Вопросы по первому модулю.

Солтан Текеев **Георгий Габолаев**

1. **Что означают записи “f(n) = Θ(g(n))”, “f(n) = O(g(n))” и “f(n) = Ω(g(n))”?**

   f(n) = Ω(g(n)) : f асимптотически ограничена снизу функцией g

   f(n) = O(g(n)) : f асимптотически ограничена сверху функцией g

   f(n) = Θ(g(n)) : f асимптотически ограничена сверху и снизу функцией g

2. **Чем плох рекурсивный алгоритм вычисления n-ого числа Фибоначчи?**

   Рекурсия глубиной n - 1 => объем доп. памяти M(n)

   Пересчет многих значений => экспоненциальная сложность

3. **Опишите алгоритм проверки числа на простоту за O(sqrt(n))?**

```cpp
#include <cmath>
bool prime(int n) {
    int sqrtFind = sqrt(n);
    for (int i = 2; i <= sqrtFind; i++)
        if (n % i == 0)
            return false;
    return true;
}
```

4. **Опишите алгоритм возведения действительного числа в натуральную степень n за O(log n)?**

```cpp
double pwr(double value, int power){

    double result = 1;
    double valueInPowerOf2 = value;

    for(; power; power >>= 1){
        if((power & 1) == 1)
            result *= valueInPowerOf2;
        valueInPowerOf2 *= valueInPowerOf2;
    }
    return result;
}
```

5. **Опишите нерекурсивный алгоритм бинарного поиска первого вхождения элемента в массиве.**

```
int binarySearch(int arr[], int left, int size, int find) {
    int right = size - 1;
    while (left < right) {

        int middle = left + (right - left) / 2;

        if (arr[middle] >= find)
            right = middle;
        else
            left = middle + 1;

    }
    return (left == size || arr[left] != find) ? -1 : left;
}
```

6. **Какова амортизированная стоимость операции Add в реализации динамического массива с удвоением буфера? Можно ли увеличивать буфер в 1.5 раза? Как это скажется на оценке?**

Амортизированная стоимость: O(1)
Можно. Стоимость останется константной, но коэффициент увеличится, и реаллокации будут происходить чаще.
Однако также изменится и “незанятая” память, причем в лучшую сторону (в худшем случае будет незанята 1/3 памяти)

7. **Сколько времени работает линейный поиск в односвязном списке в худшем и в лучшем случае? Сколько времени работает добавление и удаление элемента в середине списка (середина списка неизвестна, есть указатель на начало и конец списка)?**

Худший случай: O(n);
Лучший случай: O(1);
Само добавление или изменение элемента: O(1), но поиск середины O(n)

8. **Назовите преимущества и недостатки реализации очереди с помощью динамического массива.**

Преимущества:
Отсутствие дополнительных затрат по памяти (например, на указатели, как в списке)

Недостатки:

Ограниченый размер массива и необходимость перевыделять память и выполнять копирование данных

9. Назовите преимущества и недостатки реализации стека с помощью односвязного списка.

Преимущества:

Легкая реализация

Быстрые вставка и удаление узла (просто перецепляем указатели)

Недостатки:

Дополнительная память (указатели на следующие элементы списка)

Нет доступа по индексу

Элементы располагаются в памяти хаотично, что мешает кэшированию процессора.

10. Назовите преимущества и недостатки реализации дека с помощью динамического массива.

Преимущества:

Отсутствие дополнительных затрат по памяти (например, на указатели, как в списке)

Недостатки:

Ограниченый размер массива и необходимость перевыделять память и выполнять копирование данных

Сложность в реализации

11. Опишите подход динамического программирования для вычисления рекуррентных функций двух аргументов: F(x, y) = G(F(x - 1, y), F(x, y - 1)). Как оптимизировать использование дополнительной памяти?

Подход динамического программирования состоит в том, чтобы разбить исходную задачу на подзадачи, и решить каждую подзадачу только один раз, тем самым сократив количество вычислений.

Разобъем задачу на подзадачи.

Вычисление F(x, y) сводится к вычислению F(x - 1, y) и F(x, y - 1), то есть вычислению F дважды от меньших аргументов.

При этом многие значения в рекурсивном решении вычисляются несколько раз:

F(x - 1, y - 1) — Два раза

F(x - 1, y - 2) и F(x - 2, y - 1) — Три раза

F(x - 2, y - 2) — Шесть

Таблица числа вычислений в рек. алг: при x = y = 5

```
 0  ——>x
\|/ [70, 35, 15, 5, 1]
 Y  [35, 20, 10, 4, 1]
    [15, 10,  6, 3, 1]
    [ 5,  4,  3, 2, 1]
    [ 1,  1,  1, 1, 1]
```

Так как для вычисления F(x, y) необходимо вычислить F(i, j) for i=0..x for j=0..y, имеет смысл применить восходящее динамическое программирование.

Будем сохранять уже вычисленные значения в матрице размером [x + 1][y + 1].
Начнем заполнять её от меньших индексов к большим, так как большие индексы зависят от меньших.
Затем просто вернем значение в table[x][y]

Оптимизиация использования доп. памяти:
Нетрудно заметить, что F(x, 0..y) и F(0..x, y) используются единожды, а потому можно вычислить и сохранить значения для F(0..x-1, 0..y-1), а затем посчитать F(x, 0..y) храня только результат предыдущего вычисления, таким образом сократив память на (n+m)

Приведем пример:
G(a, b) = a + b => F(x, y) = F(x - 1, y) + F(x, y - 1)
F(x, 0) = x; F(0, y) = y; (=> F(0, 0) = 0)

12. Вычисление наибольшей общей подпоследовательности.

Давайте возьмём пример, где существуют следующие последовательности:
a = 2 3 5 4 6 (длину обозначим как N, а текущий элемент ai)
b = 1 2 3 4 (длину обозначим как M, а текущий элемент b_j)
Введём ограничение, что когда (N == 0 || M == 0), ответом задачи является пустая последовательность.

В противном случае будем идти с концов обоих последовательностей и на каждом положении i и j возможны 2 варианта:
1) Если $a_i == b_j$, то длина НОП равна $LCS(a_{i-1}, b_{j-1}) + 1$;
2) Если $a_i != b_j$, то длина НОП равна $\max(LCS(a_{i-1}, b_j), LCS(a_i, b_{j-1}))$;

Таким образом сделаем табличку NxM, где нулевые вертикальные и горизонтальные строки заполнены нулями, а остальные можем высчитывать построчно динамически, согласно вышеописанному правилу.
В данном случае:

		2	3	5	4	6
	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	1	1	1	1	1
3	0	1	2	2	2	2
4	0	1	2	2	3	3

После этого просто идём от самого углового элемента, согласно тем же правилам, чтобы восстановить, непосредственно, саму последовательность.

13. Вычисление редакторского расстояния (расстояния Левенштейна)

Расстояние Левенштейна между двумя строками — это минимальное количество операций:
1) I — вставки одного символа
2) D — удаления одного символа
3) R — замены одного символа на другой
необходимых для превращения одной строки в другую.

Алг нахождения расстояния левенштейна при помощи ДП близок к алг LCS

Пусть даны две строки A и B, и их длины N и M.

Тогда в решении потребуется таблица D[0..N+1, 0..M+1]
Элементы в этой таблице будут соответствовать следующей формуле:

$$D(i,j) = \begin{cases} 0 & i = 0,\ j = 0 \\ i & j = 0 \\ j & i = 0 \\ min \begin{bmatrix} D(i,\ j-1)+1 \\ D(i-1,\ j)+1 \\ D(i-1,\ j-1)+m(A[i],\ B[j]) \end{bmatrix} & i > 0,\ j > 0 \end{cases}$$

где `m(A[i], B[i]) = A[i] != B[i] ? 1 : 0`, то есть равно 1 если символы не равны, и 0, если равны.

Тогда 1 вариант символизирует вставку в 1 строку, 2 — удаление из 1 строки, а 3 — либо замену символа (m() == 1), либо отсутствие действий (символы одинаковы)

Можно применить восходящее дп в циклах i=0..N -> j=0..M, как в задаче на LCS.

14. Жадный алгоритм в решении задачи размена монет. Пример, когда жадный алгоритм дает неверное решение.

Скажем, что для начала мы действуем в системе монет Банка России, где существуют монеты номиналом 1,2,5,10 рублей, а сумма, которую нужно выдать, равна, например 98

Согласно жадному алгоритму, мы будем сначала 9 раз брать 10руб, затем 1 раз 5руб, затем 1 раз 2руб и наконец 1 руб. Следует заметить, что жадный алгоритм в подобных задачах действует только в нашей монетной системе, потому что каждый старший номинал (например 10руб) больше либо равен, чем удвоенный младший (5руб).

Примером, когда жадный алгоритм не сработает оптимально, может послужить монетная система, которая как раз противоречит правилу номиналов, описанному чуть выше, а именно, если, например существуют номиналы 1руб, 4 руб, 6 руб. Если в такой монетной системе нам понадобится дать сдачу размером, например, 8 рублей, жадный алгоритм сначала возьмёт 6руб, а затем два раза по 1руб, что не является самым оптимальным решением, а именно 8руб = 4руб + 4руб.

15. Жадный алгоритм в решении задачи “Покрытие отрезками”.

Жадное решение в данной задаче является оптимальным.

Сначала отсортируем отрезки по значению левого конца.

Далее:

1) Найдём отрезок, который включает в себя точку 0, и у которого наибольший правый конец. Обозначим найденный отрезок как $[aa_1,bb_1]$;

2) Найдём отрезок, который включает в себя точку bb_1, и у которого наибольший правый конец. Обозначим найденный отрезок как $[aa_2, bb_2]$;

3) Повторям эти действия, пока не покроем точку X.

16. Жадный алгоритм в решении задачи о рюкзаке.

1) Сортируем элементы по удельному весу каждого (отношение цены к весу)

2) Кидаем в рюкзак первый предмет из отсортированного массива, который влезет.

3) То же самое из оставшегося массива.

4) Выполняем пункт 3 до тех пор, пока хоть что-то из массива влезает в рюкзак.

Контрпример жадного алгоритма:

Вместимость: 90

ВЕС	ЦЕНА	УДЕЛЬНЫЙ (цена/вес)
20	60	3
30	90	3
50	100	2

Согласно жадному алгоритму, ценность предметов в рюкзаке составит 150, но это не самое оптимальное решение, поскольку можно было добавить 2 и 3 предметы, тем самым получив ценность 190.