Общие требования:

1) Проект на git'e.

2) Наличие интерактивного диалогового интерфейса для проверки корректности разработанной программы.

3) Корректное завершение программы, как в случае штатного выхода, так и в случае невосстановимых ошибок (без утечек и без использования функций мгновенного завершения программы exit, abort, std::terminate и пр.).

4) Проверка корректности ввода (работа только через потоки C++, т. е. без scanf/printf/fscanf/fprintf). В случаях ошибок формата ввода запрос повторного ввода данных. В случае невосстановимых ошибок ввода-вывода завершение программы.

5) Использование средств языка C++ для работы с динамической памятью – операторов new и delete (malloc, calloc, realloc, free запрещены).

6) Использование исключений для обработки ошибочных ситуаций (вместо кодов возврата).

7) Предпочтительно использование стандартных библиотек и функций языка C++ вместо библиотек и функций языка C (std::copy вместо memcpy, std::abs вместо abs, cstring вместо string.h и т.д.).

8) Логичная и удобная структура проекта, где каждая единица (файл/библиотека) обладает своей единой зоной ответственности (каждый класс в своих файлах .h и .cpp, диалоговые функции и main в своих).

9) Наличие средств автосборки проекта (желательно CMake, qmake и прочие, работающие “поверх” Makefile; использование самописного Makefile нежелательно, но допустимо).

10) Статический анализ кода, встроенный инструментарий в IDE (пр. VS2019: Analyze->Run Code Analysis, см. также Project -> Properties -> Configuration Properties -> Code Analysis -> Microsoft -> Active Rules) или внешние инструменты (Sonarqube + extensions, Clang Static Analyzer и д.р.) (обязательно знакомство с инструментом, исправление всех замечаний или обоснование в комментарии почему конкретное замечание исправить нельзя).

11) Динамический анализ на утечки памяти, встроенный инструментарий в IDE / библиотеки (Пр., VS2019) или внешние инструменты (valgrind, Deleaker и т.п.). Отсутствие утечек памяти и прочих замечаний анализатора.

12) Не "кривой", не избыточный, поддерживаемый и расширяемый код (разумная декомпозиция, DRY, корректное использование заголовочных файлов и т.п.).

13) Стандарт языка C++20 (рекомендуется). Допустим C++17 (если почему-то нет C++20).

Требования задачи:

1) Выбранная структура (класс) матрицы должна учитывать специфику разреженной матрицы (оптимальность по памяти), и по сложности алгоритмов из задания (оптимальность по времени обработки). Хранение нулевых элементов в разреженной матрице запрещено. Допустимы порядковые отклонения от оптимального решения в случае обоснования решения - "trade-off" анализ.
2) Использование ООП возможно (но при полностью корректной реализации, т.е. наличие необходимых конструкторов / деструкторов, перегрузка операторов, корректная сигнатура методов и т.п.), большинству не рекомендуется (дождитесь задач №2).

3) Использование контейнеров из STL возможно только для "профессионалов" в C++, подавляющему большинству не рекомендуется (дождитесь задачи №3).