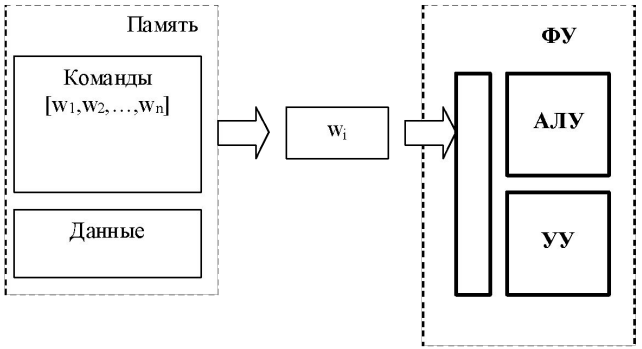
Теория компиляторов

Часть II

Лекция 3. Общие методы распараллеливания кода

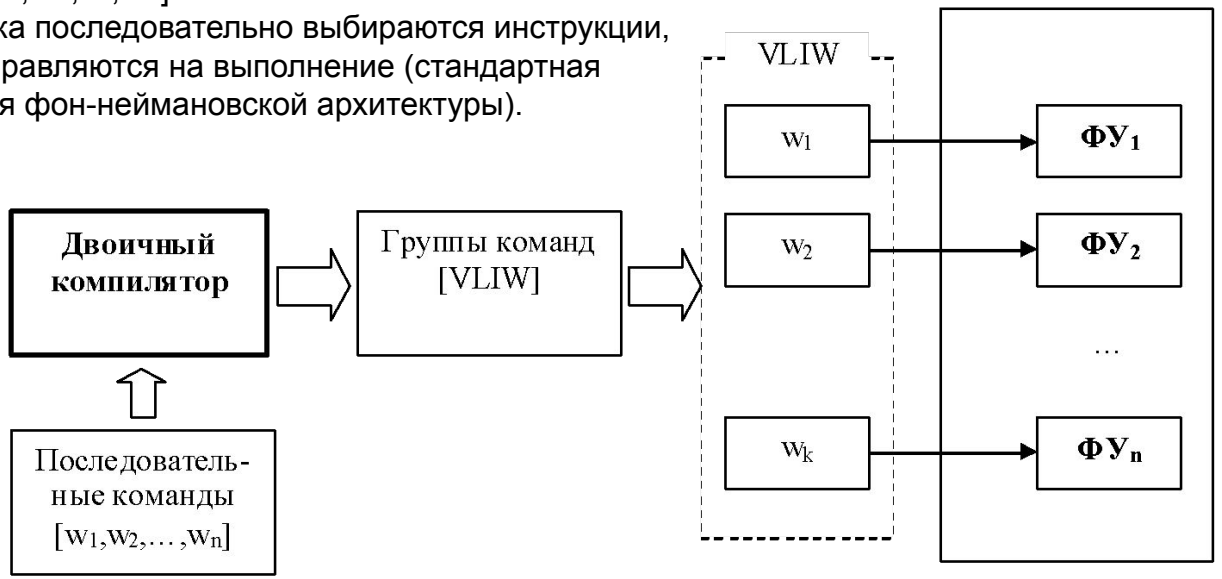
Общая схема распараллеливания программы



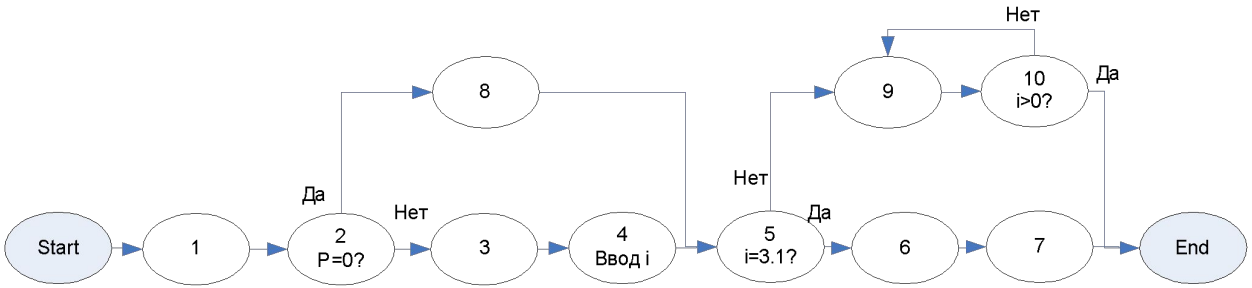
- Необходимо преобразовать линейный список инструкций $[w_1,w_2,\ldots,w_n]$ в список широких командных слов $[VLIW_1,VLIW_2,\ldots,VLIW_m]$.

$$[w_1, w_2, ..., w_n] \Rightarrow \begin{bmatrix} VLIV_1 \\ ... \\ VLIV_m \end{bmatrix} = \begin{bmatrix} [w_1^1, w_2^1, ..., w_k^1] \\ ... \\ [w_1^m, w_2^m, ..., w_l^m] \end{bmatrix}$$

Последовательность тетрад образует линейный список инструкций [w1,w2,…,wn].
Из этого списка последовательно выбираются инструкции, которые и отправляются на выполнение (стандартная процедура для фон-неймановской архитектуры).



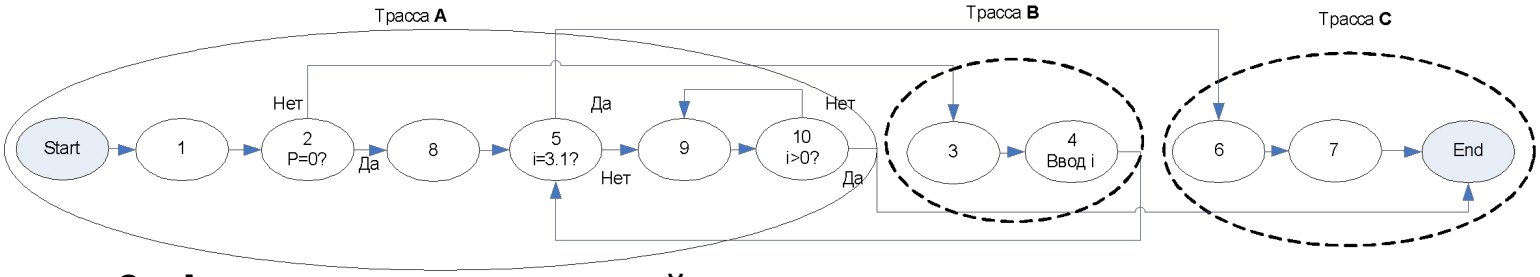
1. Управляющий граф программы



- Вершины–источники - 2, 5 и 10.
- Стоки-вершины 5 и 9.

2. Выделение трасс

Трассы – фрагменты программы (управляющего графа), которые будут выполняться с наибольшей вероятностью.



3. Формирование линейных участков

Линейный участок – это некая последовательность инструкций – блок, имеющий один вход и не более чем два выхода.

Этапы

1. Формирование модели макроуровня. Объект – исходный поток инструкций.

1.1. Расстановка меток.

1.2. Построение управляющего графа.

1.3. Планирование трасс.

1.4. Преобразование трасс.

1.5. Формирование линейных участков.

2. Формирование модели микроуровня. Объект – линейные участки.

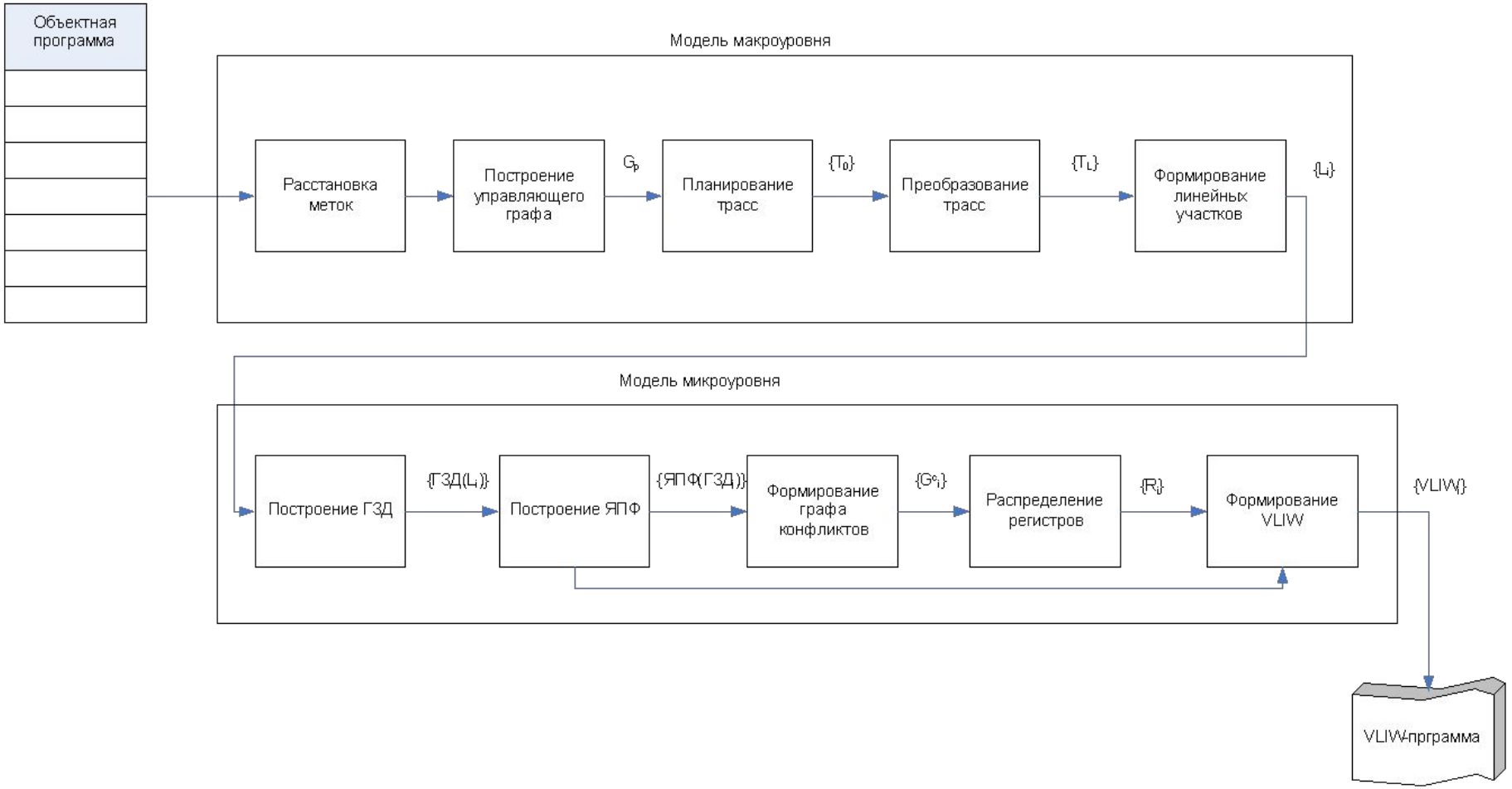
2.1. Построение графа зависимости по данным (ГЗД).

2.2. Преобразование ГЗД к ярусно-параллельной форме.

2.3. Построение графа конфликтов

2.3. Распределение регистров.

Структура ПК



Расстановка меток

- Для каждой тетрады определяются ее атрибуты, относящие тетраду к типу «развилка», «сток», «плохая инструкция».
  - «Плохая инструкция» (операции ввода-вывода, вызовы подпрограмм, операции синхронизации и т.п.)
  - Если тетрада - операция условного перехода, то это – «развилка».
  - Если адрес (номер) тетрады используется где-либо в качестве адреса перехода, то это – «сток».
- Тетрада может иметь несколько подобных атрибутов (быть и «развилкой», и «стоком»).

Построение управляющего графа

- **УГ** содержит описание линейных блоков программы
- **УГ** - орграф, вершины которого - линейные участки программы, а дуги указывают пути передачи управления.
- **УГ** имеет единственную входную и единственную выходную вершину. Каждая вершина имеет не более двух потомков.

Вход: поток тетрад.
Выход: управляющий граф в виде описания множества линейных блоков

```
Инициализация;
блок_готов := False;
Цикл Пока (поток тетрад не пуст)
Считать тетраду;
Если (тетрада с меткой) То
        Фиксация окончания очередного блока;
        блок_готов := True;
Кесли
Если (код операции = переход) То
        Фиксация окончания очередного блока;
        Фиксация метки перехода;
        блок_готов := True;
        окончание_блока_по_переходу = True;
Кесли
Если (блок_готов = True) То Добавить к графу вершину; Кесли
Если (окончание_блока_по_переходу = True) То
        Добавить к графу дугу соответствующего перехода;
Кесли
КонецЦикла
```

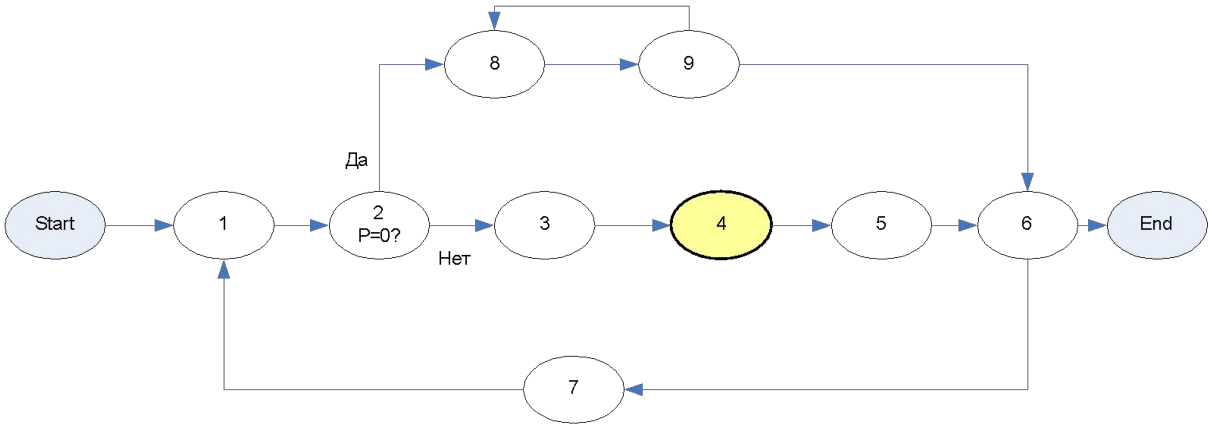
Планирование трасс

- «Хорошие» линейные участки - длинные
- «Плохие» линейные участки: короткие или содержащие «плохие» операции.
- "Плохие" инструкции:
  - вызовы внешних подпрограмм;
  - возвраты из подпрограмм;
  - операции ввода-вывода;
  - операции синхронизации по времени;
  - переходы по вычисляемым адресам (т.к. адрес перехода неизвестен заранее, то оптимизировать нельзя);
  - операции с данными, находящимися по вычисляемым адресам (невозможно проследить зависимости по данным на этапе трансляции).

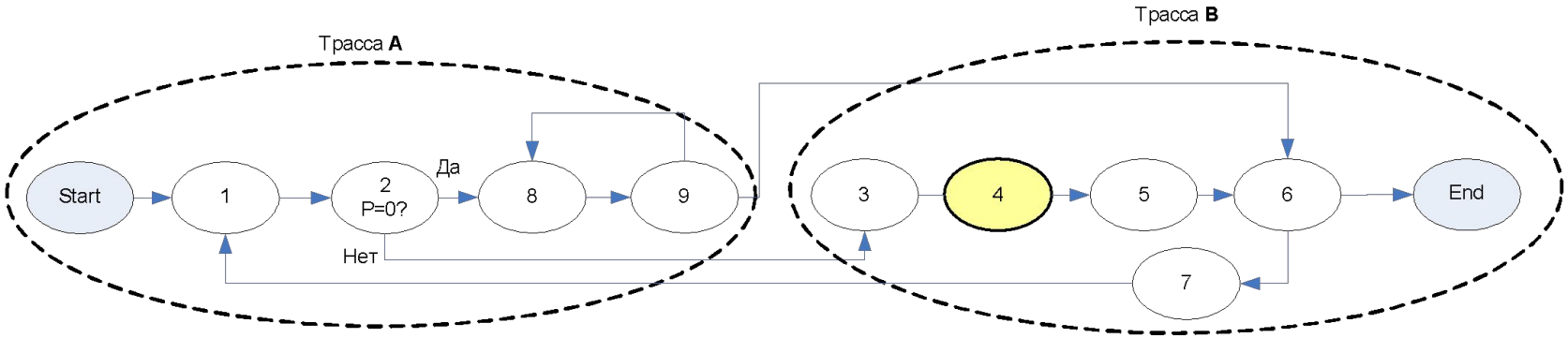
Эвристики

- ***Предсказание на основе истории ветвлений.*** Сбор статистики об исходах операций ветвления и построение на ее основе строится предположение о результате выполнения текущей операции.
- ***Предсказание на основе пробных прогонов программы.*** Для этого производится имитация выполнения программы на одном или нескольких наборах данных. Собирается статистика. Недостаток очевиден: метод работает лишь при определенных обстоятельствах и исходных данных.
- ***Использование эвристик*** выбора доминирующей ветви.
  - Избегание «плохих» инструкций.
  - Условия с указателями. Обычно справедливы условия
    - Ptr≠NULL
    - Ptr1≠Ptr2
    - if((ptr=malloc(…))!=NULL) …
    - for(p=p0;p!=NULL;p=p–>next) …
  - Эвристика исполнения циклов. Если при ветвлении одна из ветвей содержит цикл, то обычно именно она и будет доминировать. По статистике исполнение циклов занимает до 90% времени выполнения программы в целом.
  - Эвристика направления ветвления. Возврат назад более вероятен (высока вероятность неявного цикла с постусловием).
  - Предсказание по коду операции:
    - при сравнении чисел с плавающей точкой более вероятно неравенство;
    - отрицательные числа менее вероятны.

Пример



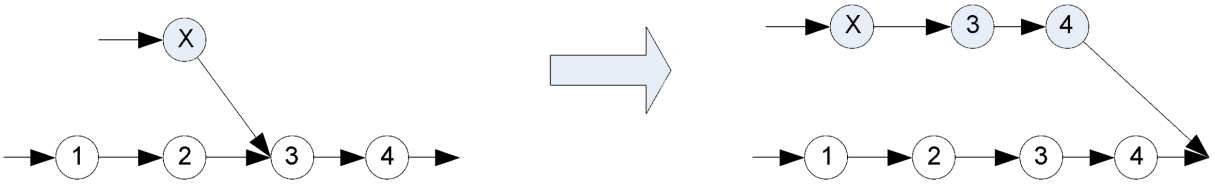
- В первую очередь, избегаем плохих инструкций (4). Если бы (4) была хорошей командой, то мы от (2) перешли на (3), т.к. чаще всего при выполнении сравнения числа не равны друг другу. После (6) идем на (7), т.к. вероятнее движение по циклу.



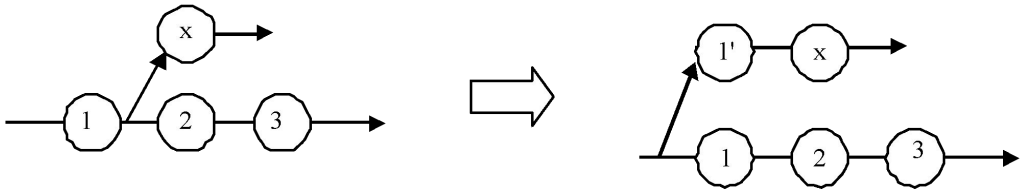
Преобразование трасс

Метод дублирования остатка

- Вход в трассу



- Выход из трассы



Выход из трассы реализуем крайне нетривиально (спекулятивного исполнение, использование предикатных файлов и проч.).

Объем инструкций при дублировании возрастает, однако параллелизм увеличивается.

Линейные участки

- ЛУ - последовательность инструкций, у которой имеется вход и два выхода.
- ЛУ заканчивается тогда, когда осуществляется переход или ставится метка.
- ЛУ – это основной объект оптимизации.
- Чем длиннее ЛУ, тем больше возможностей для параллельных вычислений.
- ЛУ → ГЗД → ЯПФ → распределение регистров → {VLIW}.

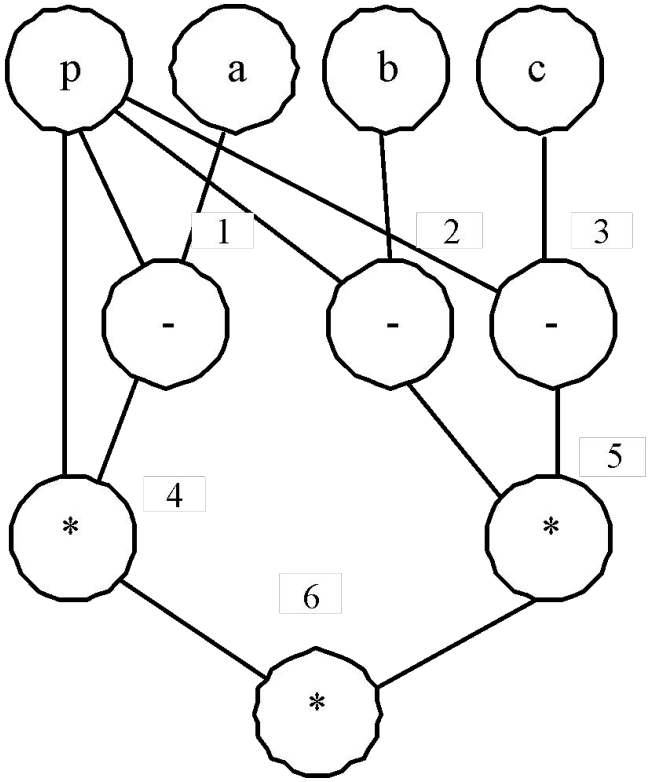
Граф зависимости по данным

- Пусть имеется участок программы – список инструкций $A=(a_1,a_2,\ldots,a_n)$
- Каждая инструкция a_i представлена в тетрадной форме $a_i=(OP_i, I_i^{(1)}, I_i^{(2)}, R_i)$
- ГЗД участка A - граф (A,V) с вершинами $a_i \in A$ и дугами $(a_i, a_j) \in V$
  $V=\{(a_i, a_j): i<j, (R_i=I_j) \vee (R_j=I_i) \vee (R_i=R_j)=true\}$

Пример ГЗД

S = p*(p–a)*((p–b)*(p–c))

1. (-, p, a, T1)
2. (-, p, b, T2)
3. (-, p, c, T3)
4. (*, T2, T3, T4)
5. (*, p, T1, T5)
6. (*, T5, T4, T6)



- Неясно, какие операции могут выполняться одновременно.
- Имеются вершины, которые одновременно участвуют в нескольких операциях (***p***, ***a***, ***b*** и ***c***).
- Удобнее работать с графом в виде дерева (один родитель = вершина участвует лишь в одной операции.)

Ярусно-параллельная форма

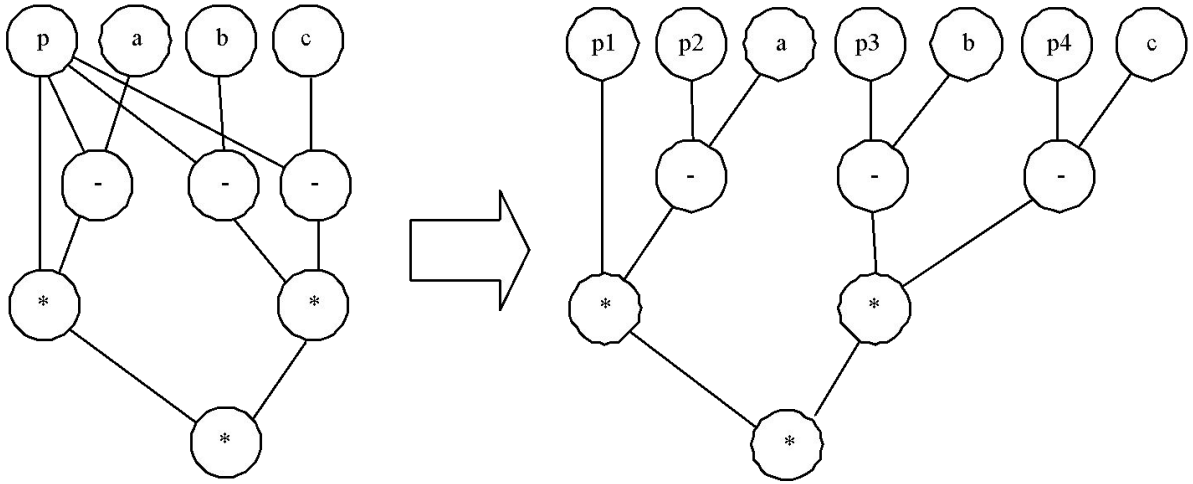
1. Построение дерева

Для преобразования ГЗД к дереву (лесу бинарных деревьев) используется *дублирование переменных*. Дублирование производится для вершин

$\deg^-(a_i \in A)>1$

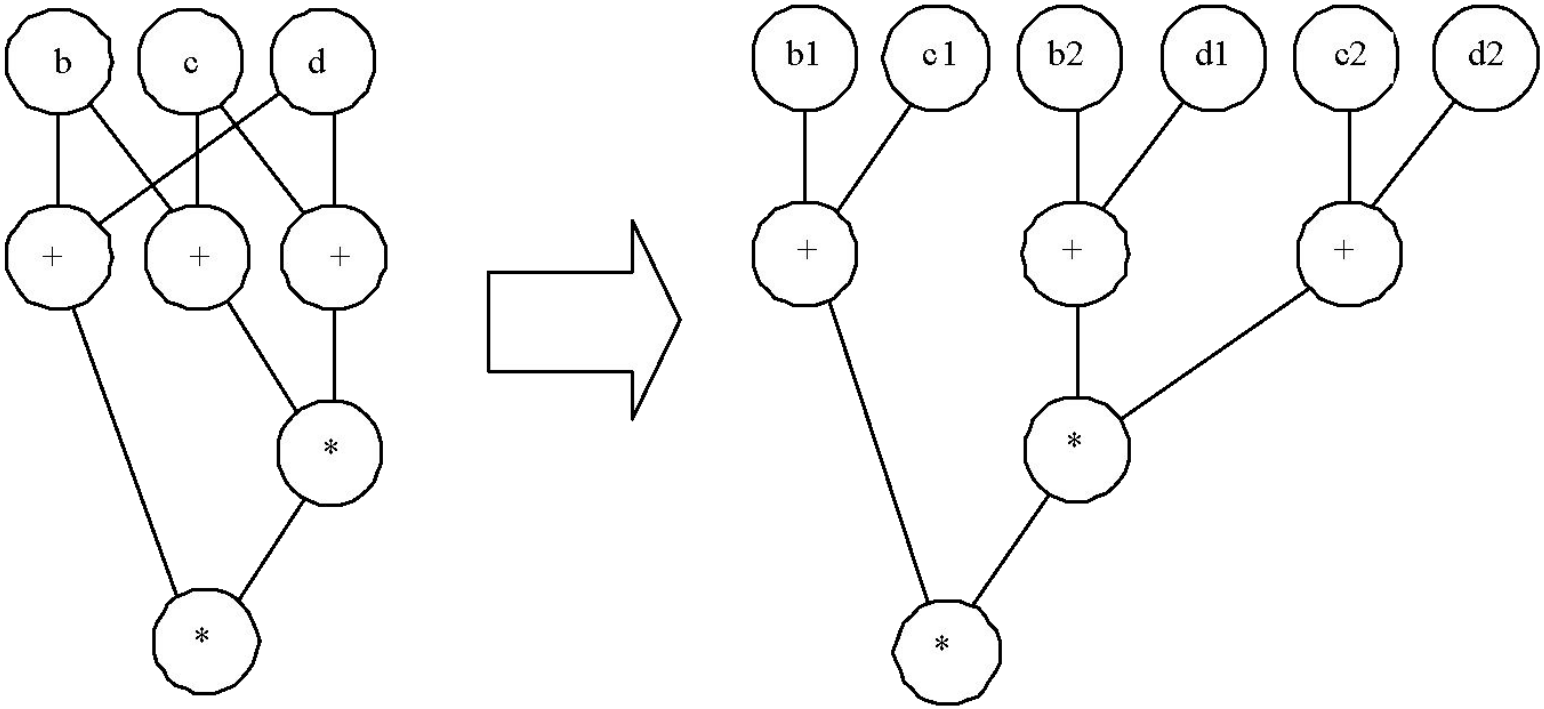
- В результате вершина дублируется $\deg^-(a_i)–1$ раз

Пример. *S = p*(p–a)*((p–b)*(p–c))*



Ярусно-параллельная форма

Пример. a=(b+c)*(c+d)*(b+d)



Построение ЯПФ

ЯПФ – эта форма разметки дерева. В ЯПФ каждой вершине графа приписывается некое число – ранг (номер яруса). Вершины, имеющие один ранг (находящиеся на одном ярусе) могут исполняться одновременно.

Пусть ГЗД оперирует двумя видами тетрад (вершин) – полными тетрадами вида

$$T4=(OP, A_1, A_2, R)$$

и неполными (вырожденными) тетрадами вида

$$T3=(OP, A, ,R)$$

(+,A,B,C) -- *C:=A+B*
(*,A,B,C) -- *C:=A*B*
(:=,A, , C) -- *C:=A*

Представление графа ЯПФ

- Граф ЯПФ – множество вершин-структур

Name	OP
left	right
rank	id

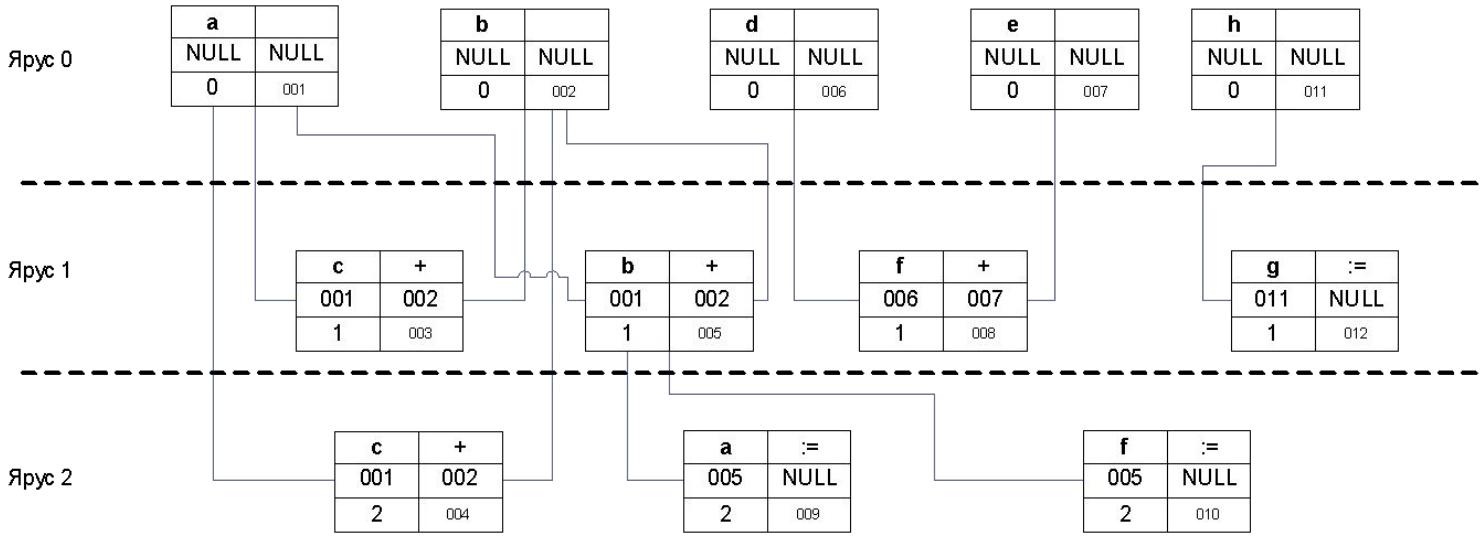
Алгоритм построения ЯПФ

Вход: поток тетрад {T}
Выход: граф G в ярусно-параллельной форме
Очистить список вершин графа G
Цикл по всем тетрадам T
Выбрать очередную тетраду T.
Если тип тетрады T соответствует T4 (T=(OP, A1, A2, R)), то
-- *Анализируем аргумент A1.*
Если ГЗД нет элемента с именем A1, то -- *добавляем новый элемент g1 в G*
g1.name := A1;
g1.rank:= 0;
g1.left := NULL;
g1.right := NULL.
добавить элемент g1 в G
иначе запомнить элемент g1 (g1.name=A1)
-- *Анализируем аргумент A2.*
Если ГЗД нет элемента с именем A2, то -- *добавляем новый элемент g2 в G*
g2.name := A2;
g2.rank:= 0;
g2.left := NULL;
g2.right := NULL.
добавить элемент g2 в G
иначе запомнить элемент g2 (g2.name=A2)
-- *Анализируем аргумент R.*
Найти элемент g3 с максимальным рангом, использующий R в качестве аргумента A1 или A2.
Найти элемент g4 максимального ранга с именем R.
Выбираем максимальный из рангов среди найденных элементов g_i: rmax = max(g1.rank, g2.rank, g3.rank, g4.rank)
(при этом если какой-либо из элементов g_i не был найден в G, то считаем его ранг равным нулю)
Помещаем элемент R на ярус со значением rmax+1.
КонецЕсли
Если тип тетрады T соответствует T3=(OP, A,, R), то
-- Далее все аналогично, только вместо двух анализируется один аргумент – операнд A.
КонецЕсли
КонецЦикла

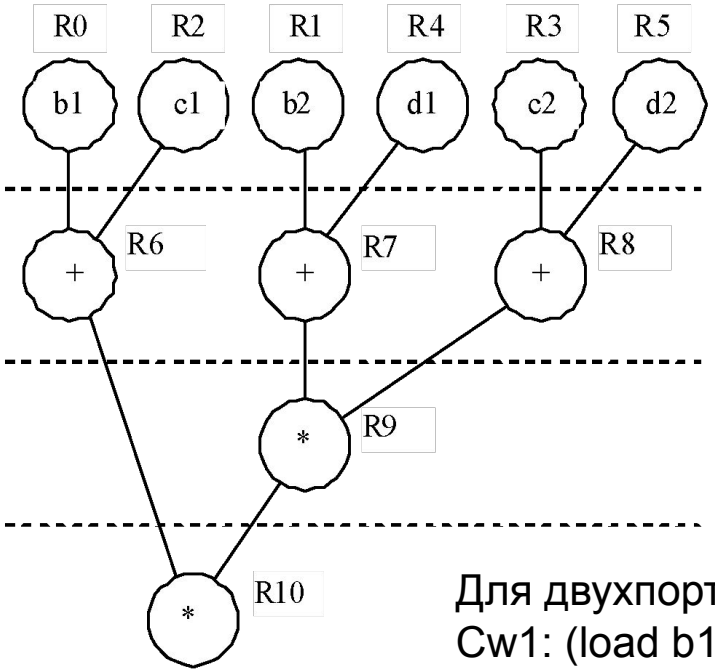
Name	OP
left	right
rank	id

Пример

1. (+,a,b,c) -- *c:=a+b*
2. (+,a,b,c) -- *c:=a+b*
3. (+,a,b,b) -- *b:=a+b*
4. (+,d,e,f) -- *f:=d+e*
5. (:=,b,,a) -- *a:=b*
6. (:=,b,,f) -- *f:=b*
7. (:=,h,,g) -- *g:=h*



Распределение регистров



11 регистров

Для двухпортового регистрового файла из ЯПФ формируем:
Cw1: (load b1, R0), (load b2, R1)
Cw2: (load c1, R2), (load c2, R3)
Cw3: (load d1, R4), (load d2, R5)
Cw4: (R6=R0+R2), (R7=R1+R4), (R8=R3+R5);
Cw5: (R9=R7*R8)
Cw6: (R10=R6*R9)
Cw7: (store R10, a)

Оптимальная загрузка регистров

- Регистров обычно не хватает
- Сведем задачу распределения регистров к задаче раскраски графа:
  - Создать граф, вершинами которого являются данные, а дуги определяют пересечение времен жизни (одновременность использования данных).
  - Раскрасить граф – приписать каждой вершине графа свой цвет – используемый регистр.
  - Количество цветов (красок) – это и есть количество регистров.

Прежде следует определиться с тем, какие команды (вершины) вообще могут **конфликтовать** друг с другом из-за регистров.

Граф конфликтов

- Граф конфликтов – это неориентированный граф, вершинами которого являются используемые переменные (данные), а ребра соединяют вершины с пересекающимися временами жизни. Строить граф конфликтов мы будем, опираясь на ЯПФ.
- Одновременно "живут" (сосуществуют) те вершины, которые находятся на одном ярусе. Кроме того:
  - сохраняются связи, полученные в ЯПФ
  - учитываются связи с вершинами предыдущего ярусами.

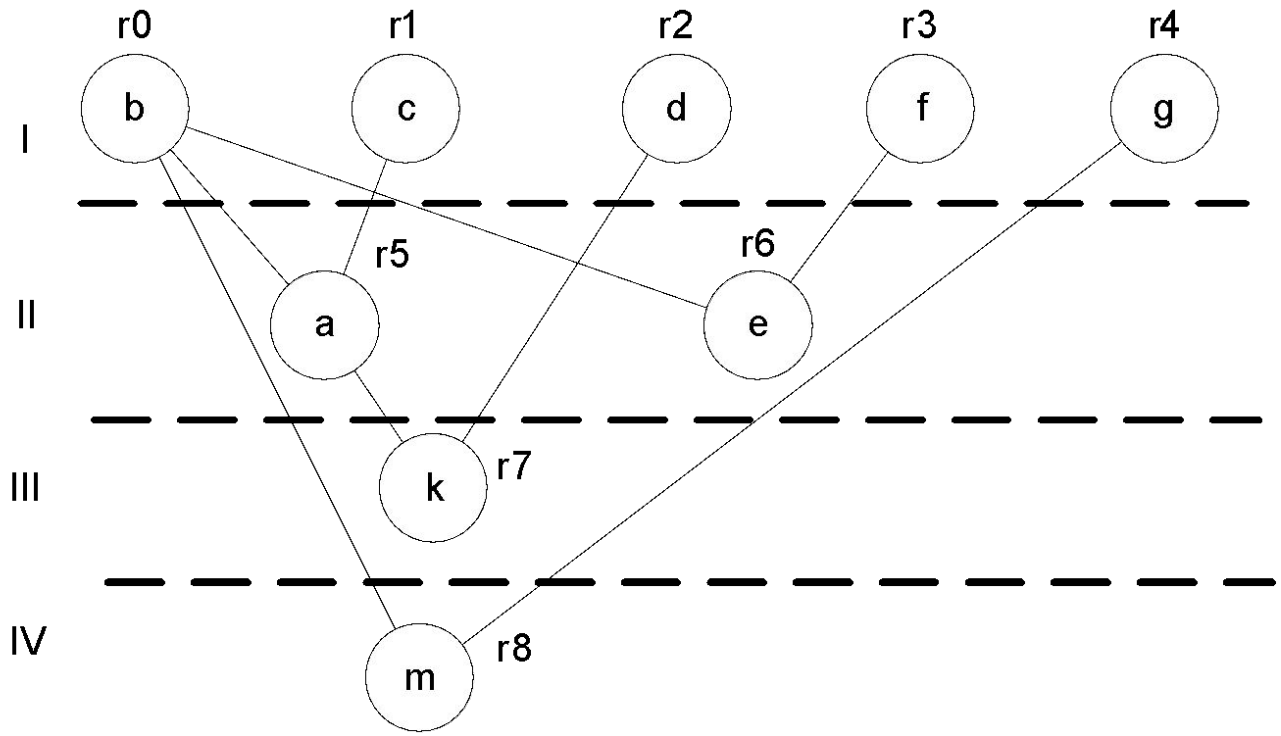
- *Определяется связь со всеми вершинами, которые находятся выше и имеют потомков ниже или на том же уровне, что и текущая:*

  Для вершины a_k строится связь (a_k, a_i) с вершиной a_i такой, что:

  $$L(a_i)<L(a_k) \text{ и } \exists\ a_j: L(a_j)\geq L(a_k)\}$$

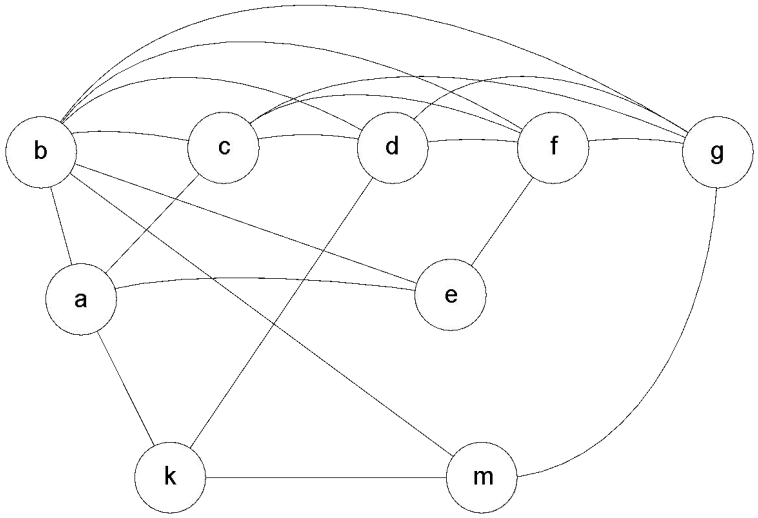
Пример

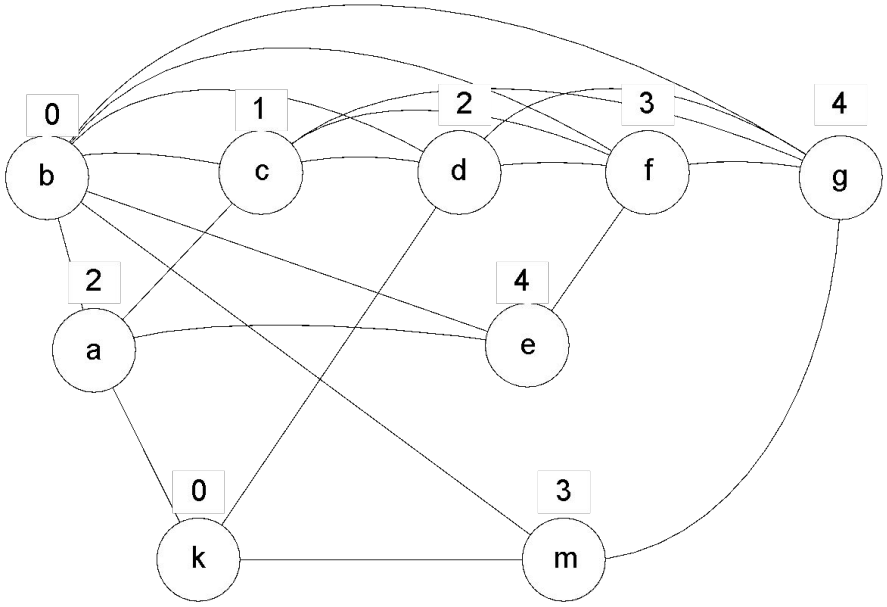
1. a:=b+c;
2. k:=a*d
3. e:=b+f
4. m:=b*g



Без раскраски – 9 регистров R0-R8

Построение графа конфликтов





Далее - раскраска графа

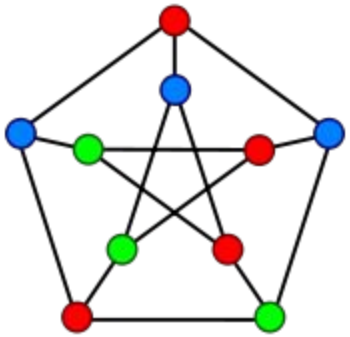
- Использованы краски с именами 0-4. Итого - 5 цветов (регистров)

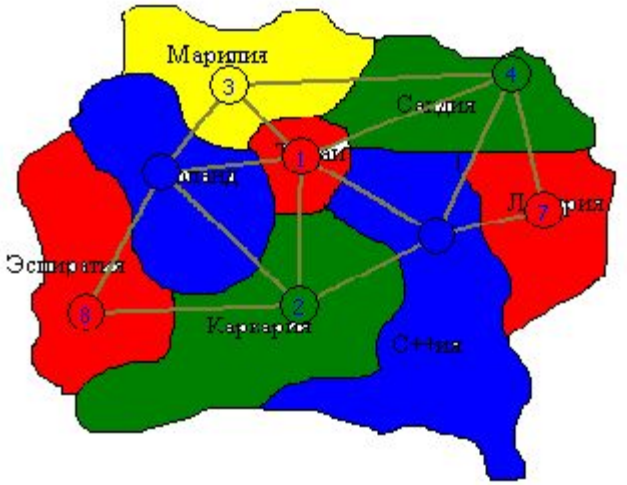
Раскраска графа (1)

Гипотеза о четырех красках:

- Хроматическое число любого **планарного** графа не превосходит 4

Но: наш граф не обязан быть планарным





Раскраска графа (2)

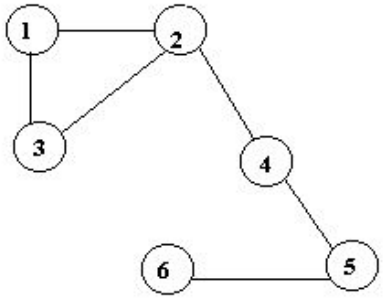
Нахождение оптимальной раскраски – это NP–полная задача. Поэтому чаще всего реализуют алгоритмы поиска субоптимального решения.

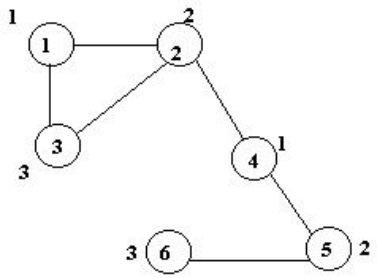
Последовательная раскраска

- Пусть дано упорядоченное множество вершин графа $v_1,\ldots,v_n$.
- вершине v_1 приписываем цвет c_1;
- если подграф $H(v_1,\ldots,v_{i-1})$, порожденный вершинами $v_1,\ldots,v_{i-1}$ k'–раскрашен, $k \le i-1$, то вершина v_i получает цвет c_m, где $m \le k+1$, т.е. цвет с наименьшим номером, не встречающимся на смежных с v_i вершинах.

Число цветов k при этом заранее не фиксируется. Этот алгоритм дает точную k–раскраску только для полных k–дольных графов.

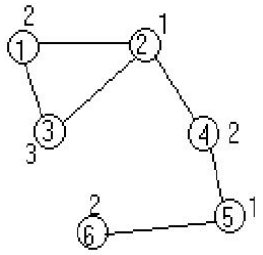
- k–дольным называется граф, множество вершин которого можно разбить на k непересекающихся подмножеств $X_1,\ldots,X_k$ так, что никакие 2 вершины из подмножества X_i, i=1,..,n, не смежны.
- k–дольный граф называется *полным k–дольным*, если каждая вершина из множества X_i смежна с каждой вершиной из X_j, $i \ne j$.





Стратегии последовательных раскрасок

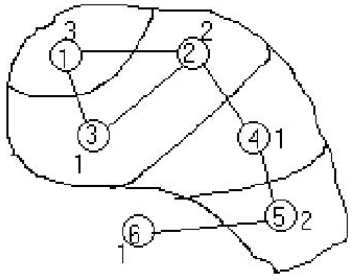
1. НП–стратегия («Наибольшие–Первыми»).
Упорядочить вершины $v_1,\ldots,v_n$ по убыванию их степеней связности, т.е. сначала раскрашиваются вершины с максимальными степенями.



- В данном случае упорядочивание может выглядеть так: {2,1,3,4,5,6}. Поэтому раскраску начнем с вершины V_1=2.

2. ПН–стратегия («Последними–Наименьшие»)

- для n=|V| в качестве v_n выбирается вершина минимальной степени в G;
- для i=n–1,n–2,…,2,1 в качестве v_i выбирается вершина минимальной степени в подграфе $H(V\backslash\{v_n,\ldots,v_{i+1}\})$.



Выберем вершину минимальной связности: V_6=6. Далее рассматриваем граф, где нет 6-й вершины. В этом графе V_5=5. Далее в оставшемся графе определим V_4=4, затем V_3=1, V_2=2 и V_1=3. Итого: {3,2,1,4,5,6}

Итоговая последовательность

1. Формирование модели макроуровня. Объект – исходный поток инструкций.

1.1. Расстановка меток.

1.2. Построение управляющего графа.

1.3. *Планирование трасс. Эвристики.*

1.4. Преобразование трасс.

1.5. Формирование линейных участков.

2. Формирование модели микроуровня. Объект – линейные участки.

2.1. Построение графа зависимости по данным (ГЗД).

2.2. Преобразование ГЗД к ярусно-параллельной форме.

2.3. Построение графа конфликтов

2.3. Распределение регистров.