Lista 3

Maciej Karczewski

zad 1

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

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

Napisz funkcję wyliczającą to prawdopodobieństwo. Nie może ona wymagać więcej niż ak + b *log(n) + c mnożeń gdzie a,b,c są stałymi.

- Na początku przekształćmy trochę wzór

$$\sum_{i=0}^{k} \binom{n}{i} \cdot p^i \cdot (1-p)^{n-i} = \sum_{i=0}^{k} \binom{n}{i} \cdot (1-p)^n \cdot \left(\frac{p}{1-p}\right)^i = (1-p)^n \cdot \sum_{i=0}^{k} \binom{n}{i} \cdot \left(\frac{p}{1-p}\right)^i$$

- Teraz znajdziemy zależność miedzy i-tym wyrazem a i+1 wyrazem

$$q = \frac{\binom{n}{i+1} \cdot p^{i+1} \cdot (1-p)^{-i-1}}{\binom{n}{i} \cdot p^i \cdot (1-p)^{-i}} = \frac{(n-i)p}{(i+1)(1-p)} = \frac{n-i}{i+1} \cdot C(p)$$

gdzie

$$C(p) = \frac{p}{1-p}$$

- Zatważmy że (1-p) możemy podnieś do n-tej potęgi w 3*log(n) mnożeniach używając algorytmu szybkiego potęgowania

```
def to_n_pow(base, power, count_mult):
    result = 1
    while power:
        power, check = divmod(power, 2)
        count_mult += 1
        if check == 0:
            base *= base
            count_mult += 1
        else:
            result *= base
            base *= base
            count_mult += 2
    return result , count_mult
```

- Każdy wyraz sumy możemy obliczyć używając 3 mnożeń , więc sume obliczymy w 3k mnożeń także potrzebujemy jednorazowo na początku wykonać dodatkowe mnożenie

```
for i in range(0,k):
    single_probabilitis[i+1] = single_probabilitis[i] * factor_p * (n-i)/(i+1)
    count_mult += 3
```

- Wynik sumy musimy przemnożyć przez wcześniej obliczoną stałą (1-p)^n więc ilość potrzebnych mnożeń nie przekroczy

$$3k + 3\log_2 n + 2$$

- Ale możemy wykorzystać zdarzenie przeciwne dla k większego niż połowa zdarzeń wtedy ilość potrzebnych mnożeń wynosi

$$3(n-k) + 3\log_2 n + 2$$

In [16]:
```
def probability(n, k, p):
    """
    Fuction count propability of less or k sucess in n attempts
    @pam n: (int) number of attempts
    @pam k: (int) number of max success in n attempts
    @pam p: (float) propability of single sucess
    @return prob: (float) propability of less or k sucess in n attempts
    @return count_mult: (int) number how many multiplicatio was done
    """

    # w ciele funkcji umieść swój kod realizujący cel zadania;
    # argument 'n' niech będzie liczbą prób;
    # argument 'k' niech będzie maksymalną liczbą sukcesów;
    # argument 'p' niech będzie prawdopodobieństwem sukcesu w pojedynczej próbie;
    # w zmiennej 'prob' zwróć oczekiwane prawdopodobieństwo;
    # w zmiennej 'count_mult' zwróć liczbę mnożeń, jaką wykonał Twój program;
    # jeśli potrzebujesz, możesz dopisać również inne funkcje (pomocnicze),
    # jednak główny cel zadania musi być realizowany w tej funkcji;

    def to_n_pow(base, power, count_mult):
        result = 1
        while power:
            power, check = divmod(power, 2)
            count_mult += 1
            if check == 0:
                base *= base
                count_mult += 1
            else:
                result *= base
                base *= base
                count_mult += 2
        return result , count_mult

    prob = 0
    count_mult = 0
    if k == n:
        return 1, 0
    elif k > n // 2: # in this case we use opposite event
        m = n-k
        factor_p = (1-p)/p
        count_mult += 1
        single_probabilitis = [1 for i in range(0,m)]
        single_probabilitis[0] , count_mult = to_n_pow(p, n, count_mult)
        for i in range(0,m-1):
            single_probabilitis[i+1] = single_probabilitis[i] * factor_p * (n-i)/(i+1)
            count_mult += 3
            prob = 1 - sum(single_probabilitis)
    else:
        m = k
        factor_p = p/(1-p)
        count_mult += 1
        single_probabilitis = [1 for i in range(0,m+1)]
        single_probabilitis[0], count_mult = to_n_pow((1-p), n, count_mult)
        for i in range(0,m):
            single_probabilitis[i+1] = single_probabilitis[i] * factor_p * (n-i)/(i+1)
            count_mult += 3
            prob = sum(single_probabilitis)


    return (prob, count_mult)
```

In [17]:
```
print(probability(64,30,0.5))
print(probability(50,35,0.7))
```

```
(0.3539903770673827, 106)
(0.5531684257419579, 58)
```

zad 2

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

- Jeżeli beziemy klasycznie podstawiać pod wzór zaczynając od zerowej potęgi kończąc na n-tej to wykonamy 2(n-1) mnożeń

In [3]:
```
def ordinary_polynomial_value_calc(coeff, arg):
    """Function calculate w(x) where w is polynomial
    @pam coeff : (list) list of coefficient of polynomial (first is a_0)
    @pam arg: (float) argument of function
    @return value: (float) value of this polynomial
    @return count_mult: (int) number how many multiplicatio was done
    @return count_add: (int) number how many adds was done
    """
    # w ciele tej funkcji zawrzyj kod wyliczający wartość wielomianu w tradycyjny sposób;
    # argument 'coeff' niech będzie listą współczynników wielomianu w kolejności od stopnia zerowego (w
yrazu wolego) wzwyż;
    # argument 'arg' niech będzie punktem, w którym chcemy policzyć wartość wielomianu;
    # w zmiennej 'count_mult' zwróć liczbę mnożeń, jakie zostału wykonane do uzyskania tego wyniku;
    # w zmiennej 'count_add' zwróć liczbę dodawań, jakie zostału wykonane do uzyskania tego wyniku;
    value = coeff[0]
    before = arg
    count_mult = 0
    count_add = 0
    for coef in coeff[1:]:
        value += coef * before
        before *= arg
        count_add += 1
        count_mult += 2
    return value, count_mult, count_add
```

In [4]:
```
print(ordinary_polynomial_value_calc([0,1,4,-5,6], 0))
print(ordinary_polynomial_value_calc([0,1,4,-5,6], 2))
```

```
(0, 8, 4)
(74, 8, 4)
```

- Możemy obliczyć wartość funkcji z schematu hornera , wtedy wykonamy n-1 mnożeń

In [5]:
```
def smart_polynomial_value_calc(coeff, arg):
    """Function calculate w(x) where w is polynomial using horner scheme
    @pam coeff : (list) list of coefficient of polynomial (first is a_0)
    @pam arg: (float) argument of function
    @return value: (float) value of this polynomial
    @return count_mult: (int) number how many multiplicatio was done
    @return count_add: (int) number how many adds was done
    """
    # w ciele tej funkcji zawrzyj kod wyliczający wartość wielomianu w sposób maksymalnie ograniczający
 liczbę wykonywanych mnożeń;
    # argument 'coeff' niech będzie listą współczynników wielomianu w kolejności od stopnia zerowego (w
yrazu wolego) wzwyż;
    # argument 'arg' niech będzie punktem, w którym chcemy policzyć wartość wielomianu;
    # w zmiennej 'count_mult' zwróć liczbę mnożeń, jakie zostału wykonane do uzyskania tego wyniku;
    # w zmiennej 'count_add' zwróć liczbę dodawań, jakie zostału wykonane do uzyskania tego wyniku;

    value = coeff[-1]
    count_mult = 0
    count_add = 1
    for n in range( len(coeff) - 2, -1, -1):
        value *= arg
        value += coeff[n]
        count_add += 1
        count_mult += 1
    return value, count_mult, count_add
```

In [6]:
```
print(smart_polynomial_value_calc([0,1,4,-5,6], 0))
print(smart_polynomial_value_calc([0,1,4,-5,6], 2))
```

```
(0, 4, 5)
(74, 4, 5)
```

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 zadaniu tworzę słownik z tekstu a następnie obliczam różnice zbiorów ze zbiorem zawierającym spacje, enter, tabulator a następnie iteruje przez zbiór i metodą count liczę ile razy ten znak pojawia się w tekście i dodaje kulcz którym jest znak a wartość to liczba powtórzeń

In [8]:
```
def counting_chars_without_ifs(filename):
    """Fuction count numers of chars in given filename
    @pam filename: (str) path to file with text
    @return char_count: (dict) dictionary with numbers of chars in text"""
    with open(filename, 'r') as file_ref:
        text = file_ref.read()
        words = text.lower()
        char_count = {}
        letters = set(words) - set(' \n \t')
    for letter in letters:
        char_count[letter] = words.count(letter)


    # uzupełnij ciało tej funkcji kodem realizującym cel zadania;
    # w zmiennej 'char_count' zwróć słownik zawierający wszystkie znaki tekstu
    # jako klucze i ich liczebność jako wartości np. {'a': 6, 'b': 2 ...};
    # jeli potrzebujesz, mozesz dopisać również inne funkcje (pomocnicze),
    # jednak główny cel zadania musi być realizowany w tej funkcji;
    return char_count
```

Kod

https://github.com/maciejkar/lista3.git