Содержание

1 Ввод-вывод

1.1 Python

1.1.1 Файловый ввод-вывод

```
fin = open("problem.in", "r")
fout = open("problem.out", "w")

# fin.readline()
# fout.write()
```

Или можно написать так:

```
import sys
sys.stdin = open("problem.in", "r")
sys.stdout = open("problem.out", "w")
```

После этого все стандартные способы типа `input`/`print` будут работать с файлами (но они медленнее, см. 2.2).

1.2 C++

Для ввода/вывода обычно используют `scanf`/`printf` (они работают быстро) или `cin`/`cout`. По умолчанию `cin`/`cout` может тормозить. Чтобы этого не происходило, нужно:

1. Написать в начале программы:

```
ios_base::sync_with_stdio(false);
cin.tie(NULL);
```

Обратите внимание, что после этого нельзя перемешивать в одной программе оба метода — и `scanf`/`printf`, и `cin`/`cout` — они не будут синхронизироваться.

2. Не использовать `endl` — он выполняет `flush`, то есть сброс буфера вывода. Вместо него можно использовать `'\n'` или `"\n"`.

1.2.1 Файловый ввод-вывод

Напишите в начале программы следующие строки и далее работайте как с `stdin`/`stdout`:

```
assert(freopen("problem.in", "r", stdin));
assert(freopen("problem.out", "w", stdout));
```

2 Ускорение программ на Python

2.1 CPython и PyPy

Имеет смысл пробовать посылать программу как под Python 3 (интерпретатор CPython), так и под PyPy 3. На PyPy 3 программы обычно работают быстрее.

2.2 Файловый ввод-вывод

Функции `input` и `print` работают медленнее, чем методы работы с файлами. Используйте файловый ввод-вывод (методы `read`, `readline`, `readlines`, `write`).

Для чтения данных со стандартного ввода можно использовать `sys.stdin.readline()` (а также `read()` и `readlines()`), для вывода на стандартный вывод можно использовать `sys.stdout().write()`.

Если вы уже написали код с использованием `input` и `print`, а хотите считывать и выводить быстро, но не хотите везде заменять ввод/вывод, можно в начале программы написать так:

```
input = sys.stdin.readline
print = lambda x: sys.stdout.write(str(x) + '\n')
```

Зачастую этих двух советов уже бывает достаточно. Но если нет, то вот ещё несколько.

2.3 Используйте `map` для ввода-вывода чисел

Считывание списка чисел, записанных через пробел:

```
A = list(map(int, sys.stdin.readline().split()))
```

Слева от присваивания можно поставить кортеж из нескольких переменных:

```
a, b, c = map(int, sys.stdin.readline().split())
```

Вывод списка чисел через пробел:

```
sys.stdout.write(' '.join(map(str, A)))
```

2.4 Используйте локальные переменные

Локальные переменные работают быстрее глобальных, поэтому весь код должен находиться внутри функции. Даже если у вас короткая программа, сделайте из неё функцию:

```
def main():
  # ...

main()
```

2.5 Заранее создавайте массив нужного размера

Вот такой код считывания:

```
n = int(sys.stdin.readline())
A = [0] * n
for i in range(n):
    A[i] = int(sys.stdin.readline())
```

работает быстрее, чем создание пустого списка и добавление элементов по одному в конец при помощи `append`.

2.6 Добавление в конец

Если `A` — список, то `A = A + [x]` работает за $\mathcal{O}(|A|)$. Нужно писать `A += [x]` или `A.append(x)`.

Строки в Python иммутабельные (неизменяемые), поэтому `S += "x"` эквивалентно `S = S + "x"`. В зависимости от реализации интерпретатора это может работать за $\mathcal{O}(1)$ или за $\mathcal{O}(|S|)$.

3 Увеличение стека

Когда вы пишете программу, в которой возникает глубокая рекурсия, у вас могут возникнуть проблемы с превышением размера стека. Обычно такие проблемы выливаются в ошибку выполнения (Runtime Error).

К сожалению, способы увеличения ограничения на стек зачастую зависят и от языка программирования, и от операционной системы, и от компилятора/интерпретатора.

В языке C++ при компиляции под MinGW достаточно указать флаг `-Wl,--stack=268435456` (стек размером в 256 мебибайт). В частности, при посылке на Codeforces всё уже сделано за вас.

С языком Python придётся повозиться. И хотя интерпретатор PyPy обычно «лучше» стандартного интерпретатора CPython в плане быстродействия, в этой ситуации с ним возникает больше проблем.

Примеры кода для CPython и PyPy см. ниже.

Не пишите функцию `gcd` таким образом! Здесь он специально написан неоптимально (с вычитанием) и рекурсивно. А ещё есть `math.gcd`.

3.1 CPython 3

```
def gcd(n, m):
  if n == 0 or m == 0:
    return n + m
  return gcd(min(n, m), max(n, m) - min(n, m))

def main():
  print(gcd(100000, 2))

#####

import sys, threading
sys.setrecursionlimit(1 << 30)
threading.stack_size(1 << 27)

main_thread = threading.Thread(target=main)
main_thread.start()
main_thread.join()
```

3.2 PyPy 3

```
# Взято отсюда, немного усовершенствовано
from _continuation import continulet

def invoke(_, callable, arg):
  return callable(*arg)

def bootstrap(c):
  callable, *arg = c.switch()
  while True:
    to = continulet(invoke, callable, arg)
    callable, *arg = c.switch(to=to)
cont = continulet(bootstrap)
cont.switch()

#####

def gcd(n, m):
  if n == 0 or m == 0:
    return n + m
  # Рекомендуется не каждый раз пользоваться cont.switch, а, например, каждый 200-й уровень
  # А остальные 199 - обычной рекурсией. Но можно и всё время:
  return cont.switch((gcd, min(n, m), max(n, m) - min(n, m)))

def main():
  print(gcd(100000, 2))

main()
```