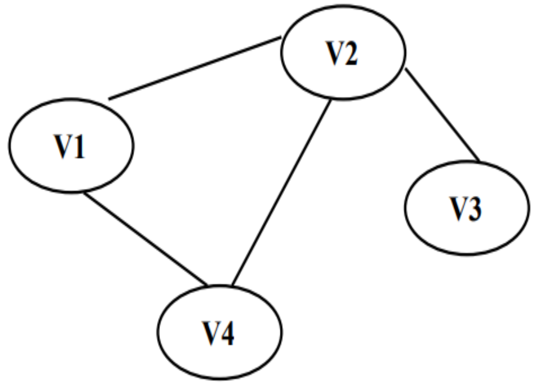
Графы

Основные понятия

Граф – это структура данных, описывающее множество вершин и множество ребер, соединяющих их

G = (V, E)



неориентированный граф

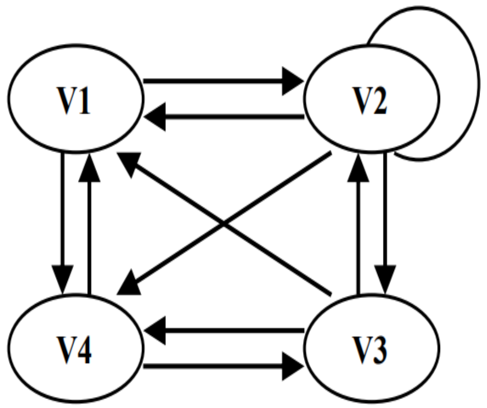
Смежные вершины – имеют соединение в одно ребро

Ориентированное ребро (дуга) – имеет направление обхода

Петля – ребро, соединяющее вершину с самой собой

Ориентированный граф – содержит все ориентированные ребра (дуги)

Простой граф – граф, содержащий между любыми парами вершин не более одного ребра



Ориентированный граф

Основные понятия

Путь - это такая последовательность ребер графа, при которой конечная вершина любого ребра этой последовательности является начальной вершиной следующего ребра

$$P = ((v1, v2), (v2, v3), \ldots, (vK\text{-}2, vK\text{-}1), (vK\text{-}1, vK))$$

$$P = (v1, v2, \ldots, vK\text{-}1, vK)$$

Длина пути – число ребер в последовательности, задающей путь
Простой путь – не содержит повторяющихся ребер
Элементарный путь – не содержит повторяющихся вершин
Цикл – простой путь, оканчивающийся в начальной вершине
Ациклический – простой граф, не содержащий циклов
Связный граф – подразумевает путь между любыми двумя вершинами

Основные понятия

Граф $G' = (V', E')$ называется **подграфом** графа $G = (V, E)$, если $V' \subset V$. $E' \subset E$

Компонента связанности – максимальный связанный подграф графа

Сильно связанный ориентированный граф – подразумевает для каждой пары вершин i и j как минимум ориентированные пути из i в j и наоборот

Взвешенный граф – граф, ребрам которого назначены веса.

Представление графов

Матричное представление графов

Пусть имеется простой граф G = (V,E) с упорядоченными вершинами {v1, v2 … vn}, где n – количество вершин

Матрица смежности для данного графа:

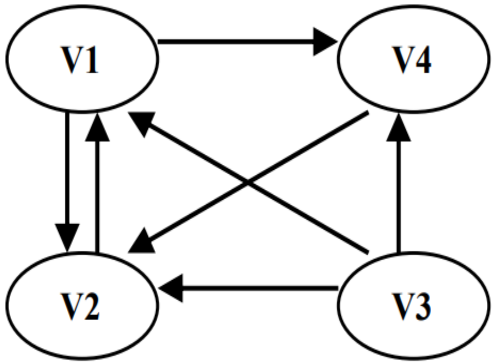
A(n,n) = $|\boldsymbol{a_{ij}}|$, где i = 1..n, j = 1..n, такая, что:

$$a_{ij} = \begin{cases} 1 \ - \text{ребро } (v_i, v_j) \text{из вершины i в вершину j существует,} \\ 0 \ - \text{ребро } (v_i, v_j) \text{из вершины i в вершину j не существует} \end{cases}$$

Матрица смежности

```
public class Graph
    {
        // количество вершин в графе
        private int size;
        // матрица смежности графа
        private bool[,] adjacency;
    }
```

	V1	V2	V3	V4
V1	0	1	0	1
V2	1	0	0	0
V3	1	1	0	1
V4	0	1	0	0



Матрица весов

Весовая матрица (матрица весов) W (n,n) = $|w_{ij}|$, i=1..n, j = 1..n, такая, что

$$w_{ij} = \begin{cases} \text{вес} \ - \text{ребро} \ (v_i, v_j) \text{из вершины i в вершину j существует,} \\ \propto \ - \text{ребро} \ (v_i, v_j) \text{из вершины i в вершину j не существует} \end{cases}$$

```
public class Graph
    {
        // количество вершин в графе
        private int size;
        // весовая матрица графа
        private int[,] weight;
    }
```

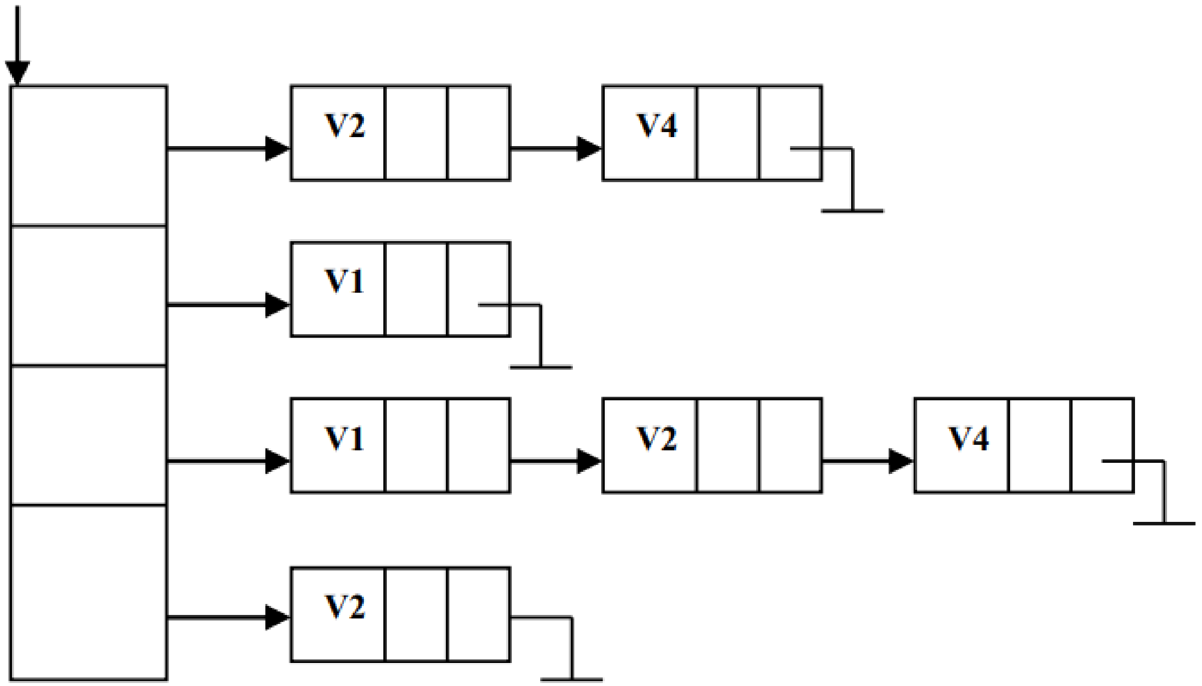
Матрица достижимости

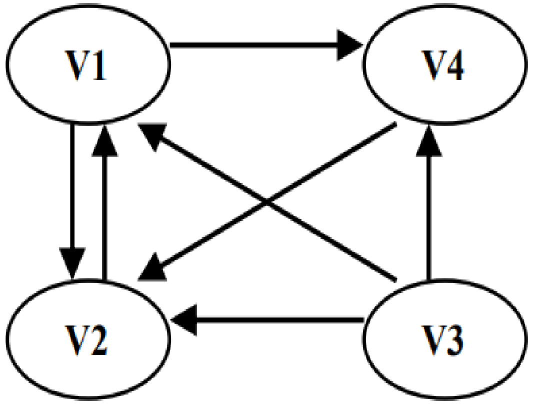
Матрица достижимости P (n,n) = $|p_{ij}|$, i=1..n, j = 1..n, такая, что

$$p_{ij} = \begin{cases} 1 \ - \text{существует путь длины } \leq \text{n из вершины i в вершину j,} \\ 0 - \text{не существует путь длины} \leq \text{n из вершины i в вершину j} \end{cases}$$

Представление графов

Представление в виде списка смежности





Программное описание вершины графа

```
public class Node //класс вершины
    {
        private int id; // идентификатор смежной вершины
        private int weight; // вес ребра
        private Node link; // ссылка на соседний элемент списка вершин
        public int Id { get; set; } // свойства
        public int Weight { get; set; }
        public Node Link { get; set; }
        public Node() { } // конструкторы
        public Node(int id, int weight)
        {
            Id = id; Weight = weight;
        }
    }
```

Программное описание графа

```
public class Graph
    {
        private int size; // количество вершин в графе
        private Node[] adjList; // ссылка на список смежности вершин графа

        public int Size { get; set; } // свойства
        public Node[] AdjList { get; set; }
        public Graph(int size) // конструктор
        { // создание и инициализация списка смежности вершин графа
            AdjList = new Node[size];
            for (int i = 0; i<=size; i++) AdjList[i] = null;
        }

        ///

    }
```

Выбор варианта представления

Разреженный граф – количество ребер меньше количества вершин. Предпочтительно использовать списки смежности

Плотный граф – содержит максимально возможное количество ребер (приближающееся к n^2-n). Предпочтительно использовать матрицы смежности.

Алгоритмы обходов графов

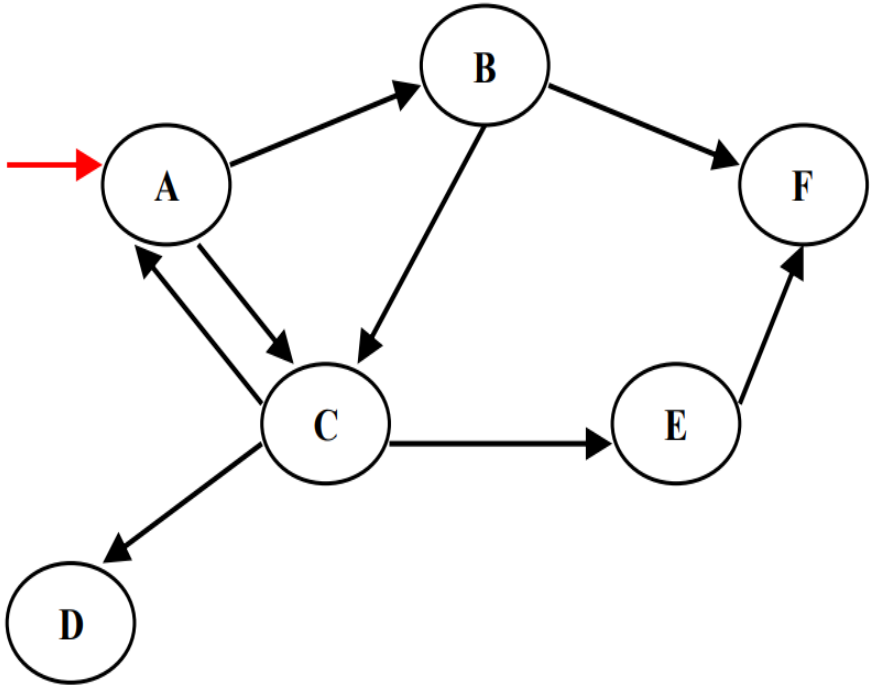
Обход в ширину

Обход в глубину

- Произвольным способом определяется вершина, с которой стартует обход.
- Рекурсивно обрабатываются остальные вершины, достижимые из первой.

Обход графа в глубину

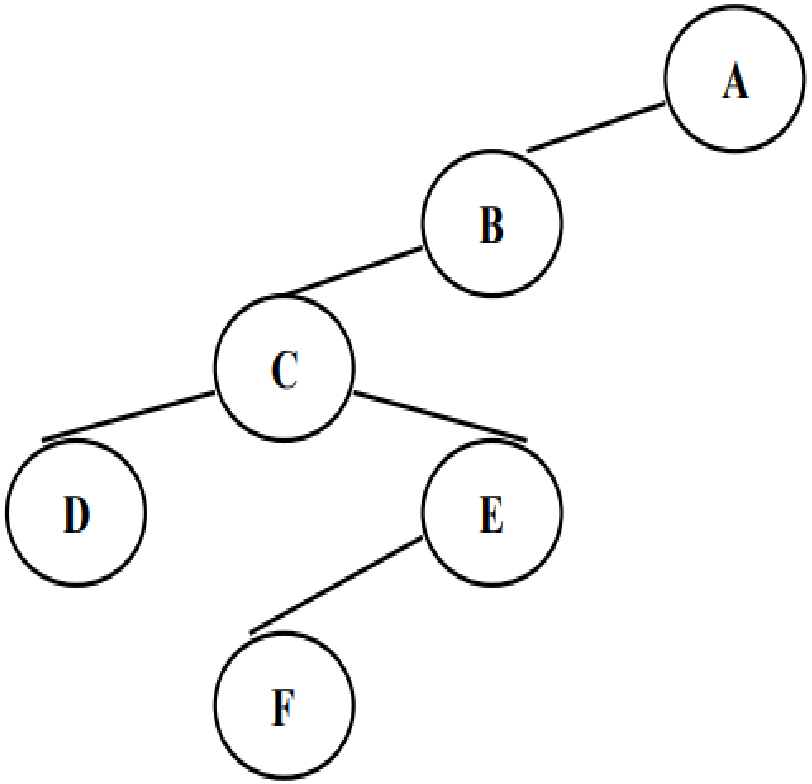
Исходные данные



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

Обход графа в глубину

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



Дерево, полученное в результате обхода графа в глубину

Программная реализация обхода в глубину

```
public class Graph
    {
        private int size;              // количество вершин в графе
        private bool[,] adjacency;   // матрица смежности графа
        private bool[] vector;        // вектор посещенных вершин

        public int Size {get; set;}
        public bool[,] Adjacency { get; set;}
        public bool[] Vector {get; set;}
```

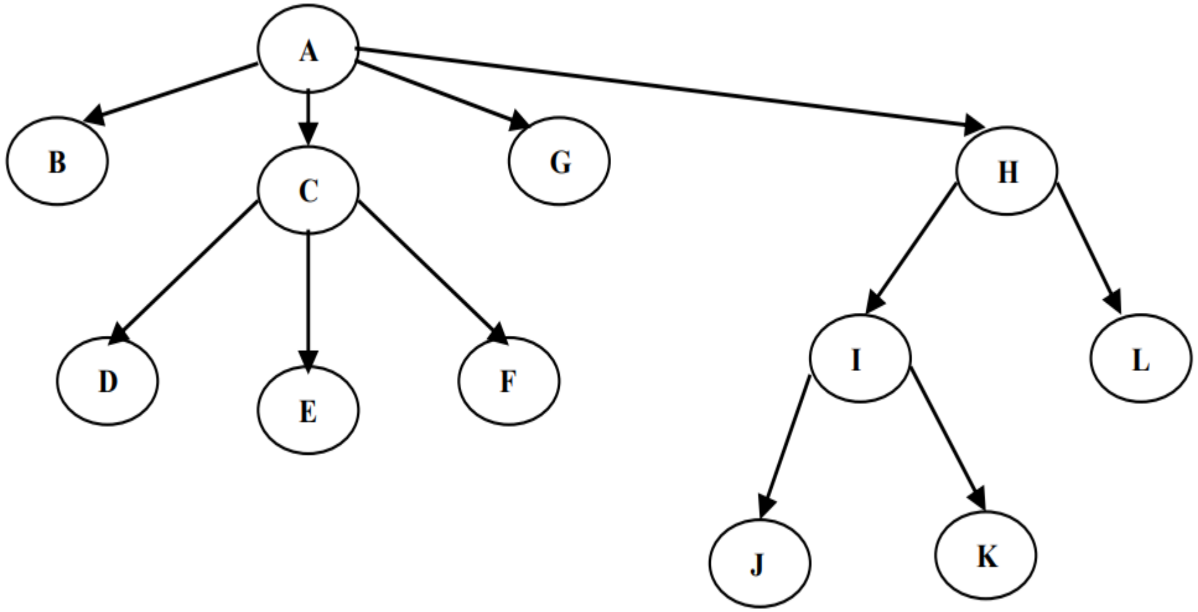
Программная реализация обхода в глубину

```
public Graph(int size, bool[,] G)        // конструктор класса «Графы»
       {
           Adjacency = new bool[size, size];   // инициализация матрицы смежности
           Adjacency = G;
           Vector = new bool[size];
           for (int i = 0; i<size; i++)
               Vector[i] = false;              // иниц-я вектора посещенных вершин
           Size = size;
       }

       public void Depth(int i)                //i – вершина, с которой начинается обход
       {
           Vector[i]=true;                     // отметить вершину i как обработанную
           Console.Write( "{0}" + ' ', i );    // распечатать номер посещенной вершины
           for (int k=0; k<Size; k++)          // найти первую встретившуюся ранее
                                          непосещенную вершину k, смежную с вершиной i
               if (Adjacency[i, k] && ! (Vector[k]) )
                   Depth(k );                  // перейти к обработке вершины k
        }
    }
```

Программная реализация обхода в глубину

```
static void Main(string[] args)
        {
            bool[,] M = new bool[4, 4]
            {
                {false, true, false, true}, // матрица смежности графа G4
                {true, false, false, false},
                {true, true, false, true},
                {false, true, false, false}
            };

            Graph graph = new Graph(4, M);
            graph.Depth(1);
        }
```

Обход графа в ширину



Узлы проходятся слева направо по горизонтали в соответствии с матрицей смежности в порядке возрастания длины пути от корня к вершине.

A B C G H D E F I L J K.