Лабораторная работа №2

Входные данные

1. *M_intM* – количество бит, используемых для генерации идентификаторов
2. *m_arPos* – идентификаторы позиций, в которых находятся узлы

Задание

Моделирование алгоритма Chord, используемого при создании структурированных пиринговых сетей. Модель должна быть реализована в виде массива объектов класса *ChordNode*. Для выполнения задания требуется выполнить следующие операции:

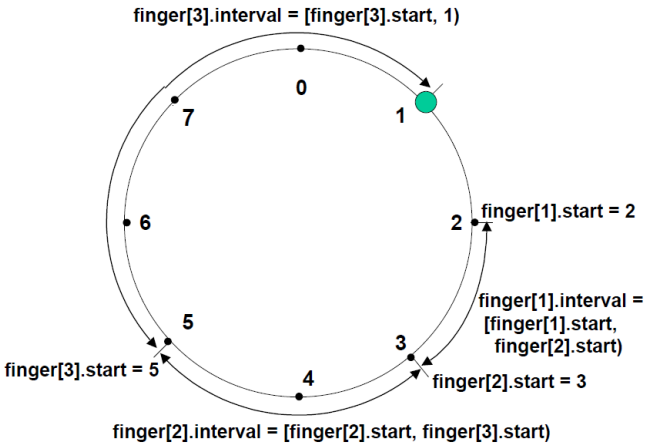
1. Реализовать класс *ChordNode*, содержащий всю необходимую информацию об узле
2. Реализовать функции:
   a. Поиск по идентификатору
   b. Добавление узла
   c. Удаление узла
   d. Стабилизация системы (доп.)

Пояснения

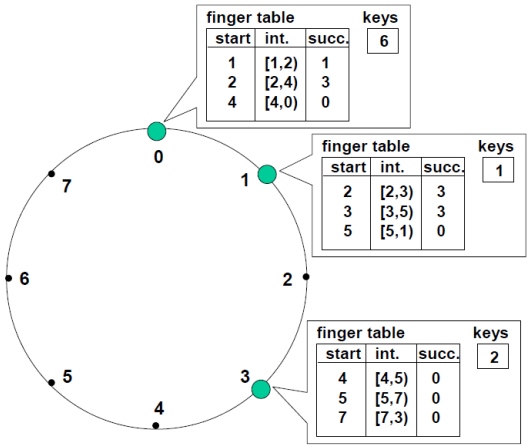
1. Данные, которыми должен располагать узел (*ChordNode*)

Параметр	**Содержание**
finger[i].start	$\left(n+2^{i-1}\right)\bmod 2^m, i \in [1,m)$
.interval	[*finger[i].start, finger[i+1].start*)
.node	Первый узел ≥ *finger[i].start*
successor	Следующий узел в кольце *finger[1].node*
predecessor	Предыдущий узел в кольце

2. Пример finger table для узлов



(a)



(b)

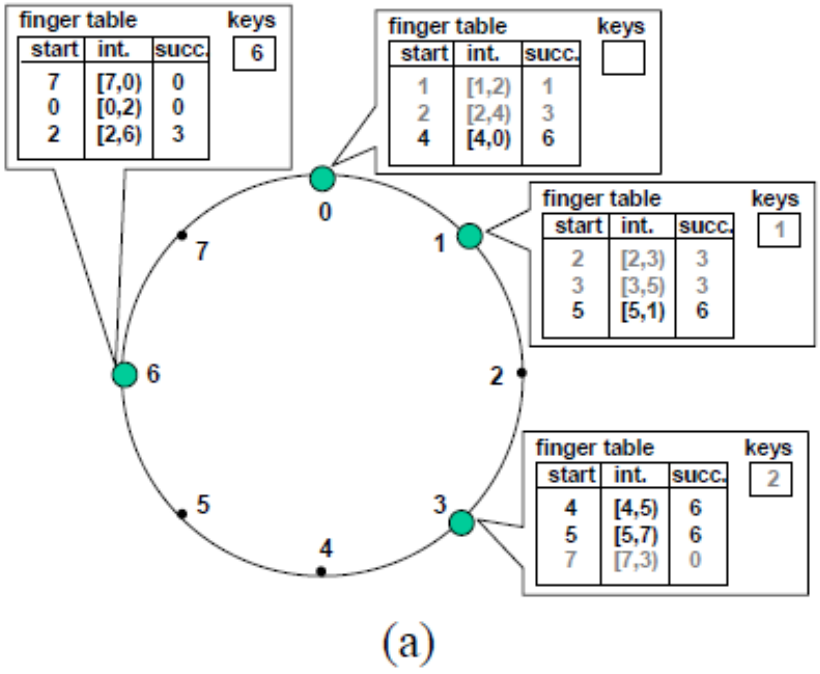
3. Псевдокод алгоритма поиска *successor* узла для идентификатора *id*

```
// ask node n to find id's successor
n.find_successor(id)
  n' = find_predecessor(id);
  return n'.successor;

// ask node n to find id's predecessor
n.find_predecessor(id)
  n' = n;
  while (id ∉ (n', n'.successor])
    n' = n'.closest_preceding_finger(id);
  return n';

// return closest finger preceding id
n.closest_preceding_finger(id)
  for i = m downto 1
    if (finger[i].node ∈ (n, id))
      return finger[i].node;
  return n;
```

4. Пример изменения системы после добавления узла с *id*=6 (неизмененная часть выделена серым)



(a)

5. Псевдокод добавления узла в систему

```
#define successor finger[1].node

// node n joins the network;
// n' is an arbitrary node in the network
n.join(n')
  if (n')
    init_finger_table(n');
    update_others();
    // move keys in (predecessor, n] from successor
  else // n is the only node in the network
    for i = 1 to m
      finger[i].node = n;
    predecessor = n;

// initialize finger table of local node;
// n' is an arbitrary node already in the network
n.init_finger_table(n')
  finger[1].node = n'.find_successor(finger[1].start);
  predecessor = successor.predecessor;
  successor.predecessor = n;
  for i = 1 to m - 1
    if (finger[i + 1].start ∈ [n,finger[i].node))
      finger[i + 1].node = finger[i].node;
    else
      finger[i + 1].node =
          n'.find_successor(finger[i + 1].start);


// update all nodes whose finger
// tables should refer to n
n.update_others()
  for i = 1 to m
    // find last node p whose i^th finger might be n
    p = find_predecessor(n - 2^(i-1));
    p.update_finger_table(n, i);

// if s is i^th finger of n, update n's finger table with s
n.update_finger_table(s, i)
  if (s ∈ [n,finger[i].node))
    finger[i].node = s;
    p = predecessor; // get first node preceding n
    p.update_finger_table(s, i);
```

6. При удалении узла, он информирует предыдущий и последующий узлы об этом и передает свои данные следующему узлу.
7. Псевдокод для выполнения стабилизации системы *(дополнительное задание)*

```
n.join(n')
    predecessor = nil;
    successor = n'.find_successor(n);

// periodically verify n's immediate successor,
// and tell the successor about n.
n.stabilize()
    x = successor.predecessor;
    if (x ∈ (n, successor))
        successor = x;
    successor.notify(n);

// n' thinks it might be our predecessor.
n.notify(n')
    if (predecessor is nil or n' ∈ (predecessor, n))
        predecessor = n';

// periodically refresh finger table entries.
n.fix_fingers()
    i = random index > 1 into finger[];
    finger[i].node = find_successor(finger[i].start);
```

Дополнительную информацию можно найти в статье авторов Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications” (статья находится в отдельном файле).