1) Ваше решение ошибается на единичку: не рассмотрен случай, когда все коэффициенты линейной комбинации равны 1. Кроме того, подключается несуществующий в репозитории заголовочный файл direct.h.

Косяк с недосчётом всех линейных комбинаций исправлен. *direct.h* в коде больше не использую

2) Прикладываем набор тестовых файлов. Для удобства тестирования на наборе файлов (и чтобы не использовать платформозависимый метод `_chdir`) реализуйте возможность передавать названия входного и выходного файлов в виде аргументов командной строки.

Из программы убрал работу с каталогами вообще и теперь подразумевается, что программа вызывается из директории, где лежит .exe, через cmd. Вводим название программы и файл входных данных.

3) Результат работы программы зависит от наличия в конце файла пустой строки. Попробуйте это исправить.
Если во входном файле содержатся ошибки, выводите сообщение об этом в поток для ошибок `std::cerr`.

Исправлено. Проверка идет по принципу разной длины строки. Также проверем, чтобы только 0/1 в файле входных данных.

4)В программе объявлен класс Set, однако он созвучен с названием контейнера std::set, который обладает свойствами множества. Подобных смысловых пересечений стоит избегать, поскольку Set на самом деле похож на контейнер `std::vector`.

Вся структура кода была переработана и частично убрана избыточность кода.

5) В методе calcWeigth операция + применяется к векторам различной длины. Такая возможность реализована с помощью добавления нулей в начало вектора, однако не совсем понятно, чем оправдано такое решение. Скорее всего, это логическая ошибка. Поэлементные операции следует применять к векторам одинаковой длины.

Исправлено.

6) Для сортировки лучше всего использовать `std::sort`, поскольку собственные реализации неуниверсальны и могут быть недостаточно тщательно отлажены. Например, проверьте, является ли ваша реализация сортировки устойчивой (stable sort).

В новой версии сортировка не понадобилась.

7) Вычисление факториала при помощи рекурсии довольно неэффективно, потому что одни и те же значения будут многократно перевычислены. Следует либо использовать мемоизацию, либо отказаться от рекурсии и однократно предварительно подсчитать все необходимые коэффициенты прямым прохождением.
Более того, биномиальные коэффициенты могут быть эффективно подсчитаны без факториалов по рекуррентной формуле, по которой и получается так называемый «треугольник Паскаля».
Однако зачем вообще нужны эти вычисления? Кажется, распределить вычисления между потоками можно и без таких сложных вычислений. Тем более, нет необходимости повторять их независимо в каждом потоке. Впрочем, можно разделить вычисления между потоками и без явного составления расписания.

Количество параллельных потоков можно выбрать равным `std::thread::hardware_concurrency`.

Вычисления эти сделаны были так как я не имею большого опыта в написании многопоточных программ. Поэтому подумал, что сделать некую схему, которая распределяет вычисления по количеству комбинаций при n единиц в векторе длиной k (n<=k; n>=0). В новой реализации убрал, чтобы вы посмотрели то или не то вы имели в виду.

8) В комментариях упоминается код Грея и некий «порядок минимального изменения», однако это свойство никак не используется. Зачем тогда эта последовательность применяется в решении? Подробно поясните это (а также логику применения биномиальных коэффициентов) в readme-файле.

Код Грея в порядке минимального изменения используется для перебора вариантов подмножества от множества бинарных слов с n количеством единиц. То есть мы можем перебирать все варианты бинарного слова (в данном случае вектора), где содержится ровно n единиц. Это используется при переборе вариантов. В новой версии кода за этот перебор отвечает функция enumeration (в старой версии функция gray). Данная функция перебирает варианты линейной комбинации в зависимости от того, сколько должно быть должно быть единиц в данной линейной комбинации

Биномиальные коэффициенты использовались для того, чтобы вычислить сколько вариантов необходимо просмотреть для n единиц в линейной комбинации и на основании этого сделать расписание для параллельных вычислений.

9) Кажется, в многопоточной версии генерируемая последовательность кодом Грея не является. Кстати, проверьте, сохраняет ли сгенерированное выше расписание порядок в коде Грея?

Конструкции вида

```
if (part) part = 1;
if (scheme.at(i)) scheme.at(i) = 1;
```

выглядят избыточными, лучше правильным образом выполнить инициализацию переменных.

Условие в цикле

```
for (size_t i = 0; i < V.getLength(); i++)
{
   if (i >= V.getLength() - order)
   V[i] = 0;
}
```

можно заменить на правильную инициализацию цикла.

Заполнение векторов `std::vector` ***оптимальнее выполнять при помощи конструкторов или метода resize.***
Операцию сложения по модулю 2 эффективнее выполнять при помощи оператора ^. Кстати, этот оператор поддерживают контейнеры `std::valarray` ***и*** `std::bitset`***.***

Используйте в коде квалификаторы const и static, они улучшают читабельность и связность программы. Используйте namespace.

Придерживайтесь в оформлении кода какого-нибудь coding style, например, такого: ***https://inkscape.org/develop/coding-style/***

Учтено в новой версии