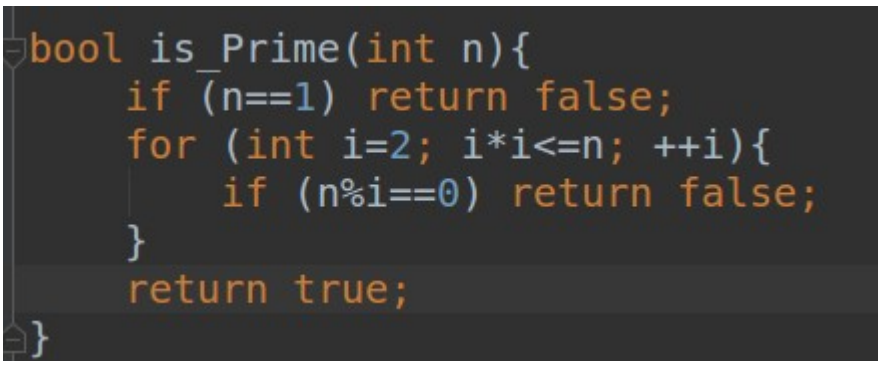
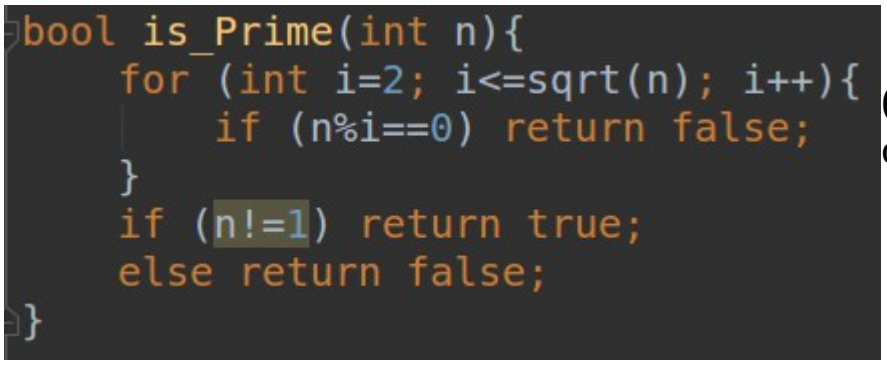
Анализ характеристик программы.

Данный алгоритм опирается на количество и порядок операторов в программе. Количественный состав операторов, если программа списана, не должен существенно отличатся от исходной, и, сравнивая только количество, нам будет не важно, если блоки кода были поменяны местами. При исследовании порядка следования операторов в программе, мы уже не сможем отследить перестановку блоков, зато сможем найти схожие части программы. При такой проверке мы также не будем зависеть от имен переменных и функций. Для примера возьмем простую программу, в которой присутствует функция проверки числа на простоту.



Программа 1



Программа 2 (считаем ее списанной)

1. Анализ частоты появления операторов.

Наши сравниваемые программы прошли процедуры нормализации и токенизации. Теперь у нас есть упорядоченные множества операторов.

(	)	{	if	(	==	)	return	;	for
(	=	;	*	<=	;	++i	)	{	if
(	%	==	)	return	;	}	return	;	}

(	)	{	for	(	=	;	<=	(	)	;
i++	)	{	if	(	%	==	)	return	;	}
if	(	!=	)	return	;	else	return	;	}	

Теперь для каждой программы пройдемся по множеству операторов и запишем для каждого в ассоциативный массив их количество в программе.

(	)	{	}	if	==	return	;	for	=	*	<=	++i	%
4	4	2	2	2	2	3	5	1	1	1	1	1	1

(	)	{	}	for	=	;	<=	i++	if	%	==	return	!=	else
5	5	2	2	1	1	5	1	1	2	1	1	3	1	1

Далее будем рассчитывать процент схожести кода. Для каждого оператора из более короткой последовательности будем искать в другой последовательности такой же.

$\frac{A}{B} * 100$ = процент схожести

1 вариант: показательным будем считать отношение всех количественно совпавших типов операторов (A) к максимальному количеству типов операторов (B). В данном примере — $\frac{9}{14} * 100$ = 64,2%

2 вариант: отношение всех количественно совпавших типов операторов более, чем на 85%, (A) к максимальному количеству типов операторов (B). Здесь, из-за маленького количества операторов, такой же процент — $\frac{9}{14} * 100$ = 64,2%

3 вариант: отношение всех совпавших операторов (A) к числу всех операторов более длинной последовательности (B). Здесь — $\frac{27}{32} * 100$ = 84,4%

2. Взаимная корреляция программ.

Если программы имеют одинаковое количество операторов, то мы один раз проходимся по 2м последовательностям операторов и увеличиваем счетчик, если операторы совпали. В данной программе у

нас различная длина последовательностей. Мы будем ориентироваться на меньшую по длине и будем сравнивать с большей столько раз, на сколько эти последовательности отличаются. Сначала первый оператор короткой последовательности будет сравниваться с первым длинной. Проходимся по двум последовательностям и смотрим на совпадения.

1	(	)	{	if	(	==	)	return	;	for
2	(	)	{	for	(	=	;	<=	(	)
1	(	=	;	*	<=	;	++i	)	{	if
2	;	i++	)	{	if	(	%	==	)	return
1	(	%	==	)	return	;	}	return	;	}
2	;	}	if	(	!=	)	return	;	else	return

Счетчик совпавших операторов:4.
Теперь будем сравнивать две последовательности так, что 1ый оператор короткой сравнивается со 2ым длинной.

1	(	)	{	if	(	==	)	return	;	for
2	)	{	for	(	=	;	<=	(	)	;
1	(	=	;	*	<=	;	++i	)	{	if
2	i++	)	{	if	(	%	==	)	return	;
1	(	%	==	)	return	;	}	return	;	}
2	}	if	(	!=	)	return	;	else	return	;

Счетчик совпавших операторов:1.

1	(	)	{	if	(	==	)	return	;	for
2	{	for	(	=	;	<=	(	)	;	i++

1	(	=	;	*	<=	;	++i	)	{	if
2	)	{	if	(	%	==	)	return	;	}
1	(	%	==	)	return	;	}	return	;	}
2	if	(	!=	)	return	;	else	return	;	}

Счетчик совпавших операторов:7.

Сдвиг на:	0	1	2
Количество операторов	4	1	7

В данном случае процент схожести кода будем рассчитывать как отношение максимального количества совпадений (A) к максимальному количеству операторов в сравниваемых множествах (B). В нашей программе это $\frac{7}{32} * 100$ = 21,8%. Также, при проценте схожести более 50%, мы будем запоминать сдвиг последовательности, чтобы можно было вручную оценить, списан ли код.

Наши анализируемые куски кода оказались довольно различными, ввиду своей маленькой величины, и, так как в списанной программе были поменяны местами некоторые условия, следовательно, поменялись местами и операторы, поэтому корреляция двух программ оказалась сравнительно небольшой. Зато, так как сами операторы практически не поменялись, процент количественного совпадения высок.

Итак, плюсами данного алгоритма мы можем назвать независимость от идентификаторов и удобство при применении дальшейших алгоритмов проверки на антиплагиат либо анализа вручную. 1 часть нашего алгоритма также не зависит от расположения блоков кода, их вложенности. 2 часть может выявить в списанной программе скопированный из оригинальной кусок кода, но уже будет зависеть от расположения блоков кода.