

Глава 1

Примеры работы со стеком и очередью

Стек — структура, которая достаточно часто используется в различных алгоритмах. В принципе, рекурсия и стек по своей структуре одно и то же. Рассмотрим несколько задач, где применяется стек.

1.1 Проверка корректности расставленных скобок

Так как вычисление математического выражение идет от «обратного» (т. е. в выражении $a * (b * (a + c))$ сначала будет подсчитана операция сложения, потом второго умножения и, наконец, первого умножения), для описания такой структуры идеально подходит стек.

Пример 1.1. УСЛОВИЕ: Дано математическое выражение. Проверить корректность расстановки скобок.

РЕШЕНИЕ: Скобок должно быть одинаковое количество и открывающая скобка должна обязательно быть раньше закрывающей. Так как в стек записываются данные в обратном порядке, то закрывающая скобка будет соответствовать открывающей, находящейся на вершине стека.

Алгоритм 1: Проверка корректности расстановки скобок в выражении

Вход: *str* — строка, содержащая скобки, буквы, цифры и знаки операций

Выход: Правильно стоят скобки или нет. Если неправильно, то каких больше

начало алгоритма

- создаем стек и инициализируем его;

цикл пока не дошли до конца строки выполнять

- пропускаем все символы;

- открывающую скобку записываем в стек;

если закрывающая скобка то

- если стек пуст то**

- неправильно, закрывающая скобка встречается раньше открывающей;

- завершаем работу;

- иначе**

- извлекаем из стека открывающую скобку;

если стек пуст то

- правильно;

иначе

- открывающих скобок больше, чем закрывающих;

конец алгоритма

Например, выражение $5 + 4 * (2 + 3 * (4 - 2) + (5 * (4 - 3) - 3) + 1) + 2$

Для наглядности будем удалять просмотренные символы (в программе этого делать необязательно).

текущий символ	стек	остаток строки
5		+4*(2+3*(4-2)+(5*(4-3)-3)+1)+2
+		4*(2+3*(4-2)+(5*(4-3)-3)+1)+2
4		*(2+3*(4-2)+(5*(4-3)-3)+1)+2
*		(2+3*(4-2)+(5*(4-3)-3)+1)+2
((2+3*(4-2)+(5*(4-3)-3)+1)+2
2	(+3*(4-2)+(5*(4-3)-3)+1)+2
+	(3*(4-2)+(5*(4-3)-3)+1)+2
3	(*(4-2)+(5*(4-3)-3)+1)+2
*	((4-2)+(5*(4-3)-3)+1)+2
(((4-2)+(5*(4-3)-3)+1)+2
4	((-2)+(5*(4-3)-3)+1)+2
-	((2)+(5*(4-3)-3)+1)+2
2	(()+(5*(4-3)-3)+1)+2
)	(+(5*(4-3)-3)+1)+2
+	((5*(4-3)-3)+1)+2
(((5*(4-3)-3)+1)+2
5	((*(4-3)-3)+1)+2
*	(((4-3)-3)+1)+2
((((4-3)-3)+1)+2
4	((-3)-3)+1)+2
-	((3)-3)+1)+2
3	(() -3)+1)+2
)	(-3)+1)+2
-	(3)+1)+2
3	() +1)+2
)	(+1)+2
+	(1)+2
1	() +2
)		+2
+		2
2		

Стек пуст, значит скобки расставлены корректно.

□

1.2 Постфиксная запись выражения

Допустим необходимо вычислить математическое выражение. Как известно, все операции имеют приоритет, поэтому для изменения порядка вычисления операций используются скобки. Но вычисления со скобками не позволяют воспользоваться обычным калькулятором. Запись выражения, где знак операции стоит между операндами, называется *инфиксной*.

В 20-ых годах XX века польским математиком Лукасевичем была предложена идея другого формата записи выражений, позволяющая избавиться от скобок. Это префиксная и постфиксная запись (часто называемые польская нотация и обратная польская нотация).

В префиксной форме записи знак операций стоит перед операндами. Так как арифметические операции всегда бинарные, то двояко выражение подсчитать невозможно, и скобки не нужны. Префиксная форма записи используется в ряде компиляторов и в языке LISP. В постфиксной записи знак операции расположен после операндов. Также скобки не нужны. Постфиксная форма записи используется в стековых языках программирования, таких, как PostScript.

Например, выражение:

$$5 + (3 + (2 + 4 * (3 - 1) + 2) / 4 + 1) * 4.$$

В префиксной форме будет выглядеть следующим образом:

$$+5 * + + 3 / + + 2 * 4 - 3 1 2 4 1 4.$$

В постфиксной форме будет выглядеть следующим образом:

$$5 3 2 4 3 1 - * + 2 + 4 / + 1 + 4 * +.$$

Видно, что никаких скобок нет. Неудобство заключается в том, что использовать можно только однозначные числа. Для других данных придется использовать какие-нибудь переменные.

Построить выражение в инфиксной записи проще всего через дерево, поэтому не будем пока рассматривать эту форму записи.

Алгоритм построения постфиксной записи прост. Предполагаем, что инфиксная запись (однозначные числа или символы, количество открывающих и закрывающих скобок совпадает, после и перед скобками — операнд, между операндами — знак операции) корректна.

Сначала определяем приоритет знака операции (функция `prior`):

- 1** — открывающая скобка;
- 2** — операции сложения и вычитания;
- 3** — операции умножения и деления.

Создаем и инициализируем стек, куда будем записывать только открывающую скобку и знаки операций. Создаем строку, куда будем записывать результат.

Далее в зависимости от значения текущего символа выполняем следующие действия:

1. Цифры или переменные записываем в результирующую строку.
2. Открывающую скобку записываем в результирующую строку.
3. Знак операции — действия зависят от приоритета:
 - (a) Если стек пуст, записываем знак операции в стек.
 - (b) Если приоритет текущего символа выше, чем приоритет элемента на вершине стека, то записываем текущий символ в стек.
 - (c) Если приоритет текущего символа меньше или равен приоритету элемента на вершине стека, то извлекаем из стека элементы и записываем их в результирующую строку. Продолжаем процесс до тех пор, пока приоритет текущего элемента не станет выше приоритета вершины стека (или стек не опустеет). После этого записываем текущий элемент в стек.
4. В случае закрывающей скобки извлекаем элементы из стека и записываем в результирующую строку до тех пор, пока не встретим открывающую скобку. Извлекаем открывающую скобку из стека. Открывающая и закрывающая скобки нигде не сохраняются.
5. Все элементы стека, которые остались после исследования всей строки, извлекаем из стека и записываем в строку.

Например, $a * (b + c * d) + e$.

Стек `znak = NULL`

Строка `res = ""`

- a — символ. Записываем в строку `res = a`.
- $*$ — знак операции. Стек пуст, записываем $*$ в стек: `znak = [*]`.
- $($ записываем в стек `znak = [(, *)]`.
- b — символ. Записываем в строку `res = ab`.
- $+$ — знак операции. Его приоритет — 2. Приоритет вершины стека $($ равен 1, следовательно, записываем $+$ в стек `znak = [+ , (, *)]`.
- c — символ. Записываем в строку `res = abc`.
- $*$ — знак операции. Его приоритет — 3. Приоритет вершины стека $(+)$ равен 2, следовательно, записываем $*$ в стек `znak = [* , + , (, *)]`.
- d — символ. Записываем в строку `res = abcd`.
- $)$. Извлекаем из стека последовательно все элементы до открывающей скобки и записываем их в результирующую строку `res = abcd*+`. Открывающую скобку просто извлекаем из стека. Результирующий стек `znak = [*]`.
- $+$ — знак операции. Его приоритет — 2. Приоритет вершины стека $(*)$ — 3. Извлекаем единственный элемент из стека и записываем в результирующую строку `res = abcd**+`. Записываем $+$ в стек `znak = [+]`.
- e — символ. Записываем в строку `res = abcd**+e`.
- Строка закончилась, поэтому извлекаем из стека оставшиеся элементы и записываем в строку `res = abcd***+e`.

Алгоритм 2: Перевод в постфиксную запись**Вход:** *str* — корректная строка (выражение в инфиксной форме)**Выход:** строка (выражение в постфиксной форме)**начало алгоритма**

- создаем стек и инициализируем его;
- создаем результирующую пустую строку;

цикл пока не дошли до конца строки выполнять

- операнд записываем в результирующую строку;
- открывающую скобку записываем в стек;

если стек пуст то

- └ · знак операции записываем в стек;

если закрывающая скобка то**цикл пока вершина стека не открывающая скобка выполнять**

- └ · извлекаем из стека вершину и записываем ее в результирующую строку;
- └ · извлекаем закрывающую скобку;

если приоритет знака операции > приоритета элемента, находящегося на вершине стека то

- └ · записываем знак операции в стек;

иначе**цикл пока приоритет знака операции \leq приоритета элемента, находящегося на вершине стека выполнять**

- └ · извлекаем из стека вершину и записываем ее в результирующую строку;
- └ · записываем знак операции в стек;

цикл пока стек не пуст выполнять

- └ · извлекаем из стека вершину и записываем ее в результирующую строку;

конец алгоритма

В примере для удобства будем удалять рассмотренные символы (в программе этого делать не обязательно.)

Рассмотрим приведенное выше выражение.

остаток строки	стек	приоритет	символ	приоритет	рез. стек	результат
$+(3+(2+4*(3-1)+2)/4+1)*4$			5			5
$(3+(2+4*(3-1)+2)/4+1)*4$			+	2	+	5
$3+(2+4*(3-1)+2)/4+1)*4$	+	2	(1	(+	5
$+(2+4*(3-1)+2)/4+1)*4$	(+	1	3		(+	53
$(2+4*(3-1)+2)/4+1)*4$	(+	1	+	2	+(+	53
$2+4*(3-1)+2)/4+1)*4$	+(+	2	(1	+(+	53
$+4*(3-1)+2)/4+1)*4$	+(+	1	2		+(+	532
$4*(3-1)+2)/4+1)*4$	+(+	1	+	2	+(+	532
$*(3-1)+2)/4+1)*4$	+(+	2	4		+(+	5324
$(3-1)+2)/4+1)*4$	+(+	2	*	3	*+(+	5324
$3-1)+2)/4+1)*4$	*+(+	3	(1	(*+(+	5324
$-1)+2)/4+1)*4$	(*+(+	1	3		(*+(+	53243
$1)+2)/4+1)*4$	(*+(+	1	-	2	-(*(+(+	53243
$) + 2)/4 + 1) * 4$	-(*(+(+	2	1		-(*(+(+	532431
$+ 2)/4 + 1) * 4$	-(*(+(+	2)		(*+(+	532431-
$+ 2)/4 + 1) * 4$	(*+(+	1)		*+(+	532431-
$2)/4 + 1) * 4$	*+(+	3	+	2	+(+	532431-*
$2)/4 + 1) * 4$	+(+	2	+	2	(+	532431-*+
$2)/4 + 1) * 4$	(+	1	+	2	+(+	532431-*+
$) / 4 + 1) * 4$	+(+	2	2		+(+	532431-*+2
$/ 4 + 1) * 4$	+(+	2)		(+	532431-*+2+
$/ 4 + 1) * 4$	(+	1)		+	532431-*+2+
$4 + 1) * 4$	+	2	/	3	/+	532431-*+2+
$+ 1) * 4$	/+	3	4		/+	532431-*+2+4
$1) * 4$	/+	3	+	2	+	532431-*+2+4/
$1) * 4$	+	2	+	2	(532431-*+2+4/+
$1) * 4$	(1	+	2	+	532431-*+2+4/+
$) * 4$	+	2	1		+	532431-*+2+4/+1
$* 4$	+	2)		(532431-*+2+4/+1+
$* 4$	(1)		+	532431-*+2+4/+1+
4	+	2	*	3	*+	532431-*+2+4/+1+
	*+	3	4		*+	532431-*+2+4/+1+4
	+	3			+	532431-+2+4/+1+4*
	+	2				532431-*+2+4/+1+4*+
						532431-*+2+4/+1+4*+

В примере рассматриваются только однозначные числа, поэтому можно обойтись строкой. Если необходимо использовать другие числа (отрицательные, двузначные и т. д.), можно вместо результирующей строки использовать очередь.

1.3 Вычисление значения выражения, записанного в постфиксной записи

Для вычисления значения выражения, записанного в постфиксной форме, будем использовать стек, но записывать туда только цифры или символы.

Если встретили знак операции, то извлекаем два первых элемента стека, производим с ними указанную операцию и результат записываем обратно в стек. В стек данные записываются в обратном порядке (было $ab-$, в стек будет записано b, a , а результатом будет $a - b$), поэтому об этом надо помнить в случае деления и вычитания.

Например, дано выражение: $3482 / + * 2 -$.

Стек `res = NULL`

- 3 записываем в стек: `res = [3]`.
- 4 записываем в стек: `res = [4, 3]`.
- 8 записываем в стек: `res = [8, 4, 3]`.
- 2 записываем в стек: `res = [2, 8, 4, 3]`.
- $/$ — знак операции. Извлекаем из стека 2 и 8. Стек имеет вид `res = [4, 3]`.. Так как запись была в обратном порядке, то надо $8/2 = 4$. Результат записываем в стек `res = [4, 4, 3]`.
- $+$ — знак операции. Извлекаем из стека 4 и 4. Стек имеет вид `res = [3]`. Складываем, результат записываем в стек `res = [8, 3]`.

- * — знак операции. Извлекаем из стека 8 и 3. Стек пуст. Умножаем, результат записываем в стек `res = [24]`.
- 2 записываем в стек `res = [2, 24]`.
- — — знак операции. Извлекаем из стека 2 и 24. Стек пуст. Так как запись была в обратном порядке, то надо $24 - 2 = 22$. Результат записываем в стек `res = [22]`.
- Строка закончена. Выводим результат — 22.

Алгоритм 3: Вычисление значения выражения

Вход: *str* — корректная строка (выражение в постфиксной форме)

Выход: число — вычисленное значение

начало алгоритма

- создаем стек и инициализируем его;
- цикл пока не дошли до конца строки выполнять**
 - если текущий символ цифра или переменная то**
 - записываем в стек;
 - иначе**
 - извлекаем 2 первых символа *b* и *a*;
 - вычисляем *a op b*, где *op* — текущий символ (один из + − */);
 - записываем результат в стек;
- в стеке содержится один элемент. Выводим его на экран;

конец алгоритма

Рассмотрим приведенный выше пример. Опять будем удалять рассмотренные символы для наглядности.

Строка	Стек до	Символ	<i>b</i>	<i>a</i>	результат	стек после
32431-*+2+4/+1+4*+		5				5
2431-*+2+4/+1+4*+	5	3				3, 5
431-*+2+4/+1+4*+	3, 5	2				2, 3, 5
31-*+2+4/+1+4*+	2, 3, 5	4				4, 2, 3, 5
1-*+2+4/+1+4*+	4, 2, 3, 5	3				3, 4, 2, 3, 5
-*+2+4/+1+4*+	3, 4, 2, 3, 5	1				1, 3, 4, 2, 3, 5
+2+4/+1+4+	1, 3, 4, 2, 3, 5	—	1	3	2	2, 4, 2, 3, 5
+2+4/+1+4*+	2, 4, 2, 3, 5	*	2	4	8	8, 2, 3, 5
2+4/+1+4*+	8, 2, 3, 5	+	8	2	10	10, 3, 5
+4/+1+4*+	10, 3, 5	2				2, 10, 3, 5
4/+1+4*+	2, 10, 3, 5	+	2	10	12	12, 3, 5
/+1+4*+	12, 3, 5	4				4, 12, 3, 5
+1+4*+	4, 12, 3, 5	/	4	12	3	3, 3, 5
1+4*+	3, 3, 5	+	3	3	6	6, 5
+4*+	6, 5	1				1, 6, 5
4*+	1, 6, 5	+	1	6	7	7, 5
*+	7, 5	4				4, 7, 5
+	4, 7, 5	*	4	7	28	28, 5
	28, 5	+	28	5	33	33
	33					

1.4 Вычисление значения выражения, записанного в префиксной записи

Алгоритм решения похож на приведенный выше. Для решения используем стек, так как вычисления происходят из «глубины» (в обратном порядке).

1. Знаки операций записываем в стек.
2. Числа записываем в стек только если вершина стека — это знак операции.
3. Если текущий элемент — число (*a*) и вершина стека — число, извлекаем 2 верхних элемента (число (*b*) и знак операции (*op*)).

4. Выполняем операцию. Если текущая вершина стека — число, повторяем шаг 3 для результата операции, если знак операции — записываем результат операции в стек. Если стек пуст — записываем результат операции в стек.

Рассмотрим пример — $* 3 + 4 / 8 2 2$

Стек `res = NULL`

- — записываем в стек: `res = [-]`.
- * записываем в стек: `res = [*, -]`.
- 3 — число, так как вершина стека — знак операции, записываем в стек: `res = [3, *, -]`.
- + записываем в стек: `res = [+ , 3, *, -]`.
- 4 — число, так как вершина стека — знак операции, записываем в стек `res = [4, +, 3, *, -]`.
- / записываем в стек `res = [/ , 4, +, 3, *, -]`.
- 8 — число. Так как вершина стека — знак операции, записываем в стек `res = [8, /, 4, +, 3, *, -]`.
- 2 — число. Так как вершина стека — число, извлекаем из стека 8 и /. Стек имеет вид: `res = [4, +, 3, *, -]`. Вычисляем $8/2 = 4$. Так как вершина стека — число, извлекаем из стека 4 и +. Стек имеет вид: `res = [3, *, -]`. Вычисляем $4 + 4 = 8$. Так как вершина стека — число, извлекаем из стека 3 и *. Стек имеет вид: `res = [-]`. Вычисляем $8 * 3 = 24$. Так как вершина стека — знак операции, записываем 24 в стек. Стек имеет вид: `res = [24, -]`.
- 2 — число. Так как вершина стека — число, извлекаем из стека 24 и —. Стек пуст. Вычисляем $24 - 2 = 22$. Записываем в стек.
- Строка закончена. Выводим результат — 22.

Алгоритм 4: Вычисление значения выражения

Вход: *str* — корректная строка (выражение в префиксной форме)

Выход: число — вычисленное значение

начало алгоритма

```

· создаем стек и инициализируем его;
цикл пока не дошли до конца строки (или очереди) выполнять
    если текущий символ — знак операции то
        · записываем в стек;
    иначе
        если голова стека — знак операции то
            · записываем текущий символ в стек;
        иначе
            цикл пока стек не пуст и текущий символ не знак операции выполнять
                · извлекаем 2 первых символа a и op;
                · вычисляем a op b, где b — текущий символ (число);
            · записываем результат в стек;
    · в стеке содержится один элемент. Выводим его на экран;
```

конец алгоритма

Рассмотрим приведенный выше пример. Опять будем удалять рассмотренные символы для наглядности.

Строка	Стек до	Сим.	b	a	Рез.	Стек после
+5*++3/++2*4-312414		+				+
5*++3/++2*4-312414	+	5				5, +
*++3/++2*4-312414	5, +	*				*, 5, +
++3/++2*4-312414	*, 5, +	+				+, *, 5, +
+3/++2*4-312414	+, *, 5, +	+				+, +, *, 5, +
3/++2*4-312414	+, +, *, 5, +	3				3, +, +, *, 5, +
/++2*4-312414	3, +, +, *, 5, +	/				/, 3, +, +, *, 5, +
++2*4-312414	/, 3, +, +, *, 5, +	+				+, /, 3, +, +, *, 5, +
+2*4-312414	+, /, 3, +, +, *, 5, +	+				+, +, /, 3, +, +, *, 5, +
2*4-312414	+, +, /, 3, +, +, *, 5, +	2				2, +, +, /, 3, +, +, *, 5, +
*4-312414	2, +, +, /, 3, +, +, *, 5, +	*				*, 2, +, +, /, 3, +, +, *, 5, +
4-312414	*, 2, +, +, /, 3, +, +, *, 5, +	4				4, *, 2, +, +, /, 3, +, +, *, 5, +
-312414	4, *, 2, +, +, /, 3, +, +, *, 5, +	—				—, 4, *, 2, +, +, /, 3, +, +, *, 5, +
312414	—, 4, *, 2, +, +, /, 3, +, +, *, 5, +	3				3, —, 4, *, 2, +, +, /, 3, +, +, *, 5, +
12414	3, —, 4, *, 2, +, +, /, 3, +, +, *, 5, +	1	3	—	2	4, *, 2, +, +, /, 3, +, +, *, 5, +
2414	4, *, 2, +, +, /, 3, +, +, *, 5, +	2	4	*	8	2, +, +, /, 3, +, +, *, 5, +
2414	2, +, +, /, 3, +, +, *, 5, +	8	2	+	10	+, /, 3, +, +, *, 5, +
2414	+, /, 3, +, +, *, 5, +	10				10, +, /, 3, +, +, *, 5, +
2414	10, +, /, 3, +, +, *, 5, +	2	10	+	12	/, 3, +, +, *, 5, +
414	/, 3, +, +, *, 5, +	12				12, /, 3, +, +, *, 5, + /, 3, +, +, *, 5, +
14	12, /, 3, +, +, *, 5, + /	4	12	/	3	3, +, +, *, 5, +
14	3, +, +, *, 5, +	3	3	+	6	+, *, 5, +
14	+, *, 5, +	6				6, +, *, 5, +
14	6, +, *, 5, +	1	6	+	7	*, 5, +
4	*, 5, +	7				7, *, 5, +
4	7, *, 5, +	4	7	*	28	5, +
	5, +	28	5	+	33	
		33				33

1.5 Обход графа в глубину

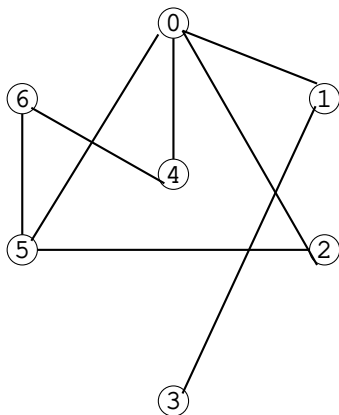
Если кратко говорить о графе, то граф — это набор вершин, соединенных ребрами. Будем пока рассматривать ситуацию, когда ребра не имеют направления, то есть, граф является неориентированным. Две вершины, соединенные ребрами, называются *смежными*.

Для описания графа можно использовать несколько способов. Пусть число вершин графа равно N .

Матрица смежности — это матрица $N \times N$, удовлетворяющая следующему свойству:

$$\begin{cases} a[i][j] = 1, \text{ если вершины } i \text{ и } j \text{ смежные} \\ a[i][j] = 0, \text{ если вершины } i \text{ и } j \text{ не смежные} \end{cases}$$

Список смежности — набор N контейнеров (любых) таких, что i -ый контейнер содержит все вершины, смежные с данной.



Например, для графа, изображенного на рисунке, матрица смежности и список смежности будут иметь следующий вид:
Матрица смежности:

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

Список смежности:

0	→ 1 → 2 → 4 → 5 → ∅
1	→ 0 → 3 → ∅
2	→ 0 → 5 → ∅
3	→ 1 → ∅
4	→ 0 → 6 → ∅
5	→ 0 → 2 → 6 → ∅
6	→ 4 → 5 → ∅

Для графа, заданного матрицей смежности, проверка на смежность двух вершин x и y означает что $a[x][y] \neq 0$.

Для графа, заданного списком смежности, проверка на смежность двух вершин x и y означает поиск вершины x в векторе $a[y]$.

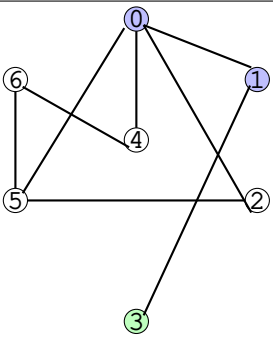
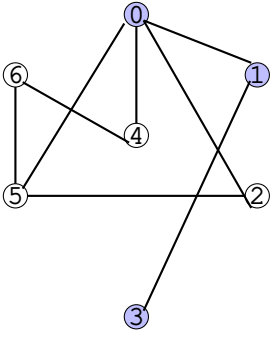
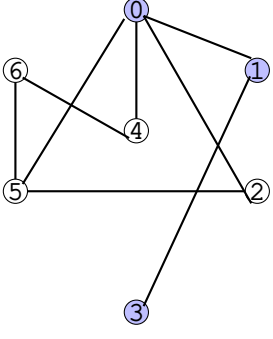
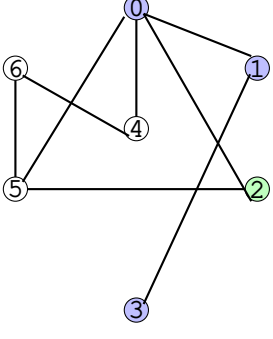
В данной главе рассмотрим только один алгоритм, связанный с графом — это обход графа. Обход графа — это посещение каждой вершины графа по одному разу. Существует два обхода — в глубину и в ширину. Результатом обхода будет последовательность посещенных вершин.

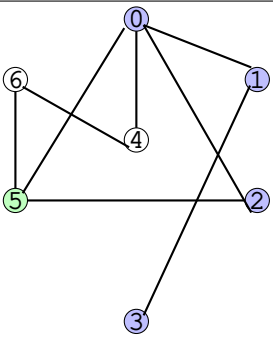
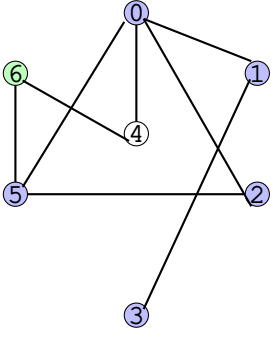
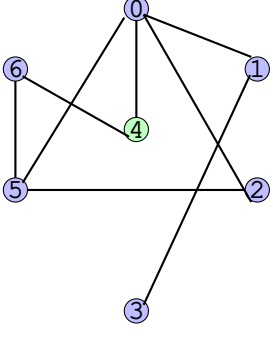
Алгоритм обхода в глубину: посетив вершину, помечаем ее меткой, что она посещена. Ищем смежную непосещенную вершину и рекурсивно вызываем обход в глубину для этой вершины. Повторяем процесс до тех пор, пока все вершины не будут посещены.

Название в глубину происходит из-за того, что идем «вглубь» графа до тех пор, пока это возможно.

Например, для заданного графа начинаем с нулевой вершины. Зеленым цветом будет отмечена текущая вершина, синим — уже посещенные.

Граф	Результат	Действие с текущей вершиной	Массив посещенных вершин	Поиск смежной вершины
	0	Помечаем 0 как посещенную	[1, 0, 0, 0, 0, 0, 0]	Смежная вершина — 1
	0, 1	Помечаем 1 как посещенную	[1, 1, 0, 0, 0, 0, 0]	Смежная вершина — 3

Граф	Результат	Действие с текущей вершиной	Массив посещенных вершин	Поиск смежной вершины
	0, 1, 3	Помечаем 3 как посещенную	[1, 1, 0, 1, 0, 0, 0]	Смежных непосещенных вершин нет. Возвращаемся на шаг назад. Ищем смежную вершину для 1.
	0, 1, 3		[1, 1, 0, 1, 0, 0, 0]	Смежных непосещенных вершин нет. Возвращаемся на шаг назад. Ищем смежную вершину для 0.
	0, 1, 3		[1, 1, 0, 1, 0, 0, 0]	Смежная вершина — 2
	0, 1, 3, 2	Помечаем 2 как посещенную	[1, 1, 1, 1, 0, 0, 0]	Смежная вершина — 5

Граф	Результат	Действие с текущей вершиной	Массив посещенных вершин	Поиск смежной вершины
	0, 1, 3, 2, 5	Помечаем 5 как посещенную	[1, 1, 1, 1, 0, 1, 0]	Смежная вершина — 6
	0, 1, 3, 2, 5, 6	Помечаем 6 как посещенную	[1, 1, 1, 1, 0, 1, 1]	Смежная вершина — 4
	0, 1, 3, 2, 5, 6, 4	Помечаем 4 как посещенную	[1, 1, 1, 1, 1, 1, 1]	Все вершины посещены. Завершаем алгоритм

В данном разделе изучаются стеки, поэтому напишем алгоритм обхода в глубину (DFS) без использования рекурсивных функций.

Алгоритм прост: Помечаем посещенную вершину (присваиваем соответствующему элементу массива значение 1) и помещаем ее в стек. Пока стек не пуст, ищем смежную вершину для головы стека. Если нашли, помещаем ее в стек, иначе извлекаем вершину из стека.

Текущая вершина	$Gr[x]$	Смежная вершина	Массив A	fl	Стек	Результат
0			[1000000]	true	0	0
0	[1, 2, 4, 5]	1	[1100000]	true	[1, 0]	0, 1
1	[0, 3]	3	[1101000]	true	[3, 1, 0]	0, 1, 3
3	[1]		[1101000]	false	[1, 0]	0, 1, 3
1	[0, 3]		[1101000]	false	[0]	0, 1, 3
0	[1, 2, 4, 5]	2	[1111000]	true	[2, 0]	0, 1, 3, 2
2	[0, 5]	5	[1111010]	true	[5, 2, 0]	0, 1, 3, 2, 5
5	[0, 2, 6]	6	[1111011]	true	[6, 5, 2, 0]	0, 1, 3, 2, 5, 6
6	[4, 5]	4	[1111111]	true	[4, 6, 5, 2, 0]	0, 1, 3, 2, 5, 6, 4
4	[0, 6]		[1111111]	false	[6, 5, 2, 0]	0, 1, 3, 2, 5, 6, 4
6	[4, 5]		[1111111]	false	[5, 2, 0]	0, 1, 3, 2, 5, 6, 4
5	[0, 2, 6]		[1111111]	false	[2, 0]	0, 1, 3, 2, 5, 6, 4
2	[0, 5]		[1111111]	false	[0]	0, 1, 3, 2, 5, 6, 4
0	[1, 2, 4, 5]		[1111111]	false	∅	0, 1, 3, 2, 5, 6, 4

Алгоритм 5: Обход в глубину. Нерекурсивный случай.

Вход: Граф, представленный списком смежности Gr , N — число вершин графа, x — вершина, с которой начинаем обход

Выход: Список последовательно посещенных вершин

начало алгоритма

· создаем стек и инициализируем его;

· создаем массив A размерности N и заполняем его нулями;

· присваиваем $A[x] = 1$ (помечаем вершину x как посещенную);

· помещаем вершину x в стек;

· выводим x на экран;

· создаем переменную типа `bool fl = false`;

цикл пока стек не пуст выполнять

· Рассматриваем вершину стека $x = h \rightarrow inf$ (не извлекая ее);

цикл для $i = 0$ до $Gr[x].size()$ выполнять

· **если существует непосещенная вершина, смежная вершине стека ($A[Gr[x][i]] == 0$) то**

· $y = Gr[x][i]$;

· $fl = true$;

· прекращаем цикл;

если нашли нужную вершину ($fl == true$) то

· помечаем y как посещенную вершину;

· помещаем ее в стек;

· выводим на экран;

иначе

· извлекаем вершину стека;

если остались непосещенные вершины то

· вызываем рассмотренный алгоритм для непосещенной вершины;

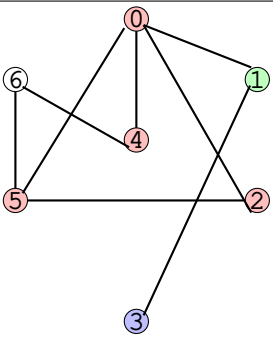
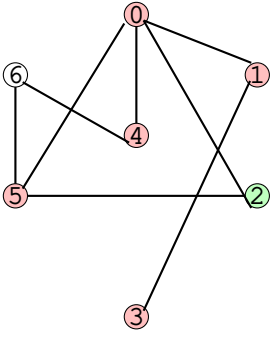
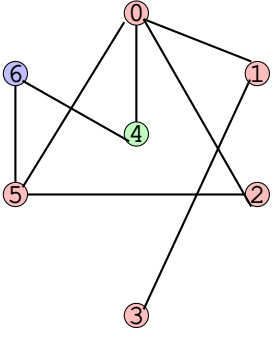
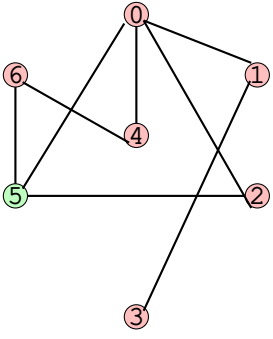
конец алгоритма

1.6 Обход графа в ширину

Другой способ обхода графа — это обход в ширину. Для работы алгоритма используем очередь. Начинаем рассматривать с некоторой вершины, записываем в очередь все смежные с ней непосещенные вершины, отмечаем их как посещенные. Извлекаем из очереди голову и повторяем процесс поиска смежных вершин до тех пор, пока очередь не пуста.

Рассмотрим тот же пример, что и в предыдущем разделе. Начинаем с вершины 0. Также создаем массив посещенных вершин. Зеленым цветом показана текущая вершина, синим — смежные, красным — уже посещенные (кроме текущей).

Граф	Результат	Действие с текущей вершиной	Массив посещенных вершин	Поиск смежных вершин
	0	Помечаем 0 как посещенную	[1, 0, 0, 0, 0, 0, 0]	Смежные вершины — 1, 2, 4, 5. Помечаем их как посещенные и записываем в очередь: 1, 2, 4, 5.

Граф	Результат	Действие с текущей вершиной	Массив посещенных вершин	Поиск смежных вершин
	0, 1, 2, 4, 5	Текущая вершина — 1. Извлекаем ее из очереди.	[1, 1, 1, 0, 1, 1, 0]	Смежная вершина — 3. Помечаем ее как посещенную и записываем в очередь: 2, 4, 5, 3.
	0, 1, 2, 4, 5, 3	Текущая вершина — 2. Извлекаем ее из очереди.	[1, 1, 1, 1, 1, 1, 0]	Смежных непосещенных вершин нет. Очередь: 4, 5, 3.
	0, 1, 2, 4, 5, 3	Текущая вершина — 4. Извлекаем ее из очереди.	[1, 1, 1, 1, 1, 1, 1]	Смежная вершина — 6. Помечаем ее как посещенную и записываем в очередь: 5, 3, 6.
	0, 1, 2, 4, 5, 3, 6	Текущая вершина — 5. Извлекаем ее из очереди.	[1, 1, 1, 1, 1, 1, 1]	Все вершины обошли, далее последовательно извлекаем вершины из очереди 3, 6.

Алгоритм 6: Обход в ширину.

Вход: Граф, представленный списком смежности Gr , N — число вершин графа, x — вершина, с которой начинаем обход

Выход: Список последовательно посещенных вершин

начало алгоритма

- создаем очередь и инициализируем ее;
- создаем массив A размерности N и заполняем его нулями;
- присваиваем $A[x] = 1$ (помечаем вершину x как посещенную);
- помещаем вершину x в очередь;
- выводим x на экран;

цикл пока очередь не пуста выполнять

- извлекаем голову очереди (x);

цикл для $i = 0$ до $Gr[x].size()$ выполнять

если существует непосещенная вершина, смежная x ($A[Gr[x][i]] == 0$) то

- $y = Gr[x][i]$;
- помечаем y как посещенную вершину;
- помещаем ее в очередь;
- выводим на экран;

если остались непосещенные вершины то

- вызываем рассмотренный алгоритм для непосещенной вершины;

конец алгоритма

Текущая вершина	$Gr[x]$	Массив A	Очередь	Результат
0	[1, 2, 4, 5]	[1 1 1 0 1 1 0]	[1, 2, 4, 5]	0, 1, 2, 4, 5
1	[0, 3]	[1 1 1 1 1 1 0]	[2, 4, 5, 2]	0, 1, 2, 4, 5, 3
2	[0, 5]	[1 1 1 1 1 1 0]	[4, 5, 3]	0, 1, 2, 4, 5, 3
4	[0, 6]	[1 1 1 1 1 1 1]	[5, 3, 6]	0, 1, 2, 4, 5, 3, 6
5	[0, 2, 6]	[1 1 1 1 1 1 1]	[3, 6]	0, 1, 2, 4, 5, 3, 6
3	[1]	[1 1 1 1 1 1 1]	[6]	0, 1, 2, 4, 5, 3, 6
6	[0, 4, 5]	[1 1 1 1 1 1 1]	[\varnothing]	0, 1, 3, 2, 5, 6