МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №2.1

по дисциплине: Дискретная математика
тема: «Алгоритмы порождения комбинаторных объектов»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
ст. пр. Рязанов Юрий Дмитриевич
ст. пр. Бондаренко Татьяна Владимировна

Белгород 2023 г.

Лабораторная работа №2.1

Алгоритмы порождения комбинаторных объектов

Цель работы: изучить основные комбинаторные объекты, алгоритмы их порождения, программно реализовать и оценить временную сложность алгоритмов.

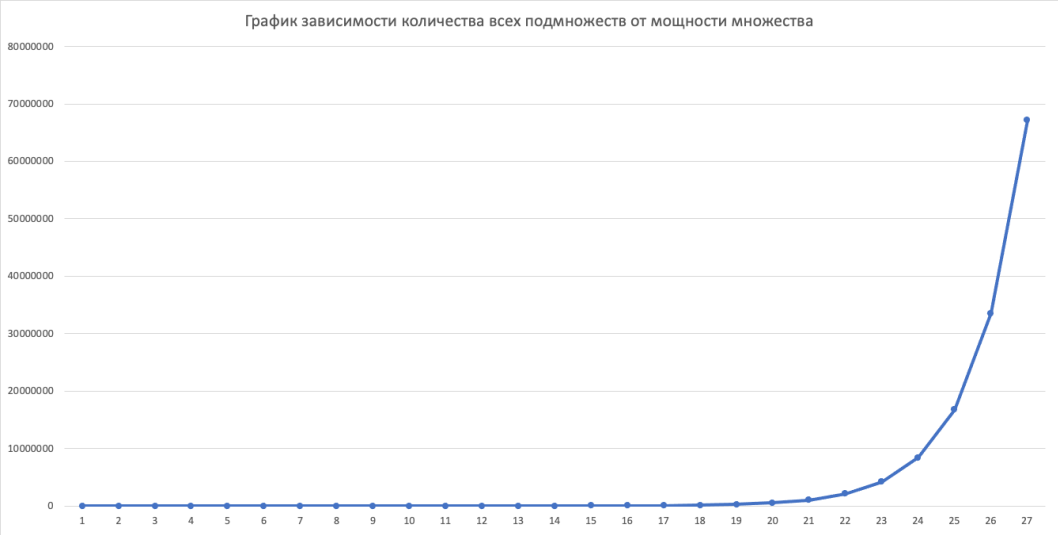
№1. Реализовать алгоритм порождения подмножеств.

```
template <typename T>
std::vector<std::vector<T>> getSubsets(std::vector<T>& baseSet, std::vector<T> currentSet,
size_t count) {
        // Если мы достигли конца исходного множества, возвращаем текущее множество
        if (count == baseSet.size())
                return {currentSet};

        // Добавляем в массив подмножеств шаг, если мы не добавляем на текущем шаге новый
↪  элемент
        std::vector<std::vector<T>> resultSubsets = getSubsets(baseSet, currentSet, count
↪  + 1);

        // Добавляем в текущее подмножество элемент массива
        currentSet.push_back(baseSet[count]);

        // Добавляем в массив подмножеств шаг, если мы добавляем на текущем шаге новый
↪  элемент
        std::vector<std::vector<T>> branch = getSubsets(baseSet, currentSet, count + 1);
        resultSubsets.insert(std::end(resultSubsets), std::begin(branch),
↪  std::end(branch));

        return resultSubsets;
}
```

Мы реализовали рекуррентный алгоритм порождения подмножеств. Функция getSubsets возвращает массив всех подмножеств исходного множества baseSet

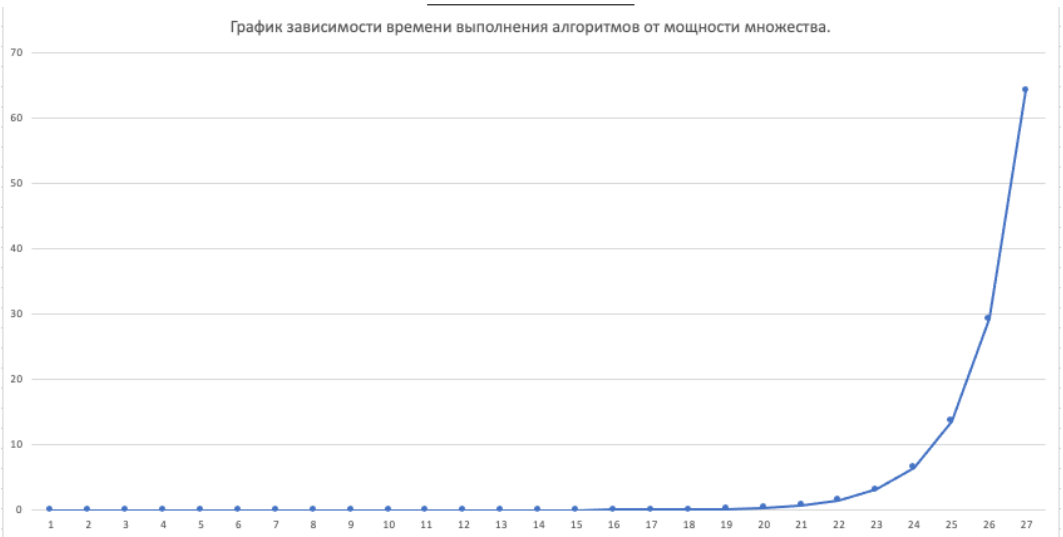
№2. Построить график зависимости количества всех подмножеств от мощности множества.

0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768
16	65536
17	131072
18	262144
19	524288
20	1048576
21	2097152
22	4194304
23	8388608
24	16777216
25	33554432
26	67108864



№3. Построить графики зависимости времени выполнения алгоритмов п.1 на вашей ЭВМ от мощности множества.

0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0,001
11	0,005
12	0,003
13	0,007
14	0,01
15	0,02
16	0,038
17	0,079
18	0,164
19	0,348
20	0,705
21	1,494
22	3,121
23	6,5
24	13,623
25	29,204
26	64,215



№4. Определить максимальную мощность множества, для которого можно получить все подмножества не более чем за час, сутки, месяц, год на вашей ЭВМ.

Воспользовавшись методом наименьших квадратов для функции $t = a \cdot 2^N + b$ на основе полученных точек получили обобщенную аппроксимацию $a = 9.35885430168442 \cdot$

$10^{-7}, b = -0.225015460816618$
$t = 9.35885430168442 \cdot 10^{-7} \cdot 2^N - 0.225015460816618$

t	N
3600 с. = 1 час	28.519 = 28
86400 с. = 1 сутки	33.104 = 33
2678400 с. = 1 месяц (31 день)	38.058 = 38
31536000 с. = 1 год (365 дней)	41.616 = 41

№5. Определить максимальную мощность множества, для которого можно получить все подмножества не более чем за час, сутки, месяц, год на ЭВМ, в 10 и в 100 раз быстрее вашей.

$$t_{\text{(ЭВМ мощнее в 10 раз)}} = \frac{9.35885430168442 \cdot 10^{-7} \cdot 2^N - 0.225015460816618}{10}$$
$$t_{\text{(ЭВМ мощнее в 100 раз)}} = \frac{9.35885430168442 \cdot 10^{-7} \cdot 2^N - 0.225015460816618}{100}$$

t	$N_{\text{(ЭВМ мощнее в 10 раз)}}$	$N_{\text{(ЭВМ мощнее в 100 раз)}}$
3600 с. = 1 час	31.841 = 31	35.163 = 35
86400 с. = 1 сутки	36.426 = 36	39.748 = 39
2678400 с. = 1 месяц (31 день)	41.38 = 31	44.702 = 44
31536000 с. = 1 год (365 дней)	44.938 = 44	48.26 = 48

№6. Реализовать алгоритм порождения сочетаний.

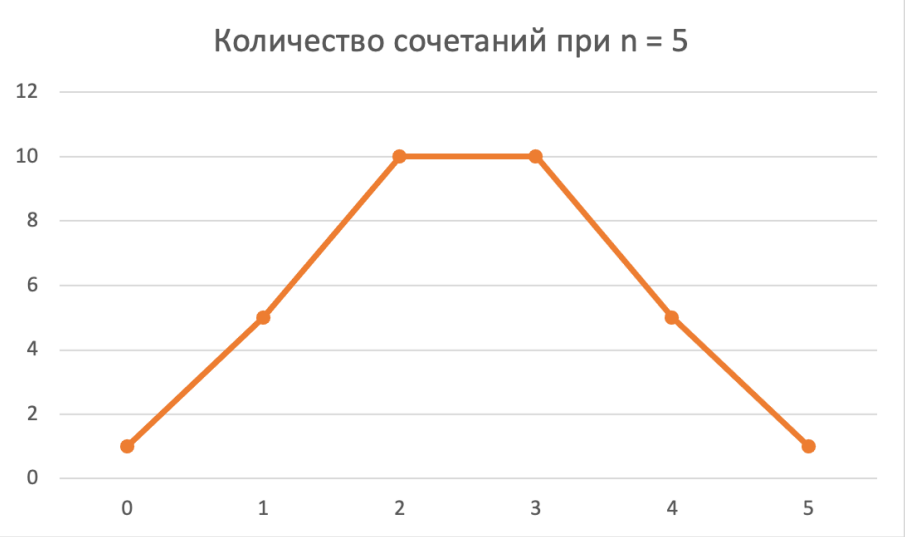
```
template<typename T>
std::vector<std::vector<T>> getCombinations(std::vector<T> &baseSet, std::vector<T>
↪  currentSet, size_t minIndex, size_t k, size_t count) {
        std::vector<std::vector<T>> resultCombs;

        // Если количество перестановок равно необходимому, мы достигли искомого
↪  множества, возвращаем его
        if (count >= k)
                return {currentSet};

        for (size_t i = minIndex; i <= baseSet.size() - k + count; i++) {
                // Добавляем в текущее множество новый элемент, Ci = x
                std::vector<T> newCurrentSet(currentSet);
                newCurrentSet.push_back(baseSet[i]);

                // Вызываем следующий шаг итерации, сохраняем его результат в общий массив
↪  множеств
                std::vector<std::vector<T>> combinations = getCombinations(baseSet,
↪  newCurrentSet, i + 1, k, count + 1);
                resultCombs.insert(std::begin(resultCombs), std::begin(combinations),
↪  std::end(combinations));
        }

        // Возвращаем массив множеств
        return resultCombs;
}
```

Мы реализовали рекуррентный алгоритм порождения сочетаний. Функция getCombinati возвращает массив всех сочетаний исходного множества baseSet

№7. Построить графики зависимости количества всех сочетаний из n по k от k при n= (5, 7, 9).

n = 5

k	Кол-во сочетаний
0	1
1	5
2	10
3	10
4	5
5	1



n = 7

k	Кол-во сочетаний
0	1
1	7
2	21
3	35
4	35
5	21
6	7
7	1

n = 9

k	Кол-во сочетаний
0	1
1	9
2	36
3	84
4	126
5	126
6	84
7	36
8	9
9	1

№8. Реализовать алгоритм порождения перестановок.

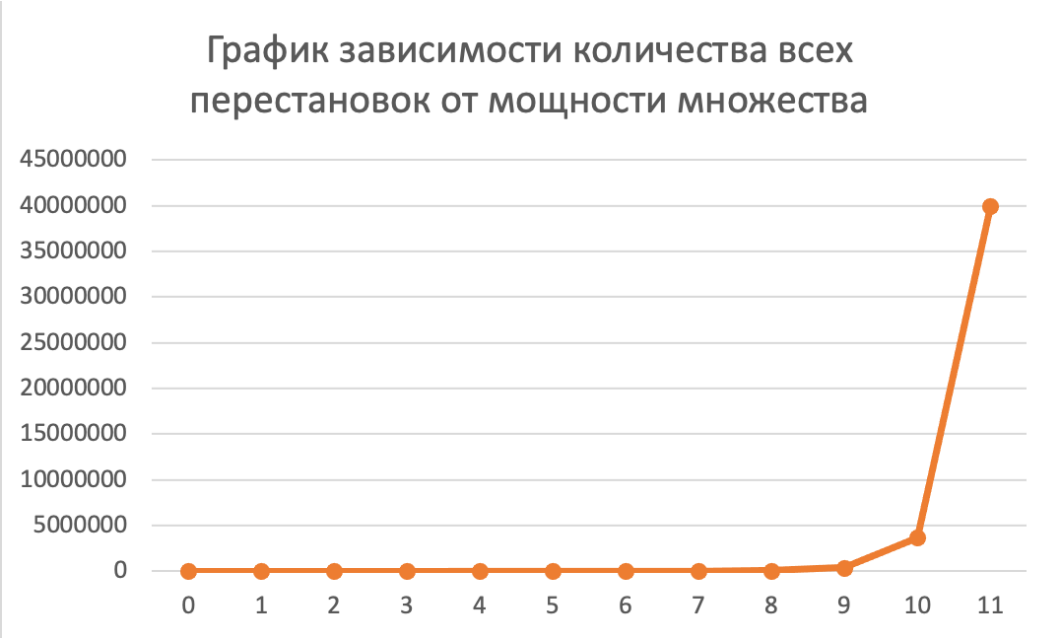
```
template<typename T>
std::vector<std::vector<T>> getPermutations(std::vector<T> baseSet, std::vector<T>
↪  currentSet) {
```

```
        std::vector<std::vector<T>> resultPerms;

        // Если элементов в изначальном множестве не осталось, получено искомое множество
        if (baseSet.size() == 0) return {currentSet};

        for (size_t i = 0; i < baseSet.size(); i++) {
                // Удаляем из исходного массива x
                std::vector<T> newBaseSet(baseSet);
                newBaseSet.erase(std::begin(newBaseSet) + i);

                // Добавляем в текущее множество новый элемент
                std::vector<T> newCurrentSet(currentSet);
                newCurrentSet.push_back(baseSet[i]);

                // Выполняем следующий шаг итерации, сохраняем в итоговый массив множеств
                auto permutations = getPermutations(newBaseSet, newCurrentSet);
                resultPerms.insert(std::begin(resultPerms), std::begin(permutations),
↪   std::end(permutations));
        }

        return resultPerms;
}
```

Мы реализовали рекуррентный алгоритм порождения перестановок. Функция getPermut возвращает массив всех перестановок исходного множества baseSet

№9. Построить график зависимости количества всех перестановок от мощности множества.

0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800
11	39916800



№10. Построить графики зависимости времени выполнения алгоритма п.8 на вашей ЭВМ от мощности множества.

0	0
1	0
2	0
3	0
4	0
5	0
6	0,003
7	0,016
8	0,111
9	1,043
10	11,39
11	137,71

№11. Определить максимальную мощность множества, для которого можно получить все перестановки не более чем за час, сутки, месяц, год на вашей ЭВМ.

Воспользовавшись методом наименьших квадратов для функции $t = a \cdot N!$ на основе полученных точек получили обобщенную аппроксимацию $a = 3.4473277768732325 \cdot 10^{-6}$

$t = 3.4473277768732325 \cdot 10^{-6} \cdot N!$

t	N
3600 с. = 1 час	12.307 = 12
86400 с. = 1 сутки	13.531 = 13
2678400 с. = 1 месяц (31 день)	14.81 = 14
31536000 с. = 1 год (365 дней)	15.704 = 15

№12. Определить максимальную мощность множества, для которого можно получить все подмножества не более чем за час, сутки, месяц, год на ЭВМ, в 10 и в 100 раз быстрее вашей.

$$t_{\text{(ЭВМ мощнее в 10 раз)}} = \frac{3.4473277768732325 \cdot 10^{-6} \cdot N!}{10}$$
$$t_{\text{(ЭВМ мощнее в 100 раз)}} = \frac{3.4473277768732325 \cdot 10^{-6} \cdot N!}{100}$$

t	$N_{\text{(ЭВМ мощнее в 10 раз)}}$	$N_{\text{(ЭВМ мощнее в 100 раз)}}$
3600 с. = 1 час	13.198 = 13	14.067 = 14
86400 с. = 1 сутки	14.393 = 14	15.237 = 15
2678400 с. = 1 месяц (31 день)	15.645 = 15	16.466 = 16
31536000 с. = 1 год (365 дней)	16.523 = 16	17.329 = 17

№13. Реализовать алгоритм порождения размещений.

```
template<typename T>
std::vector<std::vector<T>> getPlacements(std::vector<T> baseSet, std::vector<T>
↪  currentSet, size_t k) {
        // k не может быть больше размера исходного множества
        k = std::min(baseSet.size(), k);

        // Если k = 0, то количество перестановок закончилось или закончился исходный
↪  массив. В обоих случаях мы достигли
        // искомого множества, возвращаем его.
        if (k == 0) return {currentSet};

        std::vector<std::vector<T>> resultPerms;

        for (size_t i = 0; i < baseSet.size(); i++) {
                // Удаляем из исходного массива x
                std::vector<T> newBaseSet(baseSet);
                newBaseSet.erase(std::begin(newBaseSet) + i);

                // Добавляем в текущее множество новый элемент
                std::vector<T> newCurrentSet(currentSet);
```

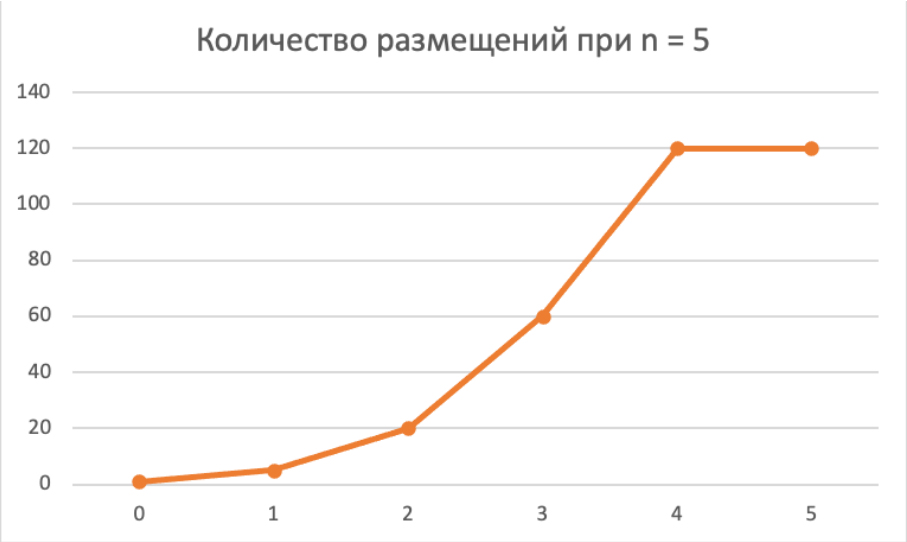
```
                newCurrentSet.push_back(baseSet[i]);

                // Выполняем следующий шаг итерации, сохраняем в итоговый массив множеств
                auto permutations = getPlacements(newBaseSet, newCurrentSet, k - 1);
                resultPerms.insert(std::begin(resultPerms), std::begin(permutations),
↪   std::end(permutations));
        }

        return resultPerms;
}
```

Мы реализовали рекуррентный алгоритм порождения размещекний. Функция getPlacem возвращает массив всех размещений исходного множества baseSet

№14. Построить графики зависимости количества всех сочетаний из n по k от k при n= (5, 7, 9).

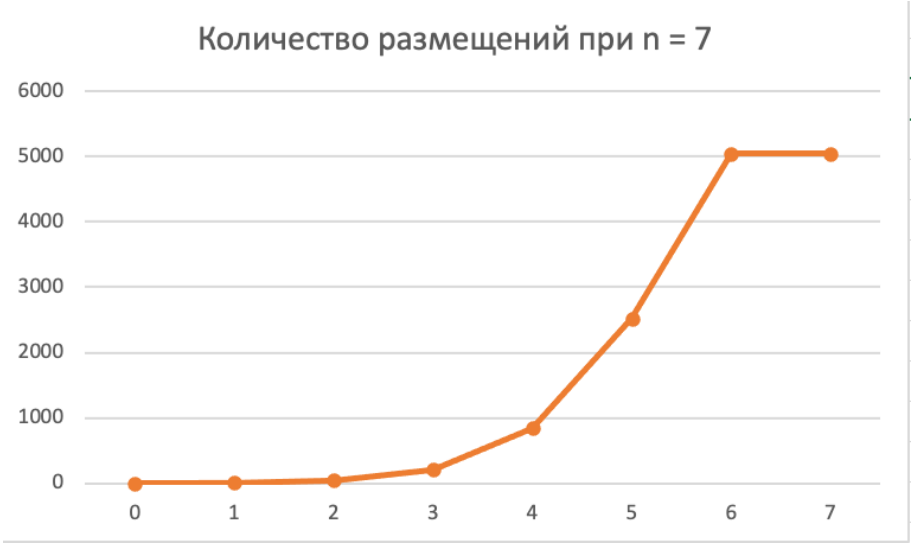
n = 5

k	Кол-во сочетаний
0	1
1	5
2	20
3	60
4	120
5	120



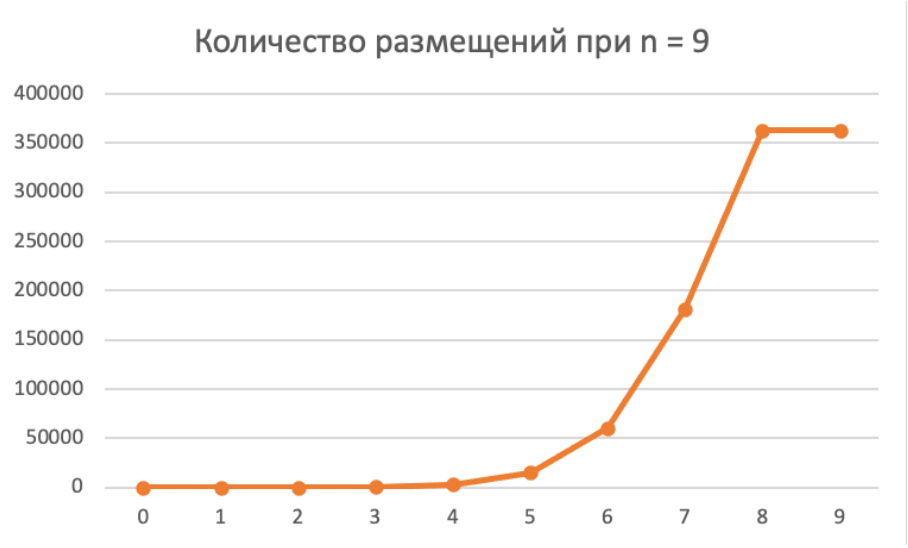
n = 7

k	Кол-во сочетаний
0	1
1	7
2	42
3	210
4	840
5	2520
6	5040
7	5040



n = 9

k	Кол-во сочетаний
0	1
1	9
2	72
3	504
4	3024
5	15120
6	60480
7	181440
8	362880
9	362880



Вывод: в ходе лабораторной работы изучили основные комбинаторные объекты, алгоритмы их порождения, программно реализовали и оценили временную сложность алгоритмов.