ЛЕКЦИЯ 3

МЕТОДИКИ РЕШЕНИЯ ТИПОВЫХ ЗАДАЧ

1. Задачи на использование оператора `if`

Рассмотрим задачу на определение принадлежности точки к определенной области. Дан график функции (см. рис. 3.1).

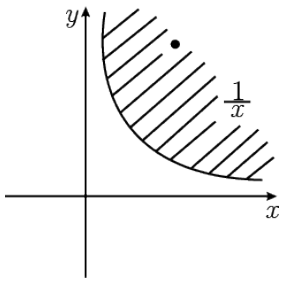


Рис. 3.1

Необходимо напечатать `yes`, если точка принадлежит заштрихованной области, и `no` в противном случае.

Даны координаты рассматриваемой точки — `x` и `y`. В рассматриваемой задаче граничные точки не заданы, поэтому можно делать не такую точную проверку, без использования типов `float` и `double` — можно ограничиться типом `int`. Проверим условия принадлежности точки заданной области. Данное условие будет являться объединением отдельных условий: `x>0 && y>1/x`. При этом отметим, что перемена условий местами — `y>1/x && x>0` — работать не будет, т. к. в данном случае для условия `y>1/x` условие `x>0` еще не наложено, поэтому возможна ситуация использования отрицательных чисел или деления на ноль, к примеру, при рассмотрении точки (0;1). Во избежание путаницы можно воспользоваться раздельной записью условий:

Listing 3.1: Использование оператора `if`.

```
...
```

```
if(x>0) {
        if(y>1/x) {
...
```

Главное в данной ситуации — в первую очередь прописать условие на `x`: `x>0` — тогда ситуации с делением на 0 удастся избежать. Иногда в данного типа задачах встречается ошибка при использовании логического оператора: вместо оператора «и» (`&&`) используется оператор «или» (`||`): `x>0 || y>1/x`. Это неверно, т. к. если хотя бы одно условие выполняется, то все логическое выражение будет являться истиной, что не соответствует определенной в задаче области.

2. Работа с длинными числами

Необходимо считать с консоли длинное число и посмотреть, делится ли оно на какое-либо число.

Рассмотрим фрагмент программного кода с инициализацией и считыванием числа.

```
...
int c,x;
c=getchar();
...
```

В переменную с помощью `getchar()` считывается символ с клавиатуры. Если на клавиатуре ввести значение 3, то в `c` запишется ASCII-символ тройки, т. е. число 51, т. к. ASCII-символ нуля равен 48. Если смотреть, к примеру, делится ли число на 10, и рассматривать вместо введенного с клавиатуры значения его ASCII-код, то можно получить ошибочный результат. Для корректности решения необходимо перейти от ASCII-кодов к цифрам по формуле: `x = c-'0'`.

2.1. Сложение очень больших чисел

Очень большие числа — это те числа, которые не поместятся ни в один из стандартных типов данных. Техника сложения таких чисел — сложение столбиком.

Первый способ такого сложения: рассмотреть два массива чисел, соответствующих цифрам каждого числа, и поэлементно складывать соответствующие цифры чисел. Как и в предыдущей задаче, необходимо не забывать о переводе считанных цифр из ASCII-кодировки в обычную.

Выводить полученное число можно различными способами:

```
putchar(x+'0');
printf("%c",x+'0');
printf("%d",x);
```

Необходимо смотреть на ограничение по длине — позволительно ли хранить каждую цифру в отдельной ячейке массива. В крайнем случае заводится массив не из элементов типа `int`, а из элементов типа `char`. К примеру, `char a[100]`. Однако данный метод зачастую бывает ошибочным, поэтому самым оптимальным решением в данном случае является хранение в ячейках массива типа `int` сразу некоторое количество цифр, вплоть до миллиона.

3. Динамическое программирование

В данном подходе каждая подзадача разбивается на малые подзадачи, которые решаются последовательно.

3.1. Задача про черепаху

Имеем поле 3×4, в каждой ячейке которого находится определенное количество кочанов капусты (см. рис. 3.2).

Черепаха выползает из верхнего угла. Ходить черепаха может только вниз и вправо, и закончит свое движение в правом нижнем углу. Вопрос — если черепаха умная и проползет по тому пути, где она съест больше всего кочанов, то сколько она их в итоге съест?

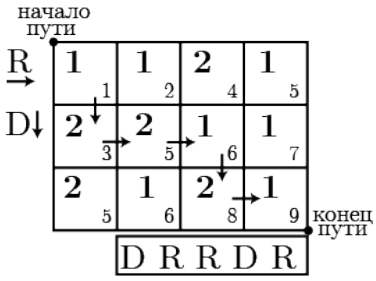


Рис. 3.2

Рассмотрим путь черепахи (см. рис. 3.2). В нижнем углу каждой ячейки записано суммарное количество кочанов, которое черепаха съест, если будет ползти по определенному пути. Перебрав все пути и найдя максимальное количество съеденных кочанов, можно восстановить оптимальным путь обратными действиями — двигаясь только влево и вверх. Для этого достаточно одного массива, в который будет записываться движение черепахи. Оптимальный путь для данного случая — D-R-R-D-R (см. рис. 3.2). Размер пути и, соответственно, размер используемого массива равен 5, т. е. полупериметру прямоугольника за вычетом 2. Для общего случая, с полем размера $N \times M$, длина пути находится аналогично.

Если вдруг задача чуть усложнится, и некоторые ячейки из правой части поля отсутствуют, то в таком случае необходимо рассмотреть несколько вариантов с разными конечными ячейками. Однако, чтобы использовать для решения задачи единый код, а не разбивать его на несколько частей, используем небольшую хитрость. Вокруг основного поля создадим ячейки, заполненные нулями. Теперь для каждой ячейки, как и в предыдущей задаче, можно использовать переходы вниз и налево, поскольку поля сверху и справа всегда будут заполнены (см. рис. 3.3).

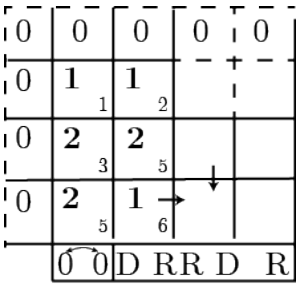


Рис. 3.3

3.2. Калькулятор с восстановлением ответа

Рассмотрим калькулятор, выполняющий только три действия: $+1$, $\times 3$, $/2$. Цель работы калькулятора — получение числа `n` из исходной единицы.

Рассмотрим процесс работы калькулятора (см. рис. 3.4).

Все действия записываются в массив. Единицу получаем нулем действий. Минимальное количество действий для получения определенной цифры записано внизу ячейки

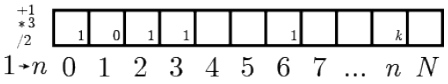


Рис. 3.4

массива, соответствующей данной цифре. Из единицы можно получить двойку и тройку +1 действием. Из двойки можно получить тройку еще +1 действием, однако она уже получается 1 действием, следовательно, этот вариант выбрасывается — количество действий не является минимально возможным. Из двойки можно получить 6 двумя действиями; это количество является важным, т. к. оно — минимальное. Аналогично ищем количество действий для получения всех чисел, пока не дойдем до числа `n`. Когда `n` достигается первый раз с числом действий `k`, то `k` и будет являться ответом к задаче.

Проблема в данной задаче может заключаться в том, что возможно сначала будет удобнее выполнить действие умножения, а затем — деления.

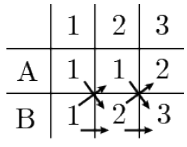
В итоге может случиться выход за границы массива — в таких случаях лучше использовать расширенный массив размером `N`, превосходящим `n`. Если рассматриваемое число достаточно велико, то возможен выход за границы рассматриваемого типа данных. При использовании `char` (возможные достигаемые значения — от −128 до 127) можно перейти к `unsigned char` (возможные достигаемые значения — от 0 до 255).

Также возможен вопрос о выводе на печать последовательности выполненных калькулятором действий. Для этого необходимо эту цепочку действий раскрутить в обратном направлении (т. е. действию сложения поставить в соответствие вычитание, действию умножения — деление) и пройти путь от `n` до 1.

3.3. Взрывоопасность

При переработке радиоактивных материалов образуются отходы двух видов — особо опасные (тип A) и неопасные (тип B). Для их хранения используются одинаковые контейнеры. После помещения отходов в контейнеры последние укладываются вертикальной стопкой. Стопка считается взрывоопасной, если в ней подряд расположено более одного контейнера типа A. Для заданного количества контейнеров N определить число безопасных стопок.

Из условия можно сделать вывод о том, что можно ставить подряд контейнеры A и B, контейнеры A и A ставить подряд нельзя. Рассмотрим постановку контейнеров в зависимости от высоты стопок (см. рис. 3.5). Строка A соответствует стопкам с основанием A, строка B — стопкам с основанием B.



... A|B
... B|A
... B|B

Рис. 3.5

В 1 столбце, со стопками высотой 1, может быть только по одному варианту расстановки. В стопку высотой 2 с основанием B можно поставить и A, и B, с основанием A — только B. Далее, стопки высотой 3 с основанием A могут выглядеть следующим образом: $A-B-A$ и $A-B-B$. Если продолжать подсчеты, то получится, что в данном случае рассматриваются 2 ряда чисел Фибоначчи (см. рис. 3.6).

	1	2	3	4	5	6
A	1	1	2	3	5	8
B	1	2	3	5	8	13

Рис. 3.6

Реализация поиска значения в ряде Фибоначчи технически довольно проста.

Рассмотрим усложнение задачи — добавление контейнера вида C, который, как и контейнер B, не является взрывоопасным. Проанализируем аналогичную первому случаю таблицу размещения контейнеров (см. рис. 3.7).

	3	8	22
	1	2	3
A	1	2	6
B	1	3	8
C	1	3	8

Рис. 3.7

С первым столбцом все очевидно. Далее, для стопок высоты 2 получаем то, что количество возможных стопок с основанием A — это сумма возможных комбинаций стопок с основаниями B и C высоты 1, т. е. 2. Количество стопок высоты 2, оканчивающихся на B — это сумма количеств возможных комбинаций стопок всех трех оснований высоты 1. Такое же количество возможных комбинаций соответствует также и стопкам с основанием C. Аналогичная итерационная система подсчета будет действовать и для стопок большей высоты. Подсчитаем сумму по всем столбцам полученной таблицы. 1 столбец — сумма равна 3, 2 столбец — сумма равна 8, 3 столбец — сумма равна 22.

Если в задаче стоит вопрос о нахождении данной суммы комбинаций для стопок, к примеру, высотой 10, то будем использовать 3 массива — A, B и C. Заполняем данные массивы полученными итерационными формулами и находим искомое значение.

3.4. Последовательность из 0 и 1

Необходимо найти количество возможных последовательностей определенной длины, составленных из 0 и 1, в которых нет трех единиц подряд.

Задача оказывается схожей с предыдущей задачей про контейнеры, только здесь запрещенной последовательностью является не AA, а 111. В качестве оснований, аналогично типам контейнеров, будут рассматриваться двухзначные комбинации, состоящие из 0 и 1: 00, 01, 10, 11. Рассмотрим таблицу размещения значений (см. рис. 3.8).

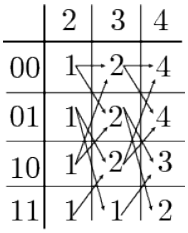


Рис. 3.8

В данной таблице названия столбцов соответствуют длине рассматриваемой последовательности. Для всех длин последовательности, равных 1, существует по одной возможной расстановке. Для второго столбца для всех рассматриваемых оснований возможно по 2 расстановки, кроме последнего, т. к. в таком случае 0 еще возможно добавить в комбинацию, а единицу уже нет — сталкиваемся с ограничением, прописанным в условии. Далее можно получить следующую закономерность — для каждого последующего столбца значение в ячейке соответствует сумме значения в соответствующей и следующей за ней ячейках предыдущего столбца. Соответственно, метод нахождения количества возможных последовательностей определенной длины сводится к методу, использующемуся в предыдущей задаче про контейнеры.

3.5. Последовательность из 0 и 1 длины `N`

Требуется вывести все последовательности из 0 и 1 длины `N`.

Рассмотрим массив `a` длины 3. Программный код для решения задачи будет выглядеть следующим образом.

Listing 3.2: Функция присваивания `set` для вывода всех последовательностей из 0 и 1 длины 3.

```
int N=3;
int a[N];
void set(int i) {
        if(i==N) {
                printf(a);
                return;
        }
        a[i]=0;
        set(i+1);
        a[i]=1;
        set(i+1);
}
```

Задача функции присваивания заключается в том, что на `i`-м месте в массиве ставится либо 0, либо 1 и помещается на `(i+1)`-е место. Получается рекурсия, и для ее использования необходимо начальное значение. Для этого рассмотрим функцию `main` с вызовом результата `set`, который и будет являться начальным значением.

Listing 3.3: Функция `main` для корректного использования функции `set`.

```
int main() {
        set(0);
        return 0;
}
```

Рассмотрим работу функции `set` для данного случая с помощью дерева вызова.

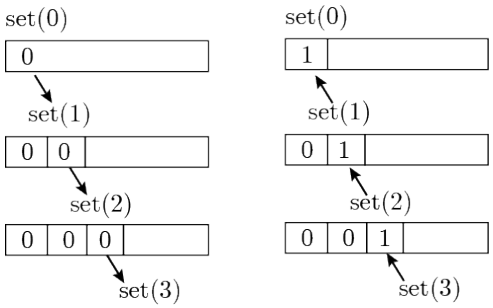


Рис. 3.9

Функция останавливается после `set(2)` и выводит полученный массив на печать. Далее функция будет продолжать печатать все возможные комбинации с 0 и 1 длины 3.

Между прочим, функцию `set` можно переписать способом, представленным в нижеприведенном программном коде. В таком случае формирование массива будет начинаться не с нулевого элемента, а с `N-1`-го, а результат останется прежним.

Listing 3.4: Альтернативный способ записи функции `set`.

```
void set(int i) {
        if(i<0) {
                printf(a);
                return;
        }
        a[i]=0;
        set(i-1);
        a[i]=1;
        set(i-1);
}
int main() {
        set(2);
        return 0;
}
```

Далее рассмотрим небольшое усложнение данной задачи — при выводе последовательностей нельзя использовать те, в которых используется 2 единицы подряд. Программный код функции в данном случае немного изменится.

Listing 3.5: Альтернативный способ записи функции `set`.

```
void set(int i) {
        if(i==N) {
                printf(a);
```

```
                return;
            }
            a[i]=0;
            set(i+1);
            if(a[i-1]==0) {
                a[i]=1;
                set(i+1);
            }
    }
```