Algorytmy i stuktury danych - Lista 3

Alicja Myśliwiec - gr. wtorek 7:30

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
```

Zad. 1 Jeżeli prawdopodobieństwo pojedynczego sukcesu wynosi p, to prawdopodobieństwo osiągnięcia co najwyżej k sukcesów wyrazi się wzorem:

$$P(n,k) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i}$$

Napisz funkcję wyliczającą to prawdopodobieństwo. Nie może ona wymagać więcej niż $3k + \log n$ mnożeń.

Najpierw można sumę przekształcić w następujący sposób:

$$(1 - p)^n \cdot \sum_i^k \binom{n}{i} (\frac{p}{1-p})^i$$

Dwumian Newtona również można zapisać w sposób bardziej optymalny

$$\binom{n}{k} = \frac{n^{\underline{k}}}{k!} = \frac{n(n-1)(n-2)\cdots(n-(k-1))}{k(k-1)(k-2)\cdots 1} = \prod_{i=1}^{k} \frac{n+1-i}{i}$$

In [2]:
```python
def binomial_coeff(n, k):
    result, end, count_mult = 1, k + 1, 0

    for i in range(1, end):
        result *= (n - i + 1) / i
        count_mult += 2
    return result, count_mult
```

Dwumian generuje nam następującą liczbę mnożeń: $\sum_{i=1}^{k} 2i = k(k+1)$

Aby zredukować ilość mnożeń, skorzystam z potęgowania poprzez podnoszenie do kwadratu.

In [3]:
```python
def to_any_pow(base, any_pow):
    count_mult, result = 0, 1

    if any_pow < 12:   # dopiero od 12 jest opłacalne używać
        result = base ** any_pow
        count_mult +=  any_pow - 1

        return result, count_mult

    while any_pow > 0:
        if any_pow % 2 == 0:
            any_pow //= 2
            base *= base
            count_mult += 2
        else:
            any_pow = (any_pow - 1) // 2
            result *= base
            base *= base
            count_mult += 3

    return result, count_mult
```

In [4]:
```python
to_any_pow(15, 14)[1]
```

Out[4]:
```
11
```

Otrzymany wynik mnożeń podzielony przez log(15) (czyli log(n)) daje nam w przybliżeniu 3, zbliżone wyniki dawały inne próby. Dlatego też tą metodą nie przekroczymy liczby 3log(n) mnożeń.

Przechodząc już do funkcji liczącej prawdopodobieństwo

In [5]:
```python
def probability(n, k, p):
    count_mult, prob = 1, 0

    if n == k:
        return 1, 0

    iter_list = [it for it in range(0, k + 1)]
    p_elem = p / (1 - p)
    data = to_any_pow((1 - p), n)
    data_value = data[0]

    for i in iter_list:
        newton_step = binomial_coeff(n, i)
        func = newton_step[0] * data_value
        prob += func
        data_value *= p_elem
        count_mult += (newton_step[1] + 2)

    return prob, count_mult
```

In [6]:
```python
probability(16, 7, 0.3)
```

Out[6]:
```
(0.9256484499474198, 73)
```

In [7]:
```python
probability(16, 16, 0.378)
```

Out[7]:
```
(1, 0)
```

In [8]:
```python
probability(13, 8, 0.5)
```

Out[8]:
```
(0.8665771484375, 91)
```

In [9]:
```python
probability(12, 9, 0.85)
```

Out[9]:
```
(0.26418191377654937, 111)
```

Wiemy już, że na pewno mamy 3log(n) mnożeń. Należy teraz dodać odpowiednie mnożenia: dzielenie p/(1 - p), (1 - p)^n razy reszta wyrażenia, k(k-1) które generuje nam dwumian oraz (k-1) razy dwumian * nawias pod znakiem sumy. Daje nam to :

$$3\log_2(n) + (k+1)^2$$

Sprawdzenie:

Input

$3\log_2(16) + (7+1)^2$

Result

76

a 73 < 76

Zad. 2 Ile potrzeba mnożeń, aby wyliczyć wartość wielomianu stopnia n o współczynnikach zawartych w liście **a**. Napisz funkcję realizującą Twój algorytm.

Funckja licząca 'na około', a raczej krok po kroku.

In [10]:
```python
def ordinary_polynomial_value_calc(coeff, arg):

    length = len(coeff)

    count_mult, count_add, value = 0, length - 1, coeff[0] #count_add = ilości współ. pomnijeszona o 1
    step = 1

    if arg == 0:      # wartość to jedynie wyraz wolny, nie wykonujemy mnożeń i dodawań
        count_add = 0
        return value, count_mult, count_add

    if coeff[0] == 0:
        count_add -= 1  #ponieważ współ. jest równy 0 nie trzeba go dodawać (a + 0 = a)

    for i in coeff[1:]:
        if arg == 1:    # wartość jest równa sumie współczynników, wykonujemy tylko dodawanie (a * 1 = a)
            if i == 0:
                count_add -= 1
            else:
                value += i
        else:
            if i == 0:
                count_add -= 1
            elif i == 1:
                value += arg ** step
                count_mult += step - 1
            else:
                value += i * (arg ** step)
                count_mult += step
            step += 1

    return value, count_mult, count_add
```

In [11]:
```python
ordinary_polynomial_value_calc([10, 2, 2, 0, 1, 2], 2)    # 2x^5 + x^4 + 2x^2 + 2x + 10
```

Out[11]:
```
(102, 11, 4)
```

In [12]:
```python
10 + 2 * (2) + 2 * (2 * 2) + 2 * (2 * 2 * 2) + 2 * (2 * 2 * 2 * 2 * 2)
```

Out[12]:
```
102
```

Gdyby liczyć na piechotę ilość '+' i '*' to widać, że występują one odpowienio 4 i 11 razy (wyraz ze współ. 0 został pomienięty)

In [13]:
```python
ordinary_polynomial_value_calc([0, 1, 2, 0, 3], 2)       # 3x^4 + 2x^2 + x
```

Out[13]:
```
(58, 6, 2)
```

Funckja zoptymalizowana, w której został użyty algorytm Hornera właśnie w celu zredukowania liczby mnożeń. Z wielomianu w znanej nam postaci:

$$\mathrm{w(x)} = \mathrm{a}_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$$

Możemy go łatwo przekształcić

$$\mathrm{w(x)} = \mathrm{a}_0 + x(a_1 + x(a_2 + \ldots x(a_{n-1} + xa_n)))$$

In [14]:
```python
def smart_polynomial_value_calc(coeff, arg):

    count_mult, count_add, value = 0, 0, coeff[-1]       # wartość od razu jest przyrównywana do n-tego wyrazu
    coeff.reverse()      # odwracam listę, ponieważ idę 'od środka' powyższego równania

    if arg == 0:    # wartość to jedynie wyraz wolny, nie wykonujemy mnożeń i dodawań
        value = coeff[-1]
        return value, count_mult, count_add

    for i in coeff[1:]:
        if arg == 1:      # wartość jest równa sumie współczynników, wykonujemy tylko dodawanie (a * 1 = a)
            if i == 0:
                pass
            else:
                value += i
                count_mult += 1
        else:
            if i == 0:
                value *= arg
                count_mult += 1
            else:
                value = value * arg + i
                count_mult += 1
                count_add += 1

    return value, count_mult, count_add
```

In [15]:
```python
smart_polynomial_value_calc([10, 2, 2, 0, 1, 2], 2)
```

Out[15]:
```
(102, 5, 4)
```

In [16]:
```python
smart_polynomial_value_calc([10, 2, 2, 0, 1, 2], 1)
```

Out[16]:
```
(17, 4, 0)
```

In [17]:
```python
smart_polynomial_value_calc([-3, 2, 5, 0, 2], 4)   #  2x^4 + 5x^2 + 2x - 3
```

Out[17]:
```
(597, 4, 3)
```

Zad. 3 Napisz program, który policzy, ile razy występuje każdy znak w pliku tekstowym podanym jako argument wywołania. Nie możesz przy tym używać wyrażenia warunkowego **if**.

W raporcie funckja będzie przyjmować string jako argument. W pliku .py otwiera już ona plik tak jak powinna

Wersja z otwieraniem pliku

def counting_chars_without_ifs(filename):

```
file_ref = open(filename, 'r')
text = file_ref.read()
text = text.lower()

div_text = []
for char in text:
    div_text.append(char)

accepted_chars = [chr(x) for x in range(33, 126)]

what_chars_to_count = list(set(div_text) & set(accepted_chars))
# what_chars_to_count.sort()

char_count = {}

for char in what_chars_to_count:
    how_many_chars = div_text.count(char)
    key = '{}'.format(char)
    char_count[key] = how_many_chars

return char_count
```

In [18]:
```python
def counting_chars_without_ifs(string):
    text = string
    text = text.lower()

    div_text = []
    for char in text:
        div_text.append(char)

    accepted_chars = [chr(x) for x in range(33, 126)]

    what_chars_to_count = list(set(div_text) & set(accepted_chars))
    # what_chars_to_count.sort()

    char_count = {}

    for char in what_chars_to_count:
        how_many_chars = div_text.count(char)
        key = '{}'.format(char)
        char_count[key] = how_many_chars

    return char_count
```

W funkcji tworzę listę 'akceptowalnych znaków' za pomocą wartości tych właśnie znaków zgodnie z poniższą tabelą ASCII. Nie chcemy aby funckja zliczała nam białych znaków, takich jak spacja czy enter - dlatego przedział znaków o wartościach [33, 126] jest jedynym, który nas interesuje.

Dec	Char	Dec	Char	Dec	Char
32	SPACE	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_	127	DEL

Następnie tworzę listę stworzoną z częsci wspólnej dwóch list: akceptowalnych znaków oraz unikalnych znaków w tekście, który chcemy sprawdzić. Dzięki temu w słowniku nie znajdą sie ani znaki białe ani nie będą zliczane znaki, które po prostu w tekście nie występują.

In [19]:
```python
sample_text = "Happy families are all alike; every unhappy family is unhappy in its own way.\
Everything was in confusion in the Oblonskys' house. The wife had discovered that the husband was carrying on \
an intrigue with a French girl, who had been a governess in their family, and she had announced to her husband
that she could not go on living in the same house with him. This position of affairs had now lasted three days,
and not only the husband and wife themselves, but all the members of their family and household, were painfully
conscious of it. Every person in the house felt that there was so sense in their living together, and that \
the stray people brought together by chance in any inn had more in common with one another than they, \
the members of the family and household of the Oblonskys. The wife did not leave her own room, \
the husband had not been at home for three days. The children ran wild all over the house; \
the English governess quarreled with the housekeeper, and wrote to a friend asking her to look out for a new si
for her; the man-cook had walked off the day before just at dinner time; the kitchen-maid, and the coachman \
had given warning."
```

In [20]:
```python
data = counting_chars_without_ifs(sample_text)
```

In [21]:
```python
print(data)
```

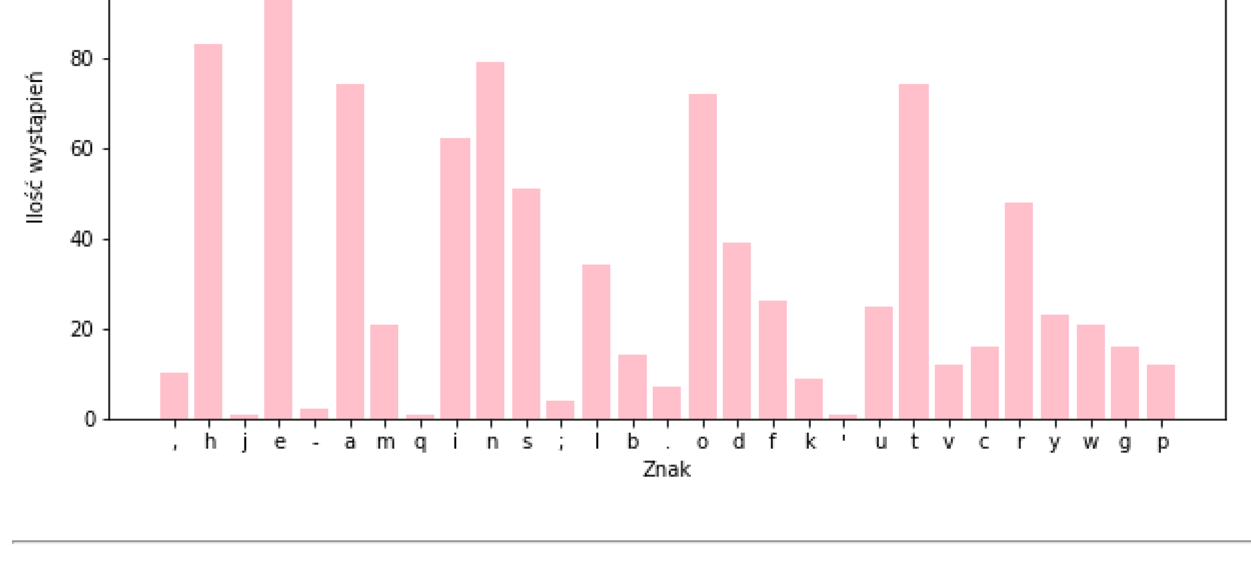
```
{',': 10, 'h': 83, 'j': 1, 'e': 115, '-': 2, 'a': 74, 'm': 21, 'q': 1, 'i': 62, 'n': 79, 's': 51, ';': 4, 'l':
34, 'b': 14, '.': 7, 'o': 72, 'd': 39, 'f': 26, 'k': 9, "'": 1, 'u': 25, 't': 74, 'v': 12, 'c': 16, 'r': 48,
'y': 23, 'w': 21, 'g': 16, 'p': 12}
```

Dodatkowa próba wizualizacji

In [22]:
```python
amount = [value for value in data.values()]
bars = (key for key in data.keys())
y_pos = np.arange(len(amount))
plt.figure(figsize=(10,5))
plt.bar(y_pos, amount, color = "pink")
plt.xticks(y_pos, bars)

plt.xlabel('Znak')
plt.ylabel('Ilość wystąpień')
plt.title('Wykresik')

plt.show()
```



LINK GITHUB

https://github.com/AlutkaMalutka/Programowanie_python/tree/main/semestr_3/Lista_3-3s-