SMWare

Анализ потока управления и потока данных в программе

Новиков Сергей

Содержание

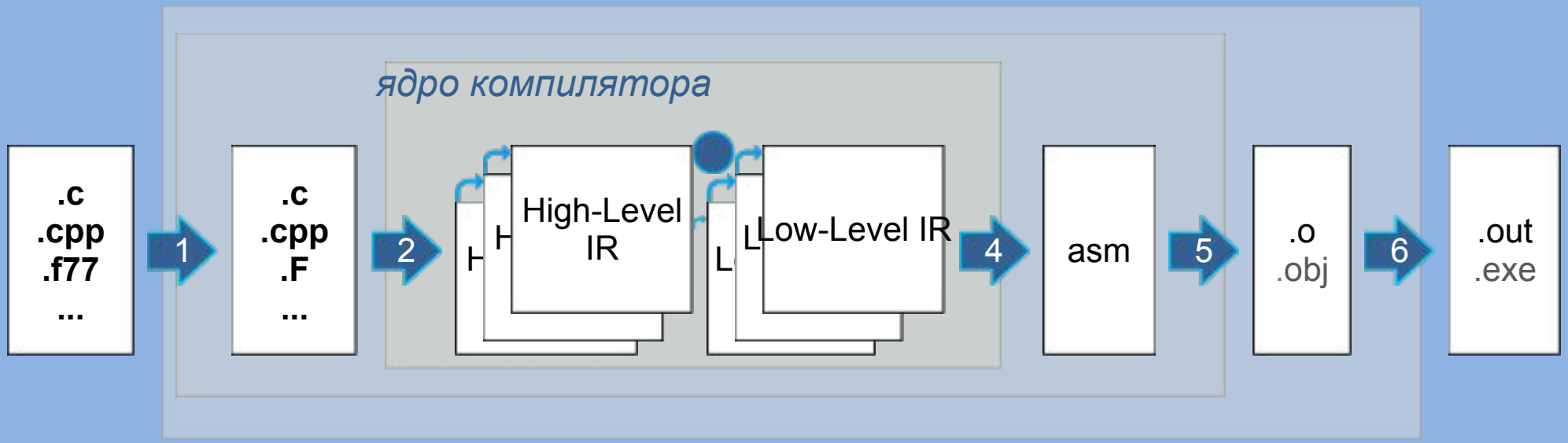
Agenda

- Структура компилятора
- Пример программы на C
- Линейная последовательность операций
- Анализ потока управления
- Анализ потока данных
- Примеры оптимизаций
- Литература к лекции

Структура компилятора

Compiler structure

Компилятор - переводит исходный код программы (написанные на языке высокого уровня) в эквивалентный код на языке целевой платформы



1. 1.Препроцессор
   - 2.Front-End
   - **3.Оптимизации**
   - 4.Кодогенератор
   - 5.Ассемблер
   - 6.Линкер

Пример (исходый код программы на C)

```
1. int func( int a, int b)
 – {
 – int res = 0;
 – int c = 10;
 – int d = 20;
 – int i, j, k = 0;
 – for ( i = 0; i < 100; i++ )
 – {
 – for ( j = 0; j < 100; j++ )
 – {
 – if ( i + j < a + b )
 – {
 – res += a + b + i;
```

Линейная последовательность операций

1. MOVE.s32 <s32:0> -> res // line:3,0
2. MOVE.s32 <s32:10> -> c // line:4,0
3. MOVE.s32 <s32:20> -> d // line:5,0
4. MOVE.s32 <s32:0> -> k // line:6,0
5. MOVE.s32 <s32:0> -> i // line:8,0
6. GOTO <mo_l0:#nil> // line:8,0
7. LABEL //

...

52. IF bool_tvar.15, <mo_l0:#nil>, <mo_l0:#nil> // line:8,0
53. LABEL //
54. MOVE.s32 res -> D.1572 // line:23,0
55. MOVE.s32 D.1572 -> D.1552 //
56. RETURN D.1552 //

Граф потока управления

Определение:

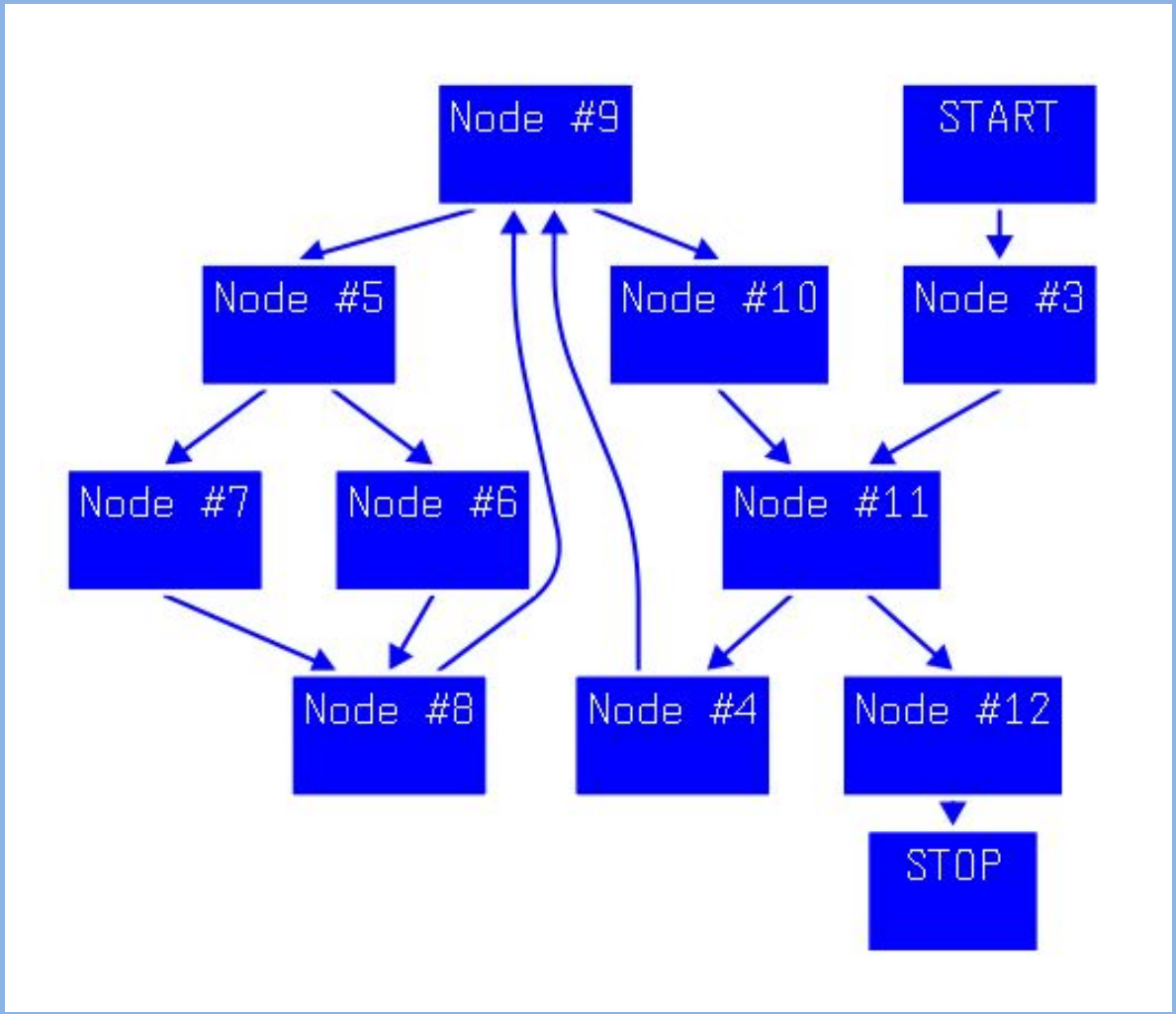
$G=(V, E, start, stop)$ - граф потока управления $\Leftrightarrow$

1. (V, E) - ориентированный граф
2. $start \in G.V$, $stop \in G.V$
3. $|in(start)|=|out(stop)|=\varnothing$
4. $\forall v \in G.V\ start \rightarrow^* v \rightarrow^* stop$

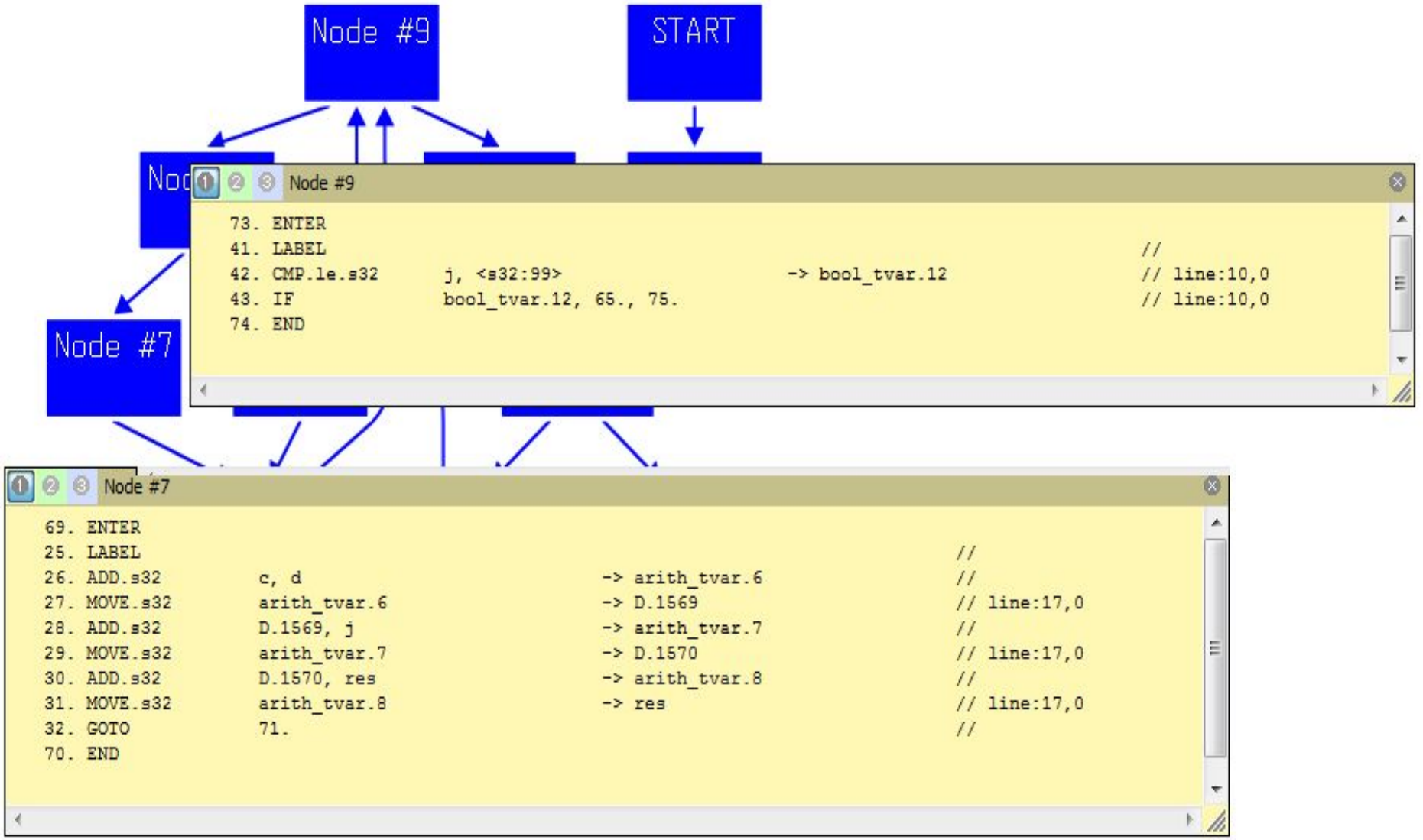
Обозначения:

1. $e=(v,w) \in E \Rightarrow beg(e)=v,\ end(e)=w$
2. $v \in V \Rightarrow in(v)=\{e \in E | end(e)=v\}$
3. $v \in V \Rightarrow out(v)=\{e \in E | beg(e)=v\}$
4. $p=(v_1,v_2,\ldots,v_k),\ \forall i=1,\ldots,k\text{-}1\ v_i \in V,\ v_k \in V,\ (v_i,v_{i+1}) \in E$

Граф потока управления



Граф потока управления с промежуточным представлением



Действия на графе потока управления

- Обход (нумерация)
  - Обход в глубину (depth first)
    1. для каждого преемника {
    2. устанавливаем номер ++
    3. обходим рекурсивно преемника }
  - Обход в ширину (reverse post order)
    1. для каждого преемника {
    2. обходим рекурсивно преемника }
    3. устанавливаем номер --
- Маркирование
- Клонирование
- Построение дерева доминаторов/постдоминаторов
- Построение дерева циклов

Обязательное предшествование (доминирование)

Обязательное предшествование

Отношение обязательного предшествования (<):

$\forall v,w \in V\ v<w \Leftrightarrow \forall p=(start,\ldots,w)\ v \in p$

Строгое обязательное предшествование (sdom):

$\forall v,w \in V\ (v\ sdom\ w) \Leftrightarrow v<w\ \&\ v \neq w$

Непосредственное обязательное предшествование (idom):

$\forall v,w \in V\ (v\ idom\ w) \Leftrightarrow (v\ sdom\ w)\ \&\ \forall u \in V\ (u\ sdom\ w) \Rightarrow u<v$

Свойства:

1. $\forall v \in V\ v<v$
2. $\forall v \in V\ start<v,\ v \neq start \Rightarrow (start\ sdom\ v)$
3. $\forall v \in V\ \forall w,u \in V\ (w\ sdom\ v)\ \&\ (u\ sdom\ v) \Rightarrow (w<u)\ \vee (u<w)$

Определение:

$(v, w) \in E$ - обратная $\Leftrightarrow w < v$

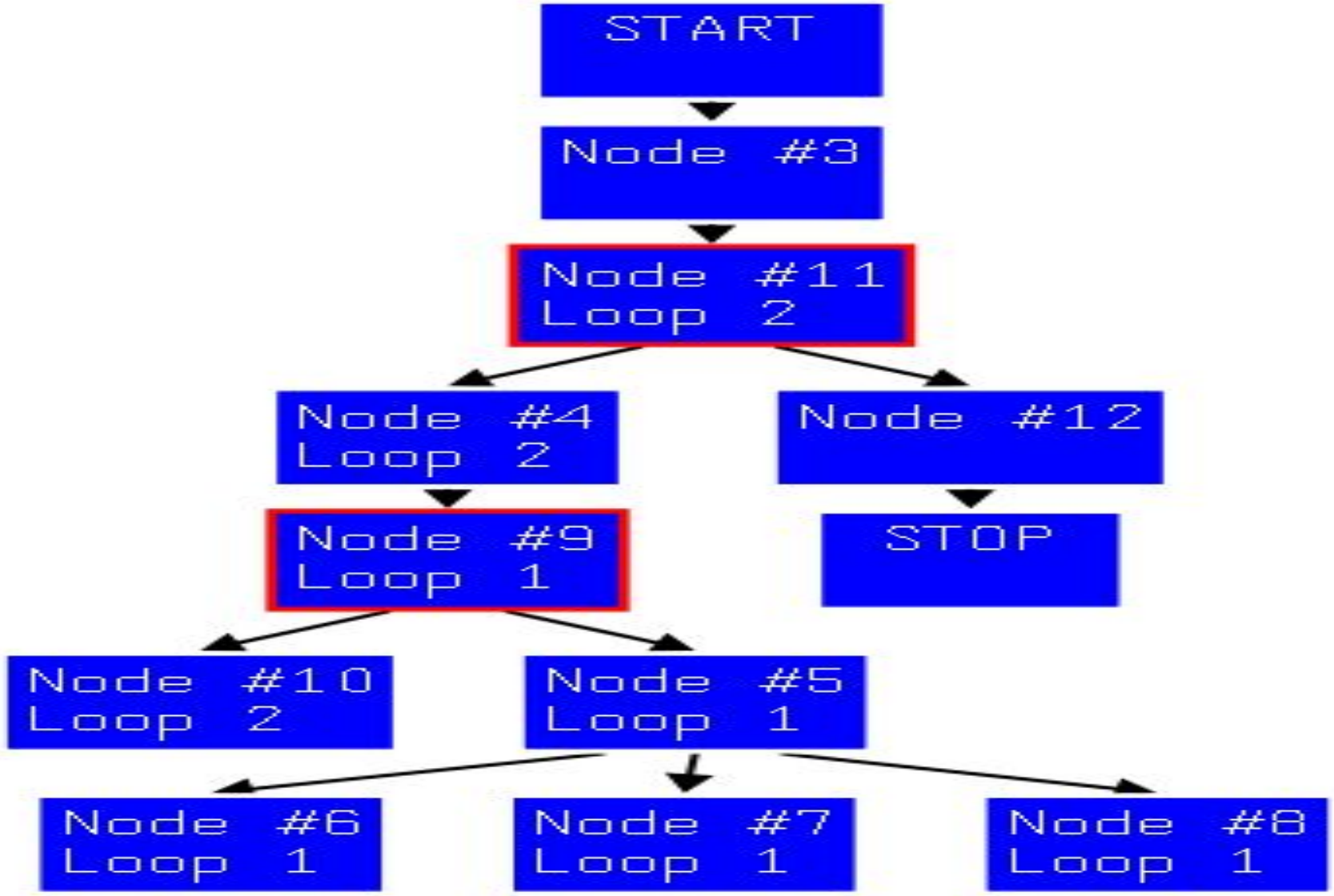
Свойство доминирования/постдоминирования

- Узел d доминирует/постдоминирует узел n если любой путь от стартового/стопового узла к n проходит через d
- Алгоритмы построения дерева доминаторов/постдоминаторов
  - Простейший алгоритм O(N*N)
  - Lengauer-Tarjan алгоритм *O((N+E)log(N+E))*

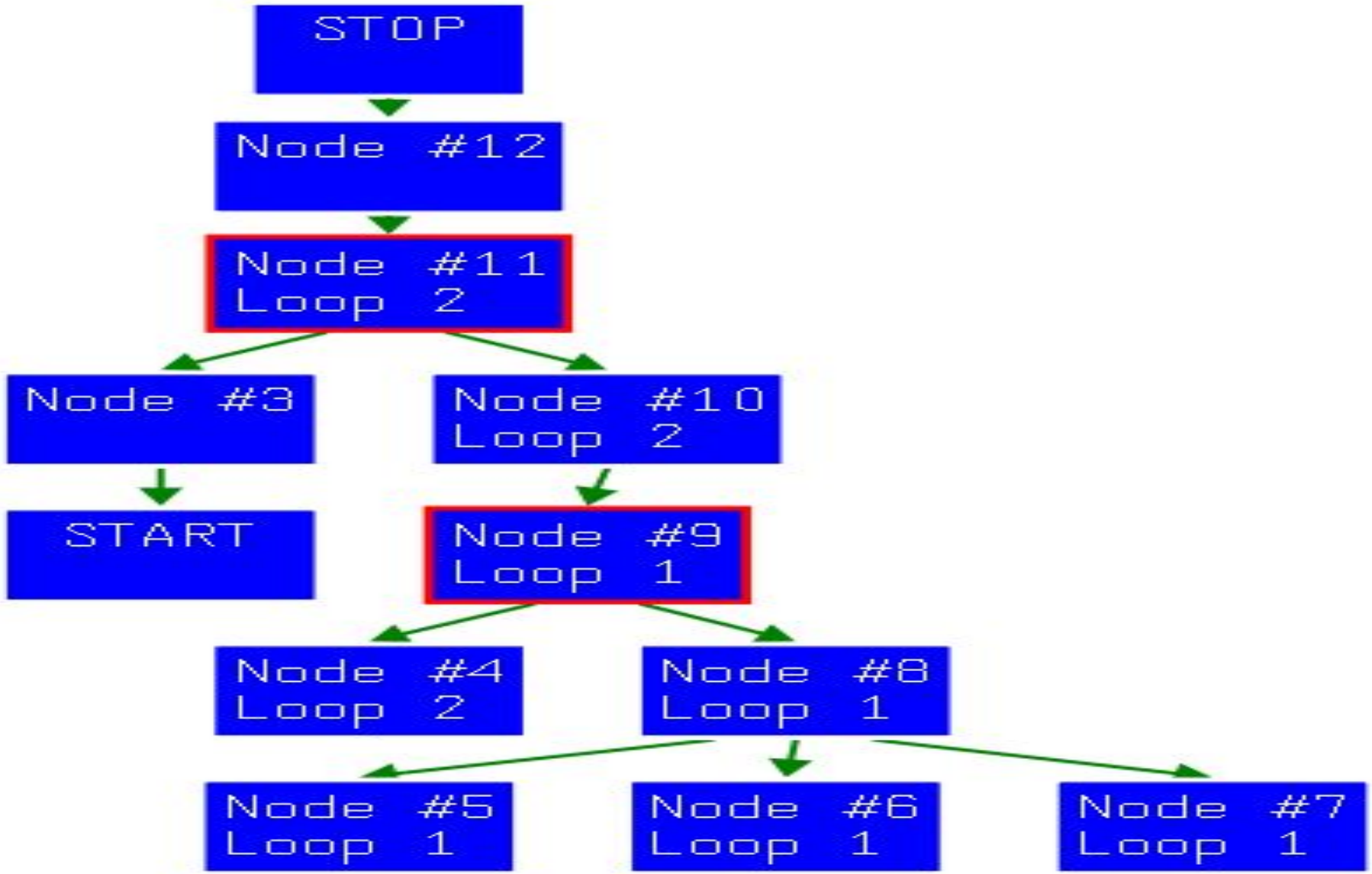
$$\mathrm{Dom}(n_o) = \{n_o\}$$

$$\mathrm{Dom}(n) = \left(\bigcap_{p \in \mathrm{preds}(n)} \mathrm{Dom}(p) \right) \bigcup \{n\}$$

Дерево доминаторов



Дерево постдоминаторов



Глубинное остовное дерево (depth-first spanning tree)

Нумерация:

$\#: V \to [1..|V|]$ - взаимно-однозначное

G - граф, $\#$ - нумерация, $e=(v, w)$

1. e - прямая в смысле $\# \Leftrightarrow \#(v)<\#(w)$
2. e - обратная в смысле $\# \Leftrightarrow \#(v)\geq\#(w)$

Глубинное остовное дерево T:

остовное дерево, полученное обходом в глубину, начиная со *start*

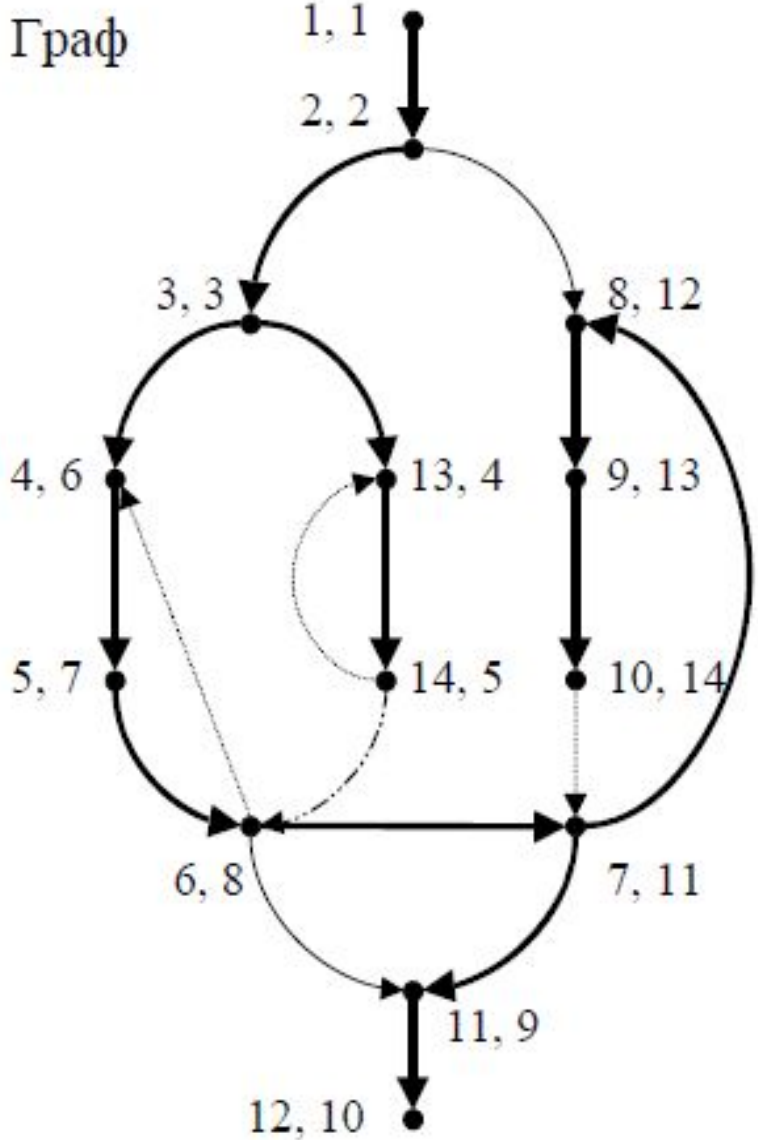
Тип дуги $e=(v, w)$ по отношению к T:

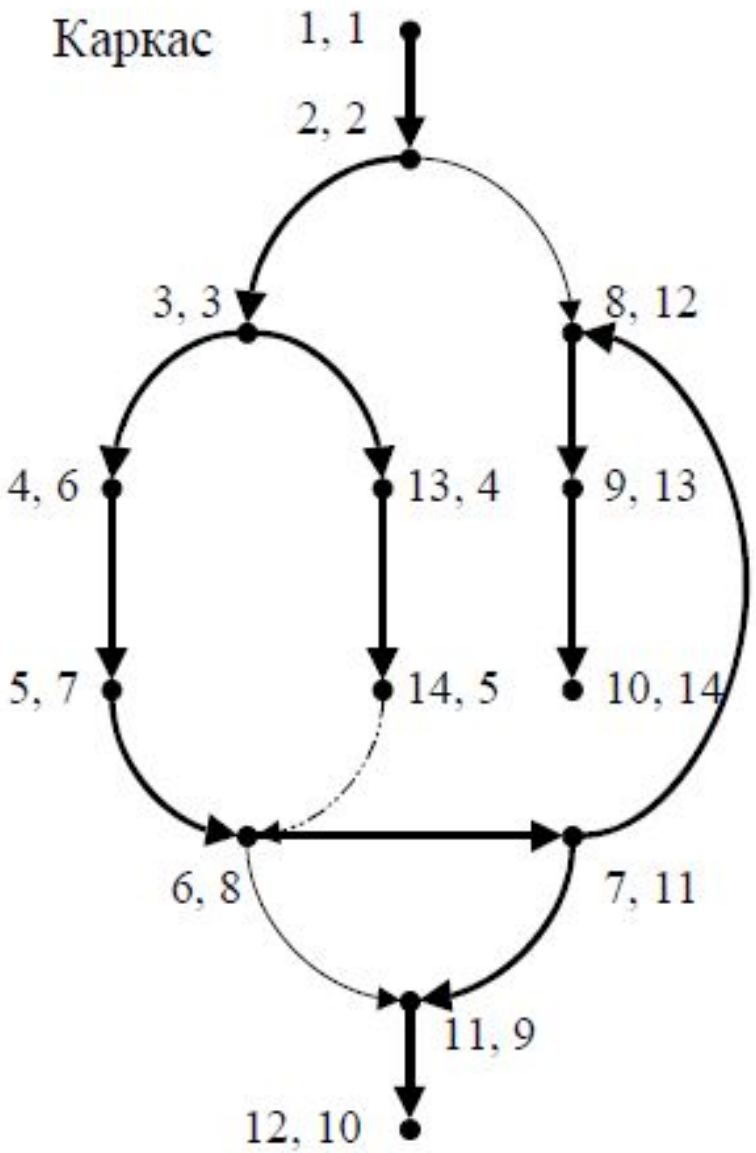
1. e - деревянная $\Leftrightarrow e \in T$
2. e - прямая $\Leftrightarrow v \to_T^* w$
3. e - обратная $\Leftrightarrow w \to_T^* v$
4. e - поперечная

Прямая и обратная нумерации:

1. *Pre:* $V \to [1..|V|]$ - прямая, отражает порядок включения вершин в T
2. *Post:* $V \to [1..|V|]$ - обратная, отражает порядок, обратный порядку исключения вершин из T

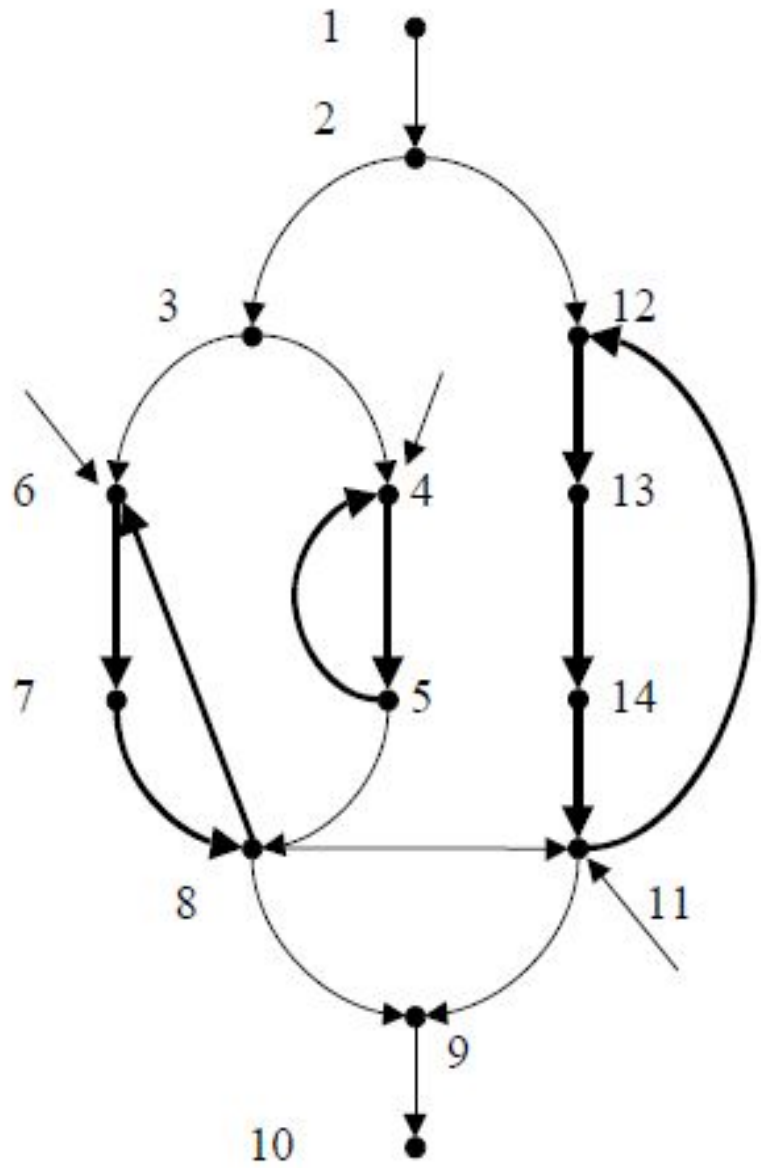
Глубинное остовное дерево (пример)



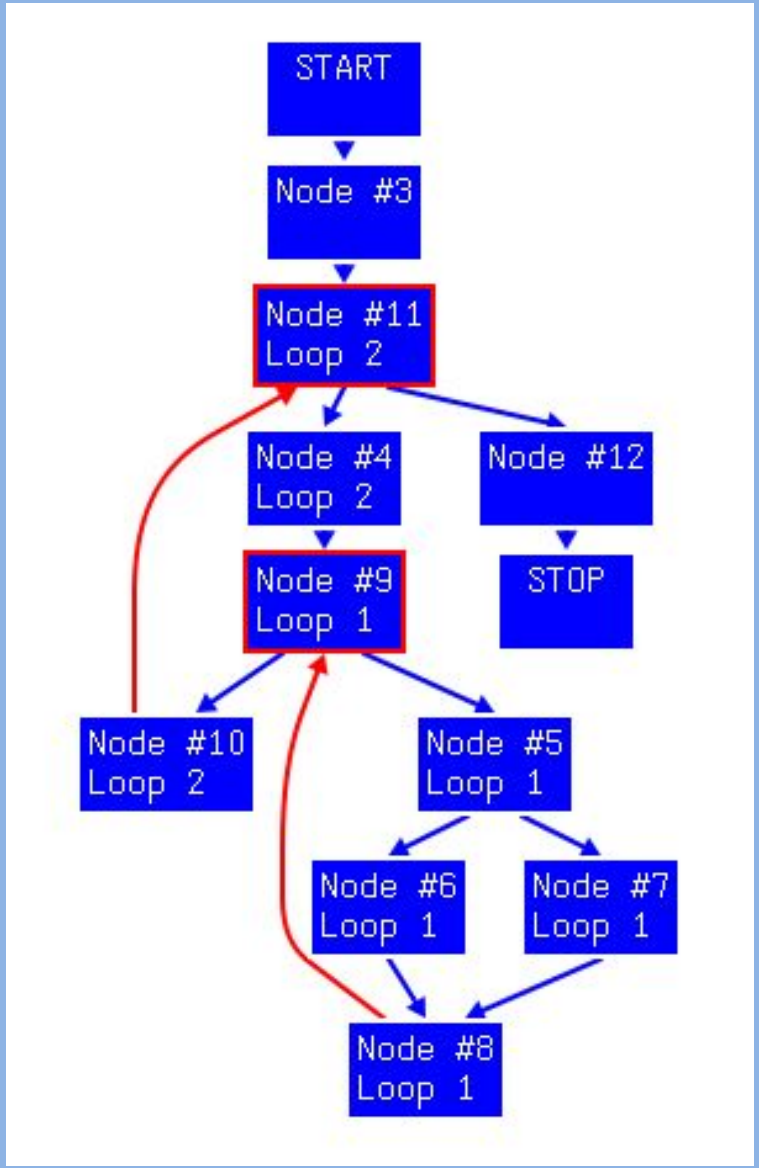


Выделение сильно связных подграфов

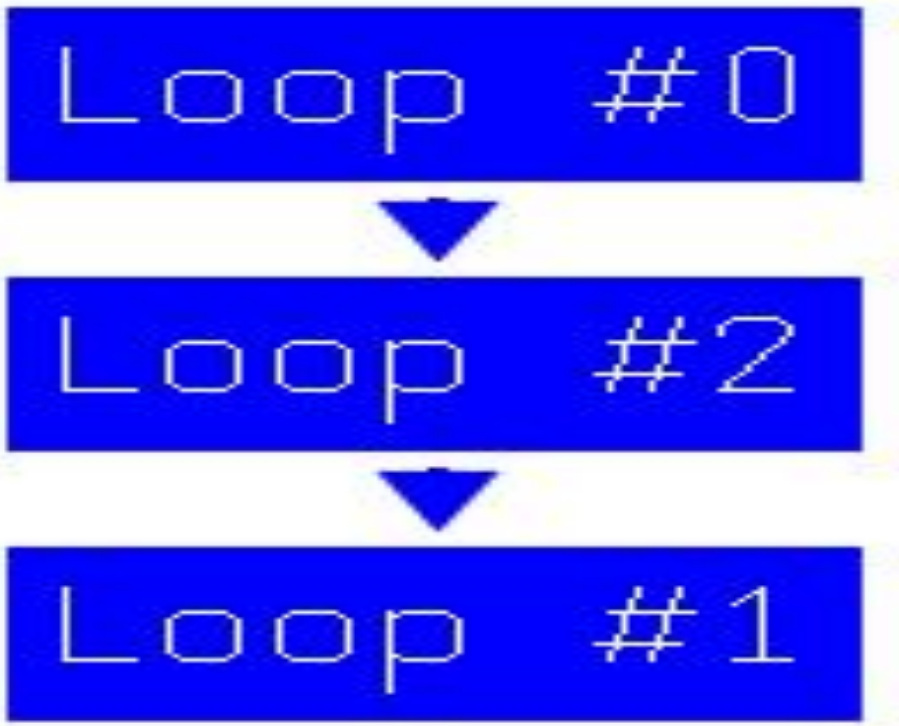
```
set<Vertex> Area (v: Vertex; N: Numbering)
{
     set<Vertex> r = {v};

     void Inner (u: Vertex, N: Numbering)
     {
          for ∀w : ∃(w,u)∈E do
               if (N(w) > N(v))
               {
                    r = r∪{w};
                    Inner (w, N);
               }
     }

     Inner (v, N);

     return r;
}
```



Разметка циклов

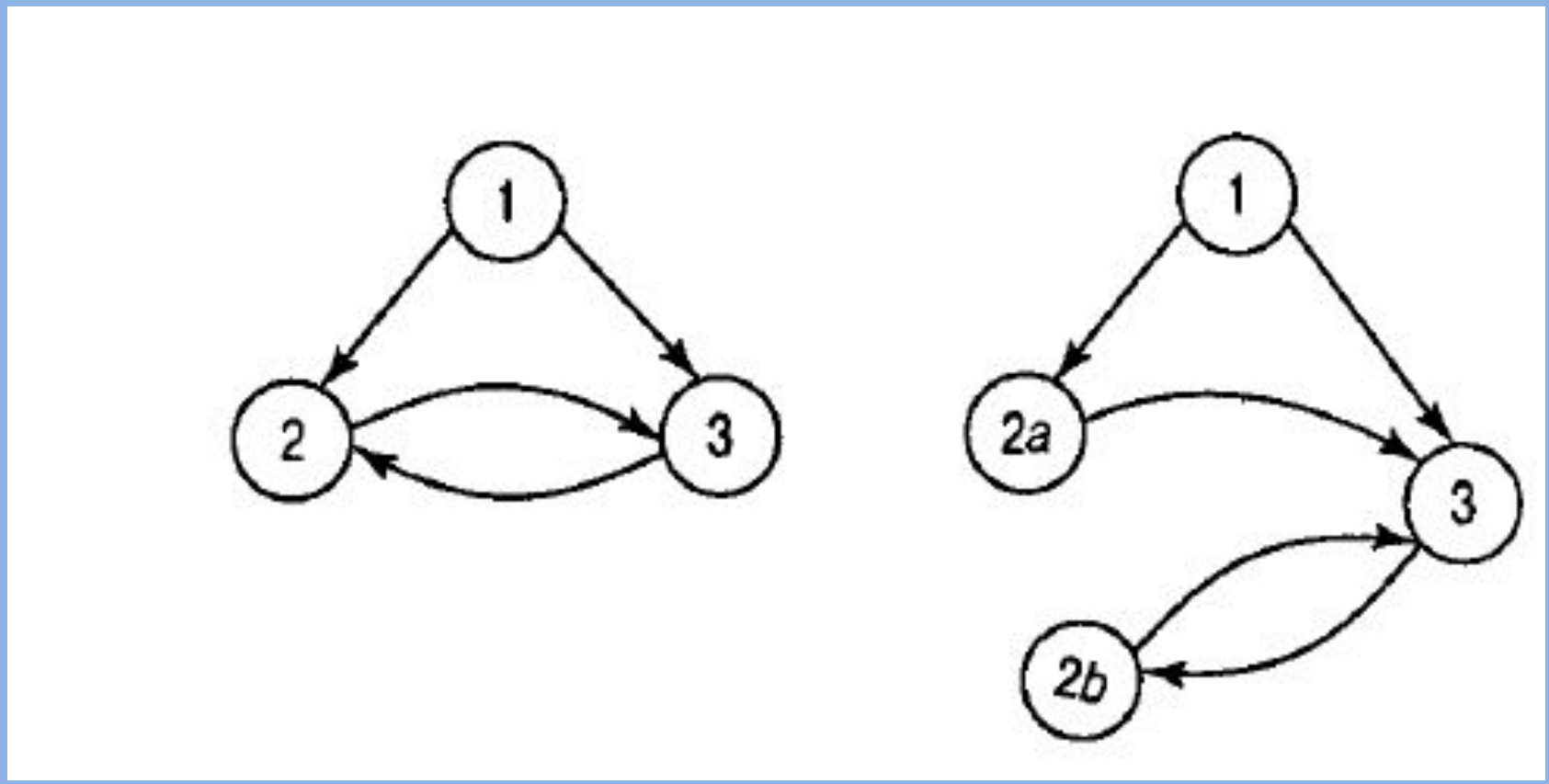


Дерево циклов

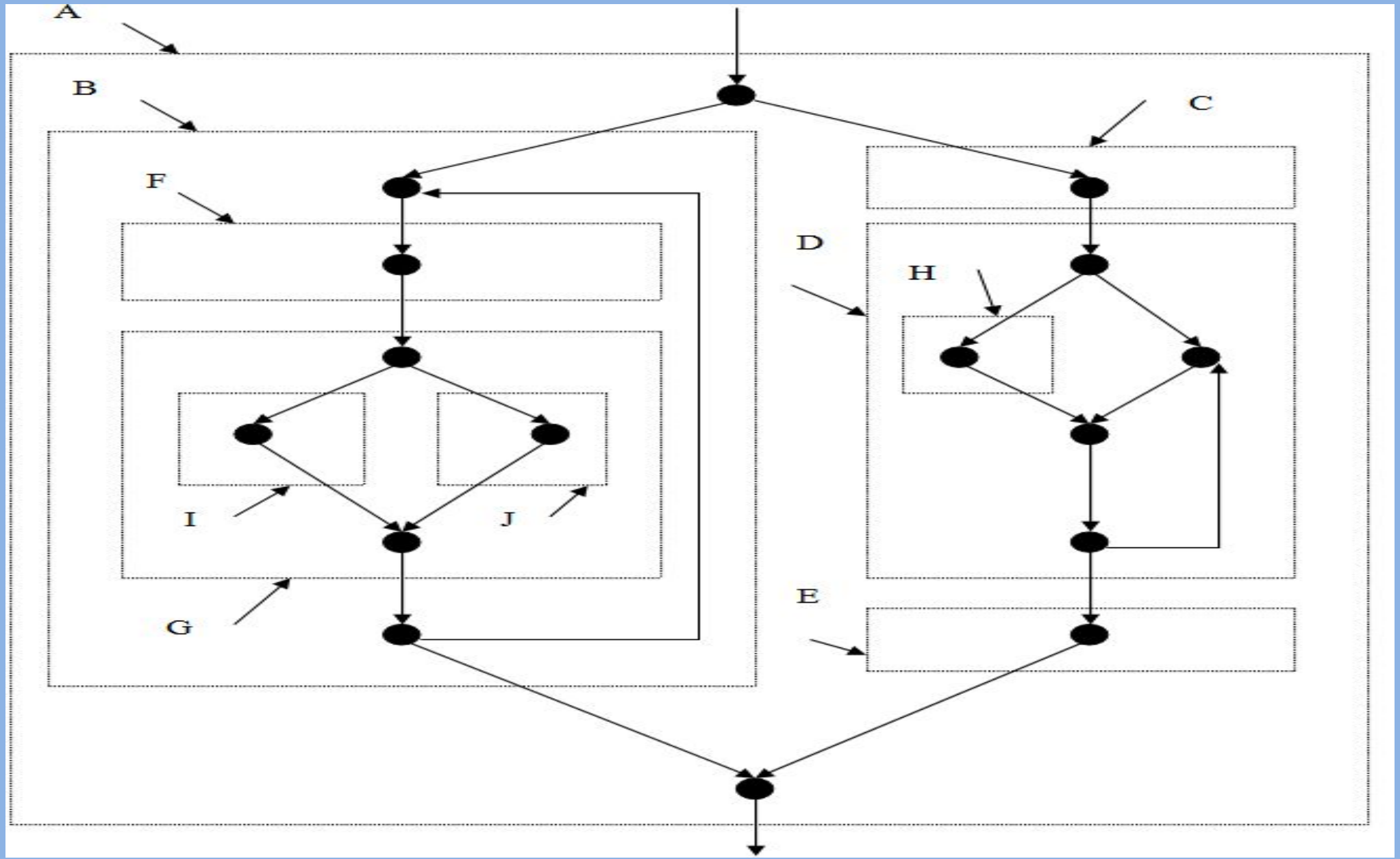


Несводимые циклы

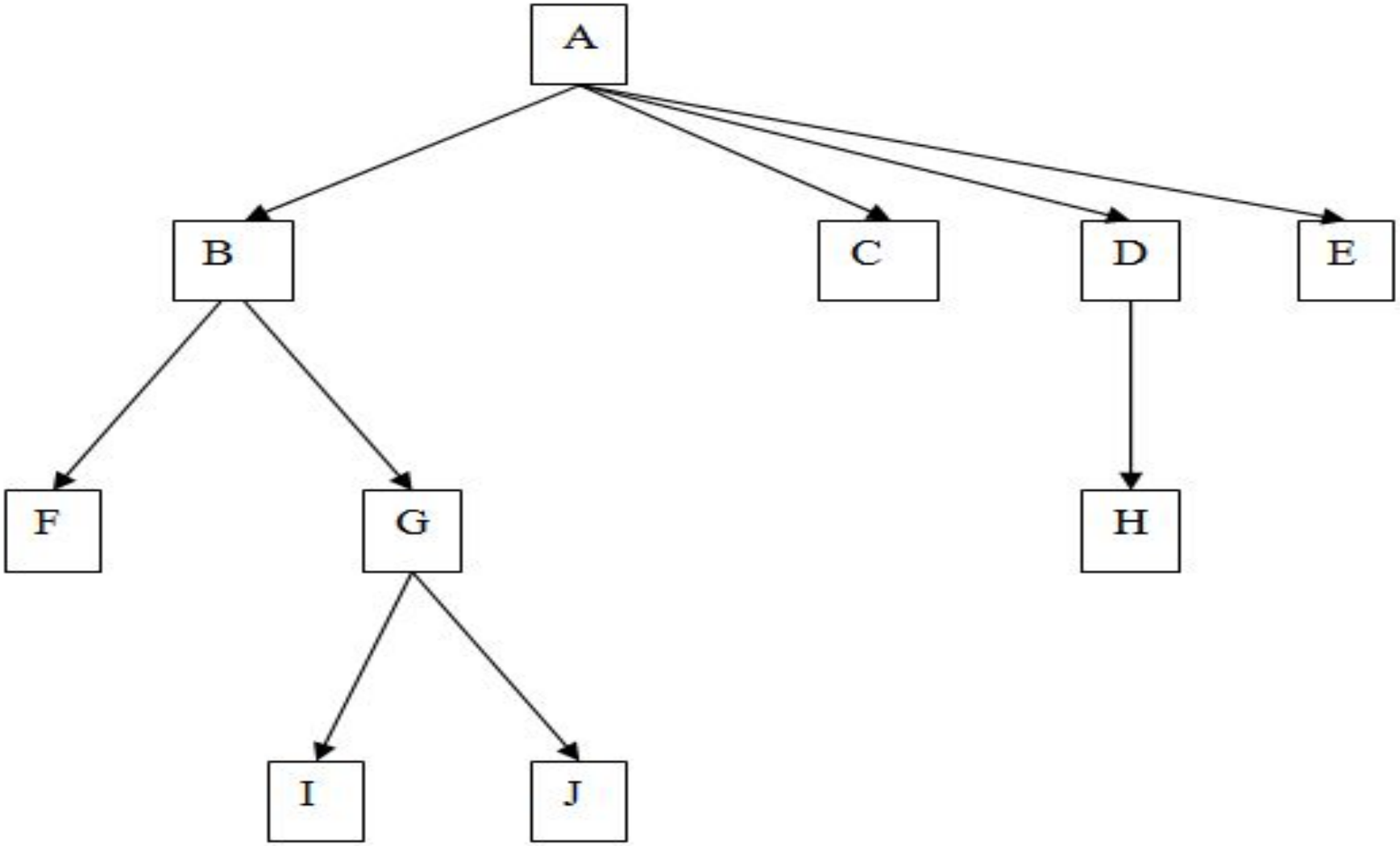
- Несводимый цикл – цикл с более, чем одним входом
- Цикл можно свести путем дублирования кода



Компоненты с одним входом и одним выходом



Дерево структуры программы (program structure tree)



Классический анализ потока данных

1) $\text{OUT}[\text{Вход}] = v_{\text{Вход}}$;
2) **for** (каждый базовый блок B, отличный от входного) $\text{OUT}[B] = \top$;
3) **while** (внесены изменения в OUT)
4) **for** (каждый базовый блок B, отличный от входного) {
5) $\text{IN}[B] = \wedge_{P - \text{предшественник } B} \text{OUT}[P]$;
6) $\text{OUT}[B] = f_B(\text{IN}[B])$;
}

а) Итеративный алгоритм для прямой задачи потока данных

1) $\text{IN}[\text{Выход}] = v_{\text{Выход}}$;
2) **for** (каждый базовый блок B, отличный от выходного) $\text{IN}[B] = \top$;
3) **while** (внесены изменения в IN)
4) **for** (каждый базовый блок B, отличный от выходного) {
5) $\text{OUT}[B] = \wedge_{S - \text{преемник } B} \text{IN}[S]$;
6) $\text{IN}[B] = f_B(\text{OUT}[B])$;
}

б) Итеративный алгоритм для обратной задачи потока данных

Время жизни переменных

Def (or definition)

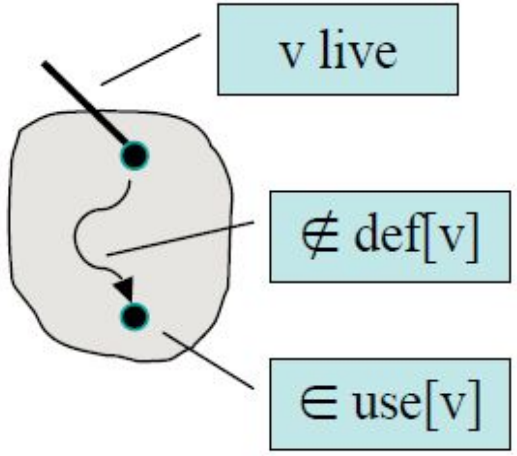
- An **assignment** of a value to a variable
- def[v] = set of CFG nodes that define variable v
- def[n] = set of variables that are defined at node n

`a = 0`

Use

- A **read** of a variable's value
- use[v] = set of CFG nodes that use variable v
- use[n] = set of variables that are used at node n

`a < 9?`



More precise definition of liveness

- A variable v is live on a CFG edge if
  (1) ∃ a directed path from that edge to a use of v (node in use[v]), and
  (2) that path does not go through any def of v (no nodes in def[v])

Итерационный алгоритм определения времени жизни переменных

Algorithm

for each node n in CFG
    in[n] = ∅; out[n] = ∅
} initialize solutions

repeat
    for each node n in CFG
        in'[n] = in[n]
        out'[n] = out[n]
} save current results
        in[n] = use[n] ∪ (out[n] – def[n])
        $out[n] = \bigcup_{s \in succ[n]} in[s]$
} solve data-flow equations

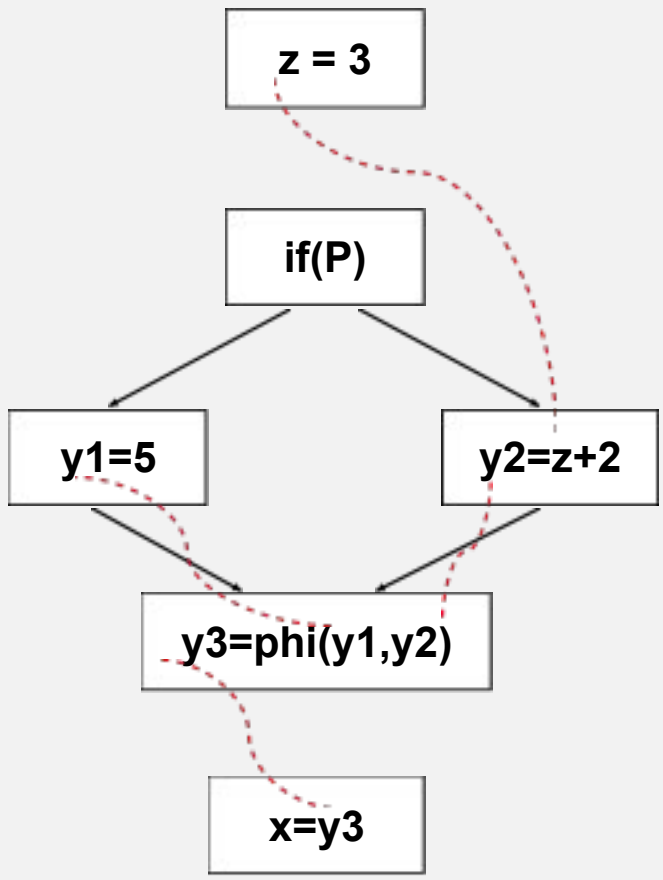
until in'[n]=in[n] and out'[n]=out[n] for all n
} test for convergence

Форма статического единственного присваивания

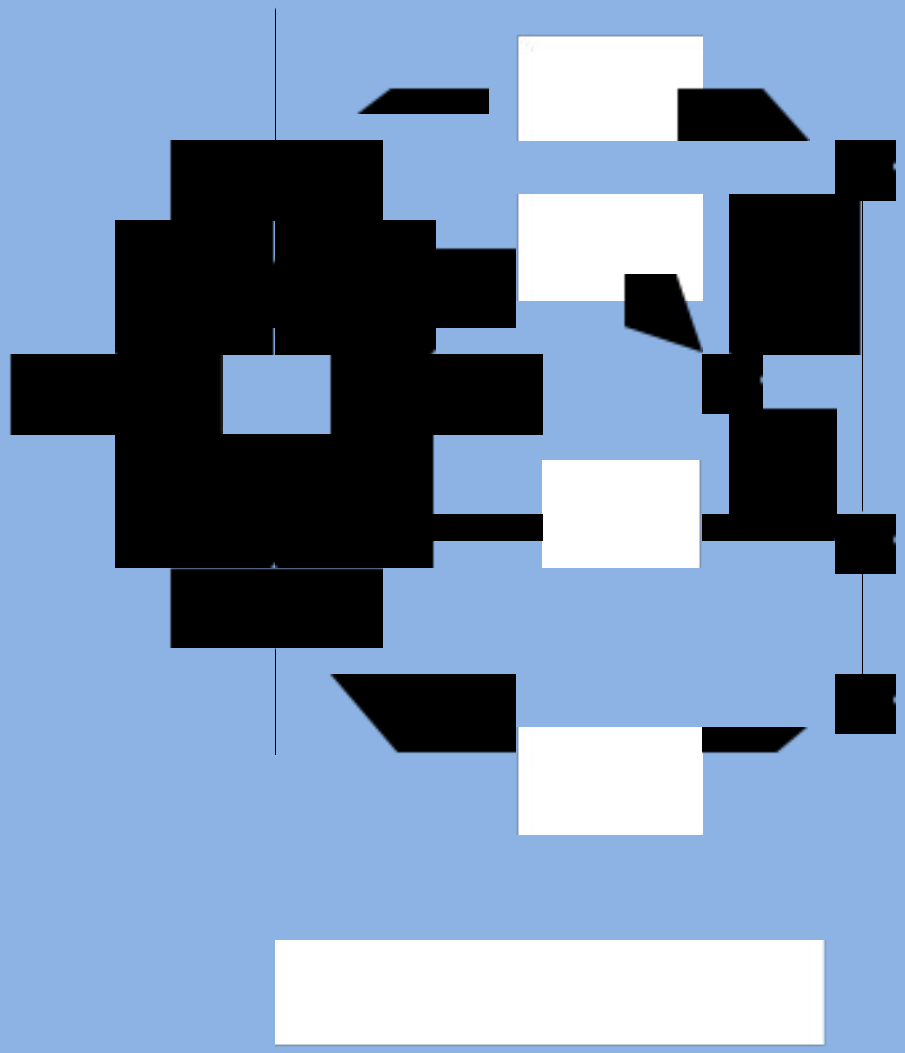
Фрагмент программы

```
z = 3;
if(P)
{
y = 5;
} else
{
y = z + 2;
}
x = y;
```

SSA - форма



Форма статического единственного присваивания в виде Def-Use графа

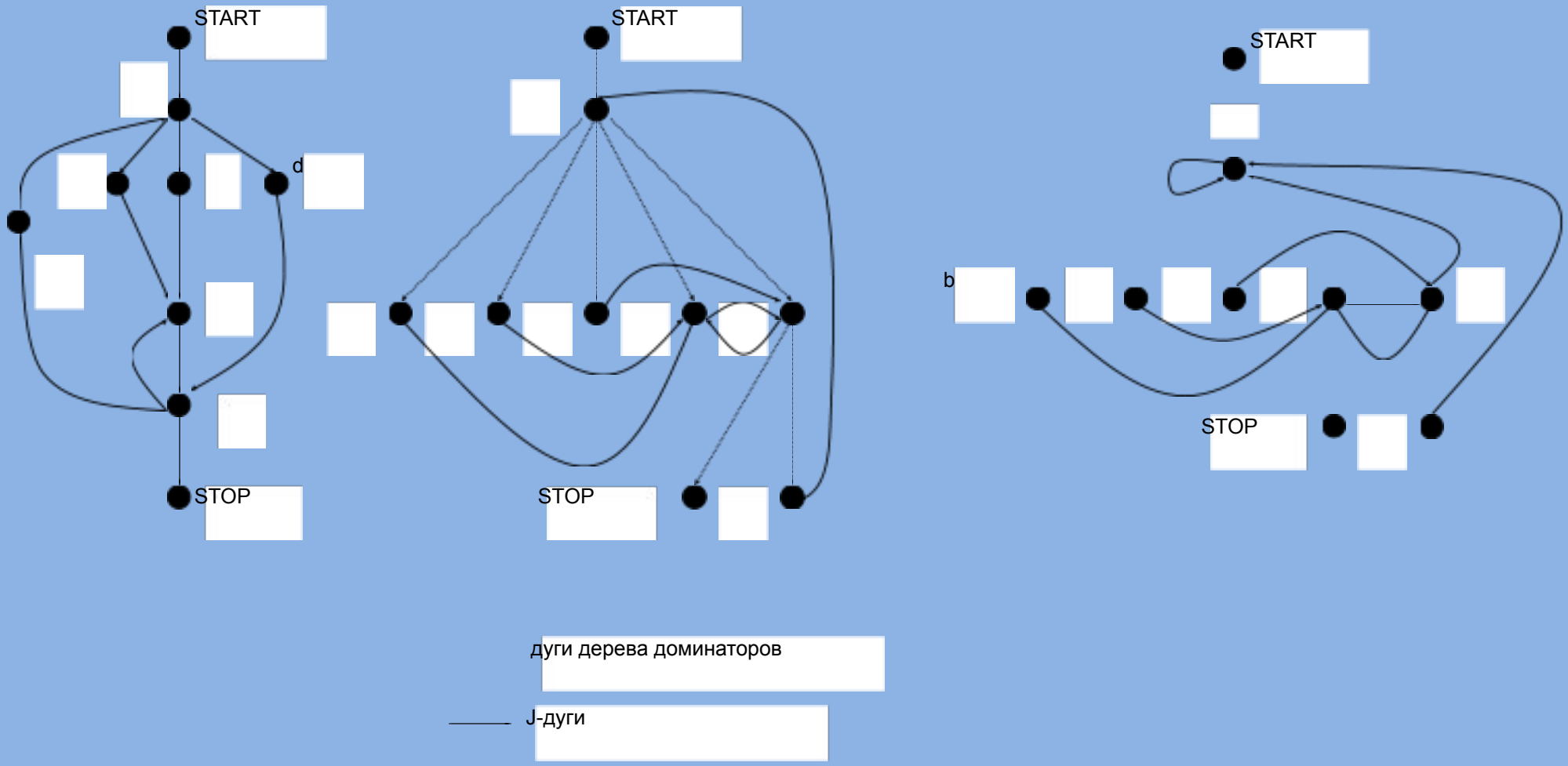


Построение SSA/Def-Use графа

- Построение phi-функций
  - Для каждой переменной определяем узлы cfg, в которых она инициализируется
  - Запускаем алгоритм поиска итерационного фронта доминирования (сложность O(|N|*|DF|)*B/size(word))
    - N – количество узлов в графе потока управления
    - DF – итерационный фронт доминирования для одного узла (в среднем 1-2 на задачах)
    - B – количество переменных
    - size(word) – размер слова в битовом векторе
  - По результатам работы алгоритма строим phi-функции
- Линковка записей и чтений

Фронт доминирования

- CFG CFG+DOM Dominance Frontier

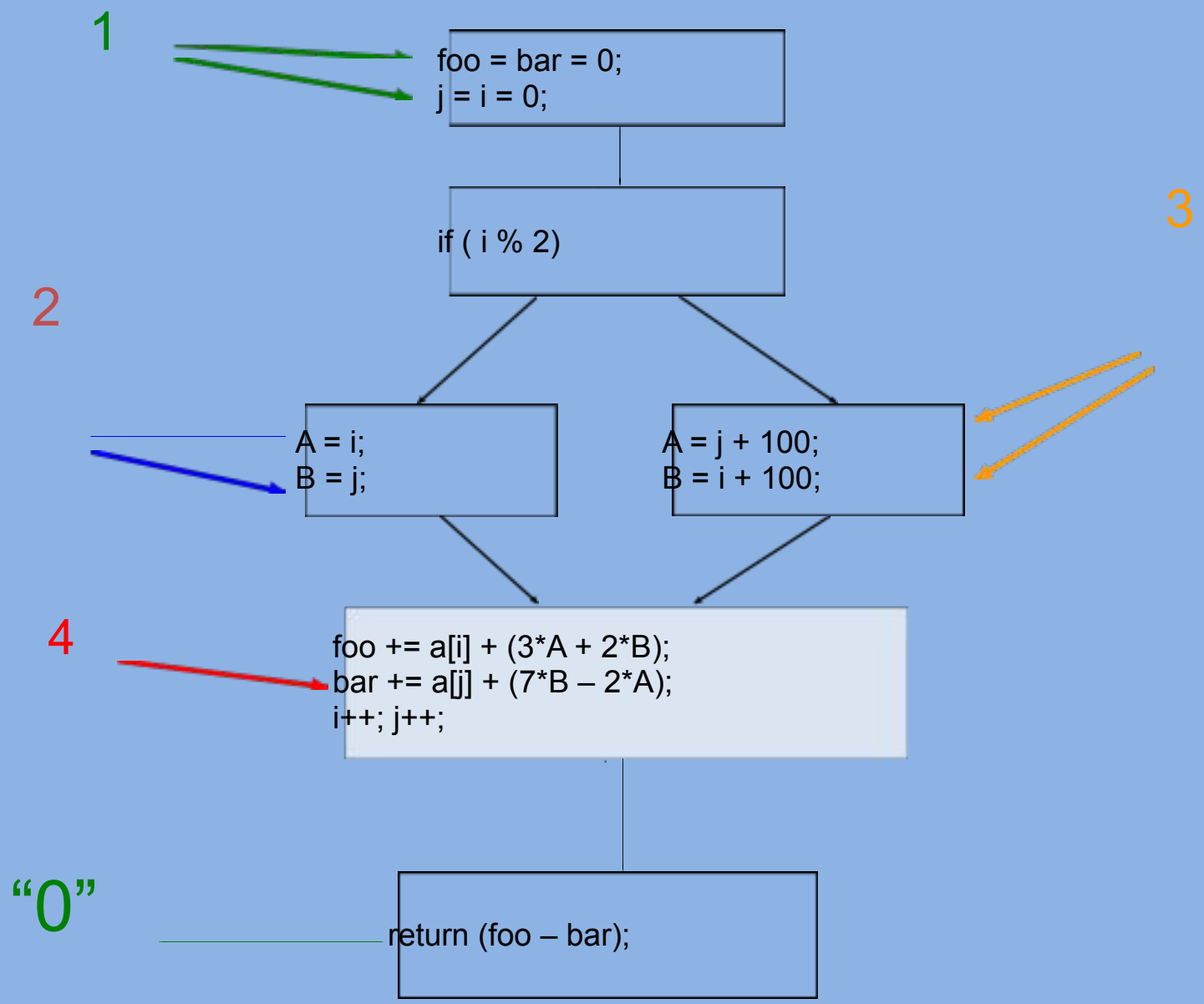


Метод нумераций значений

- Хорошо зарекомендовавшая себя техника потокового анализа.
- Анализ присваивает одинаковые номера операциям, вырабатывающие одинаковые значения. Номера называются **классами эквивалентности**.
- Алгоритмическая сложность O(N * D * Argmax)
  - N количество операций
  - D глубина дерева циклов
  - Argmax максимальное число аргументов у операции

Метод нумераций значений (пример)

- Классы эквивалентности: 1,2,3,4



Исходый код программы

```
1. int func( int a, int b)
 – {
 – int res = 0;
 – int c = 10;
 – int d = 20;
 – int i, j, k = 0;
 – for ( i = 0; i < 100; i++ )
 – {
 – for ( j = 0; j < 100; j++ )
 – {
 – if ( i + j < a + b )
 – {
 – res += a + b + i;
```

Примеры оптимизаций

- 16 (c + d) подстановка констант
- 11,13 (a+b) сбор общих подвыражений
- 13,18 (b+i) удаление частично избыточных вычислений
- 20 (k++) удаление избыточных вычислений
- 11 (a+b) вынос инвариантных вычислений из цикла

Литература к лекции

- А. Ахо, Р. Сети, Дж. Ульман "Компиляторы: принципы, технологии и инструменты", М.: "Вильямс", 2001. 768 с.
- Steven S. Muchnik "Advanced Compiler Design And Implementation", Morgan Kaufmann Publishers, July 1997. 880 pp.
- В.Н. Касьянов "Оптимизирующие преобразования программ", М., "Наука", 1988. 336 с.