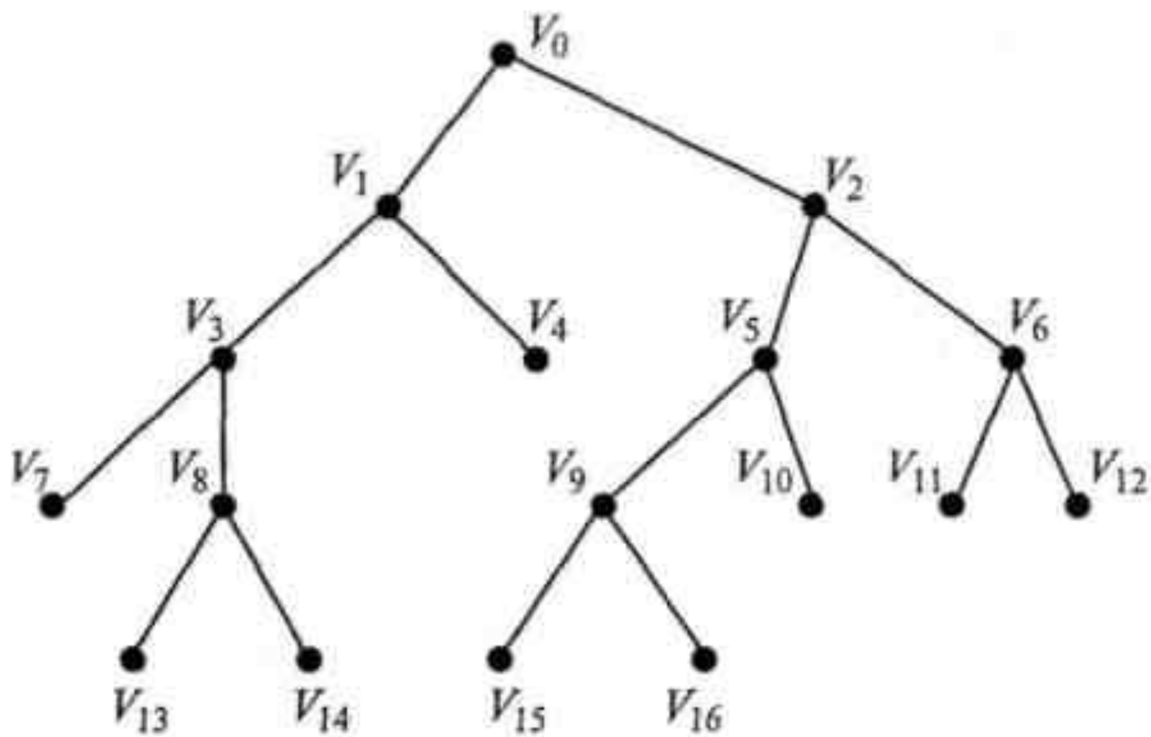
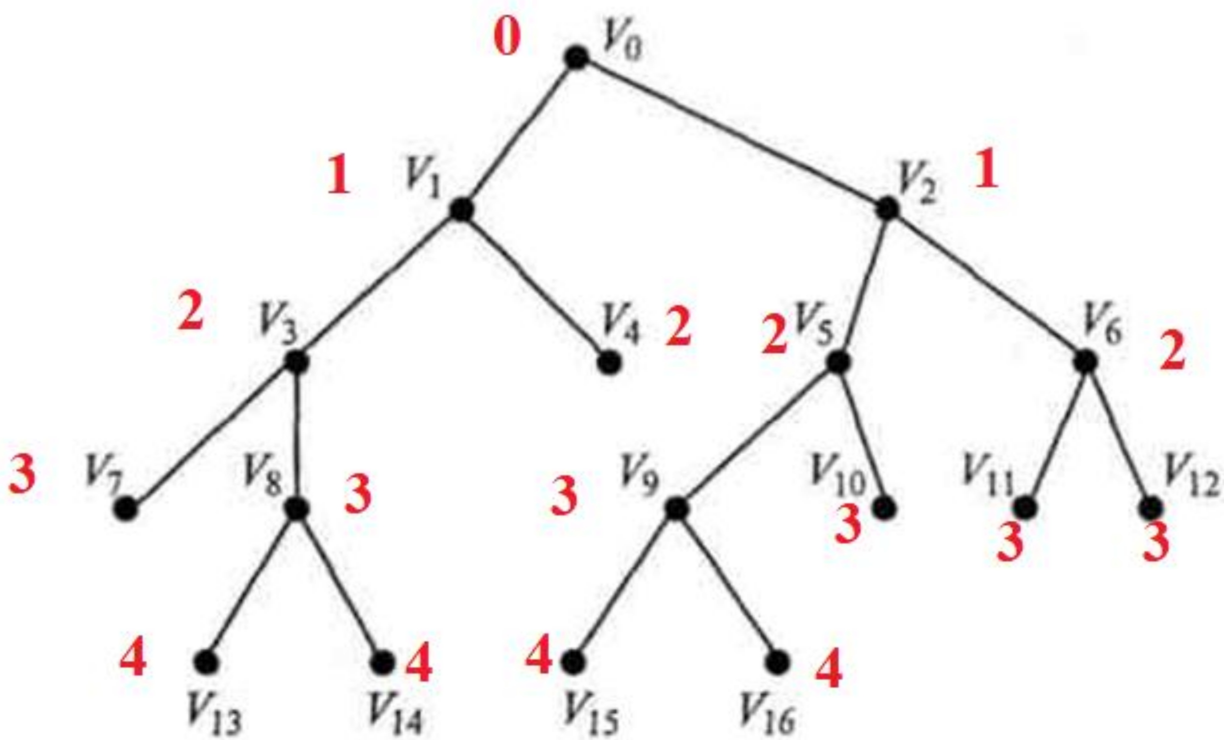
LCA

Определение

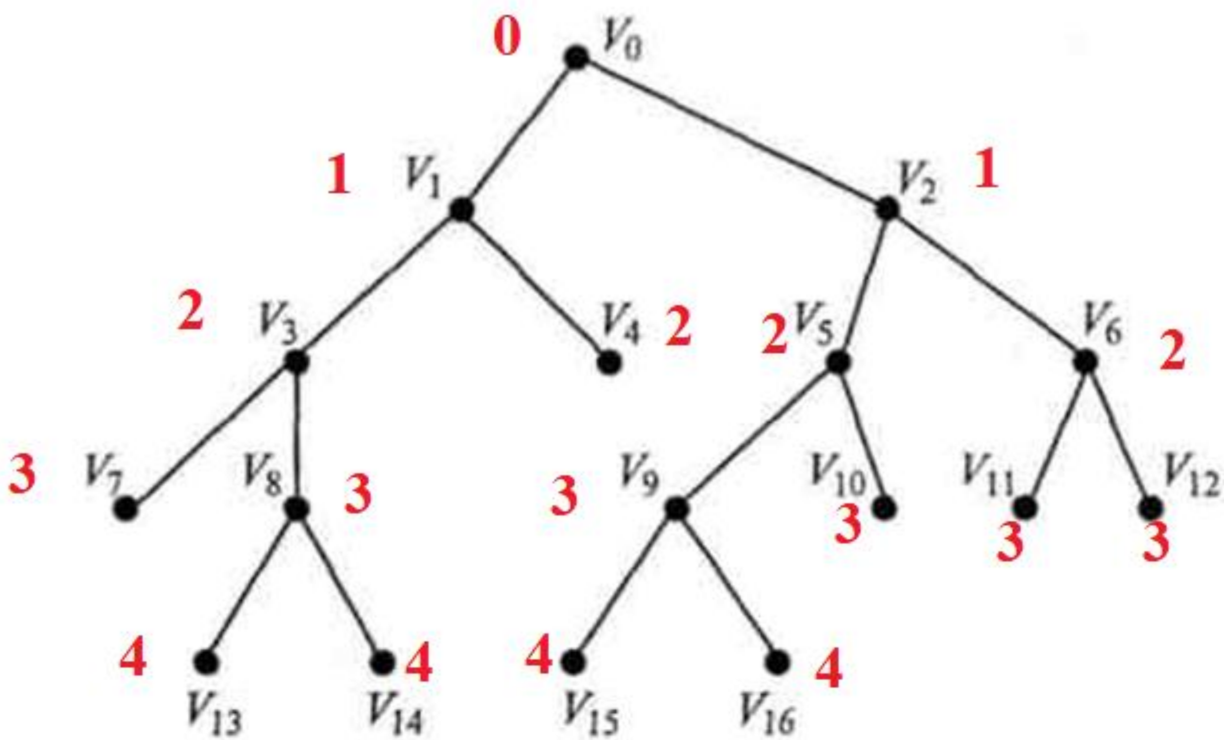
Пусть дано дерево G. На вход поступают запросы вида (V1, V2), для каждого запроса требуется найти их наименьшего общего предка, т.е. вершину V, которая лежит на пути от корня до V1, на пути от корня до V2, и из всех таких вершин следует выбирать самую нижнюю. Иными словами, искомая вершина V – предок и V1, и V2, и среди всех таких общих предков выбирается нижний. Очевидно, что наименьший общий предок вершин V1 и V2 – это их общий предок, лежащий на кратчайшем пути из V1 в V2. В частности, например, если V1 является предком V2, то V1 является их наименьшим общим предком.



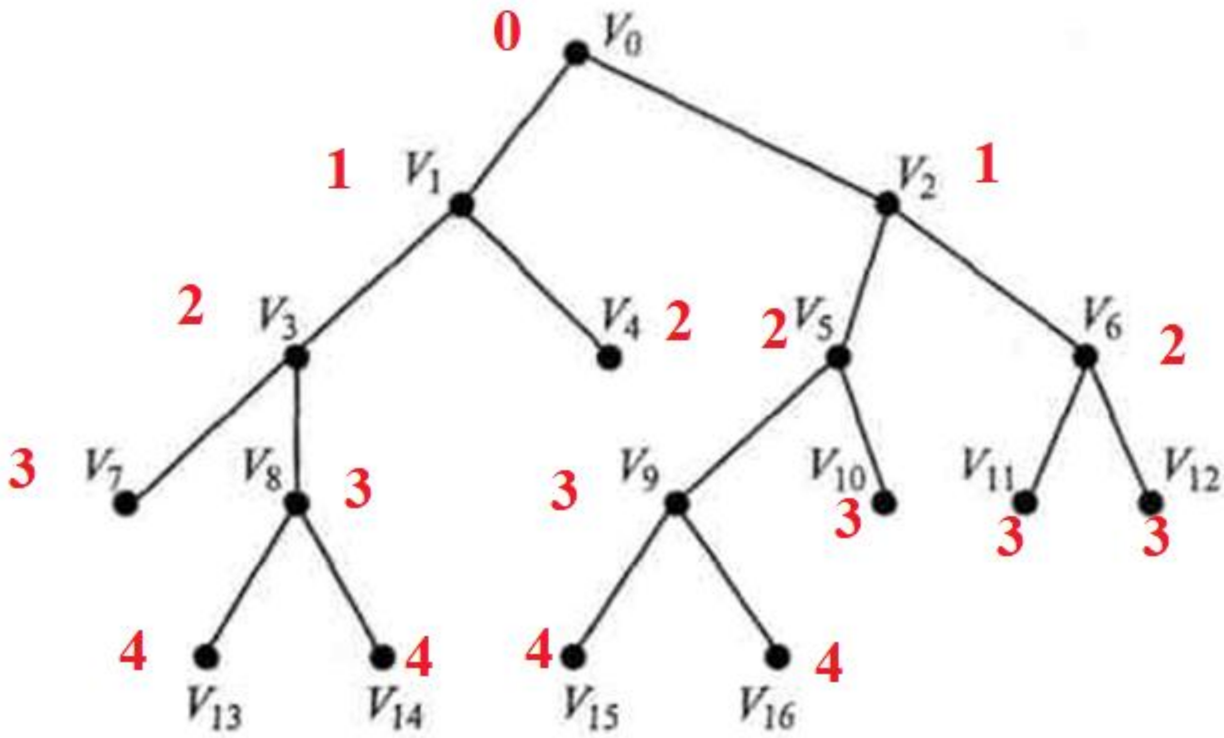
V_0
V_1
V_2
V_3
V_4
V_5
V_6
V_7
V_8
V_9
V_{10}
V_{11}
V_{12}
V_{13}
V_{14}
V_{15}
V_{16}



0
V_0
1
V_1
V_2
1
2
V_3
V_4
2
2
V_5
V_6
2
3
V_7
V_8
3
3
V_9
V_{10}
3
V_{11}
3
V_{12}
3
4
V_{13}
V_{14}
4
4
V_{15}
V_{16}
4


0
V_0
1
V_1
V_2
1
2
V_3
V_4
2
2
V_5
V_6
2
3
V_7
V_8
3
3
V_9
V_{10}
3
V_{11}
3
V_{12}
3
4
4
4
4
V_{13}
V_{14}
V_{15}
V_{16}

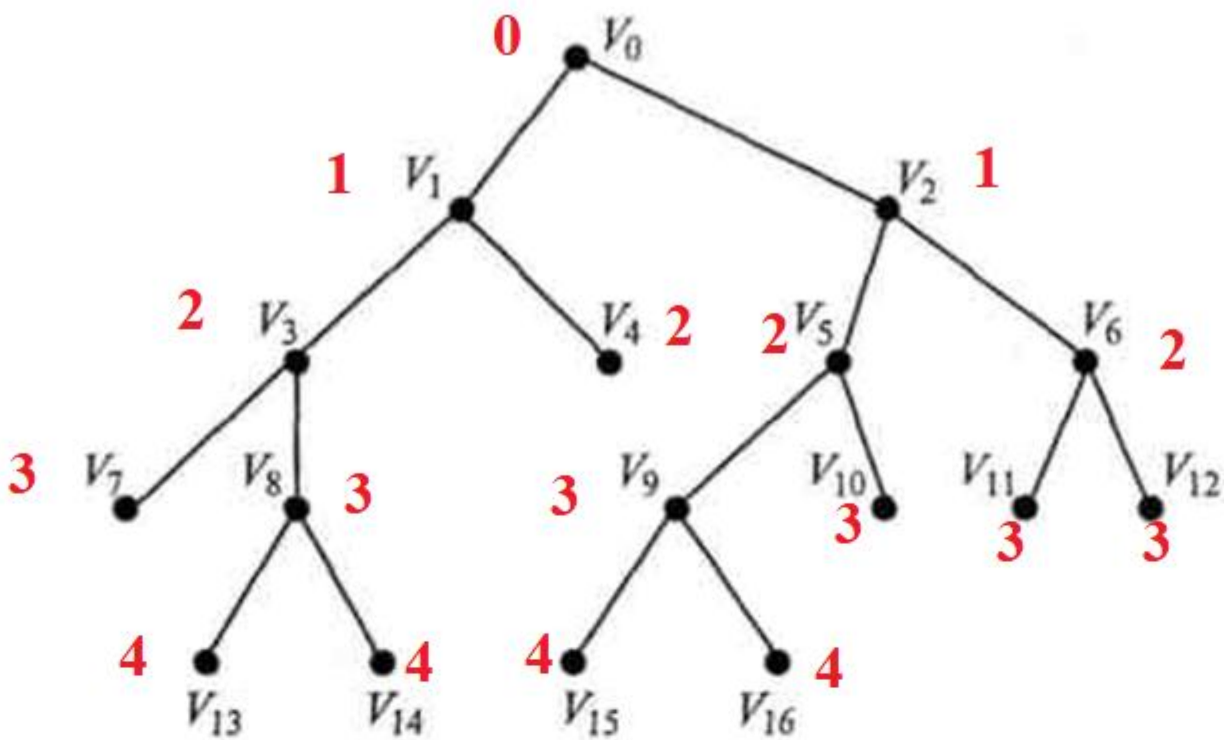
LCA(V15, V6) = ?

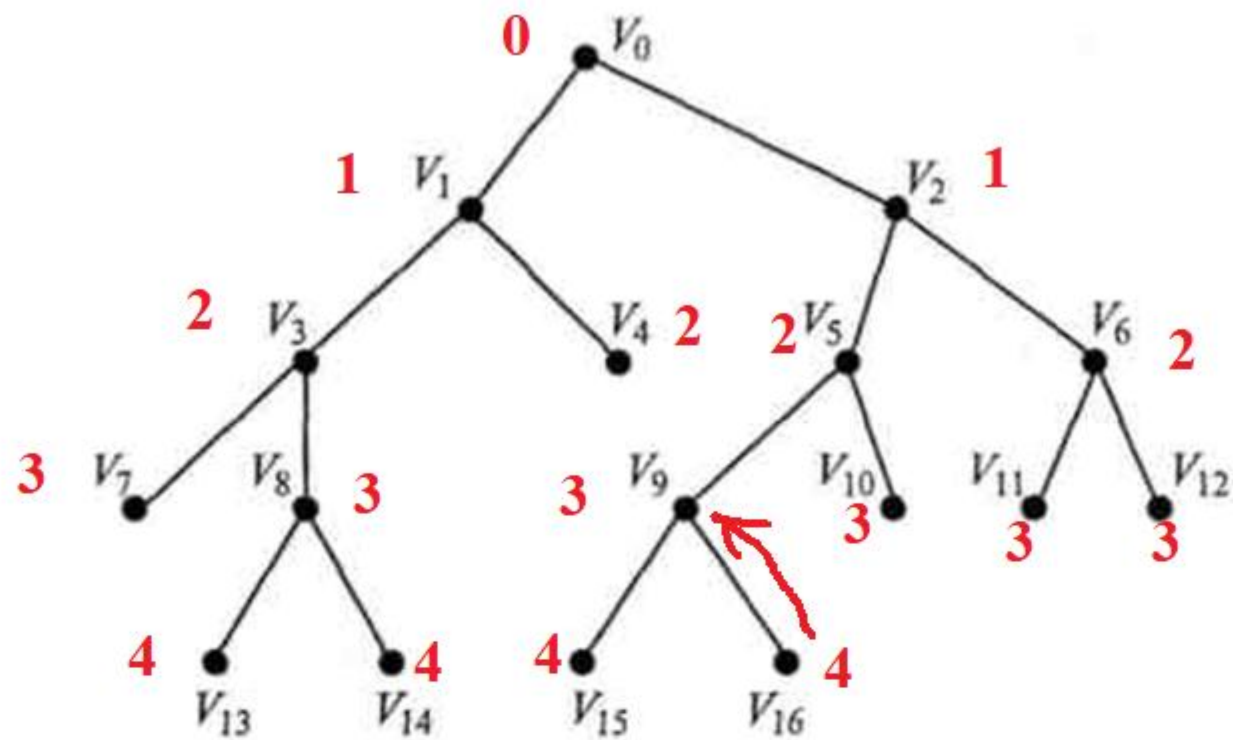

0
V_0
1
V_1
V_2
1
2
V_3
V_4
2
2
V_5
V_6
2
3
V_7
V_8
3
3
V_9
V_{10}
3
V_{11}
3
V_{12}
3
4
V_{13}
4
V_{14}
4
V_{15}
V_{16}
4

LCA(V15, V6) = V2

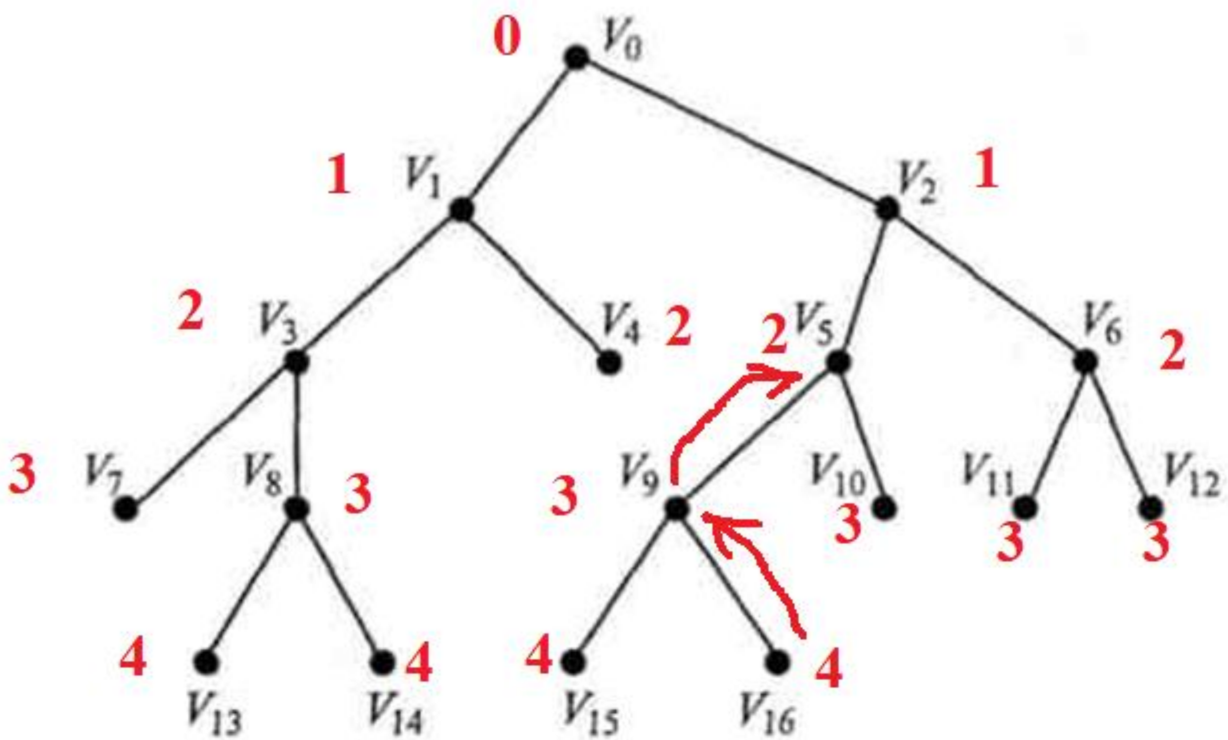
Наивный алгоритм

1. Поднимем от u или v до min(h[u], h[v])
2. Затем, пока предки вершин u и v не совпали, поднимаемся на 1 уровень вверх

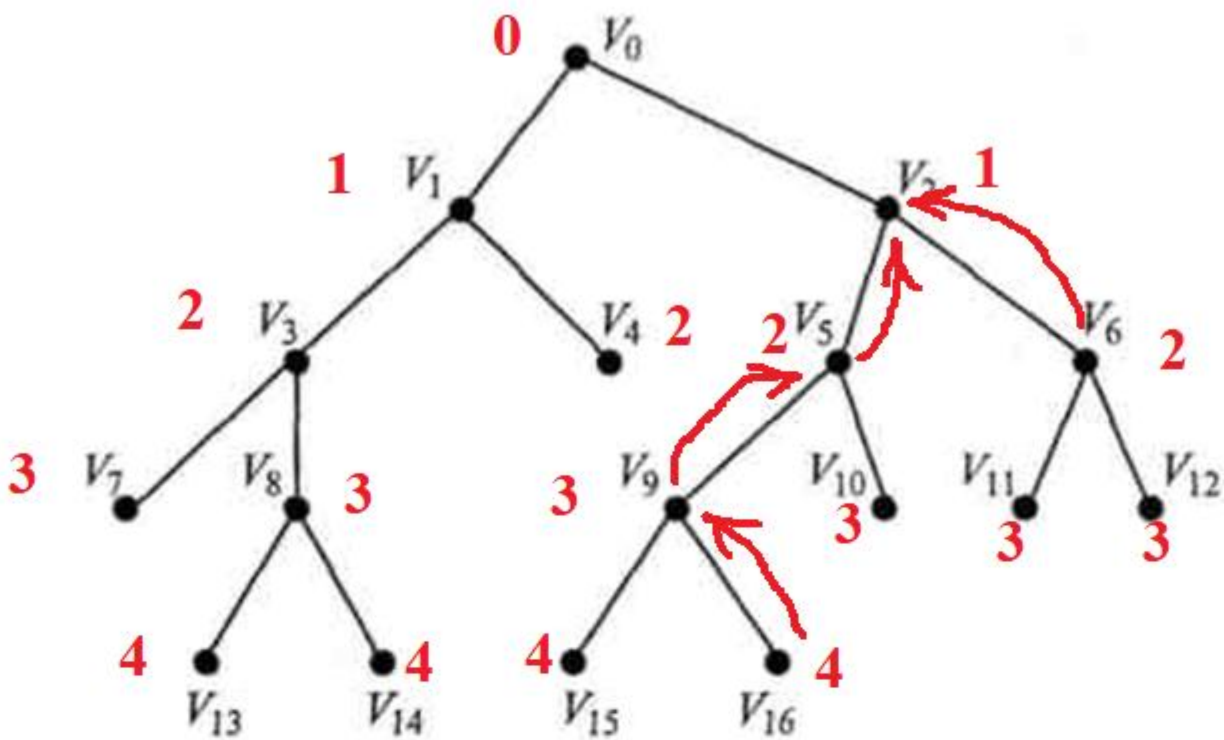

0 V_0
1 V_1
V_2 1
2 V_3
V_4 2
2 V_5
V_6 2
3 V_7
V_8 3
3 V_9
V_{10} 3
V_{11} 3
V_{12} 3
4 V_{13}
V_{14} 4
4 V_{15}
V_{16} 4



0
V_0
1
V_1
V_2
1
2
V_3
V_4
2
2
V_5
V_6
2
3
V_7
V_8
3
3
V_9
V_{10}
3
V_{11}
3
V_{12}
3
4
4
4
4
V_{13}
V_{14}
V_{15}
V_{16}



0
V_0
1
V_1
V_2
1
2
V_3
V_4
2
2
V_5
V_6
2
3
V_7
V_8
3
3
V_9
V_{10}
V_{11}
V_{12}
3
3
3
4
4
4
4
V_{13}
V_{14}
V_{15}
V_{16}



0
V_0
1
V_1
V_2
1
2
V_3
V_4
2
2
V_5
V_6
2
3
V_7
V_8
3
3
V_9
V_{10}
3
V_{11}
3
V_{12}
3
4
4
4
4
V_{13}
V_{14}
V_{15}
V_{16}

Наивный алгоритм

```
Процедура LCA(u, v):
    h1 := depth(u)            // depth(x) = глубина вершины x
    h2 := depth(v)

    while h1 ≠ h2:
       if h1 > h2:
          u := parent(u)
          h1 := h1 - 1
       else:
          v := parent(v)
          h2 := h2 - 1

    while u ≠ v:
       u := parent(u)         // parent(x) = непосредственный предок вершины x
       v := parent(v)

    return u
```

Наивный алгоритм

Время работы – $o(h)$, h – высота дерева

Двоичные подъемы

Двоичные подъемы

Для каждой вершины запомним её предка, стоящего на 1, 2, 4, …, 2^k уровней выше ($2^k \leq n$).

Двоичные подъемы

Для каждой вершины запомним её предка, стоящего на 1, 2, 4, …, 2^k уровней выше ($2^k \leq n$).
Если мы приходим в корень дерева, то мы остаемся в нём.

Двоичные подъемы

1. Для каждой вершины найдем её непосредственного предка (если вершина корень, то она является предком самой себя)

Двоичные подъемы

1. Для каждой вершины найдем её непосредственного предка (если вершина корень, то она является предком самой себя)
2. Применим ДП:

Двоичные подъемы

1. Для каждой вершины найдем её непосредственного предка (если вершина корень, то она является предком самой себя)
2. Применим ДП:
   dp[v][i] – предок вершины v, находящийся от неё в 2^i шагах

Двоичные подъемы

1. База динамики:

Двоичные подъемы

1. База динамики:
   dp[v][0] = p[v]

Двоичные подъемы

1. База динамики:
   dp[v][0] = p[v]
2. Переход:

Двоичные подъемы

1. База динамики:
   dp[v][0] = p[v]

2. Переход:
   Чтобы найти 2^i-го предка, поднимемся на 2^{i-1} шагов вверх и прыгнем ещё на 2^{i-1} шагов вверх

Двоичные подъемы

1. База динамики:
   dp[v][0] = p[v]
2. Переход:
   Чтобы найти 2^i-го предка, поднимемся на 2^{i-1} шагов вверх и прыгнем ещё на 2^{i-1} шагов
   Формула:
   $dp[v][i] = dp[dp[v][i-1]][i-1], i \leq \log_2 N$

Ответы на запросы

Будем считать, что $d[u] > d[v]$

Ответы на запросы

Будем считать, что d[u] > d[v]

Ответы на запросы

Будем считать, что $d[u] > d[v]$

Если $c = LCA(u, v)$, то $d[c] \leq \min(d[u], d[v])$

Ответы на запросы

Будем считать, что $d[u] > d[v]$

Если $c = LCA(u, v)$, то $d[c] \le \min(d[u], d[v])$

Найдем вершину u' такую, что u' – предок u и $d[u'] = d[v]$

Ответы на запросы

Будем считать, что $d[u] > d[v]$
Если $c = LCA(u, v)$, то $d[c] \leq \min(d[u], d[v])$
Найдем вершину u' такую, что u' – предок u и $d[u'] = d[v]$
Найдем предков вершин u' и v x и y таких, что $x \neq y$, но $p[x] = p[y]$

Ответы на запросы

Будем считать, что d[u] > d[v]

Если c = LCA(u, v), то $d[c] \leq \min(d[u], d[v])$

Найдем вершину u’ такую, что u’ – предок u и d[u’] = d[v]

Найдем предков вершин u’ и v x и y таких, что $x \neq y$, но p[x] = p[y]

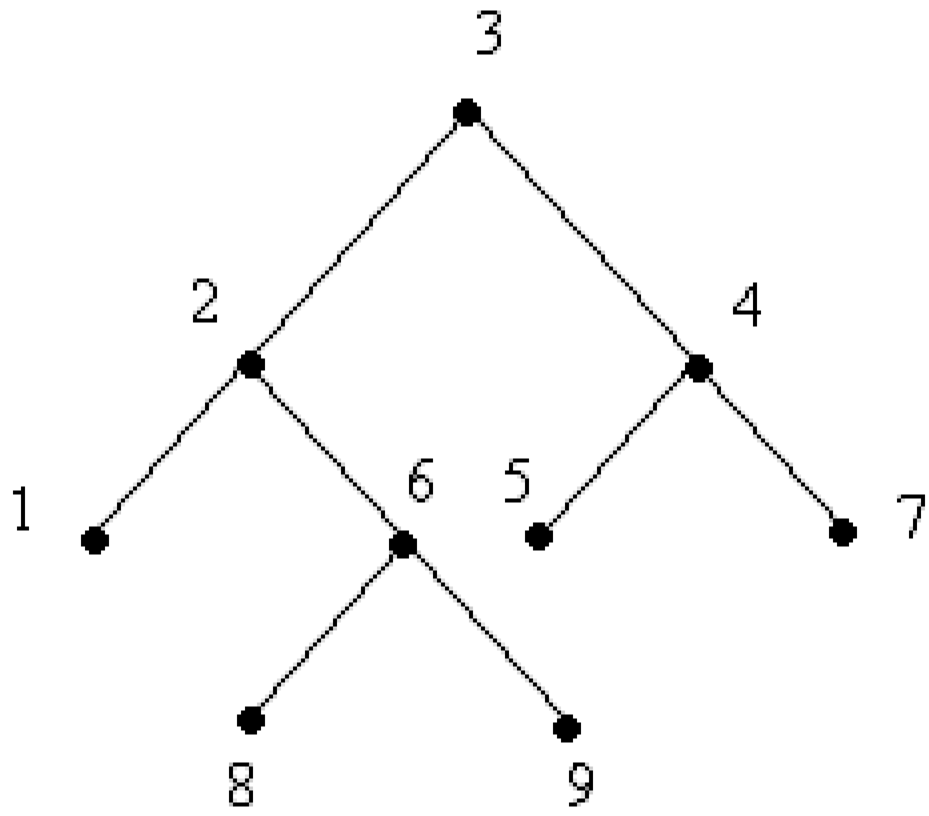
С помощью прыжков найдем x и y

Реализация

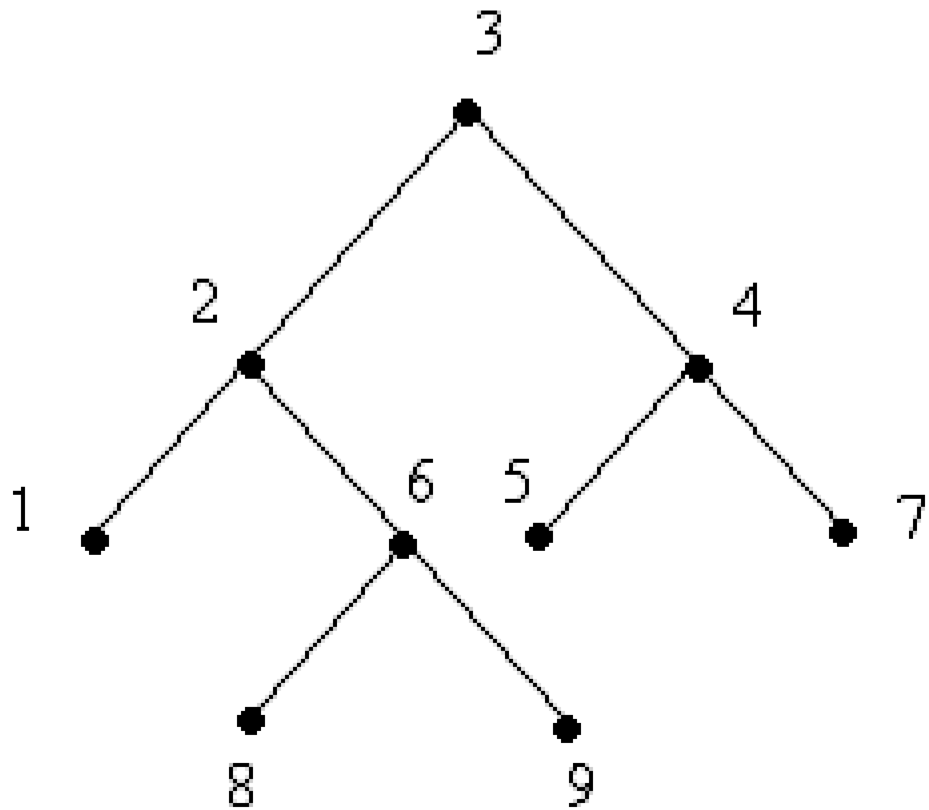
```
function preprocess():
   int[] p = dfs(0)
   for i = 1 to n
      dp[i][0] = p[i]
   for j = 1 to log(n)
      for i = 1 to n
         dp[i][j] = dp[dp[i][j - 1]][j - 1]

int lca(int v, int u):
   if d[v] > d[u]
      swap(v, u)
   for i = log(n) downto 0
      if d[dp[u][i]] - d[v] >= 0
         u = dp[u][i]
   if v == u
      return v
   for i = log(n) downto 0
      if dp[v][i] != dp[u][i]
         v = dp[v][i]
         u = dp[u][i]
   return p[v]
```

Сведение LCA к RMQ

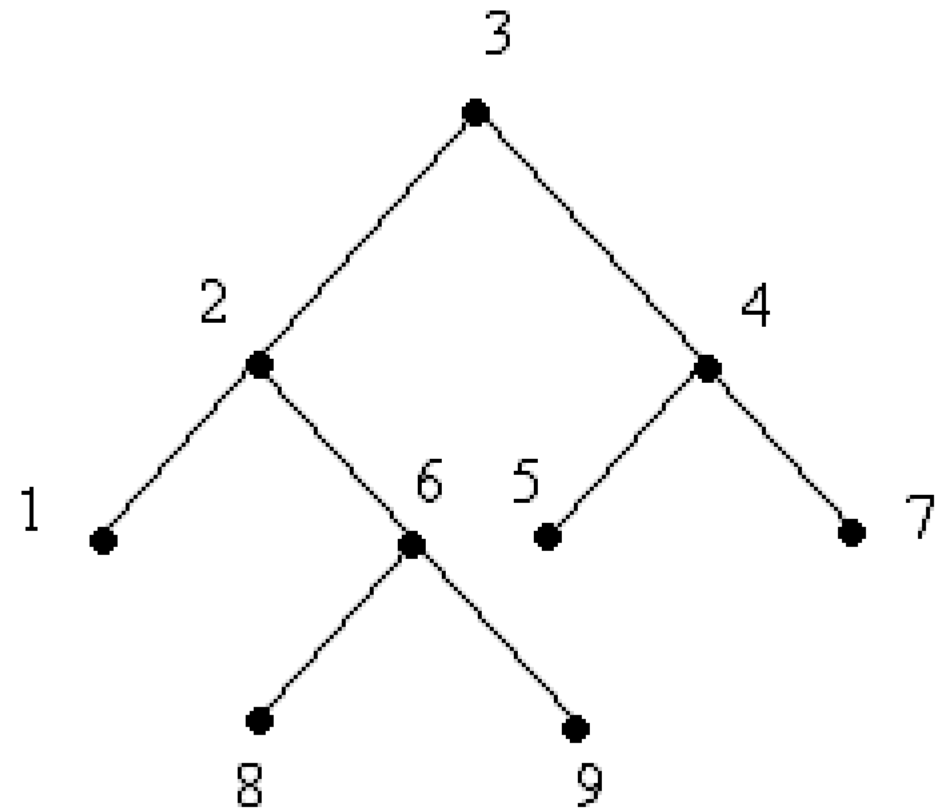


Сведение LCA к RMQ



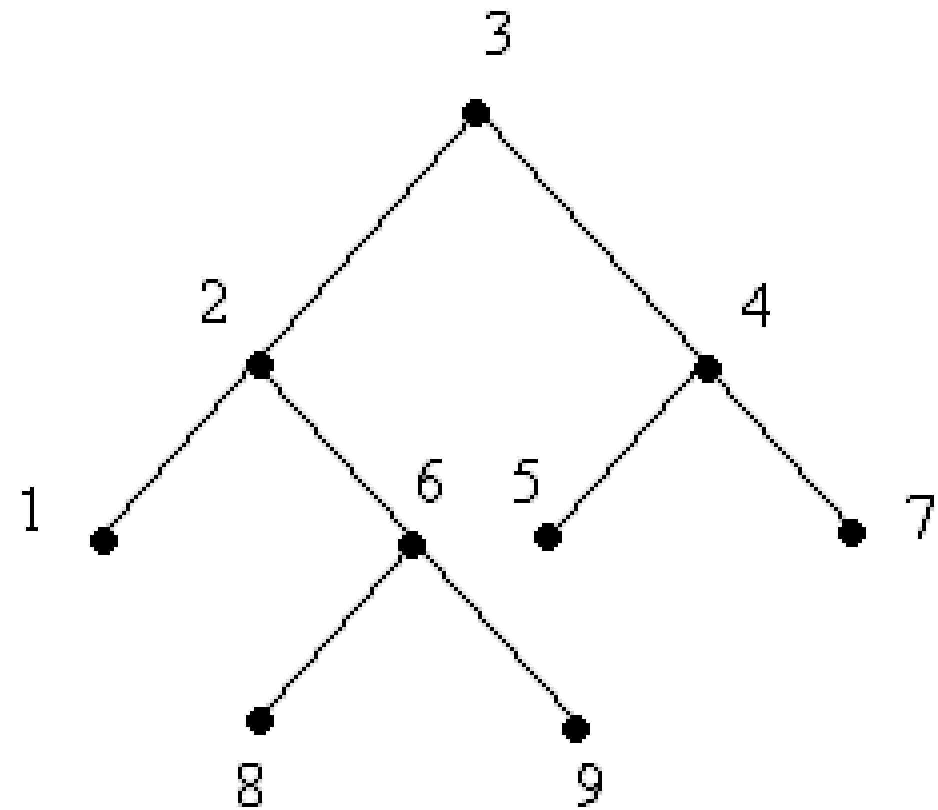
order[]

Сведение LCA к RMQ



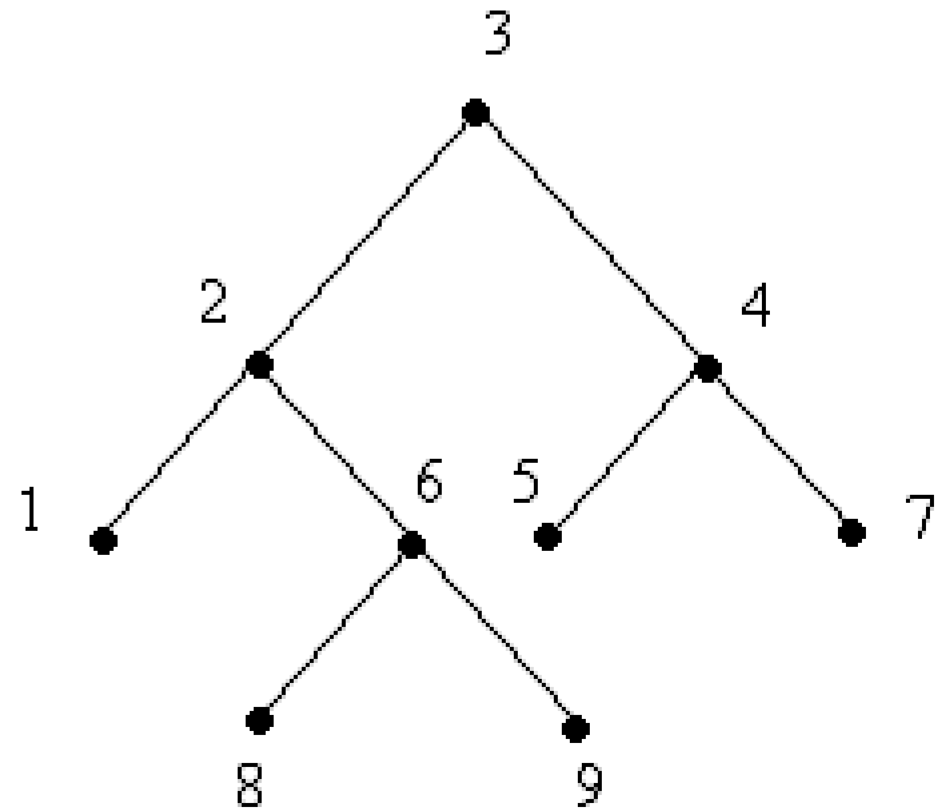
order[] = 3 2 1 2 6 8 6 9 6 2 3 4 5 4 7 4 3

Сведение LCA к RMQ



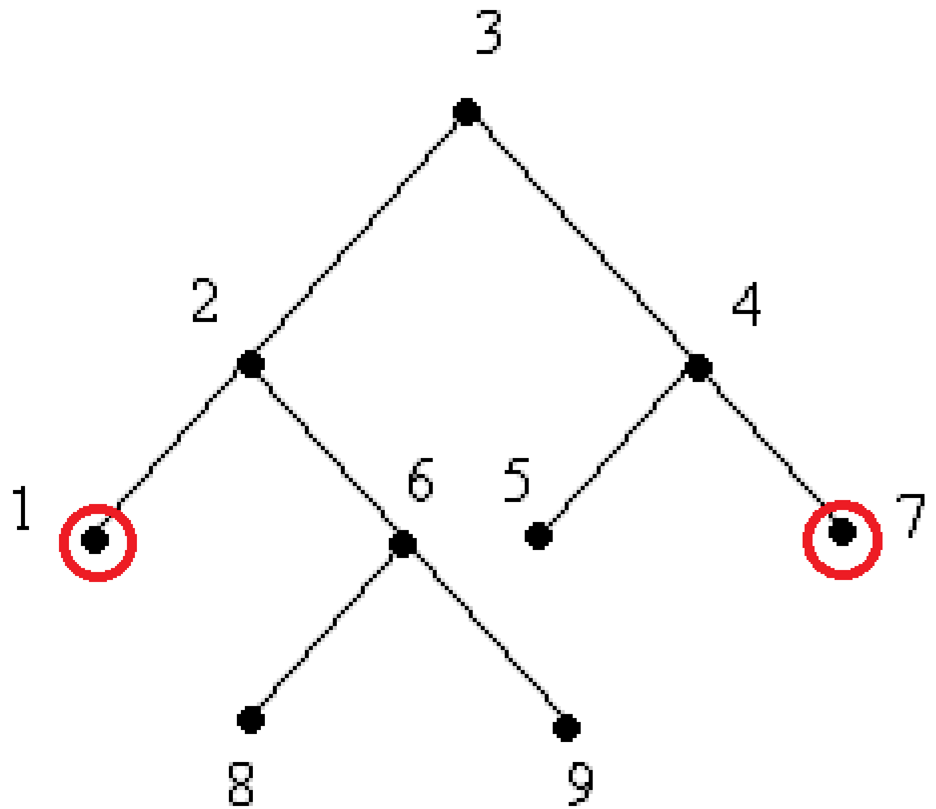
order[] = 3 2 1 2 6 8 6 9 6 2 3 4 5 4 7 4 3
depth[] = 0 1 2 1 2 3 2 3 2 1 0 1 2 1 2 1 0

Сведение LCA к RMQ



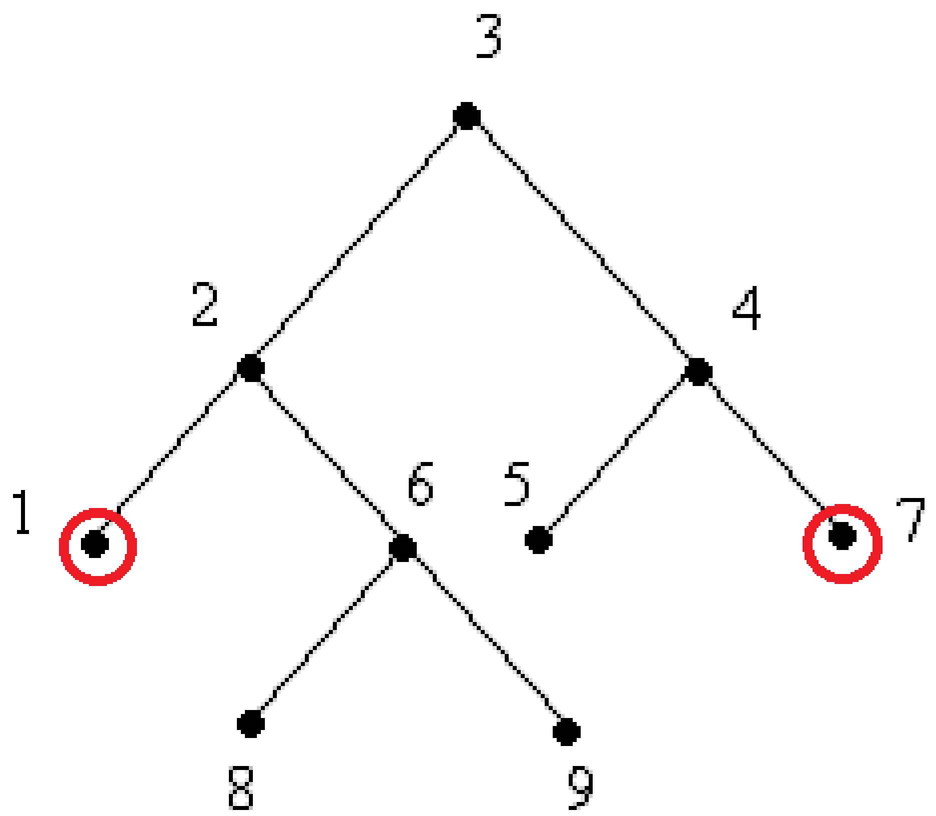
order[] = 3 2 1 2 6 8 6 9 6 2 3 4 5 4 7 4 3
depth[] = 0 1 2 1 2 3 2 3 2 1 0 1 2 1 2 1 0

Сведение LCA к RMQ



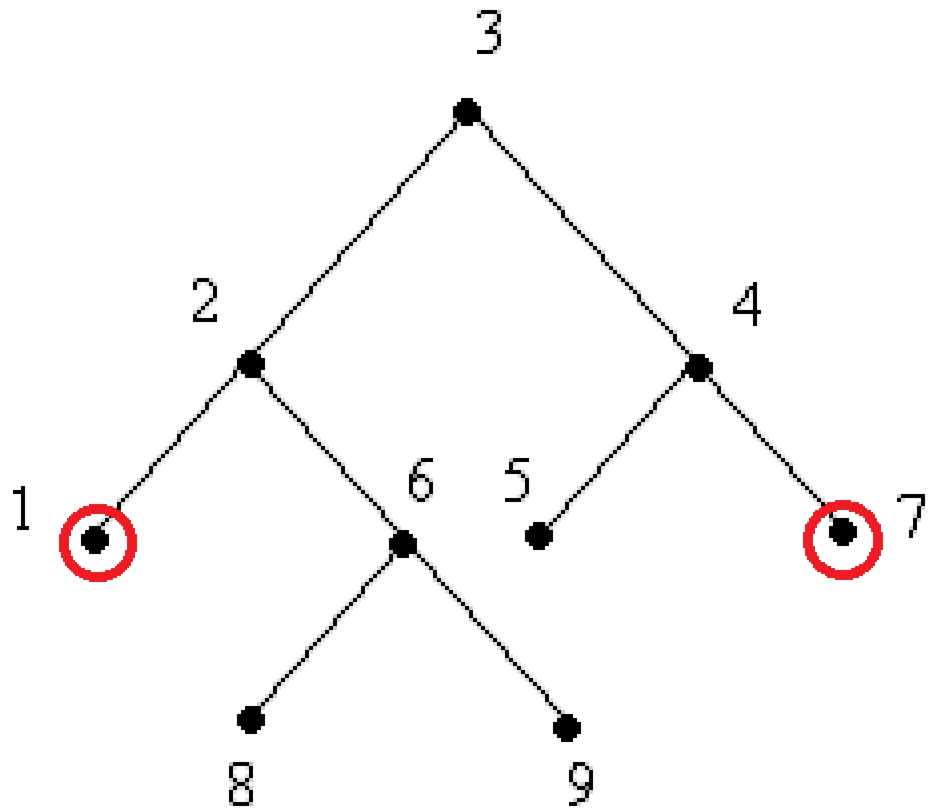
order[] = 3 2 1 2 6 8 6 9 6 2 3 4 5 4 7 4 3
depth[] = 0 1 2 1 2 3 2 3 2 1 0 1 2 1 2 1 0

Сведение LCA к RMQ



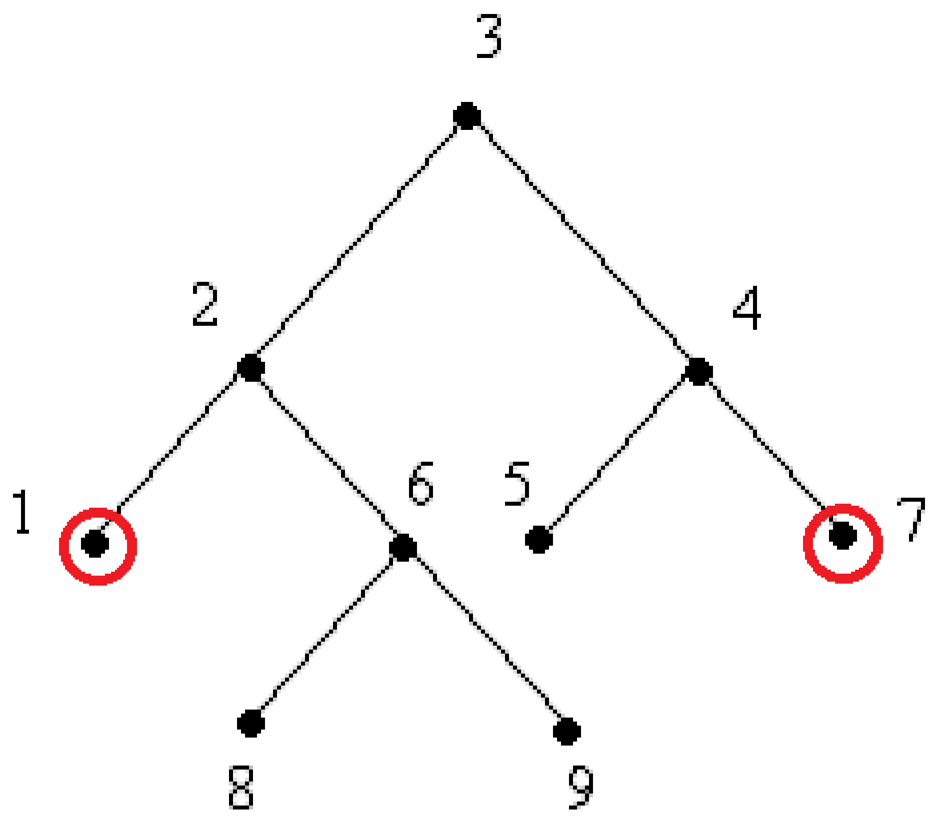
order[] = 3 2 1 2 6 8 6 9 6 2 3 4 5 4 7 4 3
depth[] = 0 1 2 1 2 3 2 3 2 1 0 1 2 1 2 1 0

Сведение LCA к RMQ



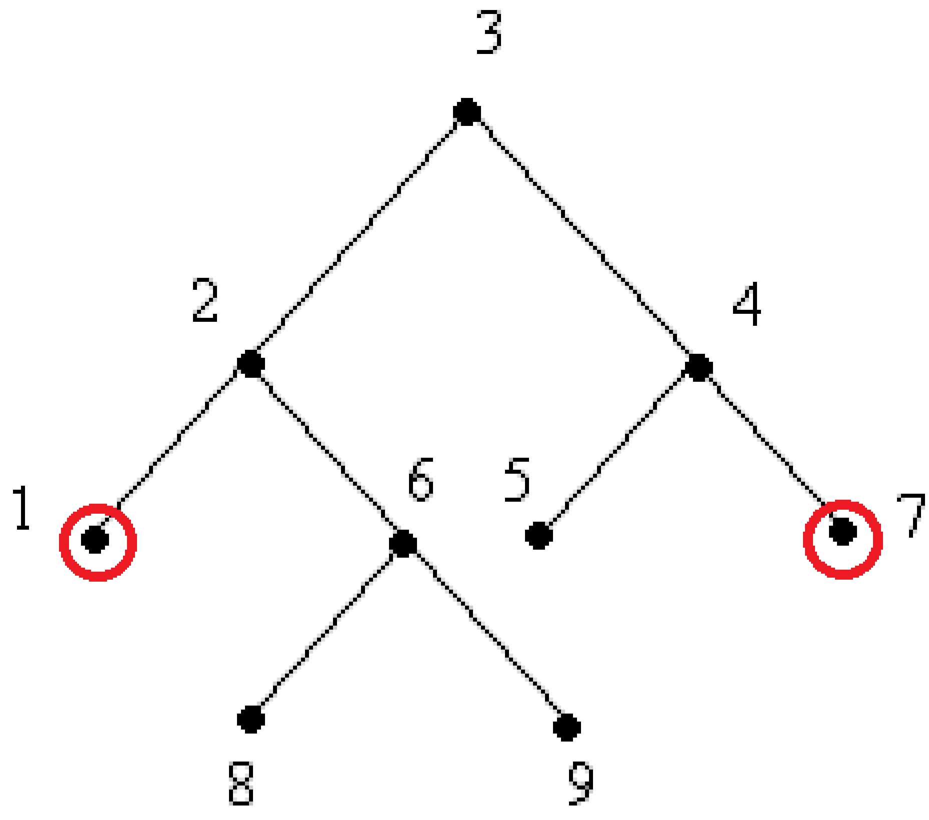
order[] = 3 2 1 2 6 8 6 9 6 2 3 4 5 4 7 4 3
depth[] = 0 1 2 1 2 3 2 3 2 1 0 1 2 1 2 1 0

Сведение LCA к RMQ



order[] = 3 2 1 2 6 8 6 9 6 2 3 4 5 4 7 4 3
depth[] = 0 1 2 1 2 3 2 3 2 1 0 1 2 1 2 1 0
first[] = -1 2 1 0 11 12 4 14 5 7

Сведение LCA к RMQ



order[] = 3 2 1 2 6 8 6 9 6 2 3 4 5 4 7 4 3
depth[] = 0 1 2 1 2 3 2 3 2 1 0 1 2 1 2 1 0
first[] = -1 2 1 0 11 12 4 14 5 7
Ответ: RMQ(first[u], first[v])

Время работы

Метод	Предподсчёт	Запрос
Двоичные подъёмы	O(NlogN)	O(logN)
LCA + ДО	O(N)	O(logN)
LCA + Sparse	O(NlogN)	O(1)

Реализация

```
vector <char> used;
vector <int> h, p;
vector < vector <int> > dp;
int log_2;

void dfs(vector <vector <int> > &gr, int v){
    used[v] = 1;
    for(int  to: gr[v]){
        if (!used[to]){
            p[to] = v;
            h[to] = h[v] + 1;
            dfs(gr, to);
        }
    }
}
```

Реализация

```
void preprocess(vector <vector <int> > &gr){
    dfs(gr, 1);
    p[1] = 1;
    int n = gr.size() - 1;
    for(int i = 1; i <= n; ++i){
        dp[i][0] = p[i];
    }
    for(int j = 1; j <= log_2; ++j){
        for(int i = 1; i <= n; ++i){
            int v = dp[i][j - 1];
            dp[i][j] = dp[v][j - 1];
        }
    }
}
```

Реализация

```
int lca(int u, int v){
    if (h[v] > h[u]){
        swap(u, v);
    }
    for(int i = log_2; i >= 0; --i){
        int t = dp[u][i];
        if (h[t] - h[v] >= 0){
            u = t;
        }
    }
    if (u == v){
        return u;
    }
    for(int i = log_2; i >= 0; --i){
        if (dp[u][i] != dp[v][i]){
            u = dp[u][i];
            v = dp[v][i];
        }
    }
    return p[u];
}
```

Реализация

```
int n, u, v;
cin >> n;
used.resize(n + 1);
p.resize(n + 1);
h.resize(n + 1);
log_2 = 0;
while ((1 << log_2) < n){
    ++log_2;
}
dp.resize(n + 1, vector <int>(log_2 + 1));
vector <vector <int> > gr(n + 1);
for(int i = 0; i < n - 1; ++i){
    cin >> u >> v;
    gr[u].push_back(v);
    gr[v].push_back(u);
}
preprocess(gr);
// cout << lca(4, 9) << endl;
// cout << lca(6, 2) << endl;
// cout << lca(7, 4);
```