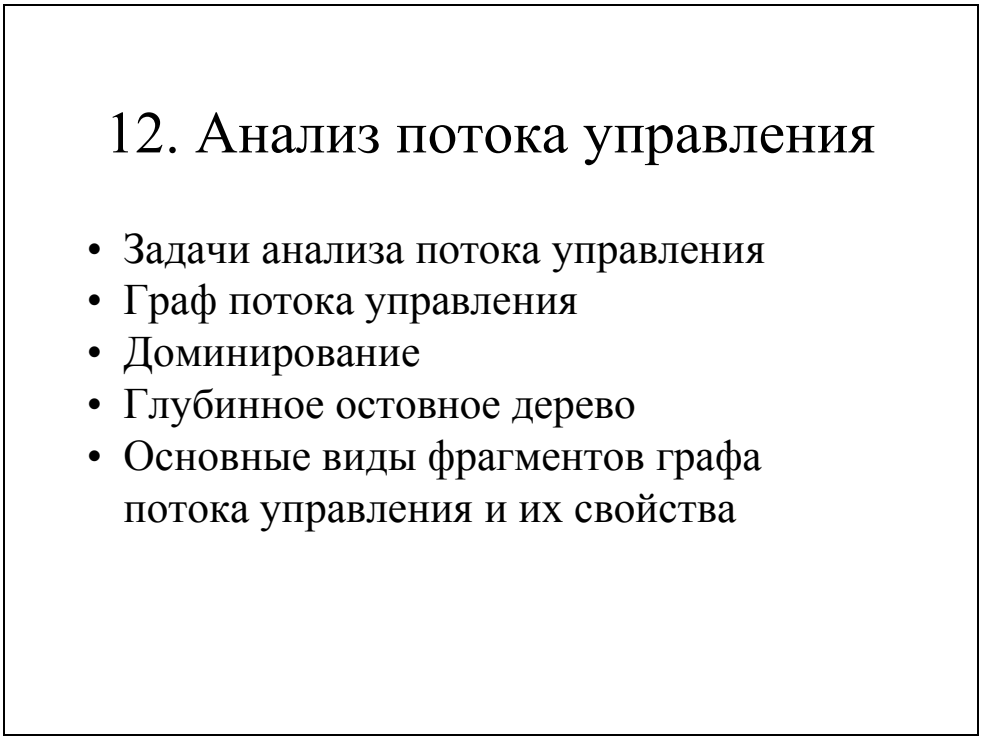
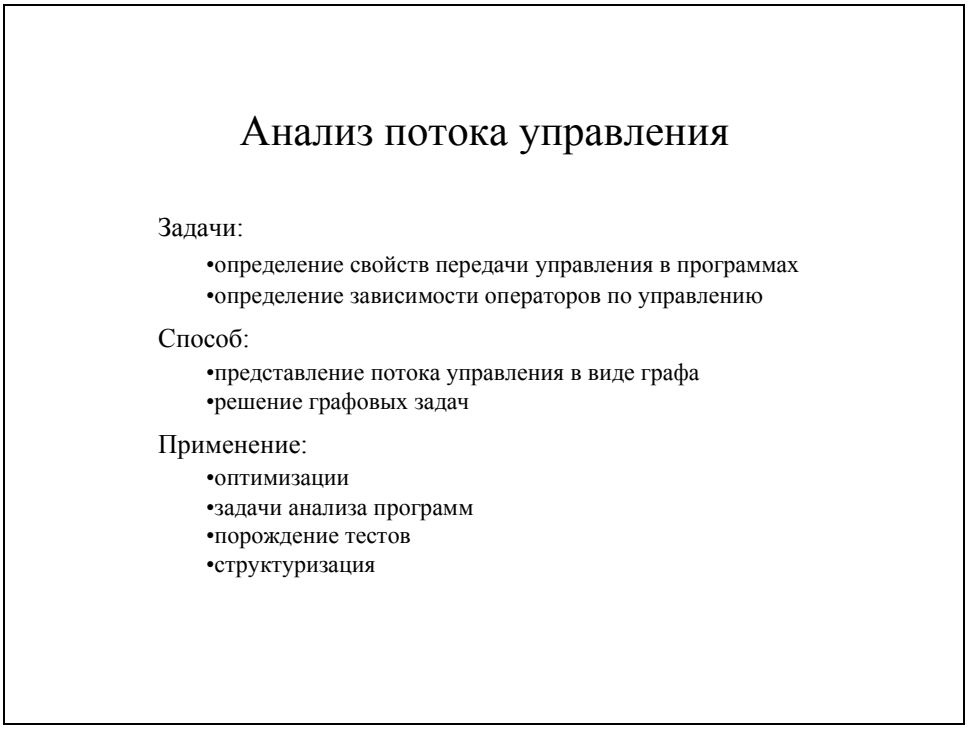


Лекция 12. Анализ потока управления

В этой лекции рассматриваются следующие вопросы:

- Задачи анализа потока управления
- Граф потока управления
- Доминирование
- Глубинное остовное дерево
- Основные виды фрагментов графа потока управления и их свойства



Анализ потока управления

В задачу *анализа потока управления* (*control flow analysis*) входит определение свойств передачи управления между операторами программы. Проверка многих свойств этого вида необходима для решения задач оптимизации, преобразований программ и т.д.

При решении этих задач обычно используется формальная графовая модель (т.е. анализ производится над графом потока управления), а сами задачи формулируются в теоретико-графовой форме.

Основное употребление анализа потока управления в оптимизации – это фрагментация, то есть представление графа потока управления в виде совокупности фрагментов определенного вида. Как было показано в лекции 11, такое представление необходимо для применения практически всех оптимизирующих преобразований.

Ниже будут рассмотрены основные понятия в области анализа потока управления, приведены определения некоторых фрагментов и алгоритмы их распознавания.

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. $p=(v_1,v_2,....,v_k),\ \forall i=1,...,k\text{-}1\ v_i \in V,\ v_k \in V,\ (v_i,v_{i+1}) \in E$

Граф потока управления

Основным способом представления потока управления программы является граф потока управления (см. лекцию 11) – ориентированный граф с двумя выделенными вершинами *start* и *stop*, такими, что

- в *start* не заходит ни одна дуга
- из *stop* не выходит ни одна дуга
- произвольная вершина принадлежит хотя бы одному пути из *start* в *stop*

Для произвольной дуги *e* обозначим через *beg(e)* ее начало, а через *end(e)* – ее конец.

Для произвольной вершины *v* обозначим через *in(v)* множество входящих в нее дуг, а через *out(v)* – множество исходящих дуг.

Путем в графе назовем последовательность вершин, такую, что между каждой последующей и предыдущей вершиной в графе существует ребро.

Обязательное предшествование

Отношение обязательного предшествования (<):

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

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 $\Leftrightarrow w < v$

Обязательное предшествование

Вершина *v обязательно предшествует* вершине *w*, если *v* принадлежит каждому пути в графе от *start* до *w*. В частности, любая вершина обязательно предшествует себе самой. Отношение обязательного предшествования будем обозначать символом '<'. Легко видеть, что это отношение рефлексивно и транзитивно, но не симметрично. Таким образом, отношение обязательного предшествования задает частичный порядок на множестве вершин графа.

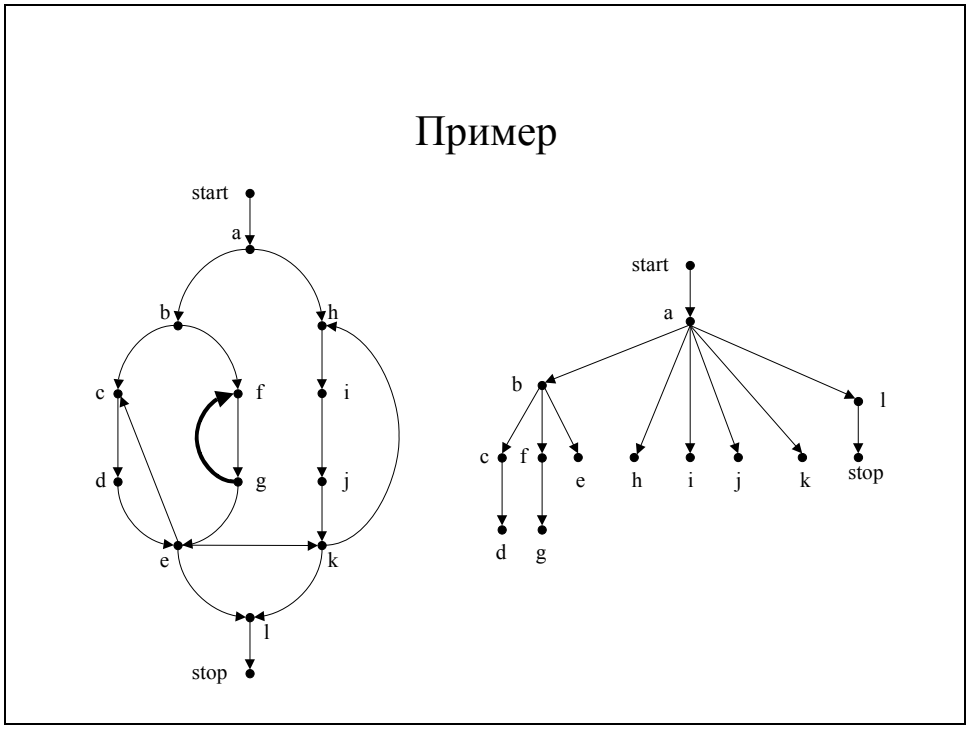
Вершина *v строго обязательно предшествует* вершине *w*, если она обязательно ей предшествует, но не совпадает с ней. Отношение строгого обязательного предшествования будем обозначать символом *sdom*.

Вершина *v непосредственно предшествует вершине w*, если она является ближайшей к *w* вершиной, которая ей строго предшествует.

Можно показать, что множество строгих обязательных предшественников для данной вершины линейно упорядочено, а непосредственный предшественник – это максимальный элемент в этом множестве. Таким образом, отношение непосредственного предшествования – это дерево. Легко показать, что корнем этого дерева является *start*.

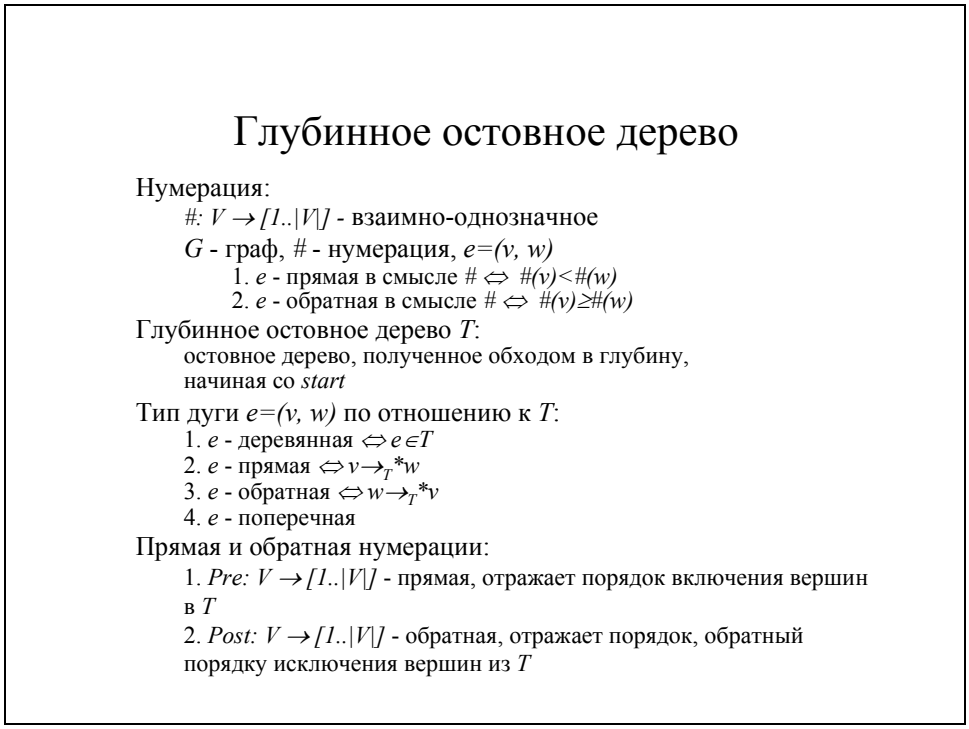
Дуга *(v, w)* называется обратной в том и только том случае, когда $w<v$.

Отношение обязательного предшествования играет чрезвычайно важную роль в задачах анализа потока управления. Например, с его помощью могут быть решены задачи проверки сводимости (см. ниже), построения статической формы единственного присваивания (в этой лекции не рассматривается) и т.д.



Пример

На слайде приведен пример графа потока управления и соответствующего ему дерева непосредственного предшествования. Дуга *(g, f)* является обратной.



Глубинное остовное дерево

Нумерацией называется взаимно однозначное отображение множества вершин графа на отрезок натурального ряда *[1..|V|]*. Для заданной нумерации # дуга *(v, w)* – прямая, если *#(v)<#(w)* и обратная в противном случае.

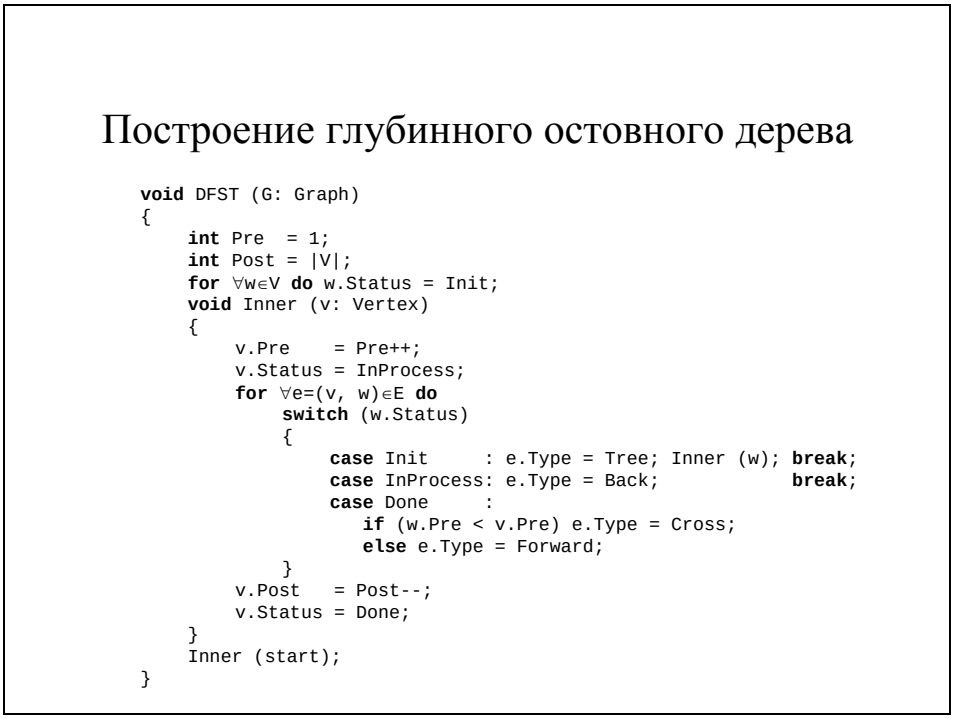
Остовное дерево – это дерево, содержащее все вершины графа и некоторые его дуги. Глубинное остовное дерево – это остовное дерево, полученное при обходе в глубину. Данный обход начинается в вершине *start* и заключается в последовательном включении вершин и ребер графа в дерево, если их еще там нет. При этом следующий сын вершины посещается лишь тогда, когда посещены все вершины, достижимые из предыдущего сына вершины.

Существует четыре типа дуг графа по отношению к данному глубинному остовному дереву: деревянные, прямые, обратные и поперечные. Следует отличать просто обратные дуги в графе (для определения которых было использовано отношение обязательного предшествования) и обратные дуги по отношению к глубинному остовному дереву.

К деревянным относятся те дуги, которые входят в состав остовного дерева.
К прямым дугам относятся те, чей конец достижим из начала в остовном дереве.
К обратным дугам относятся те, чье начало достижимо из конца в остовном дереве.
К поперечным дугам относятся все остальные.

Легко видеть, что при построении остовного дерева возникает порядок, в котором вершины графа включаются в рассмотрение. Нумерация, которая описывает этот порядок, называется прямой (*Pre*).

Аналогично, существует порядок, в котором вершины графа исключаются из рассмотрения. Нумерация, описывающая порядок, обратный к нему, называется обратной (*Post*).



Построение глубинного остовного дерева

На слайде приведен алгоритм построения остовного дерева, определения типов дуг графа по отношению к нему и построения нумераций *Pre* и *Post*.

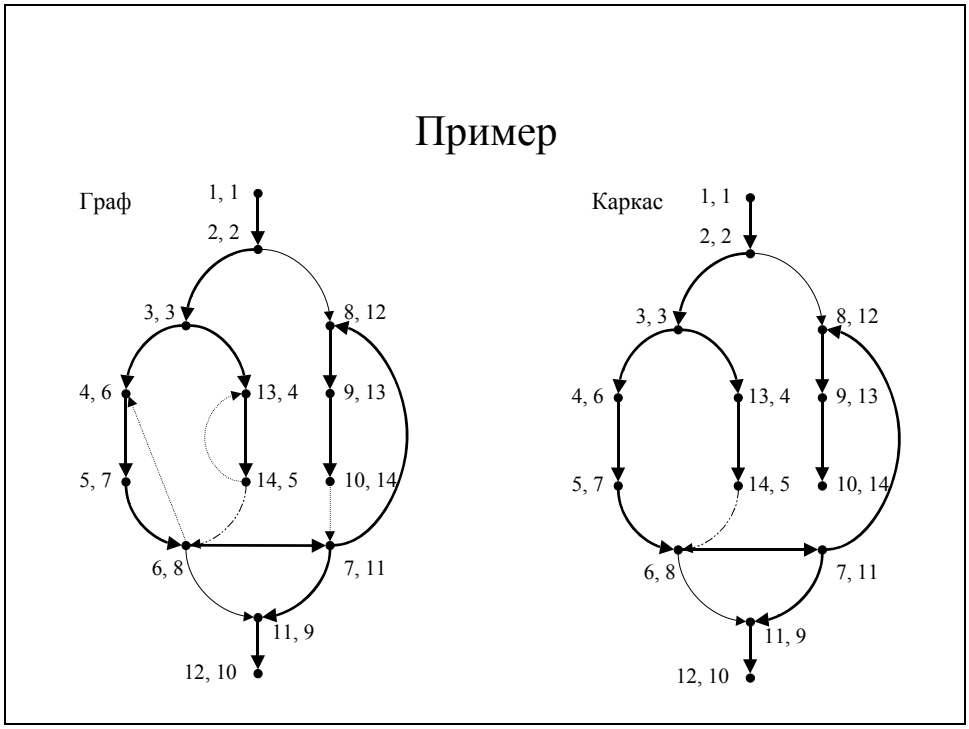
Алгоритм обходит вершины графа, начиная со *start*. При входе в очередную вершину ей присваивается очередной номер нумерации *Pre*, при этом номера *Pre* присваиваются в порядке возрастания. Далее рассматриваются все потомки этой вершины, которые еще не рассматривались. К каждому потомку применяется этот же самый шаг алгоритма – таким образом, обеспечивается обход в глубину. Наконец, после рассмотрения всех потомков текущей вершины ей присваивается очередной номер нумерации *Post*. При этом номера *Post* присваиваются в порядке убывания.

По ходу работы алгоритма поддерживается три состояния вершин:

- Init – вершина еще не рассматривалась алгоритмом
- InProcess – вершина еще рассматривается алгоритмом (т.е. алгоритм находится в процессе обработки вершин, достижимых из данной)
- Done – вершина уже исключена из рассмотрения (т.е. все достижимые из нее вершины уже обработаны).

Для определения типа дуги используется состояние конечной вершины и нумерация *Pre*. Если вершина, в которую можно попасть из данной, еще не рассматривалась, то ребро, по которому в нее можно попасть, объявляется деревянным. Для определения остальных типов дуг используются следующие очевидные утверждения:

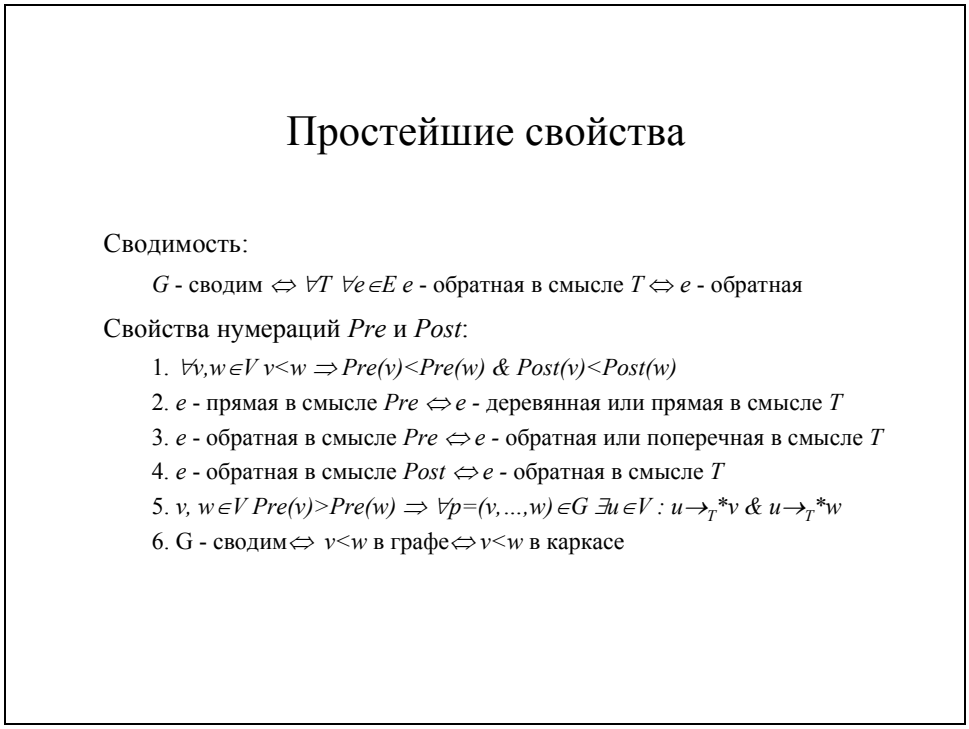
- если в дуге *(v, w) w* находится в состоянии InProcess, то эта дуга – обратная
- если в дуге *(v, w) w* находится в состоянии Done, и $Pre(w)<Pre(v)$, то эта дуга – поперечная
- если в дуге *(v, w) w* находится в состоянии Done, и $Pre(w)>Pre(v)$, то эта дуга – прямая



Пример

На слайде приведен пример графа потока управления и одного из его глубинных остовных деревьев. Каждая вершина помечена парой номеров, первый из которых соответствует нумерации *Pre*, а второй – нумерации *Post*. Деревянные дуги показаны толстыми линиями, прямые – тонкими, пунктирными линиями показаны обратные дуги и штрих-пунктирной – единственная поперечная дуга.

Граф, полученный удалением обратных по отношению к остовному дереву дуг, называется каркасом (показан в правой части слайда). Можно показать, что каркас графа при произвольном глубинном остовном дереве не содержит контуров.



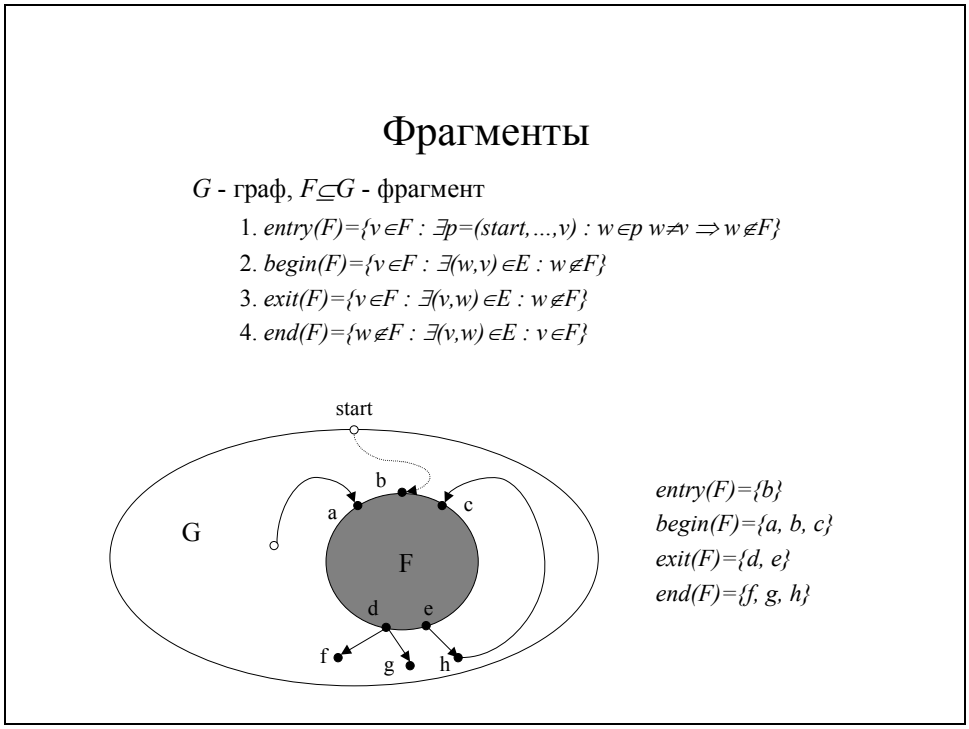
Простейшие свойства

Граф называется сводимым тогда и только тогда, когда множество обратных дуг совпадает с множеством обратных дуг относительно глубинного остовного дерева для любого такого дерева.

Простейшие свойства нумераций *Pre* и *Post*:

- если вершина v обязательно предшествует вершине w, то $Pre(v)<Pre(w)$, $Post(v)<Post(w)$
- прямые в смысле нумерации *Pre* дуги являются прямыми или деревянными относительно дерева
- обратные в смысле нумерации *Pre* дуги являются обратными или поперечными относительно дерева
- обратные в смысле нумерации *Post* дуги являются обратными в смысле дерева; остальные дуги являются прямыми относительно нумерации *Post*
- если $Pre(v)>Pre(w)$, то произвольный путь в графе от v до w содержит общего предка v и w в дереве
- граф сводим тогда и только тогда, когда отношение обязательного предшествования в нем и его каркасе совпадают

Сводимость является одной из основных характеристик графов потока управления. Как видно из определения, в сводимом графе обратные дуги для произвольного остовного дерева совпадают с обратными дугами относительно обязательного предшествования. Учитывая, что каждая обратная дуга определяет в графе потока управления контур (т.е. является абстракцией циклических конструкций языков программирования), циклическая структура сводимого графа является более регулярной, что упрощает анализ его свойств.



Фрагменты

Фрагментом называется произвольный подграф графа управления (под подграфом мы понимаем некоторое непустое подмножество вершин графа, содержащее все ребра графа между ними).

Для фрагмента F определяются четыре множества вершин:

- входные вершины (*entry (F)*) – множество вершин F, до которых существует путь из *start*, не содержащий других вершин F
- начальные вершины (*begin(F)*) – множество вершин F, в которые входит хотя бы одна дуга извне F
- выходные вершины (*exit(F)*) – множество вершин F, из которых исходит хоть одна дуга вовне F
- конечные вершины (*end(F)*) – множество вершин вовне F, в которые входит хоть одна дуга из F

Схематически соотношения между фрагментом и этими его множествами вершин показаны на слайде.

Фрагменты графа потока управления являются абстракциями конструкций управления языков программирования. Известно, что практически все языки программирования определяют сходный набор конструкций управления. Следовательно, перейдя от анализа конкретных конструкций к их обобщениям, можно разработать способы анализа свойств программ независимо от входных языков.

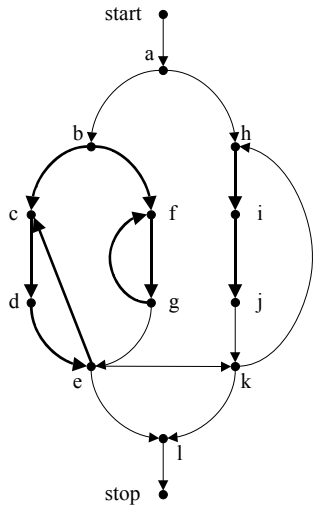
Далее мы рассмотрим некоторые виды фрагментов.

Альт

$F \subseteq G$ - альт $\Leftrightarrow |begin(F)|=1$

Свойства:

1. G - альт
2. $\{v \in G\}$ - альт
3. $begin(F)=entry(F)=\{p\}$ $\forall v \in F \Rightarrow p<v$
4. F_1, F_2 - альты, $F_1 \cap F_2 \neq \varnothing \Rightarrow$
   - либо $begin(F_1) \subseteq F_2$ и $F_1 \cup F_2 = F$ - альт, $begin(F)= begin(F_2)$
   - либо $begin(F_2) \subseteq F_1$ и $F_1 \cup F_2 = F$ - альт, $begin(F)= begin(F_1)$



Альт

Альтом называется фрагмент, имеющий одну начальную вершину. Пример альтов показан на рисунке на слайде (дуги, принадлежащие альтам, выделены жирными линиями).

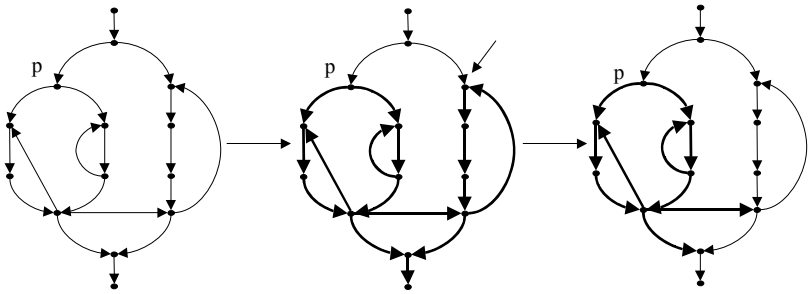
Легко могут быть доказаны следующие простейшие свойства альтов:

- весь граф является альтом
- произвольная вершина графа является альтом
- начальная вершина альта обязательно предшествует любой его вершине
- если пересечение двух альтов не пусто, то один из них содержит начальную вершину другого и их объединение также есть альт.

Выделение максимального альта

```
set<Vertex> Alt (p: Vertex)
{
     for ∀v∈V do v.Color = Black;
     for ∀v∈V : p→*v do v.Color = Gray;
     while ∃v∈V\{p} : (v.Color = Gray) && (∃(w,v)∈E : w.Color = Black)
          do v.Color = Black;

     return {v∈V : Color(v)=Gray}
}
```



Выделение максимального альта

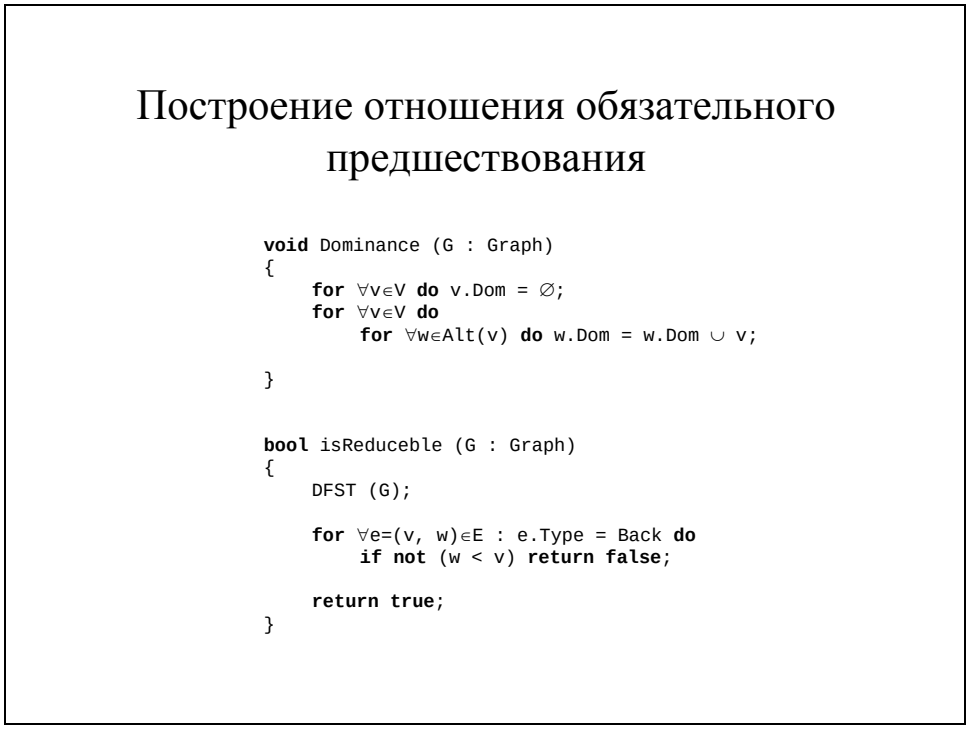
Приведем алгоритм выделения максимального альта, для которого данная вершина *p* является начальной.

Алгоритм состоит из трех шагов:

- вначале все вершины графа помечаются как "черные"
- затем все вершины, достижимые из *p*, помечаются как "серые"
- если среди серых вершин найдется вершина *v*, которая отлична от *p* и имеет хотя бы одно входящее ребро, ведущее из "черной" вершины, то эта вершина красится в "черный" цвет.

Можно показать, что в конце работы алгоритма множество "серых" вершин и будет максимальным альтом, начинающимся в вершине *p*.

Пример работы алгоритма показан на слайде. Крайний левый граф – это граф после первого шага. Средний рисунок изображает граф после того, как все вершины, достижимые из *p*, покрашены в "серый" цвет. Здесь же стрелочкой обозначена вершина, имеющая своим предком "черную" вершину. Окончательный альт показан на крайнем правом рисунке.



Построение отношения обязательного предшествования

Свойства альтов дают возможность использовать их для определения отношения обязательного предшествования. Для этого достаточно для каждой вершины построить максимальный альт, начинающийся в ней, и затем включить эту вершину во множество обязательных предшественников для каждой вершины из этого альта.

После построения отношения обязательного предшествования сводимость графа может быть проверена элементарно – достаточно построить глубинное остовное дерево и проверить, что в любой обратной по отношению к нему дуге конец обязательно предшествует началу.

Луч

$R \subseteq G$ - луч $\Leftrightarrow$

1. R - альт
2. $\forall v \in R \setminus (begin(R) \cup exit(R)) \Rightarrow in(v)=\{(w, v)\}$ & $out(v)=\{(v, u)\}$ & $w, u \in R$

Свойство:

$e=(v, w)$ - прямая, обратная или поперечная дуга относительно глубинного остовного дерева $\Rightarrow e$ не принадлежит никакому лучу

Нумерация # - правильная $\Leftrightarrow$

$\forall R$ - луч, $R=(begin(R) \rightarrow v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_k=end(R))$ $\#(v_i)=\#(begin(R))+i$

Свойство: пусть # - правильная нумерация, R - максимальный луч. Тогда

1. $\{p\}= begin(R) \Leftrightarrow p=start \vee \#^{-1}(\#(p)-1)=exit(R')$
2. $\{q\}=exit(R) \Leftrightarrow p=stop \vee \#^{-1}(\#(p)+1)=begin(R')$

Правильные нумерации: *Pre, Post.*

Луч

Лучом называется фрагмент, который, во-первых, является альтом, а во-вторых, обладает тем свойством, что произвольная его вершина, отличная от начальной и выходной, имеет одного предка и одного потомка, каждый из которых принадлежит лучу. Иными словами, луч – это линейная последовательность вершин.

Легко видеть, что в состав лучей могут входить только деревянные дуги.

Нумерация # называется правильной, если она приписывает вершинам произвольного луча последовательные номера.

Если # – правильная нумерация, а R – максимальный луч, то можно доказать следующие утверждения:

- вершина p является начальной вершиной R тогда и только тогда, когда либо $p=start$ либо $\#^{-1}(\#(p)-1)$ – выходная вершина некоторого максимального луча
- вершина q является выходной вершиной R тогда и только тогда, когда либо $p=stop$ либо $\#^{-1}(\#(p)+1)$ – начальная вершина некоторого максимального луча

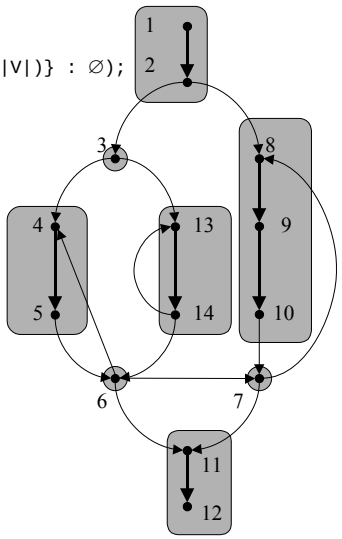
Легко показать, что нумерации *Pre* и *Post* являются правильными.

Выделение лучей

```
void Lines (G : Graph)
{
    Starts = {start} ∪ (|in (Pre⁻¹(|V|))|>1 ? {Pre⁻¹(|V|)} : ∅);
    Ends   = {stop, Pre⁻¹(|V|)};
    for i=2 to |V|-1 do
    {
        v=Pre⁻¹(i);

        if (|in(v)| > 1))
        {
            Starts = Starts ∪ {v};
            Ends   = Ends   ∪ Pre⁻¹(i-1);

        }
        if  (
                |out(v)| > 1 ||
                (|out(v)| = 1 && e.Type = Tree
                where e = (v, w)∈E)
            )
        {
            Starts = Starts ∪ Pre⁻¹(i+1);
            Ends   = Ends   ∪ {v};
        }
    }
}
```



Выделение лучей

Алгоритм выделения максимальных лучей использует свойства правильных нумераций и элементарное наблюдение, что вершина *v* является начальной вершиной максимального луча в том случае, если в нее входит более одного ребра, и выходной – если выходит более одного ребра.

Результатом работы алгоритма является два списка *Starts* и *Ends*, первый из которых содержит начальные вершины максимальных лучей, а второй – выходные.

Очевидно, что одна вершина является лучом. Таким образом, множество максимальных лучей образует разбиение множества вершин графа. На иллюстрации показано это разбиение.

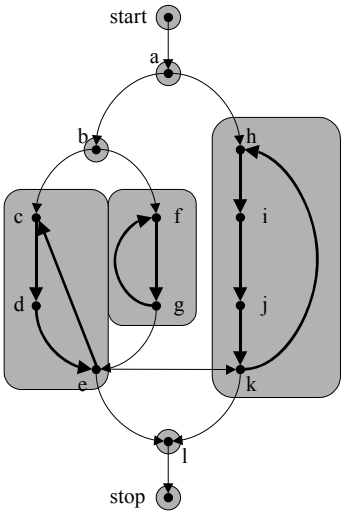
Сильно связный подграф

$F \subseteq G$ - сильно связный подграф $\Leftrightarrow$
$\forall v,w \in F\ (v \rightarrow^* w)\ \&\ (w \rightarrow^* v)$

$B \subseteq G$ - компонента сильной связности $\Leftrightarrow$
B - максимальный сильно связный подграф

Свойства:

1. $\{v \in G\}$ - сильно связный подграф
2. $\forall B_1, B_2\ B_1 \cap B_2 = \varnothing$ (B_i - компоненты сильной связности)
3. $G = \cup B_i$ (B_i - все компоненты сильной связности)
4. $\forall B\ begin(B) = entry(B)$



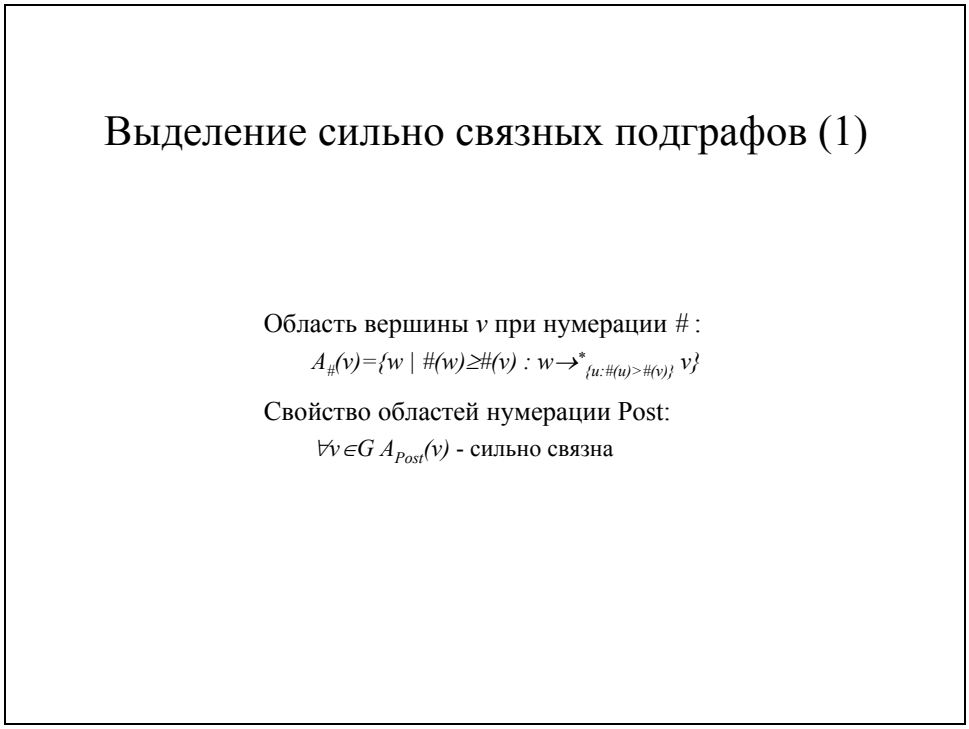
Сильно связный подграф

Сильно связный подграф – это фрагмент, состоящий из взаимно достижимых вершин. Компонента сильной связности – это максимальный сильно связный подграф.

Очевидно, произвольная вершина является сильно связным подграфом. Кроме того, легко показать, что множество компонент сильной связности задает разбиение множества вершин. Это разбиение показано на слайде.

Наконец, множества входных и начальных вершин для компонент сильной связности совпадают.

Как было показано в лекции 11, выделение сильно связных подграфов имеет определяющее значение для реализации чистки циклов.



Выделение сильно связных подграфов (1)

Для выделения сильно связных подграфов введем понятие области при нумерации #, а именно, областью вершины v при нумерации # назовем множество вершин с большими номерами, из которых *v* достижима в подграфе, состоящем из всех вершин с большими номерами.

Можно показать, что область произвольной вершины при нумерации *Post* сильно связна.

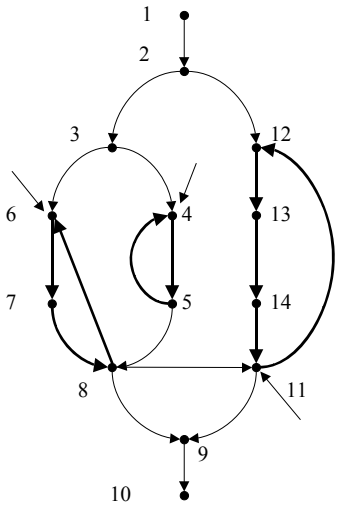
Выделение сильно связных подграфов (2)

```
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;
}
```

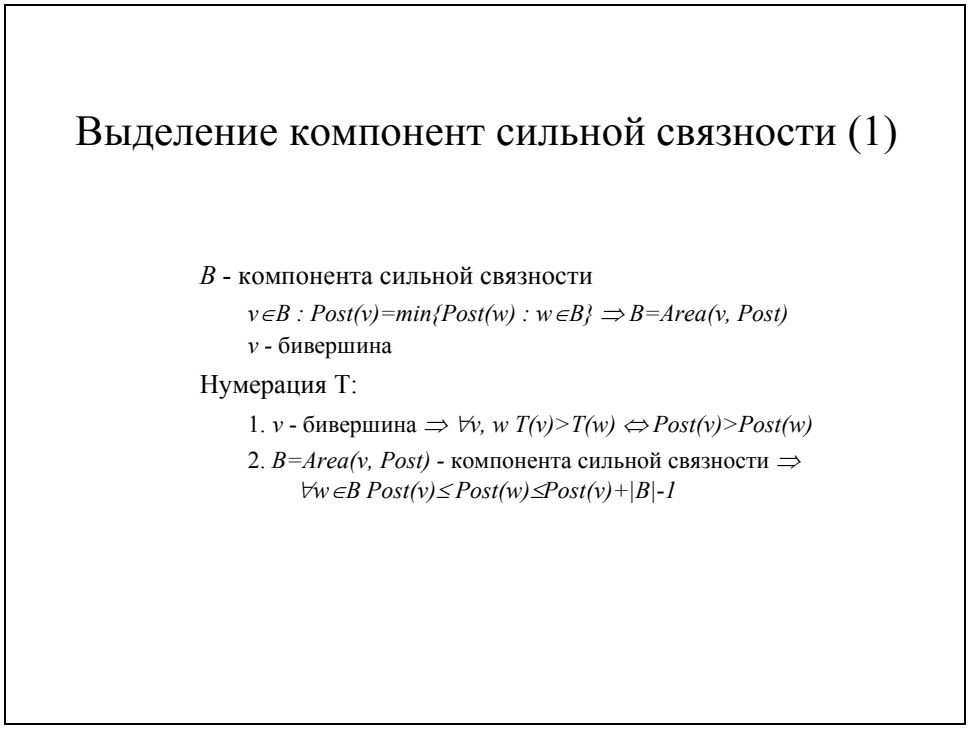


Выделение сильно связных подграфов (2)

Для выделения сильно связных подграфов, таким образом, достаточно научиться выделять области. Заметим, однако, что, несмотря на то, что области при нумерации Post сильно связны, это не означает, что произвольные сильно связные подграфы будут областями при нумерации Post.

Алгоритм построения области приведен на слайде. Видно, что он осуществляет движение от текущей вершины по встречным дугам, проходя только по вершинам, имеющим большие номера, чем данная, и включает их в область.

Примеры подграфов, выделенных с помощью алгоритма, приведены на иллюстрации. Стрелочками отмечены вершины, области которых выделены.



Выделение компонент сильной связности (1)

Можно показать, что компонента сильной связности является областью своей вершины, имеющей минимальный номер в нумерации *Post* среди всех остальных вершин этой компоненты. Такая вершина называется бивершиной.

Для выделения компонент сильной связности построим нумерацию *T*, такую, что для бивершин порядок, задаваемый *T*-номерами, совпадает с порядком, задаваемым *Post*-номерами, а все компоненты сильной связности заполнены *T*-номерами последовательно.

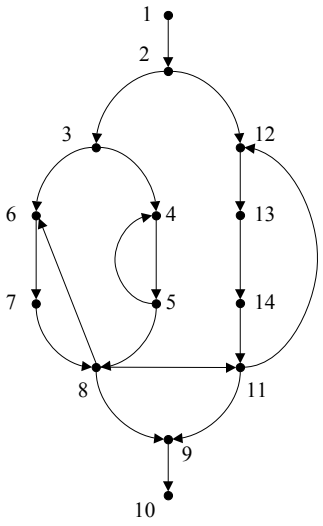
Выделение компонент сильной связности (2)

```
void T (G: Graph)
{
    currT=1;

    for ∀v∈V do v.Visited = false;

    for i=1 to |V| do
    {
        v=Post⁻¹(i);

        if (not v.Visited)
        {
            T(v)=currT++;
            v.Visited = true;
            for ∀w∈Area(v)\{v} do
            {
                T(w)=currT++;
                w.Visited = true;
            }
        }
    }
}
```

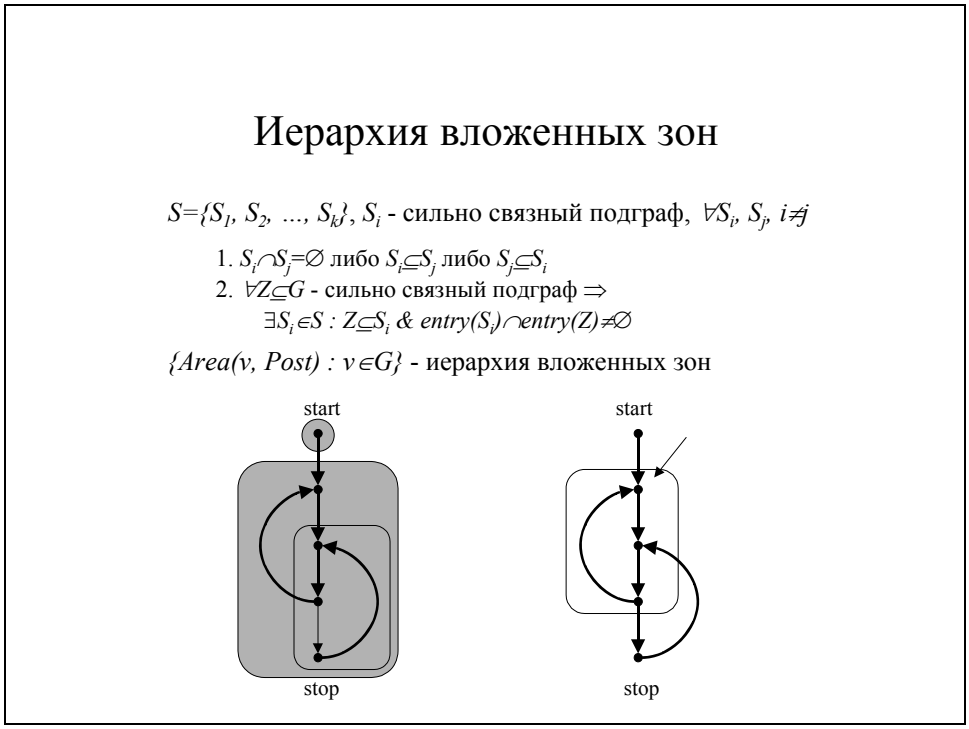


Выделение сильно связных подграфов (2)

Алгоритм построения *T*-нумерации обходит граф в порядке возрастания *Post*-номеров вершин. При этом каждая вершина может находиться в двух состояниях: обработанная или необработанная. Первоначально все вершины находятся в необработанном состоянии.

Обнаружив необработанную вершину, алгоритм присваивает ей очередной номер, выделяет ее область и присваивает вершинам области очередные номера.

На рисунке на слайде показан граф с его *T*-нумерацией.



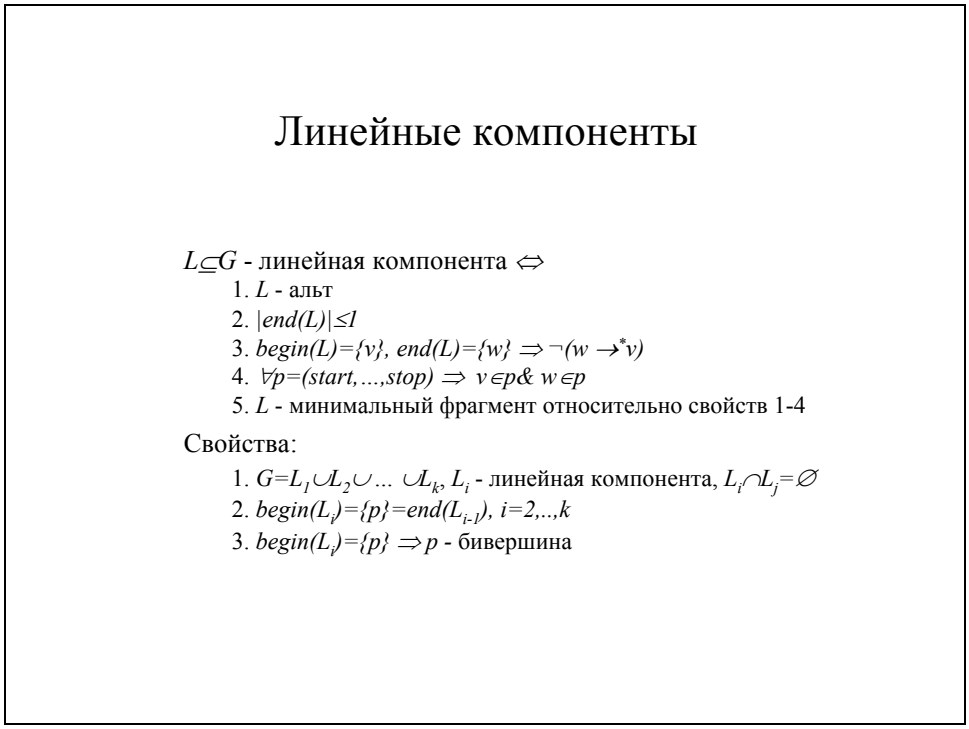
Иерархия вложенных зон

Как было отмечено выше, не любой сильно связный подграф является областью некоторой вершины при нумерации *Post*. Для того, чтобы конструктивно описать все сильно связные подграфы, используется понятие иерархии вложенных зон.

Иерархией вложенных зон называется совокупность сильно связных подграфов S, два любых из которых либо не пересекаются, либо один из них содержит другой, такая, что для произвольного сильно связного подграфа Z найдется содержащий его элемент S, который имеет с Z общую входную вершину.

Может быть показано, что набор областей всех вершин при нумерации *Post* является иерархией вложенных зон.

Иерархия вложенных зон – это один из способов описать циклическую структуру программы. Несмотря на то, что эта иерархия, вообще говоря, не содержит всех циклов программы, она тем не менее дает возможность рассмотреть некоторое приближение множества всех циклов.



Линейные компоненты

Линейной компонентой называется фрагмент L, обладающий следующими пятью свойствами:

1. L является альтом
2. L имеет не более одной конечной вершины
3. Начальная вершина L не достижима из его конечной вершины
4. Начальная и конечная вершины L принадлежат любому пути в графе от *start* до *stop*
5. L – минимальный подграф, обладающий предыдущими свойствами

Множество всех линейных компонент образует разбиение множества вершин графа. Стягивание линейных компонент переводит граф в луч.

Можно также показать, что начальная вершина линейной компоненты есть либо *start*, либо конечная вершина другой линейной компоненты. Наконец, можно показать, что начальная вершина произвольной линейной компоненты является бивершиной.

Выделение линейных компонент в исходном графе позволяет применять к нему оптимизирующие преобразования, рассчитанные на линейный участок.

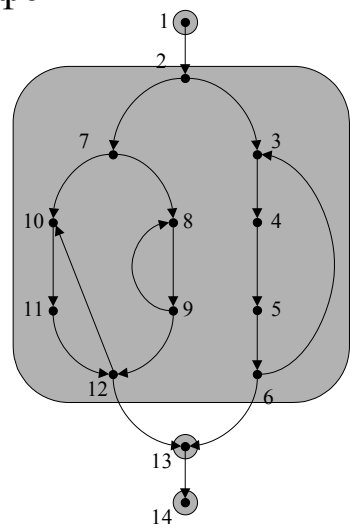
Выделение линейных компонент в сводимом графе

```
void LinearComponents (G: Graph)
{
    LCs = ∅;
    L   = ∅;
    max = 1;

    for i=1 to |V| do
    {
        v = T-1(i);
        L = L ∪ {v};

        for ∀(v, w)∈E do
            if (max < T(w)) max = T(w);

        if (i+1 = max && bivertex (T-1(i+1)))
        {
            LCs = LCs ∪ L;
            L   = ∅;
        }
    }
}
```



Выделение линейных компонент в сводимом графе

Идея алгоритм выделения линейных компонент в сводимом графе опирается на использование свойств, сформулированных выше.

Алгоритм обходит граф в порядке возрастания *T*-номеров и добавляет вершины в текущую линейную компоненту. Признаком завершения линейной компоненты является то, что вершина со следующим номером является, во-первых, бивершиной, а во-вторых, ее номер – максимальный среди номеров всех потомков вершин текущей линейной компоненты.

Разбиение графа на линейные компоненты изображено на рисунке на слайде.

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.

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.