Алгоритм Хаффмана

Проект выполнили
студентки 24-КНТ-8

Буинская Анастасия
Тигина Алёна

Структура кода

```
10  typedef struct Node
11  {
12      unsigned char symb;
13      unsigned int freq;
14      struct Node* left, * right, * next;
15  }NODE;
16
17  //Очередь(с приоритетами)
18  typedef struct Queue
19  {
20      NODE* nodes[ALPHABET];
21      int size;
22  } QUEUE;
23
24  // Битовый буфер
25  typedef union bit2char
26  {
27      char symb;
28      struct bit
29      {
30          unsigned b1 : 1;
31          unsigned b2 : 1;
32          unsigned b3 : 1;
33          unsigned b4 : 1;
34          unsigned b5 : 1;
35          unsigned b6 : 1;
36          unsigned b7 : 1;
37          unsigned b8 : 1;
38      } mbit;
39  }BIT2CHAR;
```

```
40
41  // Очередь (с приоритетами)
42  QUEUE* createQueue();
43  void enQueue(QUEUE* queue, NODE* node);
44  NODE* deQueue(QUEUE* queue);
45
46  // Дерево Хаффмана
47  NODE* createNode(unsigned char symb, unsigned int freq);
48  NODE* MakeNodeFromNode(NODE* left, NODE* right);
49  void Add2List(NODE** head, NODE* newNode);
50  NODE* buildTree(int* freq);
51
52  // Кодирование и декодирование
53  void generateHuffmanCodes(NODE* root, char codes[ALPHABET][MAX_CODE_SIZE], char* code, int level);
54  void encodeFile(FILE* inputFile, FILE* outputFile, char codes[ALPHABET][MAX_CODE_SIZE]);
55  void decodeFile(FILE* inputFile, FILE* outputFile, NODE* root);
56
57  void loadCodesFromFile(char codes[ALPHABET][MAX_CODE_SIZE], const char* filename);
58  void saveCodesToFile(const char codes[ALPHABET][MAX_CODE_SIZE], const char* filename);
59
60  // Освобождение памяти
61  void freeTree(NODE* root);
62
63  #endif
```

Main -> Main (for res.)

```c
int main()
{
    int freq[ALPHABET] = { 0 };

    FILE* fin = fopen("war_and_peace.txt", "rb");
    if (!fin) {
        fprintf(stderr, "Can't open input file\n");
        return -1;
    }

    fseek(fin, 0L, SEEK_END);
    long int length = ftell(fin);
    fseek(fin, 0, SEEK_SET);

    for (int i = 0; i < length; i++) {
        freq[(unsigned char)fgetc(fin)]++;
    }
    fclose(fin);

    NODE* root = buildTree(freq);
    if (!root) {
        fprintf(stderr, "Error: failed to build Huffman tree\n");
        return -1;
    }

    char codes[ALPHABET][MAX_CODE_SIZE] = { 0 };
    char tempCode[MAX_CODE_SIZE];
    generateHuffmanCodes(root, codes, tempCode, 0);

    saveCodesToFile(codes, "codes.txt");

    fin = fopen("war_and_peace.txt", "rb");
    FILE* fout = fopen("encoded.bin", "wb");
    encodeFile(fin, fout, codes);
    fclose(fin);
    fclose(fout);

    char loadedCodes[ALPHABET][MAX_CODE_SIZE] = { 0 };
    loadCodesFromFile(loadedCodes, "codes.txt");

    fin = fopen("encoded.bin", "rb");
    fout = fopen("decoded.txt", "wb");
    decodeFile(fin, fout, root);
    fclose(fin);
    fclose(fout);

    freeTree(root);
    return 0;
}
```

```c
void measureExecutionTime(void (*func)(const char*, int*), const char* description, const char* filename, int* freq)
{
    clock_t start_time = clock();
    func(filename, freq);
    clock_t end_time = clock();
    double time_taken = (double)(end_time - start_time) / CLOCKS_PER_SEC;
    printf("%s time: %f seconds\n", description, time_taken);
}
```

```c
int main()
{
    int freq[ALPHABET] = { 0 };
    const char* filename = "42 MB.mov";
    FILE* fin = fopen(filename, "rb");
    if (!fin)
    {
        fprintf(stderr, "Error: cannot open codes file\n");
        exit(1);
    }
    fseek(fin, 0L, SEEK_END);
    long int length = ftell(fin);
    fseek(fin, 0, SEEK_SET);
    for (int i = 0; i < length; i++)
    {
        freq[(unsigned char)fgetc(fin)]++;
    }
    fclose(fin);

    measureExecutionTime(encode, "Encoding", filename, freq);
    measureExecutionTime(decode, "Decoding", filename, freq);
    return 0;
}
```

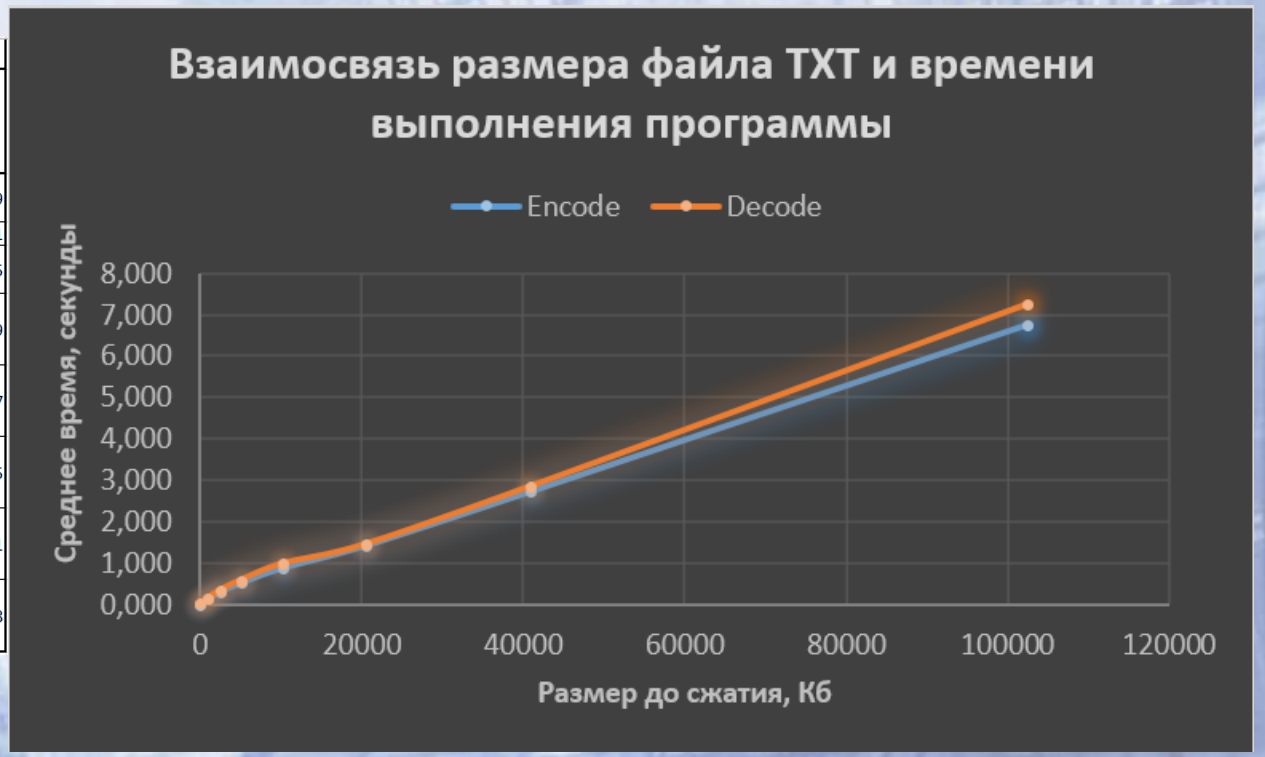
```c
void encode(const char* filename, int* freq)
{
    NODE* root = buildTree(freq);
    char codes[ALPHABET][MAX_CODE_SIZE] = { 0 };
    char tempCode[MAX_CODE_SIZE];
    generateHuffmanCodes(root, codes, tempCode, 0);

    FILE* fin = fopen(filename, "rb");
    FILE* fout = fopen("encoded.bin", "wb");
    encodeFile(fin, fout, codes);
    fclose(fin);
    fclose(fout);

    saveCodesToFile(codes, "codes.txt");
    freeTree(root);
}

void decode(const char* filename, int* freq)
{
    NODE* root = buildTree(freq);
    char codes[ALPHABET][MAX_CODE_SIZE] = { 0 };
    char tempCode[MAX_CODE_SIZE];
    generateHuffmanCodes(root, codes, tempCode, 0);

    FILE* f = fopen("encoded.bin", "rb");
    FILE* fout = fopen("decoded.mov", "wb"); ///
    decodeFile(f, fout, root);
    fclose(f);
    fclose(fout);

    freeTree(root);
}
```

Распределение задач

Буинская А.А.	**Тигина А.Р.**
Функции очереди	Объявление структур
Основной main	Функции создания дерева
Decode-функция	Объявление функций
Улучшение time-функции (привязка к en/decode)	Encode-функция
Архив файлов для тестирования	Функция замера времени
Тестирование	Организация данных исследования
Презентация	Очная презентация

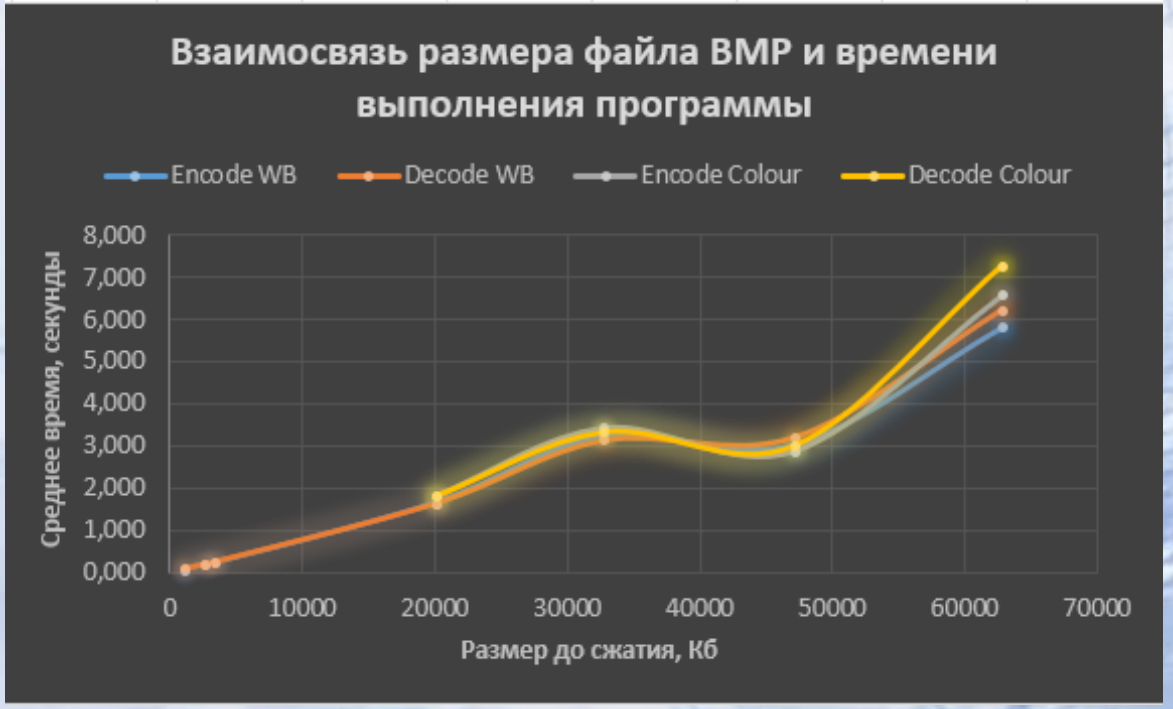
Исследование: txt-файлы

Файл				Encode				Decode			
Название	Размер до сжатия, Кб	Размер после сжатия, Кб	Коэффициент сжатия, %	Запуск 1	Запуск 2	Запуск 3	Среднее время выполнения, секунды	Запуск 1	Запуск 2	Запуск 3	Среднее время выполнения, секунды
Games of Thrones	14	8	57,143	0,005	0,004	0,004	0,004	0,011	0,008	0,009	0,009
gistfile1	1032	449	43,508	0,168	0,145	0,109	0,141	0,164	0,139	0,151	0,151
war and peace	2556	1337	52,308	0,433	0,259	0,259	0,317	0,417	0,309	0,28	0,335
5mb examplefile com	5111	2673	52,299	0,527	0,541	0,513	0,527	0,571	0,597	0,568	0,579
10mb examplefile com	10241	5383	52,563	0,943	0,863	0,864	0,890	0,989	0,943	1	0,977
20mb examplefile com	20481	10766	52,566	1,44	1,425	1,419	1,428	1,427	1,441	1,468	1,445
40mb examplefile com	40961	21531	52,565	2,753	2,731	2,733	2,739	2,821	2,855	2,876	2,851
100mb examplefile com	102401	53826	52,564	6,883	6,707	6,704	6,765	7,753	7,022	7	7,258



Исследование: bmp-файлы

Файл					Encode				Decode			
	Название	Размер до сжатия, Кб	Размер после сжатия, Кб	Коэффициент сжатия, %	Запуск 1	Запуск 2	Запуск 3	Среднее время выполнения, секунды	Запуск 1	Запуск 2	Запуск 3	Среднее время выполнения, секунды
Черно-белые	wolf-white-black	1006	533	52,982	0,082	0,076	0,083	0,080	0,105	0,093	0,106	0,101
	birds-white-black	2514	1155	45,943	0,199	0,189	0,184	0,191	0,219	0,208	0,209	0,212
	patterns-white-blac	3280	1308	39,878	0,254	0,225	0,219	0,233	0,284	0,258	0,246	0,263
	metro-white-black	19997	13869	69,355	1,664	1,649	1,642	1,652	1,648	1,639	1,654	1,647
	feild-white-black	32690	25772	78,838	3,211	3,248	3,266	3,242	3,161	3,112	3,144	3,139
	fire-white-black	47125	20048	42,542	3,024	3,019	3,038	3,027	3,281	3,182	3,17	3,211
	NN-white-black	62833	47344	75,349	5,821	5,766	5,859	5,815	6,519	5,866	6,329	6,238
Цветные	metro-colour	19998	14335	71,682	1,84	1,819	1,799	1,819	1,827	1,788	1,827	1,814
	feild-colour	32693	26035	79,635	3,459	3,411	3,451	3,440	3,353	3,28	3,354	3,329
	fire-colour	47125	16010	33,973	2,911	2,881	2,889	2,894	2,955	3,063	3,021	3,013
	NN-colour	62833	51069	81,277	6,624	6,579	6,568	6,590	7,747	6,99	7,083	7,273

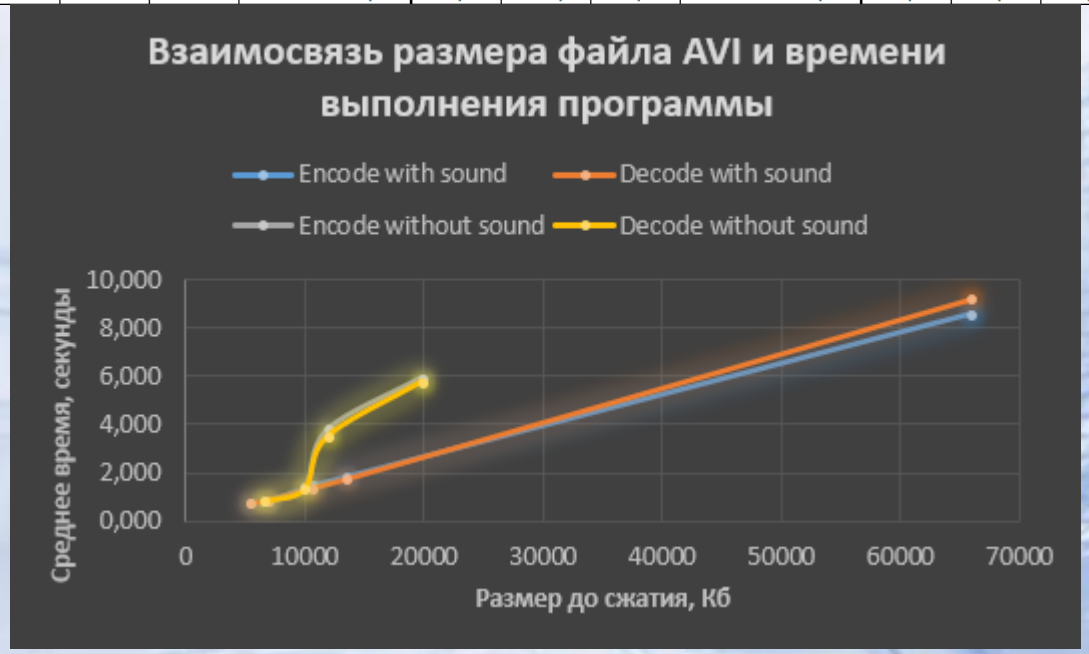


Исследование: wav-файлы

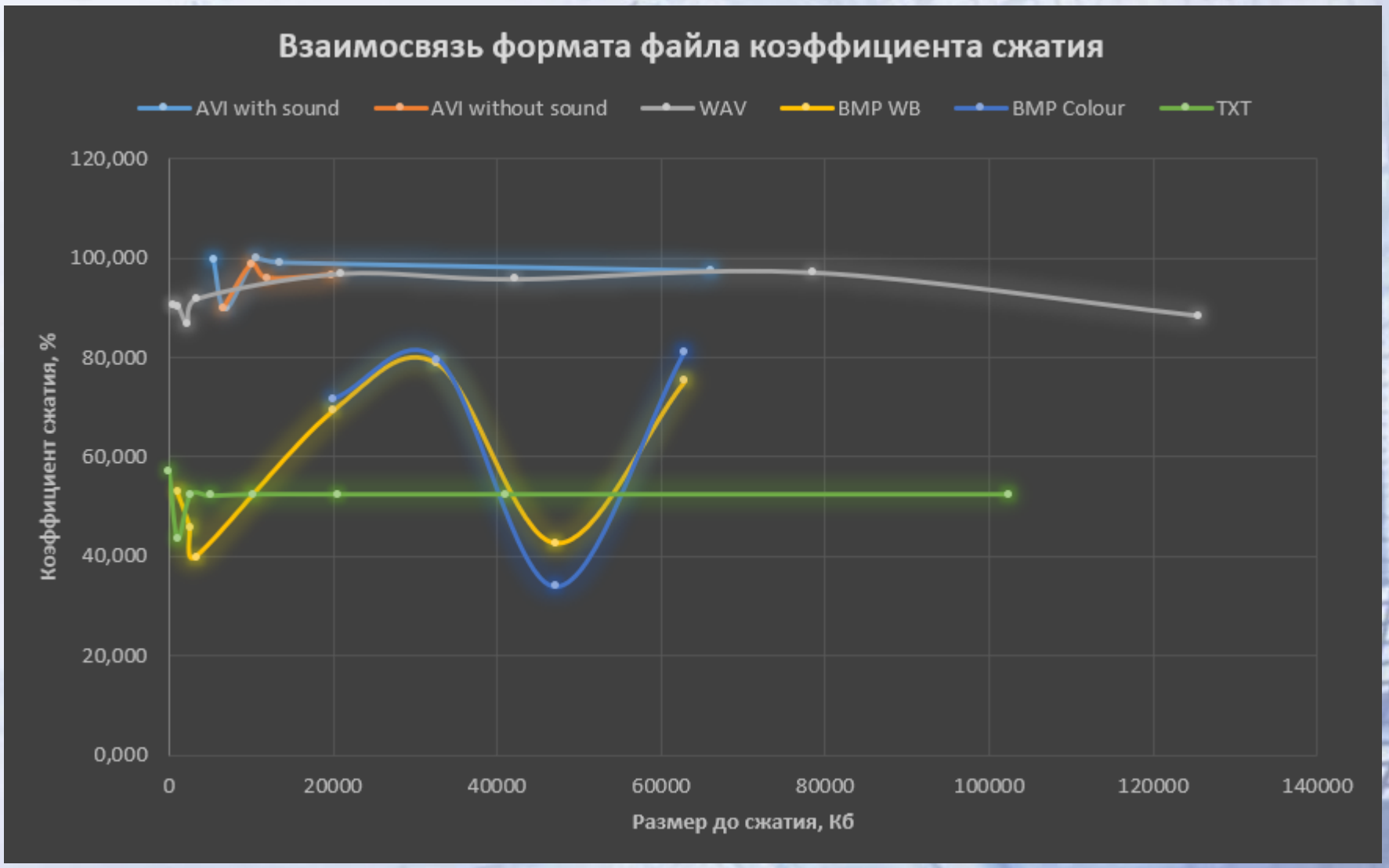
Файл				Encode				Decode			
Название	Размер до сжатия, Кб	Размер после сжатия, Кб	Коэффициент сжатия, %	Запуск 1	Запуск 2	Запуск 3	Среднее время выполнения, секунды	Запуск 1	Запуск 2	Запуск 3	Среднее время выполнения, секунды
TestMelody1	551	500	90,744	0,172	0,086	0,112	0,123	0,143	0,127	0,094	0,121
TestMelody2	1102	994	90,200	0,226	0,168	0,223	0,206	0,223	0,18	0,196	0,200
TestMelody3	2203	1915	86,927	0,475	0,361	0,324	0,387	0,491	0,375	0,42	0,429
TestMelody4	3304	3035	91,858	0,594	0,543	0,513	0,550	0,587	0,61	0,644	0,614
TestMelody5	21033	20401	96,995	3,792	3,994	3,786	3,857	3,608	3,61	3,862	3,693
TestMelody6	42113	40379	95,883	7,166	7,238	7,224	7,209	7,092	7,266	7,171	7,176
TestMelody7	78561	76392	97,239	13,245	13,171	10,083	12,166	15,07	13,078	10,716	12,955
TestMelody8	125425	110938	88,450	16,041	16,778	15,401	16,073	16,537	16,04	17,932	16,836

Исследование: avi-файлы

Файл					Encode				Decode			
	Название	Размер до сжатия, Кб	Размер после сжатия, Кб	Коэффициент сжатия, %	Запуск 1	Запуск 2	Запуск 3	Среднее время выполнения, секунды	Запуск 1	Запуск 2	Запуск 3	Среднее время выполнения, секунды
Со звуком	aaaaa-with	5465	5461	99,927	0,788	0,73	0,733	0,750	0,74	0,738	0,756	0,745
	Moments-with	6935	6236	89,921	0,85	0,871	0,853	0,858	0,877	0,878	0,88	0,878
	tiktok-with	10606	10603	99,972	1,616	1,425	1,435	1,492	1,447	1,359	1,345	1,384
	blure-with	13439	13326	99,159	1,881	1,856	1,827	1,855	1,764	1,75	1,752	1,755
	lection-with	65966	64310	97,490	8,48	8,586	8,724	8,597	9,334	8,996	9,333	9,221
Без звука	Moments-without	6581	5913	89,850	0,896	0,811	0,809	0,839	0,868	0,823	0,828	0,840
	aaaaa-without	10024	9900	98,763	1,424	1,385	1,377	1,395	1,409	1,318	1,312	1,346
	tiktok-without	11921	11449	96,041	3,814	3,647	3,93	3,797	3,52	3,508	3,527	3,518
	blure-without	19808	19157	96,713	5,809	5,86	6,158	5,942	5,712	5,891	5,737	5,780



Исследование: выводы



Над проектом работали

Буинская Анастасия
Тигина Алёна