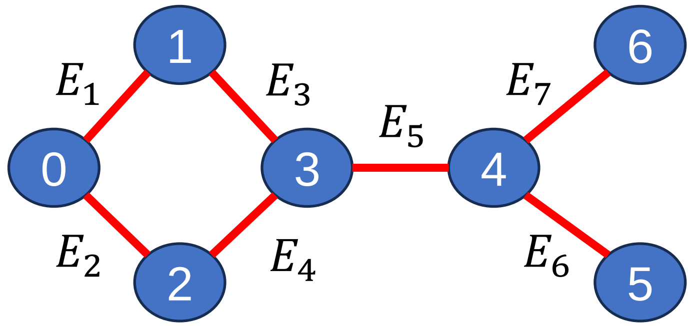
Графы также могут быть представлены с помощью мультисписков, которые можно назвать модифицированной версией списков смежности.

Мультисписок смежности — это представление графов на основе ребер, а не вершин. Мультисписочное представление в основном состоит из двух частей — каталога информации об узлах и набора связанных списков, хранящих информацию о ребрах. Хотя в каталоге узлов есть одна запись для каждого узла, с другой стороны, каждый узел появляется в двух списках смежности (по одному для узла на каждом конце ребра).

Например, запись каталога для узла i указывает на список смежности для узла i. Это означает, что узлы совместно используются несколькими списками.

В мультисписочном представлении информация о ребре (v_i, v_j) неориентированного графа может храниться с использованием следующих атрибутов: M: однобитовое поле для указания того, было ли ребро проверено или нет.

Мультисписок смежности



v_i: Вершина в графе, которая соединена с вершиной v_j ребром.
v_j: Вершина в графе, которая соединена с вершиной v_i ребром.
Ссылка i для v_i: Ссылка, которая указывает на другой узел, имеющий ребро, инцидентное v_i.
Ссылка j для v_j: Ссылка, которая указывает на другой узел, имеющий ребро, инцидентное v_j.

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

Вершины	Список ребер
0	Ребро 1, Ребро 2
1	Ребро 1, Ребро 3
2	Ребро 2, Ребро 4
3	Ребро 3, Ребро 4, Ребро 5
4	Ребро 5, Ребро 6, Ребро 7
5	Ребро 6
6	Ребро 7

Алгоритмы обхода графа

Как обходить графы?

Под обходом графа мы подразумеваем метод изучения узлов и ребер графа. Существует два стандартных метода обхода графа, которые мы обсудим в этом разделе. Эти два метода:

1. Поиск в ширину
2. Поиск в глубину

В то время как поиск в ширину использует очередь в качестве вспомогательной структуры данных для хранения узлов для дальнейшей обработки, схема поиска в глубину использует стек. Но оба эти алгоритма используют переменную STATUS. Во время выполнения алгоритма каждый узел в графе будет иметь переменную STATUS, установленную на 1 или 2, в зависимости от его текущего состояния.

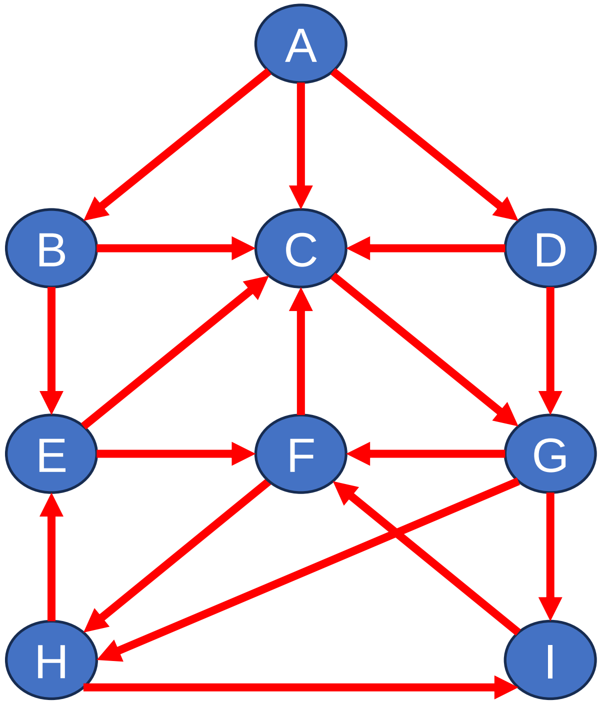
Статус	Состояние узла	Описание
1	Готов	Начальное состояние узла N
2	Ожидание	Узел N помещается в очередь или стек и ожидает обработки
3	Обработан	Узел N полностью обработан

Алгоритм поиска в ширину

Поиск в ширину (BFS) — это алгоритм поиска по графу, который начинается с корневого узла и исследует все соседние узлы. Затем для каждого из этих ближайших узлов алгоритм исследует их неисследованные соседние узлы и так далее, пока не найдет цель.

Шаг	Описание
1	УСТАНОВИТЕ СТАТУС = 1 (состояние готовности) для каждого узла в G
2	Поставьте в очередь начальный узел A и установите его СТАТУС = 2 (состояние ожидания)
3	Повторяйте шаги 4 и 5, пока ОЧЕРЕДЬ не станет пустой
4	Уберите из очереди узел N. Обработайте его и установите его СТАТУС = 3 (состояние обработки).
5	Поставить в очередь всех соседей N, которые находятся в состоянии готовности (чей STATUS = 1), и установить их STATUS = 2 (состояние ожидания) [КОНЕЦ ЦИКЛА]
6	ВЫХОД

Алгоритм поиска в ширину - пример



Список смежности

A: B, C, D
B: E
C: B,G
D: C, G
E: F
G: F, H, I
H: E, I
I: F

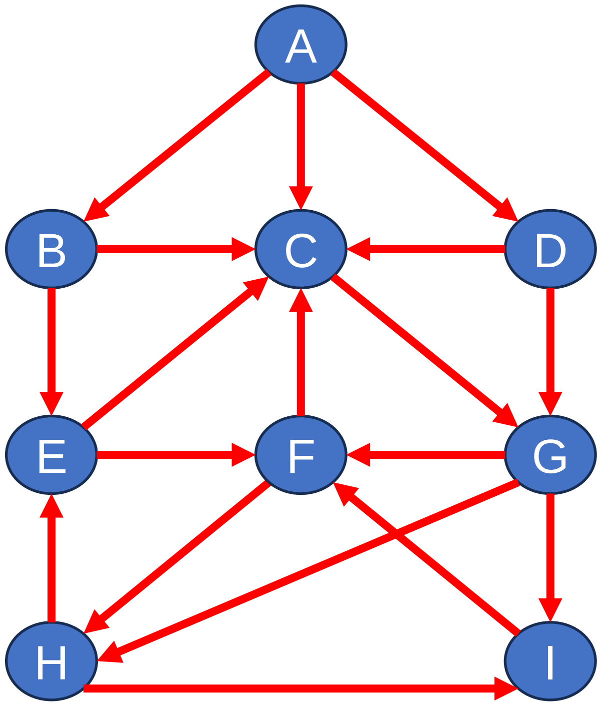
Дано:

Рассмотрим граф G и список смежности G. Предположим, что G представляет ежедневные рейсы между разными городами, и мы хотим лететь из города A в I с минимальным количеством остановок. То есть найти минимальный путь P из A в I, учитывая, что каждое ребро имеет длину 1.

Решение

Минимальный путь P можно найти, применив алгоритм поиска в ширину, который начинается в городе A и заканчивается, когда встречается I.

Алгоритм поиска в ширину - решение



Список смежности

A: B, C, D
B: E
C: B,G
D: C, G
E: F
G: F, H, I
H: E, I
I: F

Во время выполнения алгоритма мы используем два массива: QUEUE и ORIG. В то время как QUEUE используется для хранения узлов, которые должны быть обработаны, ORIG используется для отслеживания начала каждого ребра. Изначально FRONT = REAR = –1.

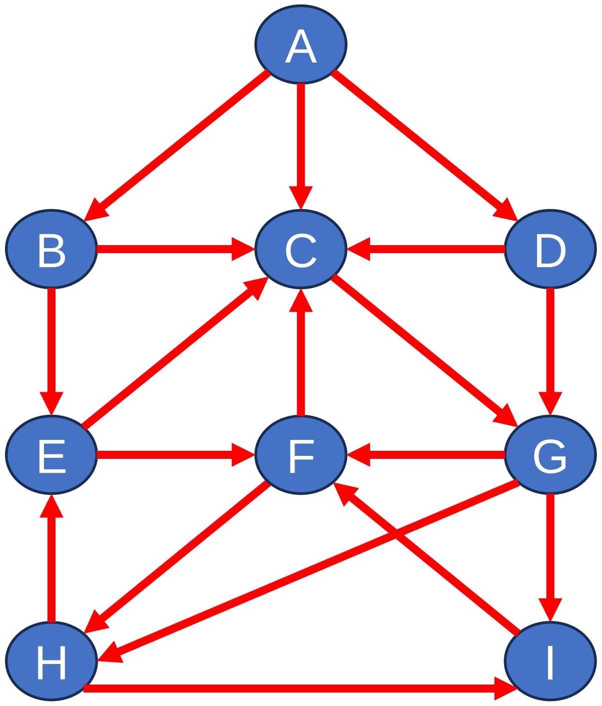
(a) Добавить A к QUEUE и добавить NULL к ORIG.

FRONT = 0	QUEUE = A
REAR = 0	ORIG = \0

(b) Исключите узел из очереди, установив FRONT = FRONT + 1 (удалите элемент FRONT из QUEUE) и поставьте в очередь соседей A. Также добавьте A как ORIG его соседей.

FRONT = 1	QUEUE = A B C D
REAR = 3	ORIG = \0 A A A

Алгоритм поиска в ширину - решение



Список смежности

A: B, C, D
B: E
C: B,G
D: C, G
E: F
G: F, H, I
H: E, I
I: F

(c) Исключите узел из очереди, установив FRONT = FRONT + 1, и поставьте в очередь соседей B. Также добавьте B как ORIG его соседей.

FRONT = 2	QUEUE = A B C D E
REAR = 4	ORIG = \0 A A A B

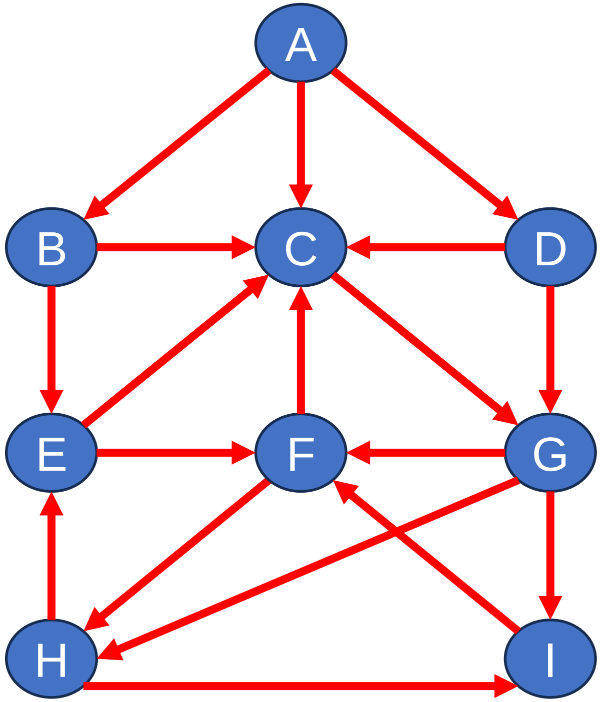
(d) Исключите узел из очереди, установив FRONT = FRONT + 1, и поставьте в очередь соседей C. Также добавьте C в качестве ORIG его соседей. Обратите внимание, что у C есть два соседа B и G. Поскольку B уже добавлен в очередь и не находится в состоянии Ready, мы не будем добавлять B, а добавим только G.

FRONT = 3	QUEUE = A B C D E G
REAR = 5	ORIG = \0 A A A B C

(e) Исключите узел из очереди, установив FRONT = FRONT + 1, и поставьте в очередь соседей D. Также добавьте D в качестве ORIG его соседей. Обратите внимание, что у D есть два соседа C и G. Поскольку оба они уже добавлены в очередь и не находятся в состоянии готовности, мы не будем добавлять их снова***.***

FRONT = 4	QUEUE = A B C D E G
REAR = 5	ORIG = \0 A A A B C

Алгоритм поиска в ширину - решение



Список смежности

A: B, C, D
B: E
C: B,G
D: C, G
E: F
G: F, H, I
H: E, I
I: F

(f) Исключите узел из очереди, установив FRONT = FRONT + 1, и добавьте в очередь соседей E. Также добавьте E как ORIG его соседей. Обратите внимание, что у E есть два соседа C и F. Поскольку C уже добавлен в очередь и не находится в состоянии готовности, мы не будем добавлять C, а добавим только F.

FRONT = 5	QUEUE = A B C D E G F
REAR = 6	ORIG = \0 A A A B C E

(g) Исключите узел из очереди, установив FRONT = FRONT + 1, и поставьте в очередь соседей G. Также добавьте G как ORIG его соседей. Обратите внимание, что у G есть три соседа F, H и I.

FRONT = 6	QUEUE = A B C D E G F H I
REAR = 9	ORIG = \0 A A A B C E G G

Поскольку F уже добавлен в очередь, мы добавим только H и I. Поскольку I — наш конечный пункт назначения, мы останавливаем выполнение этого алгоритма, как только он будет обнаружен и добавлен в ОЧЕРЕДЬ. Теперь вернемся от I с помощью ORIG, чтобы найти минимальный путь P. ***Таким образом, P выглядит как A -> C -> G -> I.***

Особенности алгоритма BFS

Сложность пространства

Для графа с коэффициентом ветвления b (количество потомков в каждом узле) и глубиной d асимптотическая сложность пространства равна количеству узлов на самом глубоком уровне $O(b^d)$. Если количество вершин и ребер в графе известно заранее, сложность пространства также может быть выражена как O (| E | + | V |), где | E | — общее количество ребер в G, а | V | — количество узлов или вершин.

Временная сложность

В худшем случае поиск в ширину должен пройти по всем путям ко всем возможным узлам, поэтому временная сложность этого алгоритма асимптотически приближается к $O(b^d)$. Однако временную сложность также можно выразить как O(| E | + | V |), поскольку в худшем случае будут исследованы каждая вершина и каждое ребро.

Полнота

Поиск в ширину называется полным алгоритмом, потому что, если есть решение, поиск в ширину найдет его независимо от вида графа. Но в случае бесконечного графа, где нет возможного решения, он будет расходиться.

Особенности алгоритма BFS

Оптимальность

Поиск в ширину оптимален для графа с ребрами одинаковой длины, поскольку он всегда возвращает результат с наименьшим количеством ребер между начальным узлом и целевым узлом. Но, как правило, в реальных приложениях у нас есть взвешенные графы, у которых есть затраты, связанные с каждым ребром, поэтому цель, следующая за началом, не обязательно должна быть самой дешевой доступной целью.

Применение алгоритма поиска в ширину

Поиск в ширину можно использовать для решения многих задач, таких как:

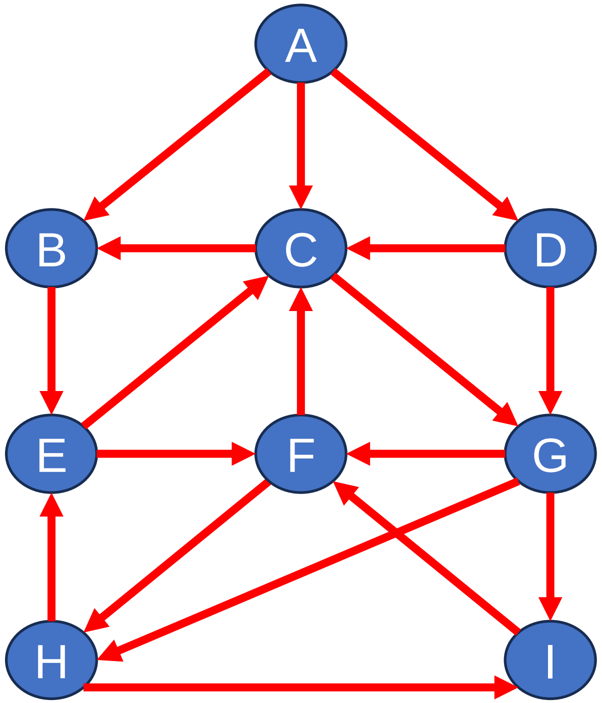
- Поиск всех связных компонентов в графе G.
- Поиск всех узлов в пределах отдельного связного компонента.
- Поиск кратчайшего пути между двумя узлами u и v невзвешенного графа.
- Поиск кратчайшего пути между двумя узлами u и v взвешенного графа.

Алгоритм поиска в глубину

Алгоритм поиска в глубину развивается путем расширения начального узла G, а затем все глубже и глубже, пока не будет найден целевой узел или пока не встретится узел, не имеющий потомков. Когда достигается тупик, алгоритм возвращается назад, возвращаясь к самому последнему узлу, который не был полностью исследован.

Шаг	Описание
1	УСТАНОВИТЕ STATUS = 1 (состояние готовности) для каждого узла в G
2	Поместите начальный узел A в стек и установите его STATUS = 2 (состояние ожидания)
3	Повторяйте шаги 4 и 5, пока STACK не опустеет
4	Извлеките верхний узел N. Обработайте его и установите его STATUS = 3 (состояние обработки)
5	Поместите в стек всех соседей N, которые находятся в состоянии готовности (чей STATUS = 1), и установите их STATUS = 2 (состояние ожидания) [КОНЕЦ ЦИКЛА]
6	ВЫХОД

Алгоритм поиска в глубину - пример

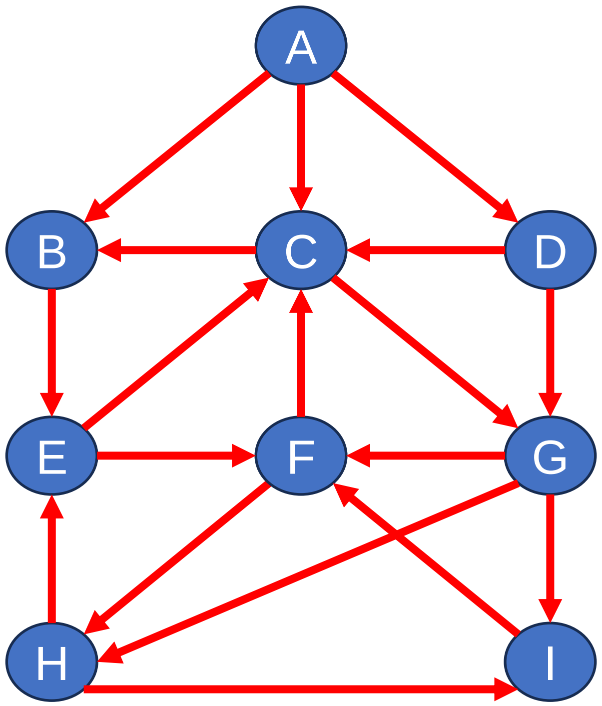


Список смежности

A: B, C, D
B: E
C: B,G
D: C, G
E: C, F
G: F, H, I
H: E, I
I: F

Дано:
Рассмотрим граф G и список смежности G. Предположим, что мы хотим вывести все узлы, которые могут быть достигнуты из узла H (включая сам H). Одной из альтернатив является использование поиска в глубину G, начиная с узла H.

Алгоритм поиска в ширину - решение



Список смежности

A: B, C, D
B: E
C: B,G
D: C, G
E: C, F
G: F, H, I
H: E, I
I: F

(a) Поместить H в стек.

STACK = H
PRINT

(b) Извлечь и вывести верхний элемент СТЕК, то есть H. Поместить всех соседей H в стек, которые находятся в состоянии готовности.

STACK = E I
PRINT H

(c) Извлечь и вывести верхний элемент СТЕК, то есть I. Поместить всех соседей I в стек, которые находятся в состоянии готовности.

STACK = E F
PRINT I

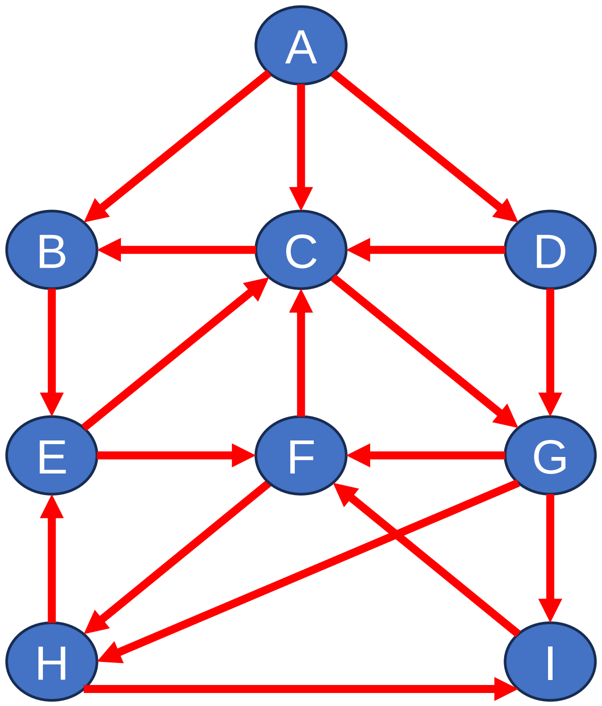
(d) Извлечь и напечатать верхний элемент СТЕК, то есть F. Поместить всех соседей F в стек, которые находятся в состоянии готовности. (H – не находится)

STACK = E C
PRINT F

(e) Извлечь и напечатать верхний элемент СТЕК, то есть C. Поместить всех соседей C в стек, которые находятся в состоянии готовности.

STACK = E B G
PRINT C

Алгоритм поиска в ширину - решение



Список смежности
A: B, C, D
B: E
C: B,G
D: C, G
E: C, F
G: F, H, I
H: E, I
I: F

(f) Извлечь и напечатать верхний элемент СТЕК, то есть G. Поместить всех соседей G в стек, которые находятся в состоянии готовности. Поскольку нет соседей G, которые находятся в состоянии готовности, операция вставки не выполняется.

STACK = E B
PRINT G

(g) Извлечь и напечатать верхний элемент СТЕК, то есть B. Поместить всех соседей B в стек, которые находятся в состоянии готовности.

STACK = E
PRINT B

(h) Извлечь и напечатать верхний элемент СТЕК, то есть E. Поместить всех соседей E в стек, которые находятся в состоянии готовности. Поскольку нет соседей E, которые находятся в состоянии готовности, операция push не выполняется.

STACK =
PRINT E

Поскольку STACK теперь пуст, поиск в глубину G, начинающийся с узла H, завершен, и напечатаны следующие узлы:
H, I, F, C, G, B, E Это узлы, достижимые из узла H.

Особенности алгоритма dfs

Пространственная сложность

Пространственная сложность поиска в глубину ниже, чем у поиска в ширину.

Временная сложность

Временная сложность поиска в глубину пропорциональна количеству вершин плюс количеству ребер в пройденных графах. Временная сложность может быть задана как (O(|V| + |E|)).

Полнота

Поиск в глубину называется полным алгоритмом. Если есть решение, поиск в глубину найдет его независимо от вида графа. Но в случае бесконечного графа, где нет возможного решения, он будет расходиться.

Применение алгоритма поиска в глубину

Поиск в глубину полезен для:

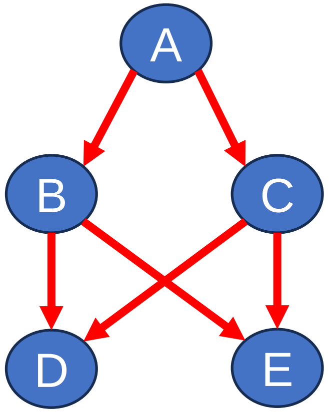
- Поиска пути между двумя указанными узлами u и v невзвешенного графа.
- Поиска пути между двумя указанными узлами u и v взвешенного графа.
- Поиска того, является ли граф связным или нет.
- Вычисления остовного дерева связного графа.

Топологическая сортировка

Топологическая сортировка направленного ациклического графа (DAG) G определяется как линейное упорядочение его узлов, в котором каждый узел предшествует всем узлам, к которым у него есть исходящие ребра. Каждый DAG имеет одно или несколько топологических сортировок.

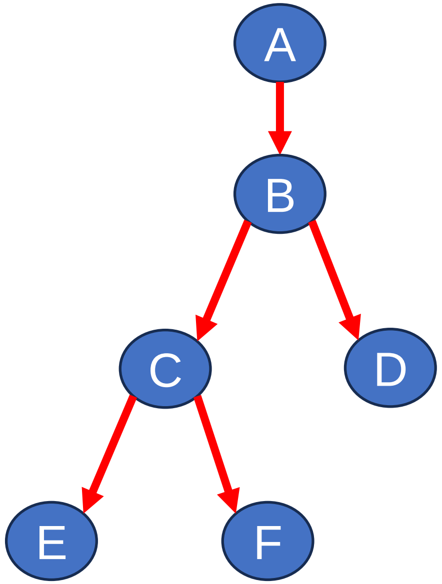
Топологическая сортировка DAG G — это упорядочение вершин G, такое что если G содержит ребро (u, v), то u появляется перед v в упорядочении. Обратите внимание, что топологическая сортировка возможна только для направленных ациклических графов, которые не имеют циклов. Для DAG, содержащего циклы, линейное упорядочение его вершин невозможно. Проще говоря, топологическое упорядочение DAG G — это упорядочение его вершин, такое что любой направленный путь в G проходит вершины в порядке возрастания. Топологическая сортировка широко используется при планировании приложений, заданий или задач. Задания, которые должны быть выполнены, представлены узлами, и есть ребро от узла u к v, если задание u должно быть выполнено до того, как задание v может быть запущено. Топологическая сортировка такого графа дает порядок, в котором данные задания должны быть выполнены.

Топологическая сортировка



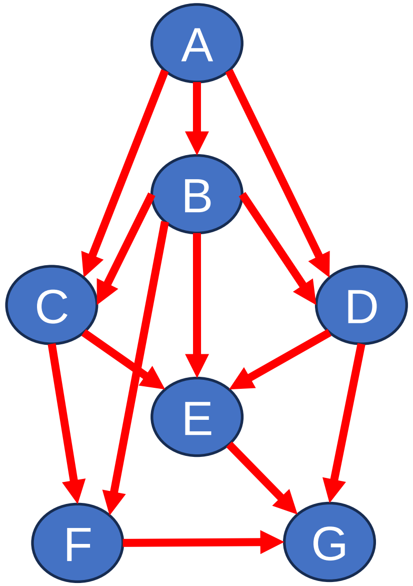
Топологическая сортировка может быть задана как:

- A, B, C, D, E
- A, B, C, E, D
- A, C, B, D, E
- A, C, B, E, D



Топологическая сортировка может быть задана как:

- A, B, D, C, E, F
- A, B, D, C, F, E
- A, B, C, D, E, F
- A, B, C, D, F, E



Топологическая сортировка может быть задана как:

- A, B, C, F, D, E, G
- A, B, C, D, E, F, G
- A, B, C, D, F, E, C
- A, B, D, C, E, F, G

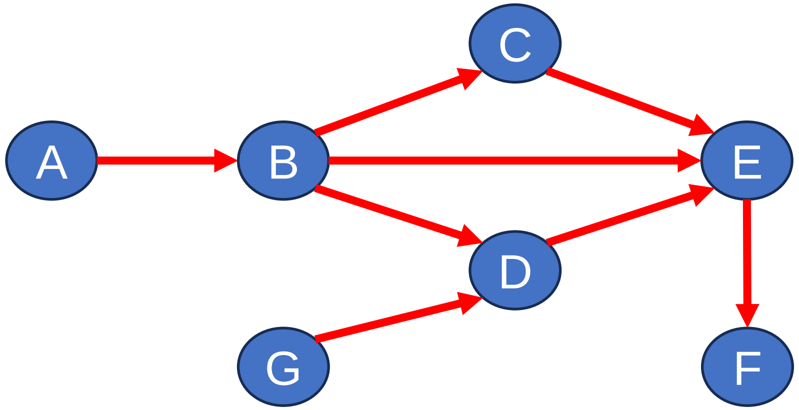
Топологическая сортировка

Алгоритм топологической сортировки графа, который не имеет циклов, фокусируется на выборе узла N с нулевой степенью захода, то есть узла, у которого нет предшественника. Два основных шага, включенных в алгоритм топологической сортировки, включают:

- Выбор узла с нулевой степенью захода
- Удаление N из графа вместе с его ребрами

Шаг	Описание
1	Найти степень захода INDEG(N) каждого узла в графе
2	Поставить в очередь все узлы с нулевой степенью захода
3	Повторить шаги 4 и 5, пока ОЧЕРЕДЬ не опустеет
4	Удалить передний узел N ОЧЕРЕДИ, установив FRONT = FRONT + 1 Повторить для каждого соседа M узла N:
5	a) Удалить ребро от N до M, установив INDEG(M) = INDEG(M) - 1 b) ЕСЛИ INDEG(M) = 0, то Поставить в очередь M, то есть добавить M в конец очереди [КОНЕЦ ВНУТРЕННЕГО ЦИКЛА] [КОНЕЦ ЦИКЛА]
6	ВЫХОД

Топологическая сортировка - пример



Список смежности
A: B
B: C, D, E
C: E
D: E
E: F
G: D

Рассмотрим ориентированный ациклический граф G, представленный на рисунке. Мы используем алгоритм, чтобы найти топологическую сортировку T графа G.

Шаг 1: Найдите входящую степень INDEG(N) каждого узла в графе

N	INDEG(N)
A	0
B	1
C	1
D	2
E	3
F	1
G	0

Шаг 2: Поставьте в очередь все узлы с нулевой входящей степенью

FRONT = 1	QUEUE = A, G
REAR = 2	

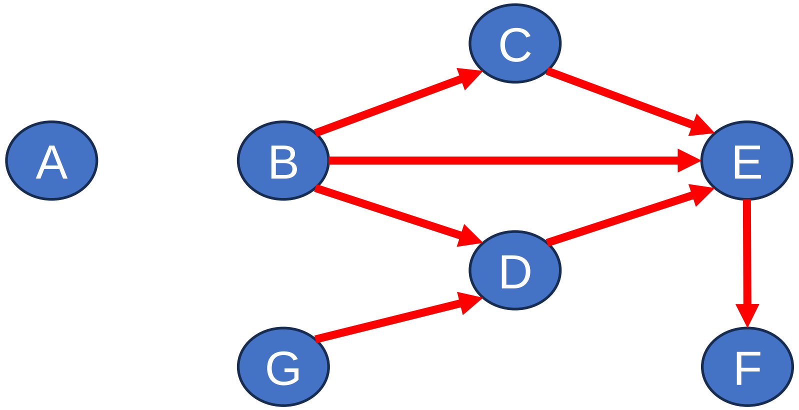
Шаг 3: Удалите передний элемент A из очереди, установив FRONT = FRONT + 1, поэтому

FRONT = 2	QUEUE = A, G
REAR = 2	

Шаг 4: Установите INDEG(B) = INDEG(B) – 1, так как B является соседом A. Обратите внимание, что INDEG(B) равен 0, поэтому добавьте его в очередь.

FRONT = 2	QUEUE = A, G, B
REAR = 3	

Топологическая сортировка - пример



Список смежности
A: B
B: C, D, E
C: E
D: E
E: F
G: D

Удалите ребро от A до B.

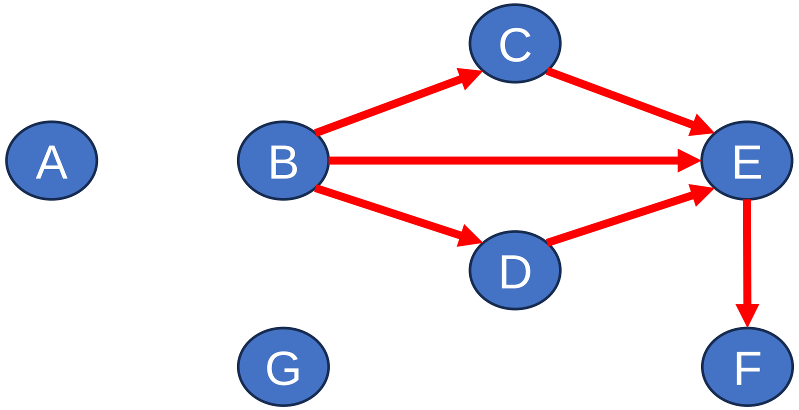
Шаг 5: Удалите передний элемент G из очереди, установив FRONT = FRONT + 1

FRONT = 3	QUEUE = A, G, B
REAR = 3	

N	INDEG(N)
A	0
B	0
C	1
D	1
E	3
F	1
G	0

Шаг 6: Установите INDEG(D) = INDEG(D) – 1, так как D является соседом G.

Топологическая сортировка - пример



Список смежности
A: B
B: C, D, E
C: E
D: E
E: F
G: D

Удалите ребро от G до D.

Шаг 7: Удалите передний элемент B из очереди, установив FRONT = FRONT + 1

FRONT = 4	QUEUE = A, G, B
REAR = 3	

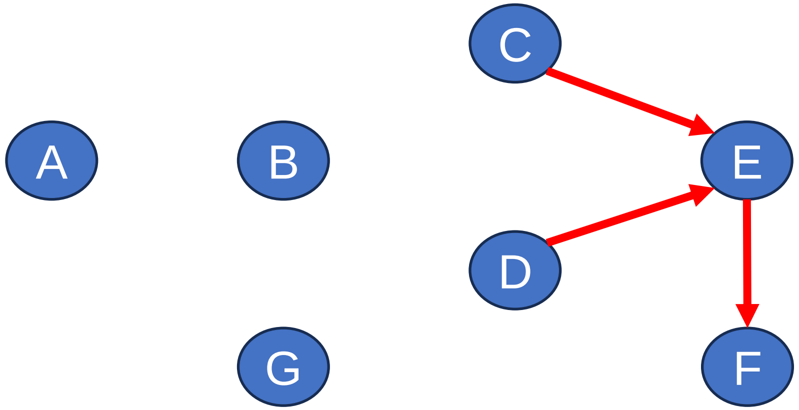
Шаг 8: Установите INDEG(C) = INDEG(C) – 1, INDEG(D) = INDEG(D) – 1, INDEG(E) = INDEG(E) – 1, так как C, D и E являются соседями B.

Шаг 9: Поскольку входящая степень узлов C и D равна нулю, добавьте C и D в конец очереди.

FRONT = 4	QUEUE = A, G, B, C, D
REAR = 5	

N	INDEG(N)
A	0
B	0
C	0
D	0
E	2
F	1
G	0

Топологическая сортировка - пример



Список смежности

A: B
B: C, D, E
C: E
D: E
E: F
G: D

Удалите ребра от B до C, E, D.

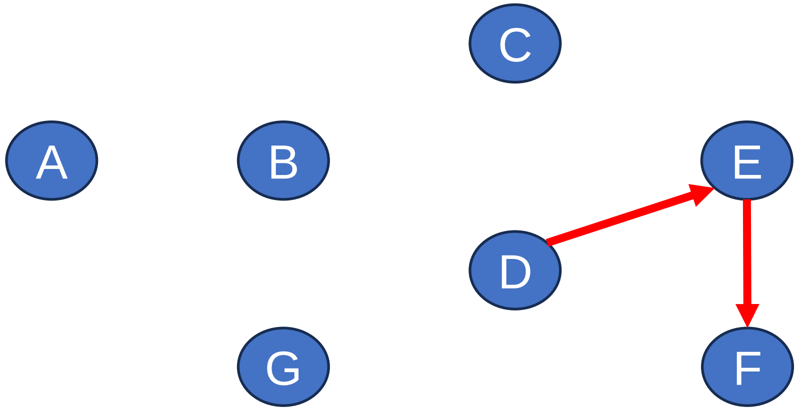
Шаг 10: Удалите передний элемент C из очереди, установив FRONT = FRONT + 1

FRONT = 5	QUEUE = A, G, B, C, D
REAR = 5	

Шаг 11: Установите INDEG(E) = INDEG(E) – 1, поскольку E является соседом C. Теперь INDEG(E) = 1

N	INDEG(N)
A	0
B	0
C	0
D	0
E	1
F	1
G	0

Топологическая сортировка - пример



Список смежности
A: B
B: C, D, E
C: E
D: E
E: F
G: D

Удалите ребра от C до E.

Шаг 12: Удалите передний элемент D из очереди, установив FRONT = FRONT + 1

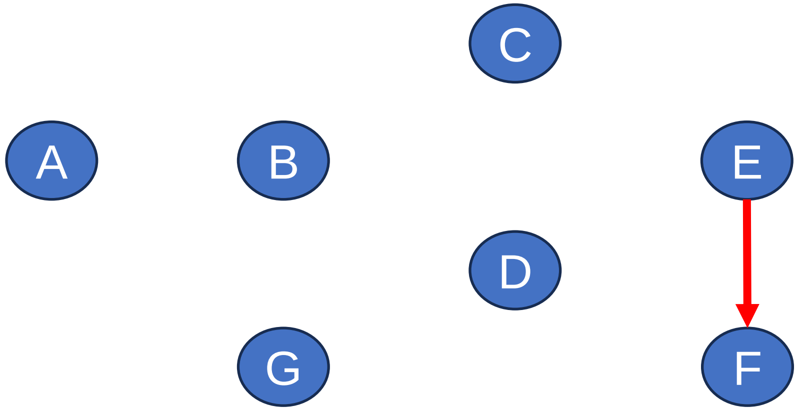
FRONT = 6	QUEUE = A, G, B, C, D
REAR = 5	

Шаг 13: Установите INDEG(E) = INDEG(E) – 1, так как E является соседом D. Теперь INDEG(E) = 0, поэтому добавьте E в очередь. Теперь очередь становится.

FRONT = 6	QUEUE = A, G, B, C, D, E
REAR = 6	

N	INDEG(N)
A	0
B	0
C	0
D	0
E	0
F	1
G	0

Топологическая сортировка - пример



Список смежности
A: B
B: C, D, E
C: E
D: E
E: F
G: D

Удалите ребра от D до E.

Шаг 14: Удалите передний элемент D из очереди, установив FRONT = FRONT + 1, поэтому

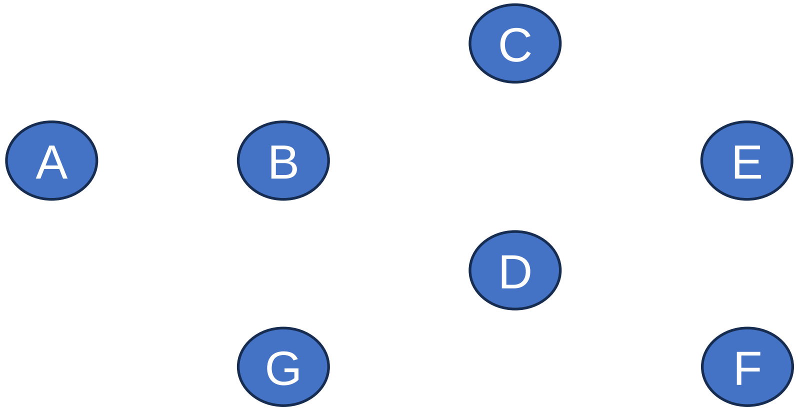
FRONT = 7	QUEUE = A, G, B, C, D, E, F
REAR = 6	

Шаг 15: Установите INDEG(F) = INDEG(F) – 1, поскольку F является соседом E. Теперь INDEG(F) = 0, поэтому добавьте F в очередь.

N	INDEG(N)
A	0
B	0
C	0
D	0
E	0
F	0
G	0

FRONT = 7	QUEUE = A, G, B, C, D, E, F, G
REAR = 7	

Топологическая сортировка - пример

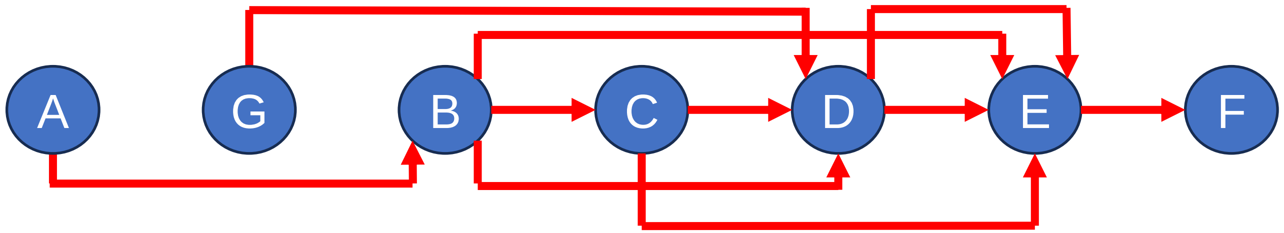


Список смежности

A: B
B: C, D, E
C: E
D: E
E: F
G: D

Удалите ребра от E до F.

В графе больше нет ребер, и все узлы добавлены в очередь, поэтому топологическая сортировка T графа G может быть задана как: A, G, B, C, D, E, F. Когда мы располагаем эти узлы в последовательности, мы обнаруживаем, что если есть ребро от u до v, то u появляется перед v.



Топологическая сортировка графа G

План лекции

Графы (основы)

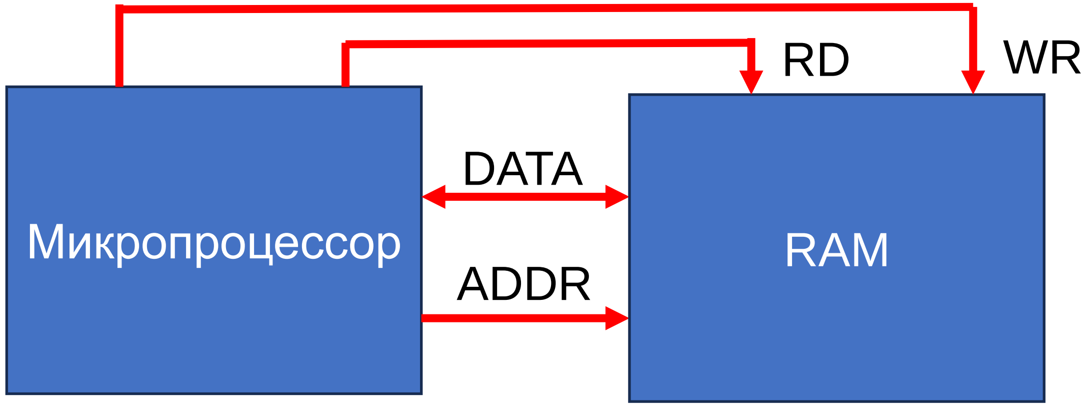
60 минут

Ассемблер <-> С

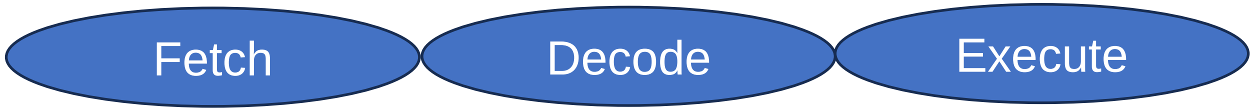
(первые слова)

60 минут

Напоминание



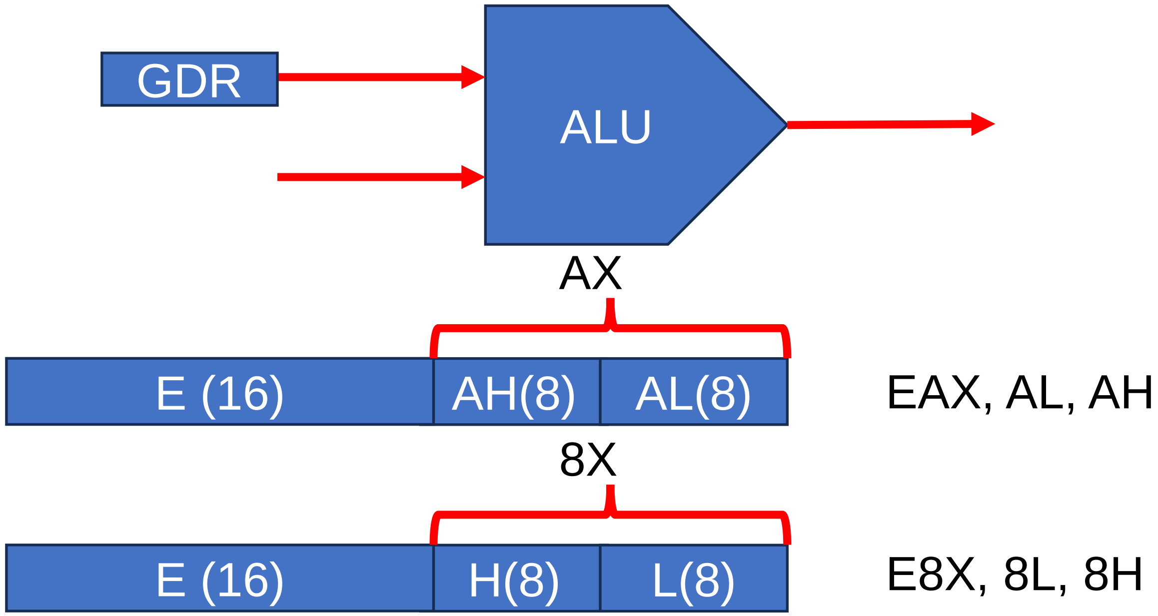
RD – чтение, WR – запись – команды к памяти.
Важно указать адрес микропроцессору.
Но мы работаем именно с логикой, физическая память для нас черный ящик.
Микропроцессор выполняет задачи извлечение + декодирование инструкций.



Напоминание

FDE – что нам нужно для этого?
Начнем с выборки:
1 требование – адрес кода. (Даем процессору адрес кода и просим начать оттуда) EIP – Extended Instruction Ponter
Таким образом, Loc_x -> EIP -> Автоматически EIP + N
Перейдем к выполнению кода:
Что нам нужно?
Есть 2 операнда левый и правый: один из них будет регистром

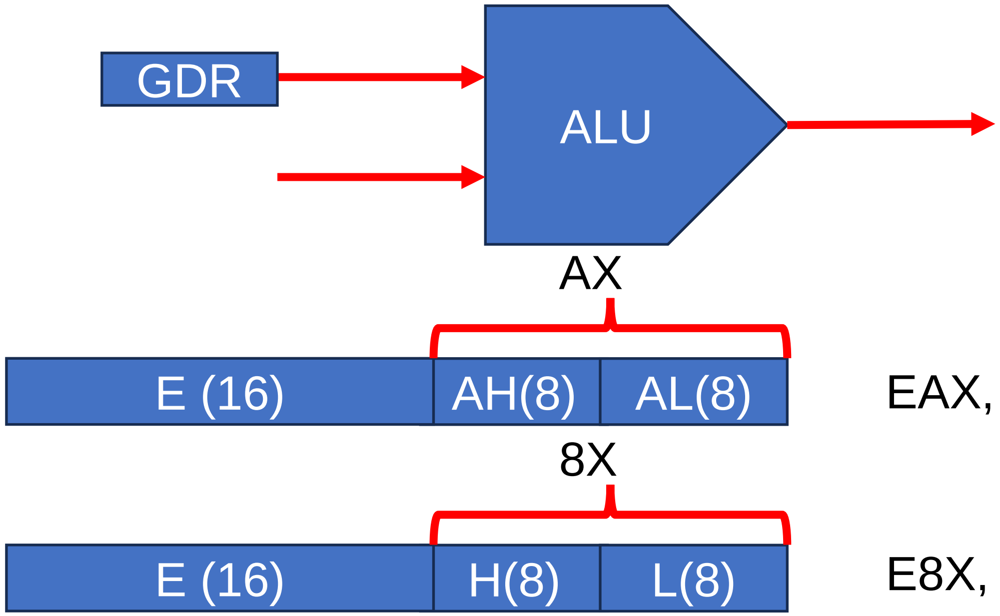
General Purpose Register – 16 бит (8086)



Напоминание

FDE – что нам нужно для этого?
Начнем с выборки:
1 требование – адрес кода. (Даем процессору адрес кода и просим начать оттуда) EIP – Extended Instruction Ponter
Таким образом, Loc_x -> EIP -> Автоматически EIP + N
Перейдем к выполнению кода:
Что нам нужно?
Есть 2 операнда левый и правый: один из них будет регистром

General Purpose Register – 16 бит (8086)

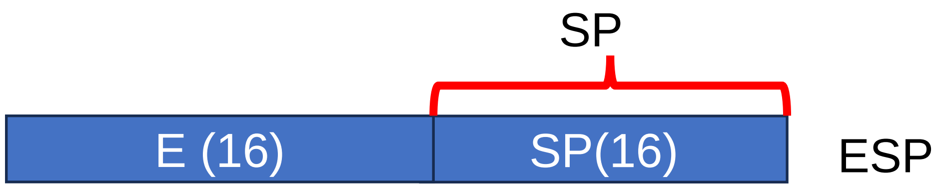


EAX, AL, AH

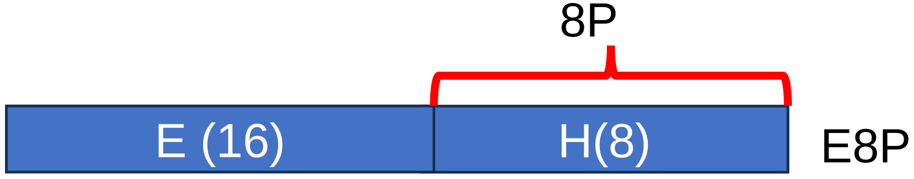
E8X, 8L, 8H

Напоминание

Стек – это последний входящий и исходящий вид из памяти. Нужно отслеживать вершину стека:



Этого недостаточно, нужно также выполнять произвольный доступ

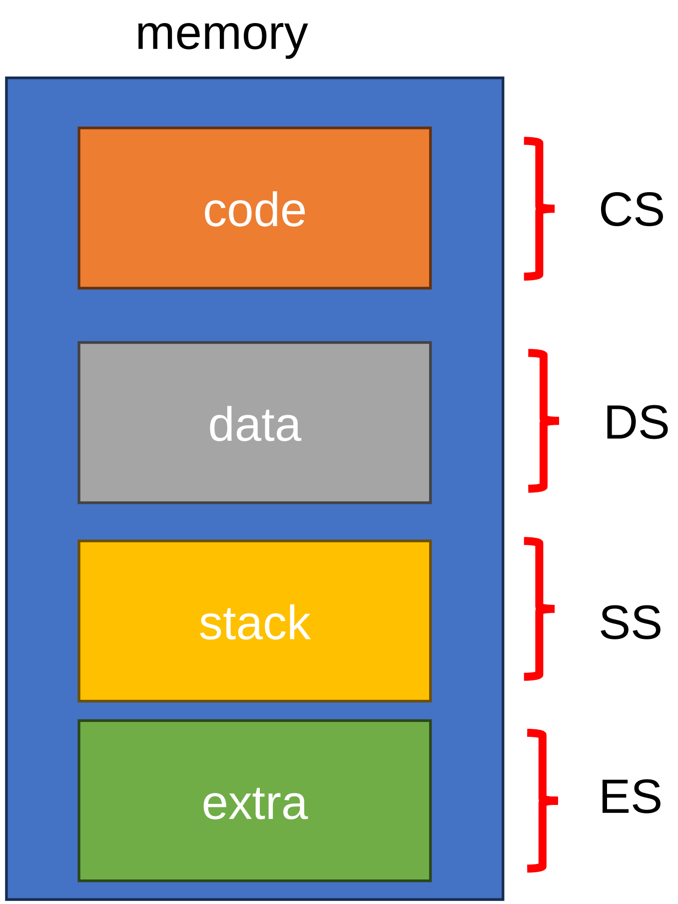


STACK Pointer and register base

Также нам нужны регистры для управления массивами и строками:
ESI – sourse index и EDI - destination

Напоминание

Стек – это последний входящий и исходящий вид из памяти. Нужно отслеживать вершину стека:



Пример:
EIP ->0x0010
Полный адрес может быть:
[ECS : EIP]
[ESS : ESP]
[ESS : EBP]

Последний регистр – регистр флага, например бит знака для отрицательных выражений

Напоминание

Набор инструкций:

1. Перемещение данных (MOVE)
2. Арифметические действия (ADD, SUB, AND, INC, CMP..)
3. Операции со стеком (PUSH, POP, EBP ..)
4. Вызывать или возвращать функции (вызывать подпрограмму в ассемблере) (in english: SUBROUTING (ASSEMBLY) / FUNCTION (C)) (CALL – раскрывается в PUSH + EIP + RET)
5. И многое другое

Доп: SCASB/W/D - сравнивает значение регистра с нужным

REPNE означает повторять пока не равно или CX = 0

CMPS B/W/D – сравнение строк

Допустим хотим скопировать 100 байт из одного массива в другой, для этого есть команда MOVSB/W/D

Первая программа

```c
#include <stdio.h>
int main(void)
{
    int x = 2;
    x = x + 4;
    printf("%d", x);
    return 0;
} // 6
```

```c
#include <stdio.h>
int main(void)
{
    int x = 2;
    __asm {                 // x = x + 4
        mov  eax, x         // eax <-x
        add eax, 0x0004     // eax <- eax + 4
        mov x, eax          // x <- eax
    }
    printf("%d", x);
    return 0;
} // 6
```

Для чего? Оптимизация?

Оптимизация

```c
#include <stdio.h>

int main(void)
{

    int x = 10, y = 5;
    int temp;
    temp = x;
    x = y;
    y = temp;
    return 0;
}
```

```c
#include <stdio.h>
int main(void)
{
    int x = 10, y = 5;
    __asm {
        push x
        push y
        pop x
        pop y
    }
    return 0;
}
```

Компилятор

Здесь будем смотреть:
https://godbolt.org/
Встроенный инструмент с открытым исходным кодом
Зачем знакомиться?
Поиграть с программами на Си, чтобы понять, как различные изменения программы влияют на нее

```c
int square(int num) {
    return num * num;
}
```

```asm
_square PROC
        push    ebp
        mov     ebp, esp
        mov     eax, DWORD PTR _num$[ebp]
        imul    eax, DWORD PTR _num$[ebp]
        pop     ebp
        ret     0
```

Пример

```c
int div() {
    int x = 10, y = 3;
    int q, r;
    q = x / y;
    r = x % y;
    return r;
}
```

```asm
        push    ebp
        mov     ebp, esp
        sub     esp, 16
        mov     DWORD PTR _x$[ebp], 10
        mov     DWORD PTR _y$[ebp], 3
        mov     eax, DWORD PTR _x$[ebp]
        cdq
        idiv    DWORD PTR _y$[ebp]
        mov     DWORD PTR _q$[ebp], eax
        mov     eax, DWORD PTR _x$[ebp]
        cdq
        idiv    DWORD PTR _y$[ebp]
        mov     DWORD PTR _r$[ebp], edx
        mov     eax, DWORD PTR _r$[ebp]
        mov     esp, ebp
        pop     ebp
        ret     0
```

Пример 2

```c
int div() {
    int x = 10, y = 3;
    int q, r, m;
    q = x / y;
    r = x % y;
    __asm{
        mov eax, q
        mov ebx, r
        imul ebx
        mov m, eax
    }
    return r;
}
```

```asm
push    ebp
mov     ebp, esp
sub     esp, 20
push    ebx
mov     DWORD PTR _x$[ebp], 10
mov     DWORD PTR _y$[ebp], 3
mov     eax, DWORD PTR _x$[ebp]
cdq
idiv    DWORD PTR _y$[ebp]
mov     DWORD PTR _q$[ebp], eax
mov     eax, DWORD PTR _x$[ebp]
cdq
idiv    DWORD PTR _y$[ebp]
mov     DWORD PTR _r$[ebp], edx
mov     eax, DWORD PTR _q$[ebp]
mov     ebx, DWORD PTR _r$[ebp]
imul    ebx
mov     DWORD PTR _m$[ebp], eax
mov     eax, DWORD PTR _r$[ebp]
pop     ebx
mov     esp, ebp
pop     ebp
ret     0
```

Пример 3 – O1

```c
int div() {
    int x = 10, y = 3;
    int q, r;
    q = x / y;
    r = x % y;
    return r;
}
```

```asm
xor     eax, eax
inc     eax
ret     0
```

Пример 4

```c
int div(int x, int y) {
    char *pA = 0;
    char *pB = 0;
    short int *pC = 0;
    pA++;
    pB++;
    pC++;
    return 0;
}
```

```asm
mov     DWORD PTR [rsp+16], edx
mov     DWORD PTR [rsp+8], ecx
sub     rsp, 40
mov     QWORD PTR pA$[rsp], 0
mov     QWORD PTR pB$[rsp], 0
mov     QWORD PTR pC$[rsp], 0
mov     rax, QWORD PTR pA$[rsp]
inc     rax
mov     QWORD PTR pA$[rsp], rax
mov     rax, QWORD PTR pB$[rsp]
inc     rax
mov     QWORD PTR pB$[rsp], rax
mov     rax, QWORD PTR pC$[rsp]
add     rax, 2
mov     QWORD PTR pC$[rsp], rax
xor     eax, eax
add     rsp, 40
ret     0
```

Пример 5

```
#include <stdio.h>
void div() {
    char *pA="This is a test string";
    int i = 0;
    while(pA[i] != '\0') {
        i++;
    }
    printf("\n String length = %d", i);
}
```

Пример 6

```
void div() {
    int x = 10, y =2;
    int q, r;
    q = 10.0/y;
    r = x % y;

}
```

При работе с операциями с плавающей точкой, процессор отправляет данные во внешний сопроцеесор
В ответе видны новые функции:
`cvtsi2sd cvttsd2si`