A. Мониторинг

Транспонирование матрицы

Решение

Complexity
Time O(nm)
Space O(nm)

```
array <array <int>> Transpose(array <array <int>> matrix)
   m = matrix.Size
   n = matrix.Front.Size
   array <array <int>> result(n, m)
   for (int j = 0; j < n; j++)
      for (int i = 0; i < m; i++)
         result[j][i] = matrix[i][j]
      return result

array <array <int>> Transpose()
   m = Input(int)
   n = Input(int)
   matrix <int> result(n, m);
   for (int i = 0; i < n; ++i)
      for (int j = 0; j < m; ++j)
         result[i][j] = Input(int)
   return result
```

Задачки на список

B. Список дел

Вывести список по его голове

Решение

```
struct Node
    string value
    Node next

void PrintList(Node node)
   while (node not null)
      Print(node->value)
      node = node->next
```

C. Нелюбимое дело

Удалить узел из списка по индексу и вернуть голову на этот список

Решение

```
struct Node
    string value
    Node next

Node GetNode(Node node, int index)
   while (index != 0)
      node = node->next
      index--
   return node

Node Remove(Node node, int index)
   if (index == 0)
      if (node != null)
         node = node->next
      return node

   Node prev = GetNode(node, index - 1)
   prev->next = prev->next->next

return head
```

D. Заботливая мама

Найти позицию узла в списке по значению

Решение

```
struct Node
    string value
    Node next

int Find(Node node, int value)
   int index = 0
   while (node != null)
      if (node->value == value)
         return index
      index += 1
      node = node->next
   return -1
```

E. Все наоборот

Развернуть список и вернуть голову этого развернутого списка

Разворот двухсвязного списка

Complexity
Time O(n)
Space O(1)

```
struct Node                Input  : 1 <-> 2 <-> 3 <-> 4
    string value           Output : 4 <-> 3 <-> 2 <-> 1
    Node next
    Node prev

Node ReverseList(Node head)
    Node node = head
    while (node != null)
        Swap(node->Prev, node->Next)
        head = node
        node = node->Prev
    return head

Node ReverseList(Node head)
    Node left = head
    Node right = head

    while (right->next != null)
        right = right->next

    while (left != right and left->prev != right)
        Swap(left->value, right->value)
        left = left->next
        right = right->prev

    return head
```

Разворот односвязного списка

Complexity
Time O(n)
Space O(1)

```
struct Node
    string value
    Node next
```

Input : 1 -> 2 -> 3 -> 4 -> 5
Output : 5 -> 4 -> 3 -> 2 -> 1

```
Node ReverseForwardList(Node head)
    Node reverse_head = null
    while (head != null)
        Node temp = head->next
        head->next = reverse_head
        reverse_head = head
        head = temp
    return reverse_head
```

G. Стек - MaxEffective

Реализовать стек поддерживающий операцию возвращения максимума из элементов за O(1)

Решение

```
class StackMaxEffective
public:
    void Push(int val)
        if (Empty)
            highs.Add(val)
        else
            highs.Add(Max(val, highs.Back))

    void Pop()
        highs.PopBack

    int GetMax()
        return highs.Back

    bool Empty()
        return values.Size == 0

private:
    array <int> highs
```

H. Скобочная последовательность

Проверить или строка является правильной скобочной последовательностью

Решение

```
bool IsCorrectBrackets(string str)
   stack <char> brackets
   for (char ch in str)
      if (bracket == '(')
         brackets.Push(')')
      else if (bracket == '{')
         brackets.Push('}')
      else if (bracket =='[')
         brackets.Push(']')
      else if (brackets.Empty or brackets.Top != bracket)
         return false
      else
         brackets.Pop
     return brackets.Empty
```

I. Ограниченная очередь

Реализовать очередь ограниченного размера

Решение

```
class MyQueueSized
public:
   MyQueueSized(int n) : capacity(n)

   bool Push(int val)
      if (data.Size == capacity)
         return false
      else
         data.PushBack(val)
         return true

   int Pop()
      int val = data.front
      data.PopFront
      return val

   int Peek()
      return data.Front

   int Size()
```

```
      return data.Size

   bool Empty()
      return Size == 0

private:
   int capacity
   list <int> data
```

J. Списочная очередь

Реализовать очередь на узлах списка

Решение

```
struct Node
   int value
   Node next

class ListQueue
public:
   ListQueue()
      size = 0
      head = null
      tail = null

   int Size()
      return size

   int Pop()
      value = head->value
      head = head->next
      size--
      return value
```

```
   void Put(int val)
      Node node(val, null)
      if (Size == 0)
         head = node
         tail = node
      else
         tail->next = node
         tail = node
      size++

private:
   int  size = 0
   Node head = null
   Note tail = null
```

Фибоначчи

K. Рекурсивные числа Фибоначчи

Рекурсивно найти числа Фибоначчи

Решение

```
int Fib(int n)
   if (n <= 1)
      return 1
   else
      return Fib(n - 1) + Fib(n - 2)
```

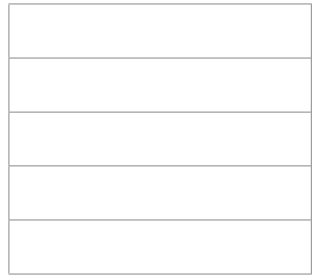
Complexity
Time $O(2^n)$
Space O(n)

Убираем хвосты

```
int Fib(int n, int lhs = 1, int rhs = 1)
   if (n <= 1)
      return rhs
   return Fib(n - 1, rhs, lhs + rhs)
```

Complexity
Time O(n)
Space O(1)

Stack



Хвостовая рекурсия — это частный случай рекурсии, при котором рекурсивный вызов единственен и является последней инструкцией

Мемоизация

Complexity
Time O(n)
Space O(n)

```
int Fib(int n)
   array <int> data { 1, 1 }
   for (int i = 1; data.Size <= n; ++i)
      data.Add(data[i] + data[i - 1])
   return data.Back
```

Мемоизация — сохранение промежуточных результатов для предотвращения повторных вычислений

Итеративная версия

```
int Fib(int n, int m)
   int lhs = 1
   int rhs = 1
   for (int i = 0; i < n; ++i)
      Swap(lhs, rhs)
      rhs = (lhs + rhs) mod m
   return lhs
```

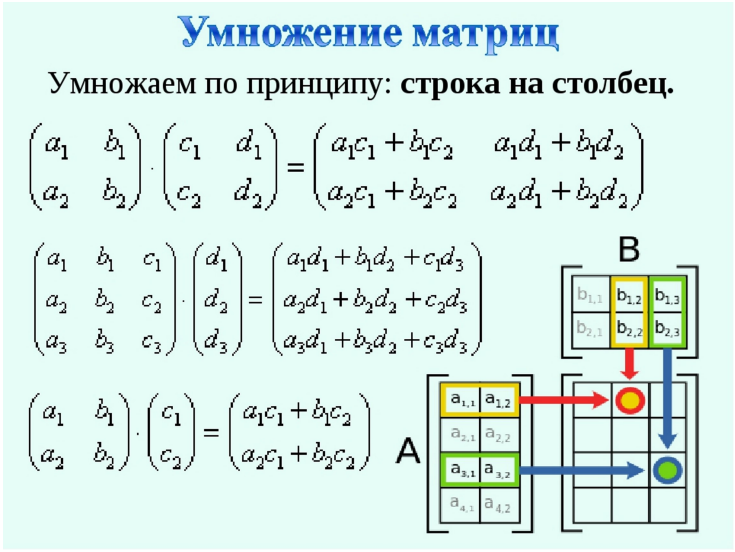
Complexity
Time O(n)
Space O(1)

Матричная формула

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{(n+1)} & F_{(n)} \\ F_{(n)} & F_{(n-1)} \end{pmatrix}$$

$$A \cdot B = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} =$$

$$= \begin{pmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} & a_{11} \cdot b_{12} + a_{12} \cdot b_{22} \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} & a_{21} \cdot b_{12} + a_{22} \cdot b_{22} \end{pmatrix}$$




```
matrix <type> == array <array <type>>

matrix <int> Multiply(matrix <int> l, matrix <int> r)
   matrix <int> temp =
   {
      { l[0][0] * r[0][0] + l[0][1] * r[1][0], l[0][0] * r[0][1] + l[0][1] * r[1][1] },
      { l[1][0] * r[0][0] + l[1][1] * r[1][0], l[1][0] * r[0][1] + l[1][1] * r[1][1] }
   }
   return temp
```

Бинарное возведение в степень

$a^n = (a^{n/2})^2 = a^{n/2} * a^{n/2}$

$a^n = a^{n/2} * a^{n/2} * a$

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{(n+1)} & F_{(n)} \\ F_{(n)} & F_{(n-1)} \end{pmatrix}$$

```
int Pow(int val, int n)
    int result = 1
    for (int i = 0; i < n; ++i)
        result *= val
    return result
```

```
int Pow(int val, int n)
    if (n == 0)
        return 1
    int result = Pow(val, n / 2)
    result *= result
    if (n mod 2 != 0)
        return result * val
    return result
```

```
int Pow(int val, int n)
    int result = 1
    while (n != 0)
        if (n mod 2 != 0)
            result *= val
        val *= val
        n /= 2
    return result
```

Рекурсивное решение

Complexity
Time O(log n)
Space O(log n)

```
matrix <type> == array <array <type>>

void Power(matrix <int> mat, int n)
   if (n <= 1)
      return

   Power(mat, n / 2)
   mat = Multiply(mat, mat)

   matrix <int> temp = { {1, 1}, {1, 0} }
   if (n % 2 != 0)
      mat = Multiply(mat, temp)

int FibRecursive(int n)
   matrix <int> mat = { {1, 1}, {1, 0} }
   Power(mat, n)
   return mat[0][0]
```

Итеративное решение

Complexity
Time O(log n)
Space O(1)

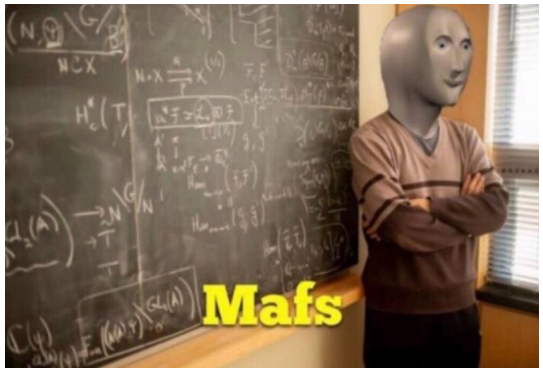
```
matrix <type> == array <array <type>>

int FibIterative(int n)
   matrix <int> temp   = { {1, 1}, {1, 0} }
   matrix <int> result = { {1, 0}, {0, 1} }
   while (n != 0)
      if (n % 2 != 0)
         result = Multiply(temp, result)
      temp = Multiply(temp, temp)
      n /= 2
   return result[0][0]
```

Формула Бине

```
int Fib(int n)
    double phi = (Sqrt(5) + 1) / 2
    return Round(phiⁿ / Sqrt(5))
```

$$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.6180339887....$$



Complexity
Time O(log n)
Space O(1)

L. Фибоначчи по модулю

Возьмем первые 10 чисел последовательности
1 1 2 3 5 8 13 21 34 55
Допустим нам нужно получить последнее число 10 числа Фибоначчи
Т.е. если считать классически, то будет
1 + 1 = 2
1 + 2 = 3
2 + 3 = 5
3 + 5 = 8
5 + 8 = 13
8 + 13 = 21
13 + 21 = 34
21 + 34 = 55
И берем от 55 последнее число
Но можно каждый раз брать остаток от 10 и будет
1 + 1 = 2
1 + 2 = 3
2 + 3 = 5
3 + 5 = 8
5 + 8 = 3
8 + 3 = 1
3 + 1 = 4
1 + 4 = 5
???
PROFIT

```
int Fib(int n, int m)
   int mod = 10^m;
   int lhs = 1
   int rhs = 1
   for (int i = 0; i < n; ++i)
      Swap(lhs, rhs)
      rhs = (lhs + rhs) % mod
   return lhs
```