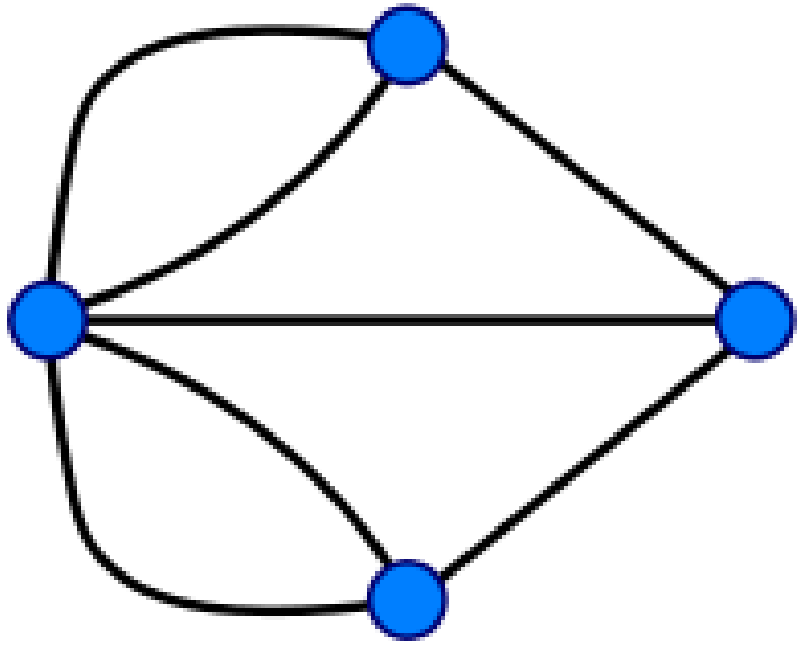
Обход графов

Графы

$$G = (V,E)$$

V – *вершины*

E = V×V – *ребра*, соединяющие пары вершин

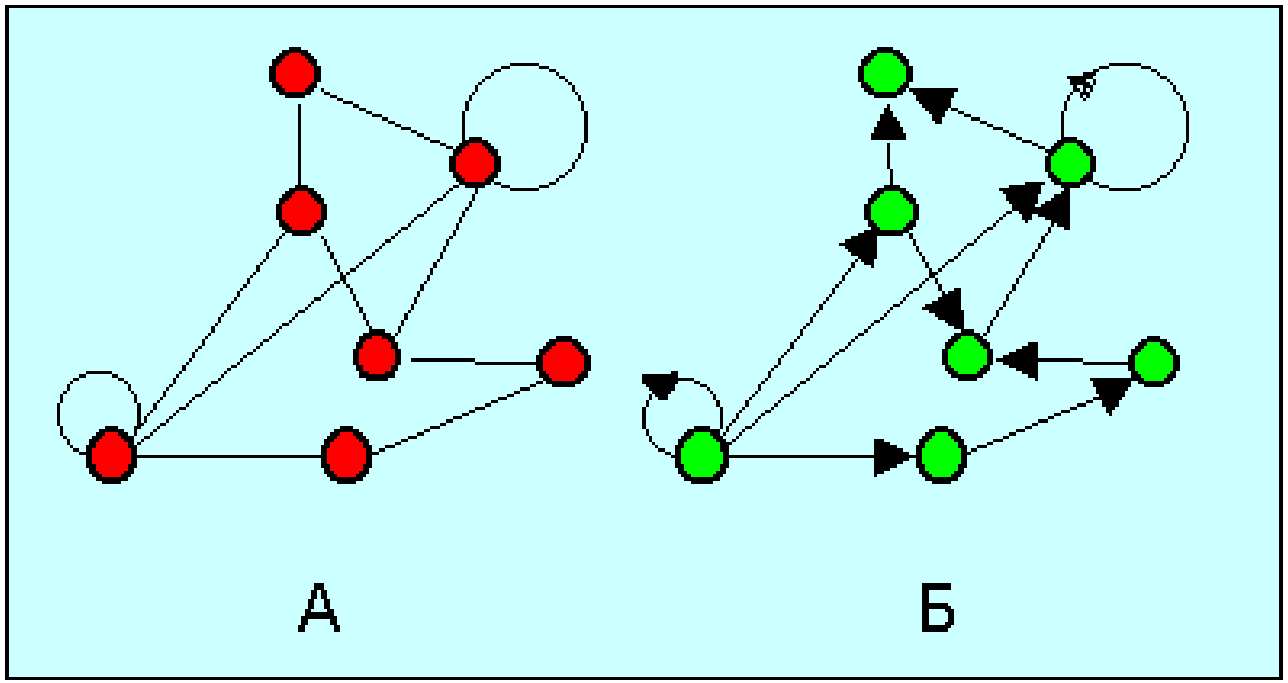


Разновидности графов

Неориентированный/ориентированный

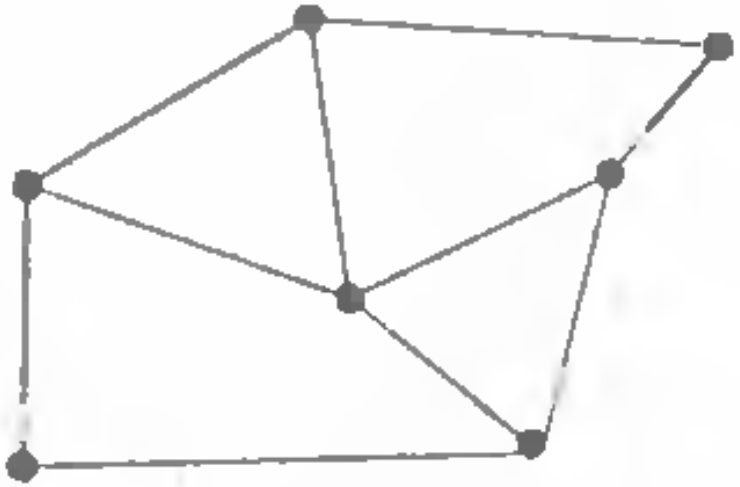
G=(V,E) – неориентированный, если из $(x,y) \in E$ следует, что (y,x) также является членом E.

В противном случае граф – *ориентированный*.

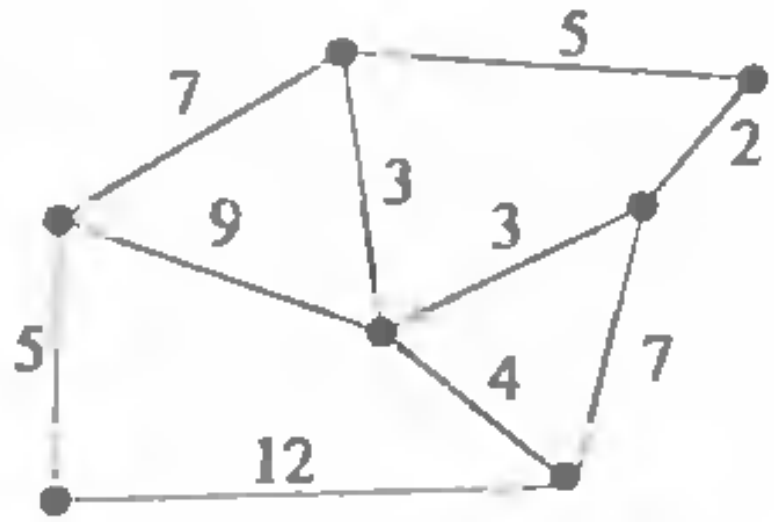


Взвешенные/невзвешенные

Каждому ребру/вершине *взвешенного* графа *G* присваивается числовое значение или вес.



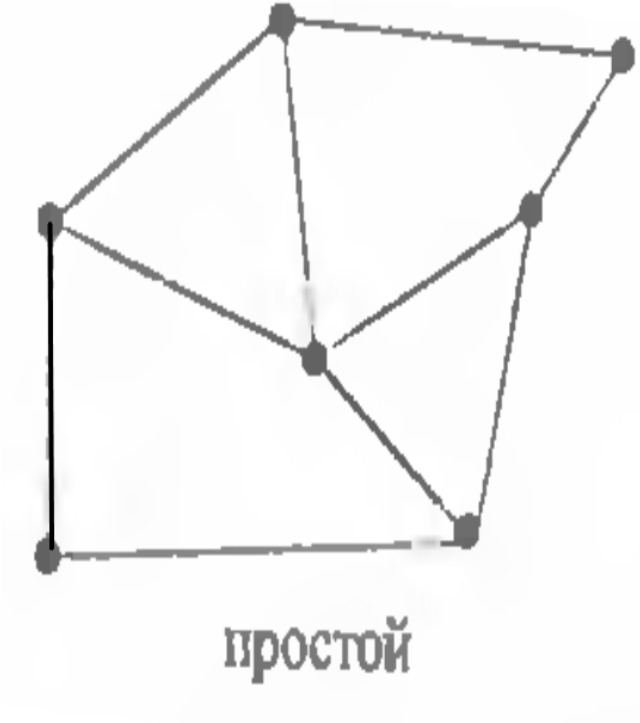
невзвешенный

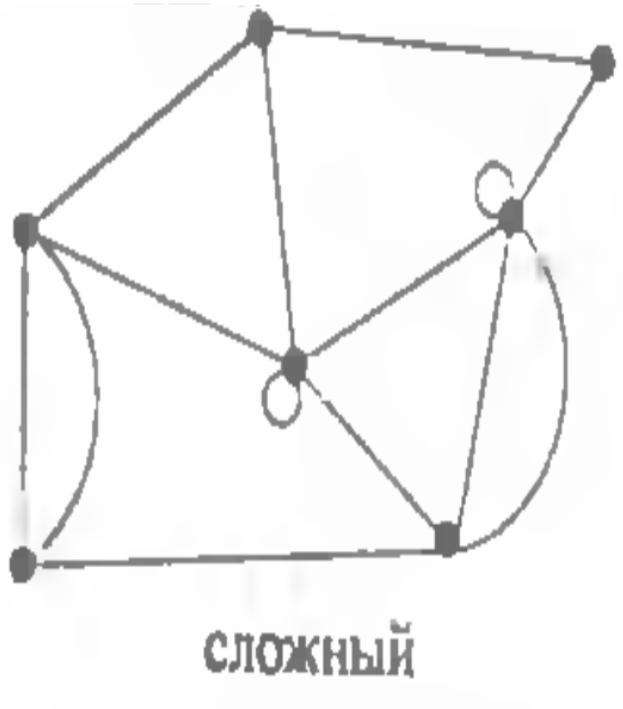


взвешенный

Простые/сложные

При наличии *петлей* или *кратного* соединения вершин возникают дополнительные затруднения при работе с графами.

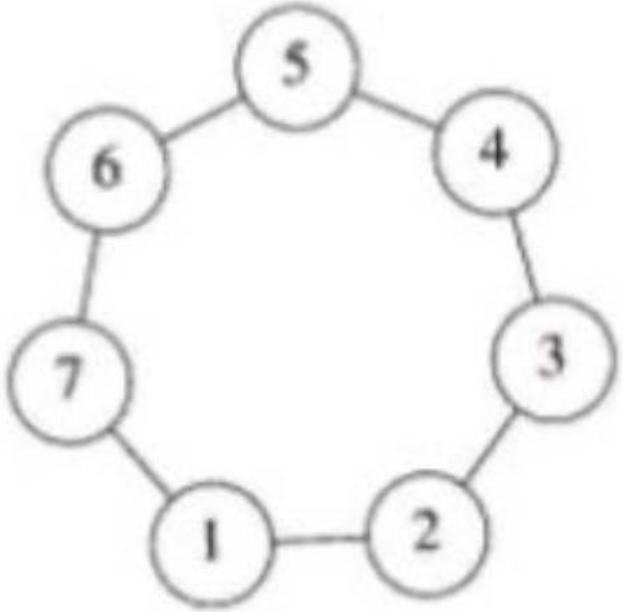


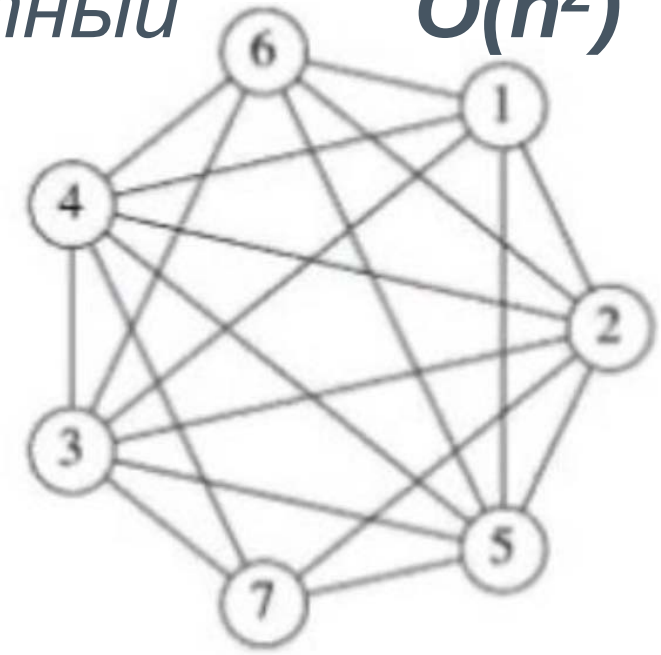


Разреженные/плотные

Граф является *разреженным*, если ребра определены для малой части возможных пар вершин. ***O(n)***

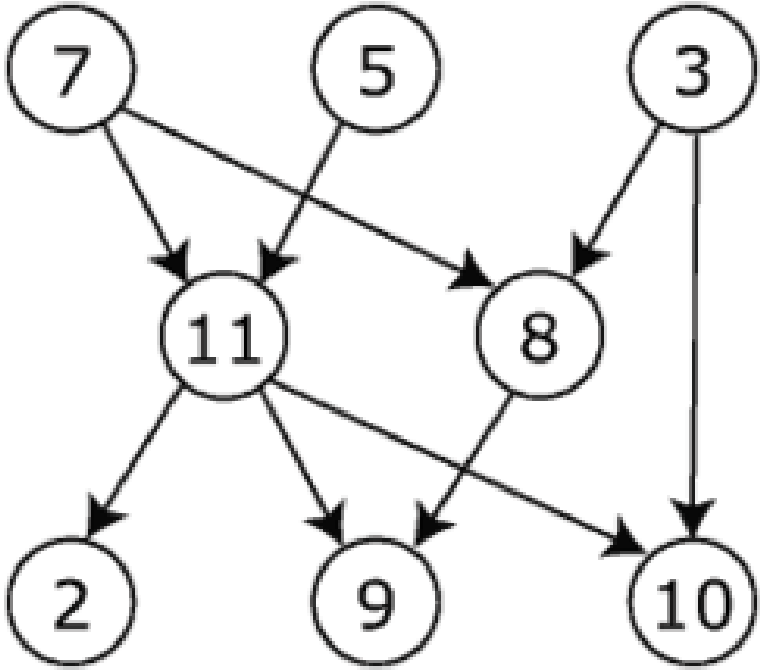
Граф, у которого ребра определены для большей части возможных вершин - *плотный* ***$O(n^2)$***

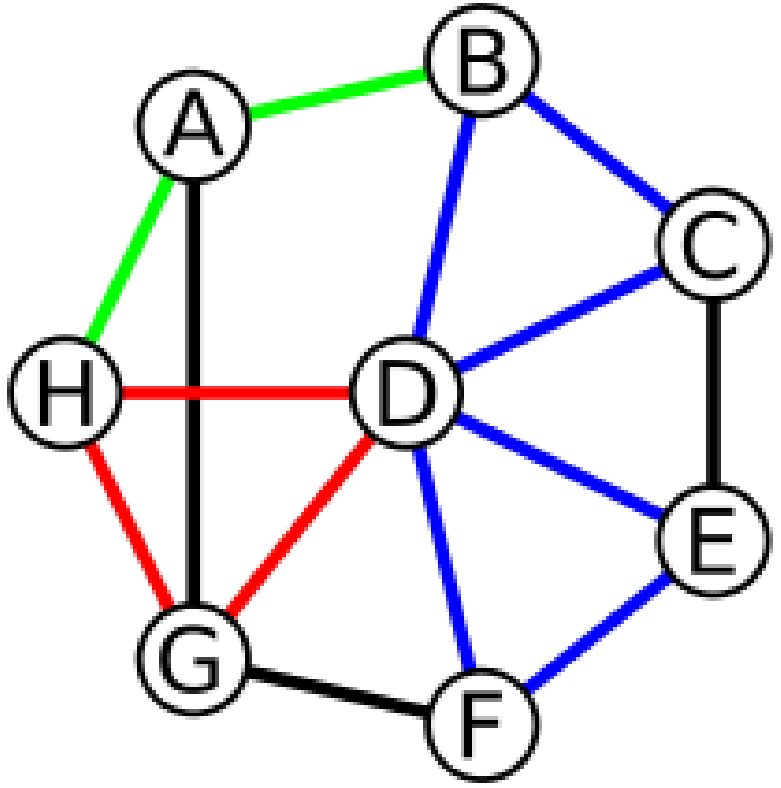




Циклический/ациклический

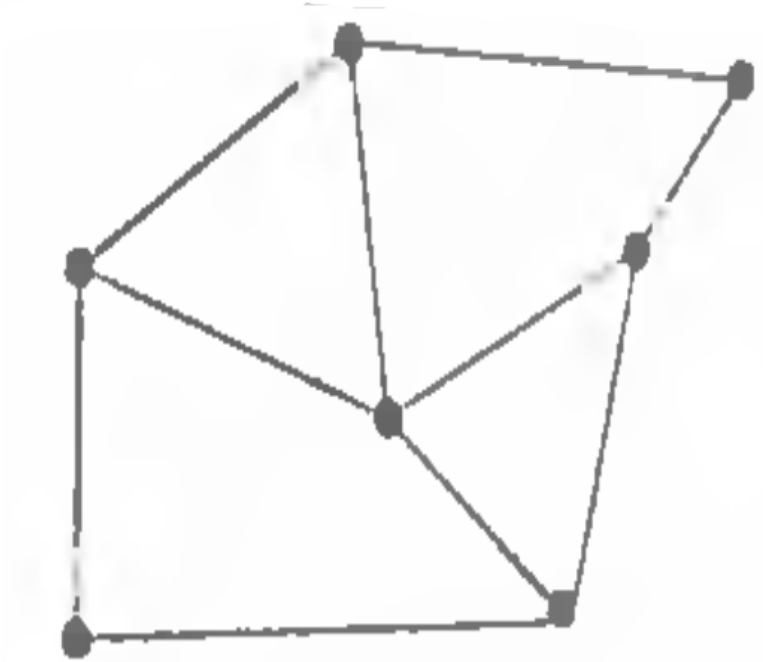
Ациклический граф не содержит циклов, т.е. невозможно преодолеть весь направленный граф, начав с одного ребра.



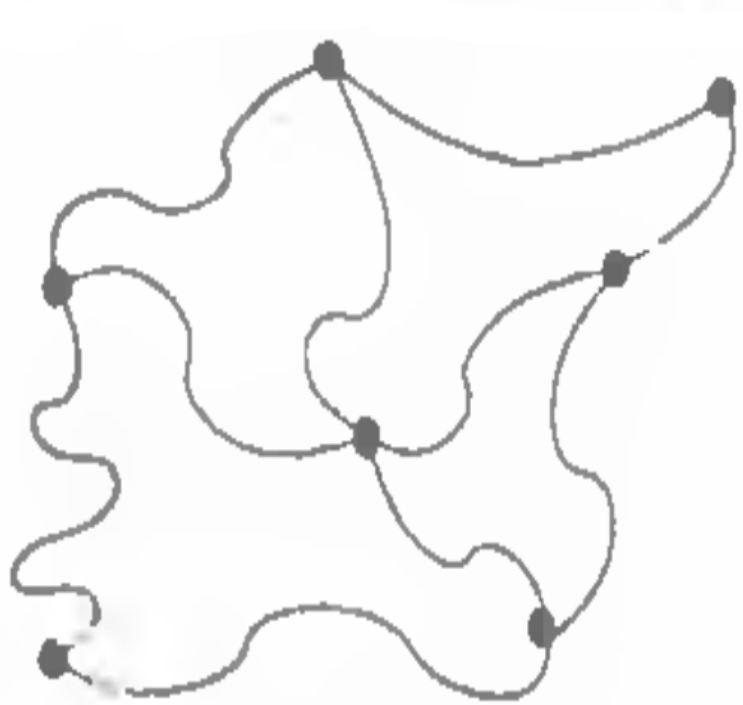


Вложенные/топологические

Граф является *вложенным*, если его вершинам и ребрам присвоены геометрические позиции.



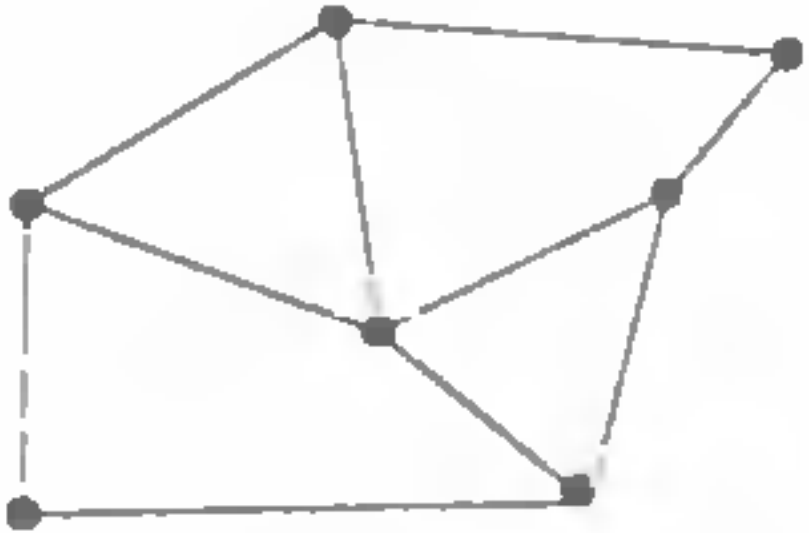
вложенный



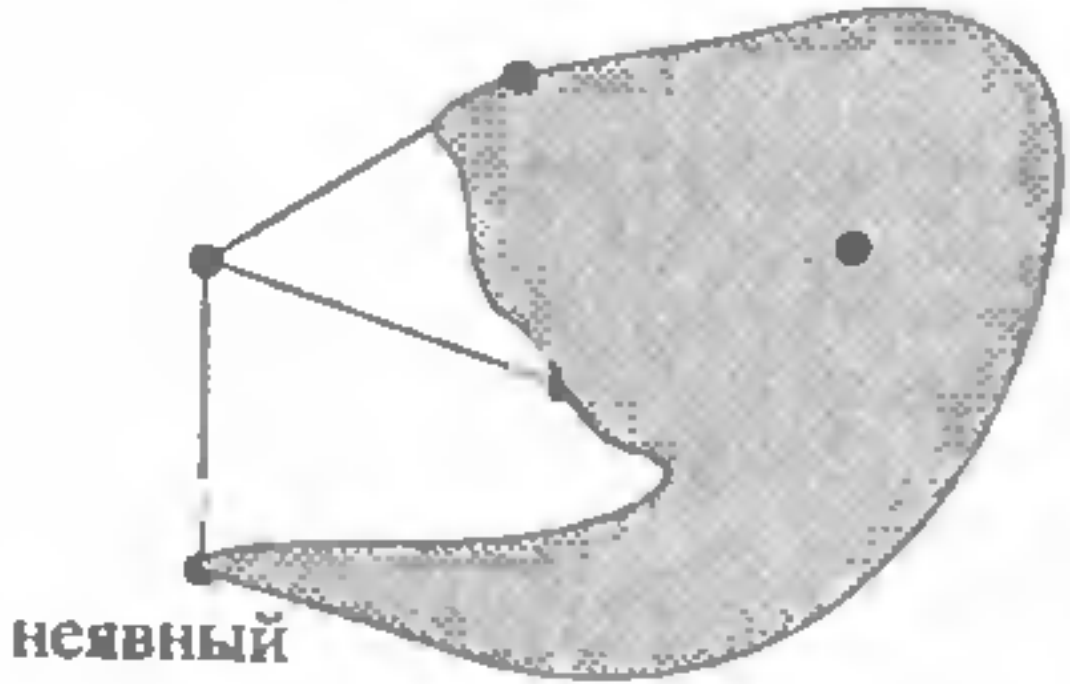
топологический

Неявные/явные

Неявный граф не создается заранее, а возникает по мере решения задачи.



явный

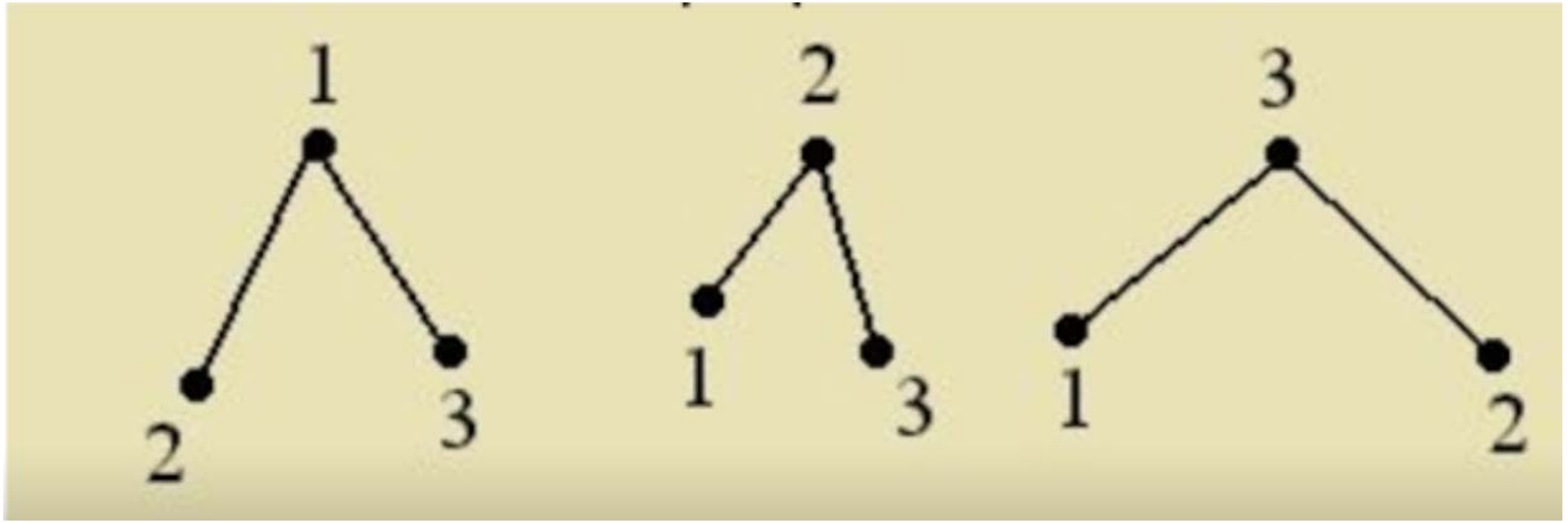


неявный

Помеченные/непомеченные

В *помеченном* графе каждая вершина имеет свою метку, что позволяет ее отличить от других вершин.

В *непомеченных* графах такие обозначения не применяются



Обход графа – фундаментальная задача

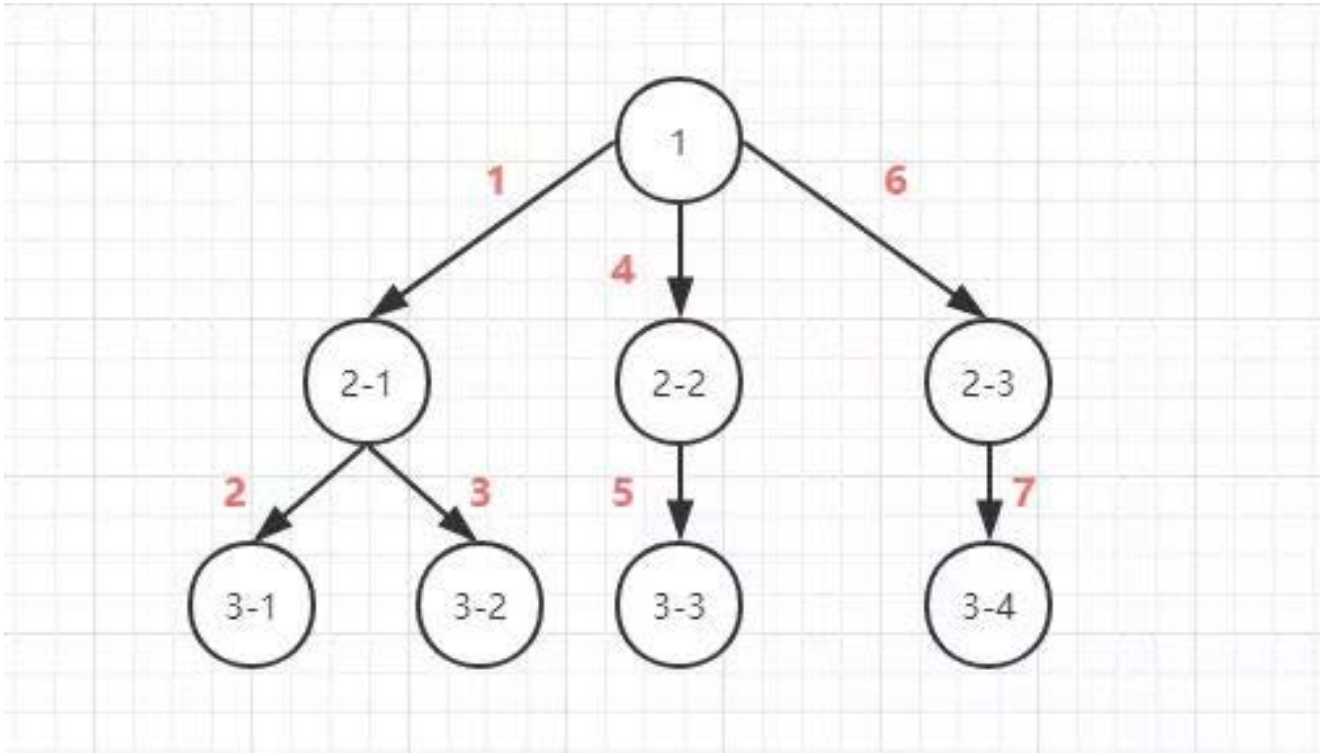
Обход графа - систематизированное посещение каждой вершины и каждого ребра графа.

Каждая из вершин находится в одном из состояний:

- *Неоткрытая* – первоначальное, нетронутое состояние вершины;
- *Открытая* – вершина обнаружена, но не проверены все ее ребра;
- *Обработанная* – все инцидентные данной вершине ребра были посещены.

Алгоритмы обхода графов – порядок выполнения обхода

- Обход в ширину (breadth-first search, BFS)
- Обход в глубину (depth-first search, DFS)



Обход графа в ширину

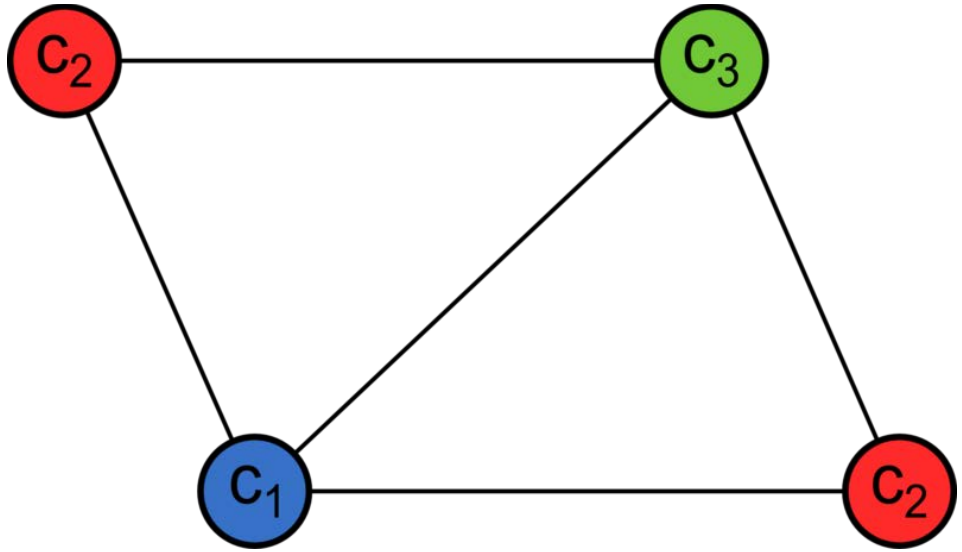
```
bfs(graph *g, int start)
{
     queue q;  /* Очередь вершин для обработки */
     int v;    /* Текущая вершина */
     int y;    /* Следующая вершина */
     edgenode *p;   /* Временной указатель */
     int_queue (&q);
     enqueuer (&q, start);
     discovered[start] = TRUE;
     while (empty_queue(&q) == FALSE) {
          v = dequeuer (&q);
          process_vertex_early(v);
```

```
        processed[v] = TRUE;
        p = g -> edges[v];
        while (p != NULL) {
                y = p->y;
                if ((processed[y]==FALSE)||g->directed)
                        process_edge(v,y);
                if (discovered[y]==FALSE) {
                        enqueuer(&q,y);
                        discovered[y]=TRUE;
                        parent[y]=v;
                }
                p=p->next;
        }
        process_vertex_late(v);
}
```

Раскраска графов

При *раскраске вершин* требуется присвоить метку (цвет) каждой вершине графа так, чтобы любые две соединенные ребром вершины были разного цвета.

G(V,E): $V \xrightarrow[\varphi]{} \{c_1, \dots, c_t\}$, для $\forall u, v\ \varphi(u) \neq \varphi(v)$



Двудольная раскраска графов

```
twocolor(graph *g)
{
   int i;     /* счетчик */
   for (i=1; i<=(g->nvertices); i++)
      color[i] = UNCOLORED;
   bipartite = TRUE;
   initialize_search(&g);
   for (i=1; i<=(g->nvertices); i++)
      if (discovered[i] == FALSE) {
         color[i] = WHITE;
         bfs(g,i);
      }
}
```

```
process_edge(int x, int y)
{
   if (color[x] == color[y]) {
      bipartite = FALSE;
      printf("Warning: not bipartite due to (%d,%d)\n",x,y);
   }
   color[y] = complement(color[x]);
}
complement(int color)
{
   if (color == WHITE) return(BLACK);
   if (color == BLACK) return(WHITE);
   return(UNCOLORED);
}
```

Обход графа в глубину

Разница

- ***Очередь* - обход в ширину**. Помещая вершины в очередь типа **FIFO**, мы исследуем самые старые не-исследованные вершины первыми. Таким образом, наше исследование медленно распространяется вширь, начиная от стартовой вершины.
- ***Стек* – обход в глубину**. Помещая вершины в стек с порядком извлечения **LIFO**, мы исследуем их, отклоняясь от пути для посещения очередного соседа, и возвращаясь назад, только если оказываемся в окружении ранее открытых вершин. Таким образом, мы в своем исследовании быстро удаляемся от стартовой вершины.

Обход графа в глубину

```
dfs(g,u)
        state[u] = “discovered”
        обрабатываем вершину u, если необходимо
        entry[u] = time
        time = time + 1
        for each v ∈ Adj[u] do
                обрабатываем ребро (u,v), если необходимо
                if state[v] = “undiscovered” then
                        p[v] = u
                        dfs(g,v)
        state[u] = “processed”
        exit[u] = time
        time = time + 1
```

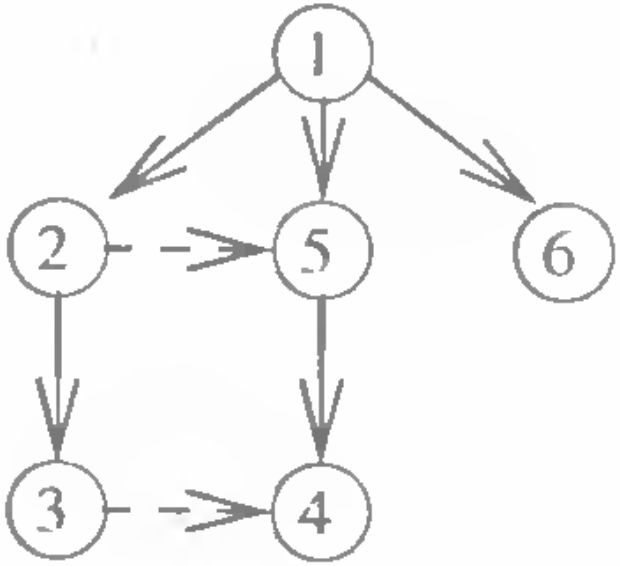
Свойства при обходе в глубину

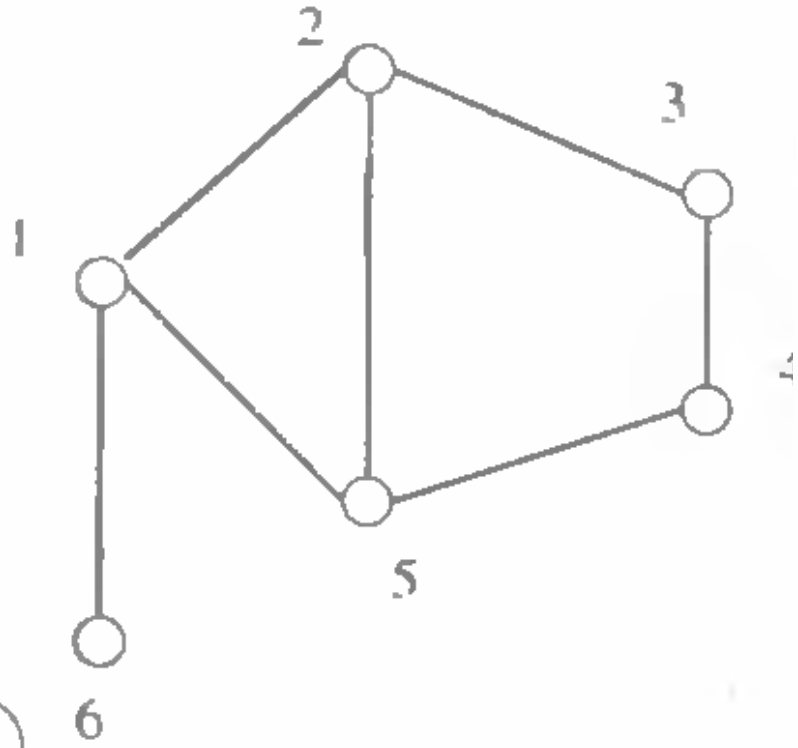
- *Посещение предшественника.* Если вершина ***x*** является предшественником вершины ***y*** в дереве обхода в глубину, то временной интервал посещения *y* должен быть корректно учтен его предшественником x
- *Количество потомков.* Разница во времени выхода и входа для вершины y свидетельствует о количестве потомков этой вершины в дереве обхода в глубину.

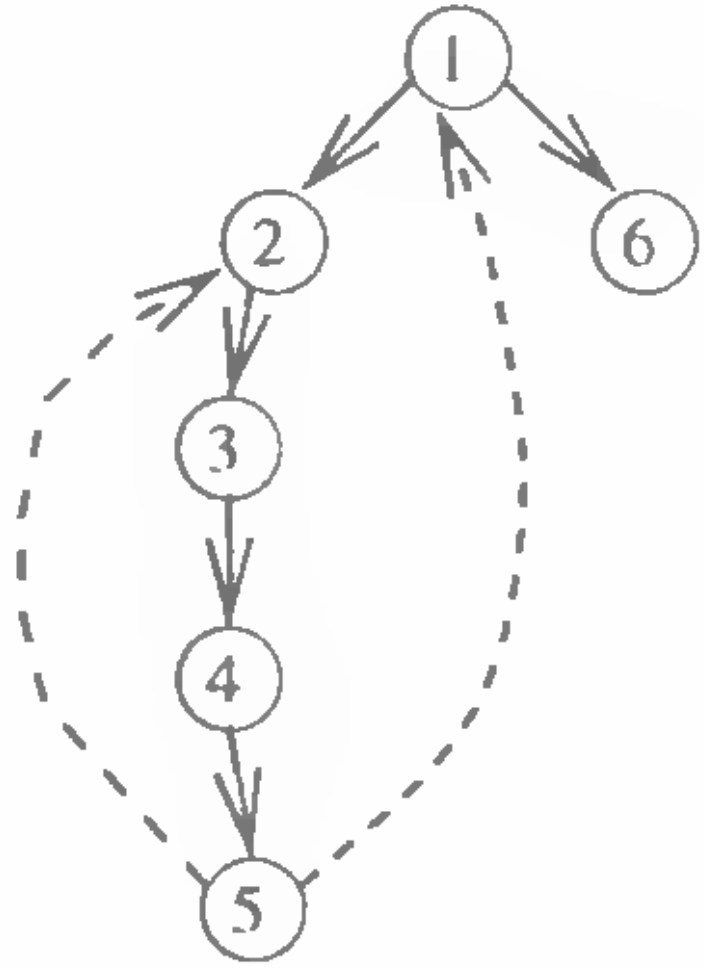
Обход в глубину разбивает ребра на два класса:

- *древесные* (tree edges) - используются при открытии новых вершин и закодированы в родительском отношении
- *обратные* (back edges) - второй конец является предшественником расширяемой вершины

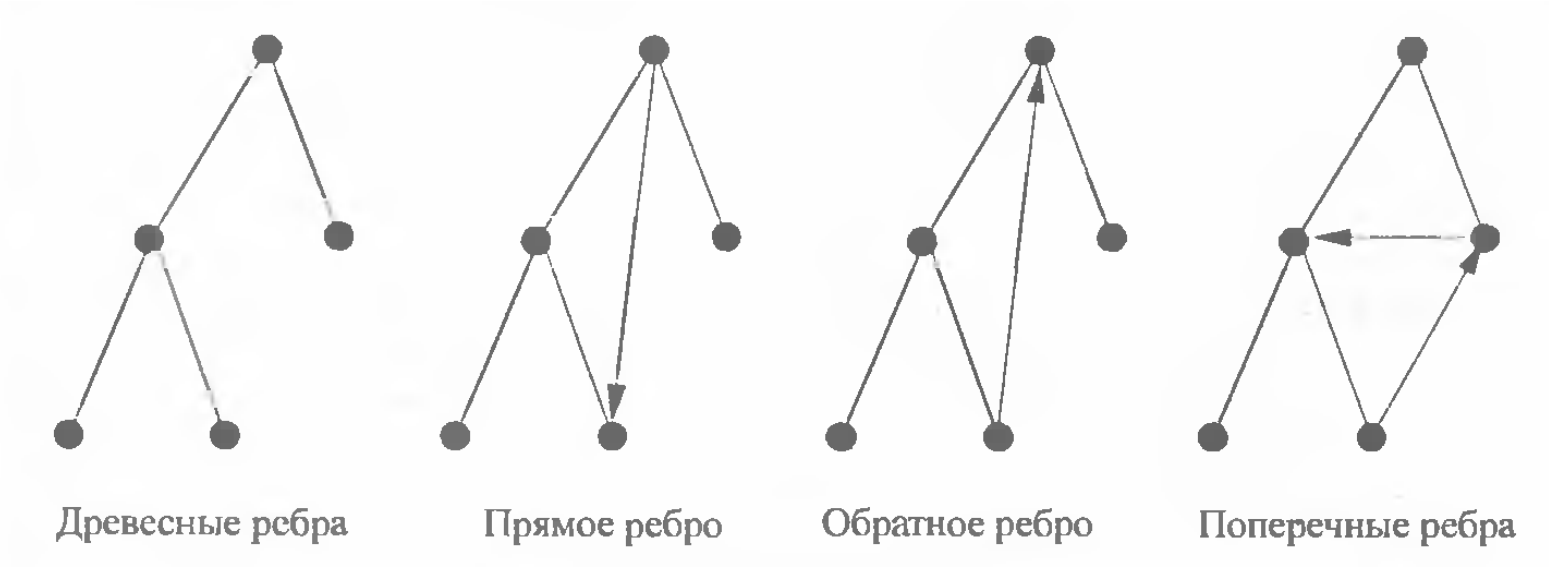
Разница обхода в ширину и глубину







Обход в глубину ориентированных графов



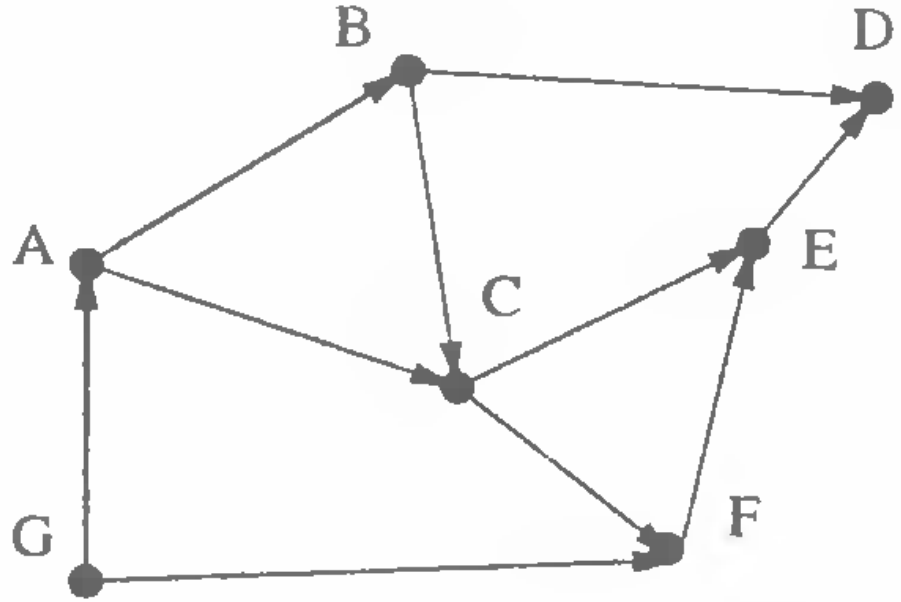

```
DFS-graph(G)
  for each vertex u ∈ V[G]do
     state[u] = "undiscovered"
        for each vertex u ∈ V[G] do
     if state[u] = "undiscovered" then
        инициализируем новую компоненту, если необходимо
        DFS(G,u)
```

Топологическая сортировка

Производится упорядочивание вершин так, что ориентированные ребра направлены слева направо.

Любой бесконтурный орграф (не содержит обратных ребер) имеет минимум одно топологическое упорядочивание.

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



Процедура топологической сортировки

- Если вершина ***y*** *не открыта*, то начинаем обход в глубину из вершины ***y***, прежде чем можем продолжать исследование вершины ***x***.
- Если вершина ***y*** *открыта*, но не *обработана*, то ребро (***x,y***) является обратным ребром, что запрещено в бесконтурном орграфе.
- Если вершина ***y*** *обработана*, то она помечается соответствующим образом раньше вершины ***x***.

Топологическая сортировка

```
process_vertex_late(int v)
{
  push(&sorted,v);
}
process_edge(int x, int y)
{
  int class; /* Класс ребра */
  class = edge_classification(x,y);
  if (class == BACK)
     printf("Warning: directed cycle found, not a DAG\n");
}
topsort(graph *g)
{
  int i; /* Счетчик */
  init_stack(&sorted);
  for (i=1; i<=g->nvertices; i++)
     if (discovered[i] == FALSE)
        dfs(g,i);
  print_stack(&sorted);  /* Выводим топологическое упорядочивание */
}
```