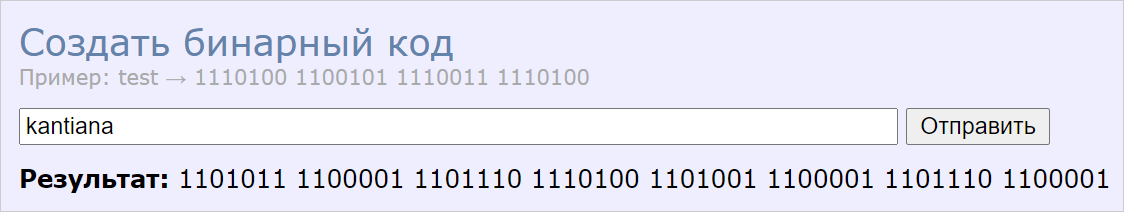
Лабораторная работа № 3

Дискретная математика для программистов

Отт Арсений ПМ-2

Вариант 52

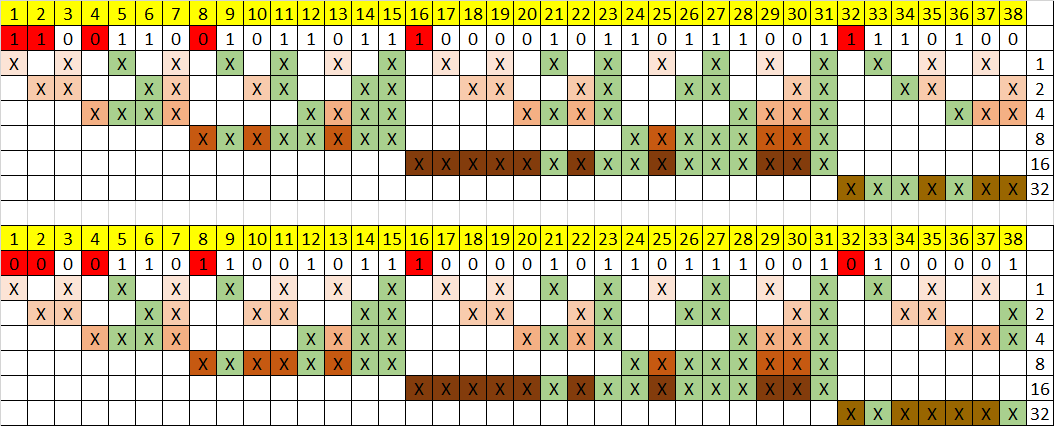
1. **Код Хемминга. Строку «kantiana» перевести в двоичный ANSI код, разбить на два блока по 32 бита, добавить контрольные биты, имитировать ошибки в 7 бите первого блока и 17 бите второго блока, восстановить исходную информацию**



Строка «» в бинарном виде, разбитая на два блока по 32 бита:

Кодировать блоки будем независимо. Вставим контрольные биты. Так как длина информационного слова бита, вставим их в позиции с номерами . Соответственно, имеем контрольных битов (они выделены красным цветом и обозначены символом «\*»):

Вычислим эти контрольные биты. Составим для каждого блока таблицу, в которой символом «» обозначим те биты, которые контролирует контрольный бит, номер которого справа. Зелёным цветом отметим позиции, в которых бит – единица. Найдём каждый контрольный бит как сумму соответствующих битов по модулю 2. Для первого и второго блоков имеем:

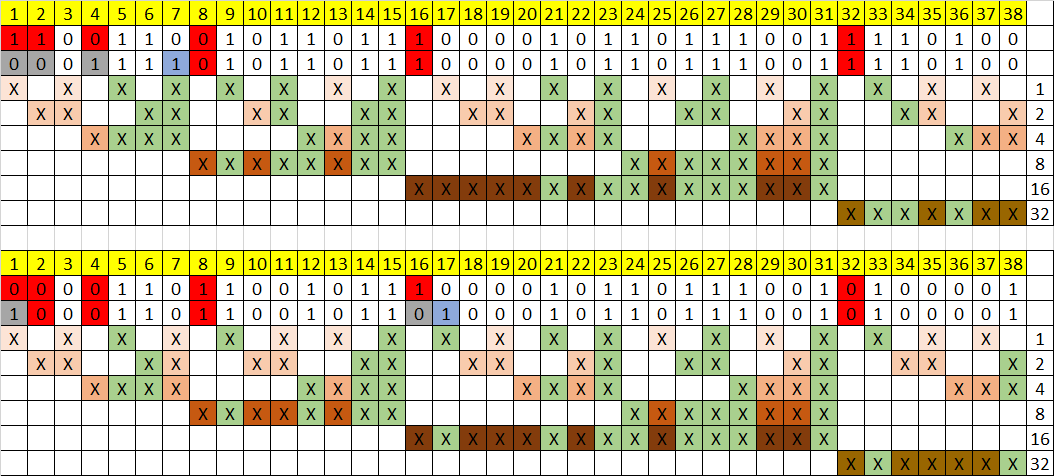


В итоге получаем:

Имитация ошибок

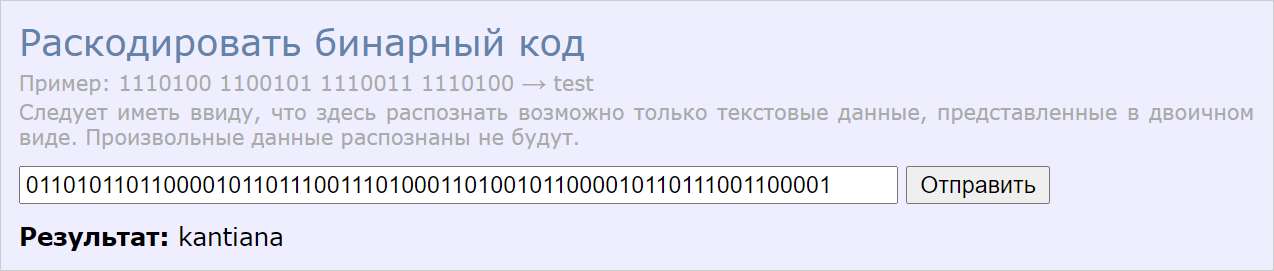
Выделим синим цветом изменённые бит первого блока и бит второго блока:

Вычислим заново все контрольные биты. Получаем, что не совпадают контрольные биты под номерами , и в первом блоке и контрольные биты под номерами и во втором блоке. Суммы номеров позиций неправильных контрольных битов равны позициям ошибочных битов:



Инвертировав изменённые биты и отбросив контрольные биты, получим исходное сообщение в первозданном виде. Инвертируем биты:

Отбрасываем контрольные:



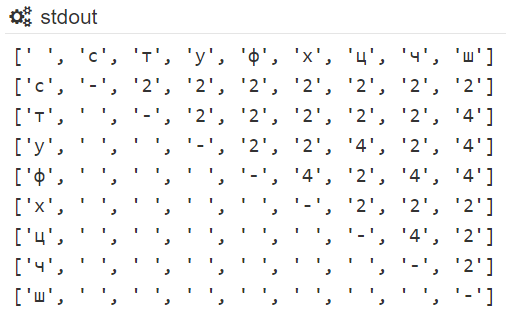
1. **Расстояние Хемминга. Для набора букв «стуфхцчш» придумать двоичные коды с расстоянием не менее 2 и продемонстрировать поиск ошибки. Придумать двоичные коды с расстоянием не менее 3 и продемонстрировать поиск и исправление ошибки**

Двоичные коды с расстоянием не менее 2

Таблица двоичных кодов:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| с |  | ф |  | ч |  |
| т |  | х |  |  |  |
| у |  | ц |  |

Таблица попарных расстояний:



1. m = [" ", 'с', 'т', 'у', 'ф', 'х', 'ц', 'ч', 'ш']
2. n = ['00000', '00011', '00101', '00110', '01001', '01010', '10001', '11000']
3. table = [m]
5. **for** i **in** range(0, len(n)):
6. line = [m[i + 1]]
7. **for** l **in** range(0, i):
8. line.append(" ")
9. line.append("-")
10. **for** j **in** range(i + 1, len(n)):
11. num = 0
12. **for** k **in** range(0, len(n[i])):
13. **if** n[i][k] != n[j][k]:
14. num += 1
15. line.append(str(num))
16. table.append(line)
18. **for** i **in** range(0, len(n) + 1):
19. **print**(table[i])

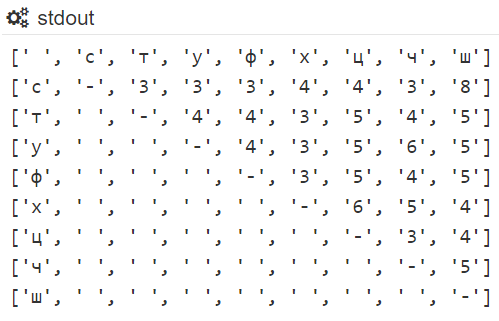
Минимальное расстояние , значит такой код может исправить ноль ошибок, обнаружить одну ошибку

Пусть было получено сообщение . Сравниваем код с кодами в таблице двоичных кодов. Совпадений не обнаружено. Посчитаем расстояния Хемминга для полученного кода и исходного набора букв:

Минимальное расстояние соответствует трём символам. Найдена одиночная ошибка, исправление невозможно

Двоичные коды с расстоянием не менее 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| с | 00000000 | ф | 11000001 | ч | 10100100 |
| т | 00000111 | х | 01010101 | ш | 11111111 |
| у | 00011001 | ц | 10110010 |



1. m = [" ", 'с', 'т', 'у', 'ф', 'х', 'ц', 'ч', 'ш']
2. n = ['00000000', '00000111', '00011001', '11000001', '01010101', '10110010', '10100100', '11111111']
3. table = [m]
5. **for** i **in** range(0, len(n)):
6. line = [m[i + 1]]
7. **for** l **in** range(0, i):
8. line.append(" ")
9. line.append("-")
10. **for** j **in** range(i + 1, len(n)):
11. num = 0
12. **for** k **in** range(0, len(n[i])):
13. **if** n[i][k] != n[j][k]:
14. num += 1
15. line.append(str(num))
16. table.append(line)
18. **for** i **in** range(0, len(n) + 1):
19. **print**(table[i])

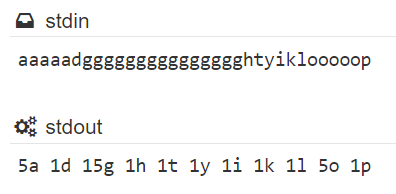
Минимальное расстояние , значит такой код может исправить единичную ошибку, обнаружить две ошибки

Пусть было получено сообщение . Совпадений с кодами в таблице двоичных кодов не обнаружено. Посчитаем расстояния Хемминга для полученного кода и исходного набора букв:

Минимальное расстояние соответствует символу «». Найдена и исправлена одиночная ошибка

1. **Сжатие способом кодирования серий (RLE). Первый байт указывает сколько раз нужно повторить следующий байт. Если первый байт равен 0, то затем идет счетчик, показывающий сколько за ним следует неповторяющихся данных.**

**Строка для сжатия: aaaaadggggggggggggggghtyiklooooop**



1. string = input()
2. amount = 1
3. final\_string = ""
4. **for** i **in** range(len(string) - 1):
5. **if** string[i] == string[i + 1]:
6. amount += 1
7. **else**:
8. a = string[i]
9. final\_string = final\_string + str(amount) + string[i] + " "
10. amount = 1
11. **if** string[len(string) - 1] != string[len(string)-2]:
12. final\_string = final\_string + str(amount) + string[len(string) - 1] + " "
13. **print**(final\_string)

Найдём степень сжатия по сравнению с простейшим равномерным кодированием. Строка содержит символа. Строка содержит символа. Одиннадцать символов можно закодировать следующим образом:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Тогда строка будет состоять из символов. Степень сжатия:

1. **Алгоритм Хаффмана. Пусть при подсчете вхождения каждого из символов в файл получили следующее:**

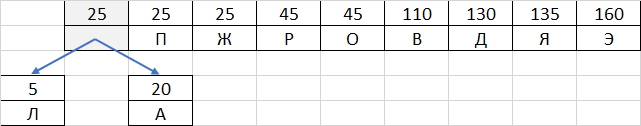
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Символ** | **В** | **А** | **П** | **Р** | **О** | **Л** | **Д** | **Ж** | **Э** | **Я** |
| **Число вхождений** | **110** | **20** | **25** | **45** | **45** | **5** | **130** | **25** | **160** | **135** |

**Построить коды символов и продемонстрировать на примерах кодирование и раскодирование**

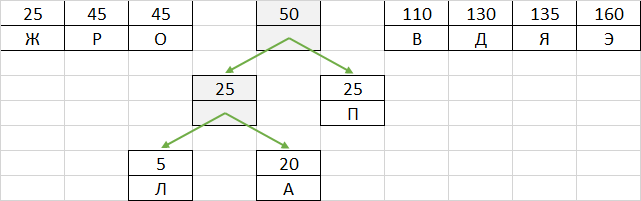
Создадим узлы бинарного дерева для каждого знака и добавим их в очередь, используя частоту в качестве приоритета:

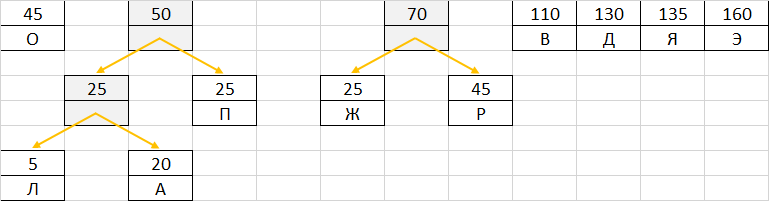


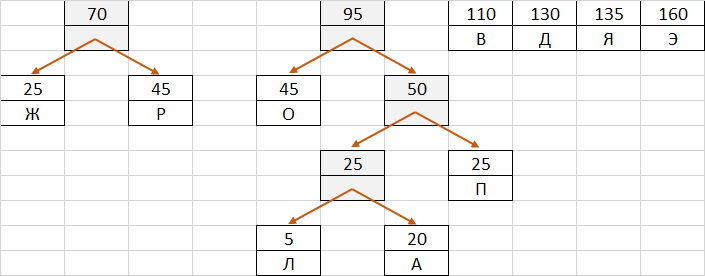
Достанем два первых элемента из очереди и свяжем их, создавая новый узел дерева, в котором они оба будут потомками, а приоритет нового узла будет равен сумме их приоритетов. Затем добавим получившийся новый узел обратно в очередь

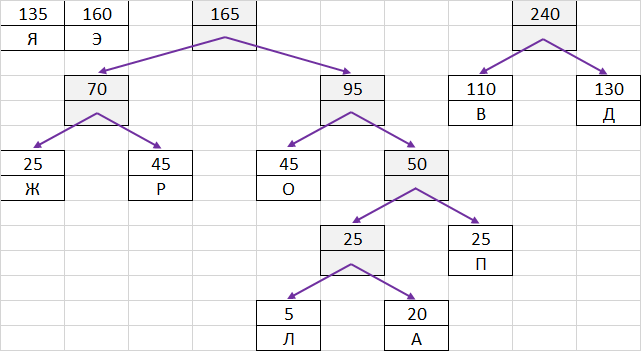


Повторим те же шаги и получим последовательно:

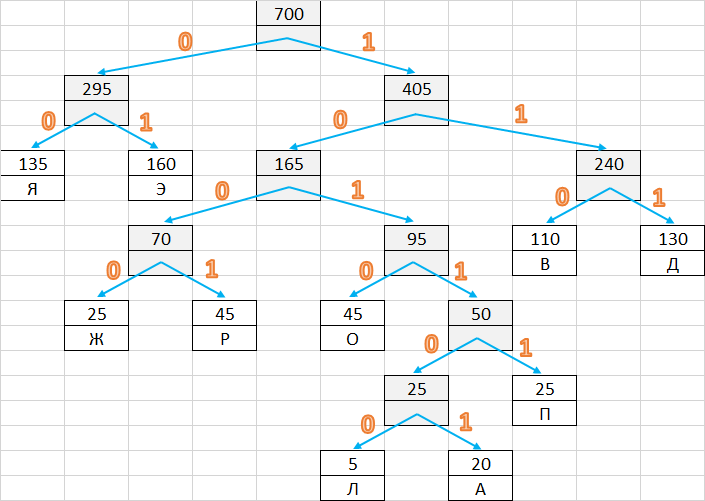








Получив итоговое дерево, найдём код для каждого символа, пройдясь по дереву. Для каждого перехода добавим , если идём влево, и – если вправо:



Получим следующие коды для символов:

|  |  |
| --- | --- |
| Символ | Код |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Закодируем строку . Пользуясь таблицей, получим: . Декодируем теперь эту строку с помощью дерева. Читаем первый бит: – идём вправо, следующий бит: снова , следующий: 1. Достигли листа дерева, получаем первый символ: . Далее аналогично.

Найдём степень сжатия по сравнению с простейшим равномерным кодированием. Строка содержит символов. Строка содержит символов. Десять символов можно закодировать следующим образом:

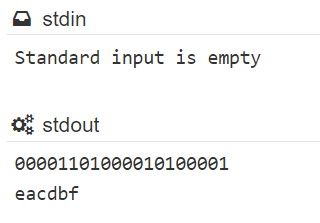
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

Тогда строка будет состоять из символов. Степень сжатия:

1. **Арифметическое кодирование. Пусть алфавит состоит из символов: a b c d e f**

**с вероятностями соответственно 0,05 0,10 0,05 0,55 0,15 0,10**

**Построить код для строки: eacdbf. Ответ дать в виде двоичной строки**



1. **def** float2bin(x, eps=1e-9):
2. res = ''
3. **while** x > eps:
4. x \*= 2
5. res += str(int(x))
6. x -= int(x)
7. **return** res

10. **def** bin2float(x):
11. **return** sum(2 \*\* (-i - 1) **for** i, digit **in** enumerate(x) **if** digit == '1')

14. **def** find\_code(a, b):
15. i = 0
16. a += '0' \* (len(b) - len(a))
17. **while** a[i] == b[i]:
18. i += 1
19. res = a[:i] + '0'
20. cnt = 0
21. **while** a[i] == 1:
22. i += 1
23. cnt += 1
24. res += '1' \* (cnt + 1)
25. **return** res

28. **def** coding(word, alphabet, p):
29. left, right = 0, 1
30. **for** letter **in** word:
31. i = alphabet.index(letter)
32. left, right = (left + (right - left) \* sum(p[:i]),
33. left + (right - left) \* sum(p[: i + 1]))
34. **return** find\_code(\*map(float2bin, (left, right)))

37. **def** decoding(code, length, alphabet, p):
38. code = bin2float(code)
39. word = ''
40. left, right = 0, 1
41. **for** \_ **in** range(length):
42. **for** i, letter **in** enumerate(alphabet):
43. interval = (left + (right - left) \* sum(p[:i]),
44. left + (right - left) \* sum(p[:i + 1]))
45. **if** interval[0] <= code < interval[1]:
46. word += letter
47. code = (code - interval[0]) / (interval[1] - interval[0])
48. **break**
49. **return** word

52. alphabet = 'aecdfb'
53. p = (0.05, 0.10, 0.05, 0.55, 0.15, 0.10)
54. word = 'eacdbf'
55. code = coding(word, alphabet, p)
56. **print**(code)
57. **print**(decoding(code, len(word), alphabet, p))

Найдём степень сжатия по сравнению с простейшим равномерным кодированием. Строка содержит символов. Строка содержит символов. Шесть символов можно закодировать следующим образом:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Тогда строка будет состоять из символов. Степень сжатия: