Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2 ПО ДИСЦИПЛИНЕ: ТИПЫ И СТРУКТУРЫ ДАННЫХ

Записи с вариантами. Обработка таблиц

Студент **Ширяев А.А.**

Группа **ИУ7-33Б**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Ширяев А.А.**

Преподаватель _____ **Никульшина Т.А.**

2024

Цель работы - приобрести навыки работы с типом данных «запись» («структура») содержащим вариантную часть, и с данными, хранящимися в таблицах. Оценить

относительную эффективность программы (в процентах) по времени и по используемому объему памяти в зависимости от используемого алгоритма и от объема сортируемой информации

Условие задачи

ВАРИАНТ 6 Вывести список квартир, содержащий: адрес, общую площадь, количество комнат, стоимость квадратного метра, тип: 1. Первичное a. С отделкой / без отделки 2. Вторичное a. Год постройки b. Количество предыдущих собственников c. Были ли животные Вывести всё вторичное двухкомнатное жилье в указанном ценовом диапазоне без животных

ПРИМЕЧАНИЕ! В отчёте будут использоваться обозначения для действий:
<read> - чтение таблицы из файла
<show> - Вывод таблицы
<add> - Добавление квартиры в таблицу
<del> - Удаление квартиры в таблицу
<sort> - Сортировка таблицы
<byprice> - Вывод всёх вторичных двухкомнатных квартир в указанном ценовом диапазоне без животных
<statistics> - Получение и вывод статистики по сортировкам (и создание массива ключей)
<keys_output_by_aparts> - Вывод таблицы квартир по массиву ключей
<keys_output> - Вывод массива ключей
<save> - Сохранение таблицы в файл
<exit> - Выход из программы

Описание техзадачи

a. Описание исходных данных

Данные на входе: Меню. Код действия. Далее для каждого действия:

<read> - Название файла, файл
<add> - Индекс, в какой место необходимо добавить квартиру, информация от квартиры
<del> - Удаление квартиры по индексу
<sort> - Код сортировки, код ключа сортировки
<byprice> - Минимальная цена, максимальная цена
<save> - Название файла, файл

Данные на выходе:

<read> -Таблица, массив ключей

<show> - Информация таблицы
<add> - Таблица, массив ключей
<del> - Таблица, массив ключей
<sort> - Таблица
<byprice> - Информация по вторичным квартирам с 2-мя комнатами без животных из введенного диапазона
<statistics> - Информация по среднему времени выполнения сортировок, выполненной разными способами, используя таблицу и массив ключей (в микросекундах). Процентное соотношение относительно менее эффективного способа. Процентное соотношение ресурсозатратности (по памяти) (в процентах)
<keys_output_by_aparts> - Таблица по массиву ключей, массив ключей
<keys_output> - Массив ключей
<save> - Изменённый файл с таблицей

b. Описание задачи, реализуемой программой

Программа реализует ряд действий, совершаемых над таблицей:

<read> - считывание данные с файла
<show> - Вывод данных таблицы
<add> - Добавление информации о квартире в таблицу
<del> - Удаление информации о квартире из таблицы
<sort> - Сортировка таблицы
<byprice> - Вывод всёх вторичных двухкомнатных квартир в указанном ценовом диапазоне без животных
<statistics> - Получение и вывод статистики по сортировкам (и создание массива ключей)
<keys_output_by_aparts> - Вывод таблицы по массиву ключей
<keys_output> - Вывод массива ключей
<save> - Сохранение таблицы в файл
<exit> - Выход из программы

c. Способ обращения к программе

Для обращения к программе запускается файл *app.exe*.

d. Описание возможных аварийных ситуаций и ошибок пользователя

Программа может не вывести результат, а вывести сообщение об ошибке. Данная ситуация может произойти при условии:

Ошибки пользователя

1. Введен неверный код действия
2. Код действия содержит иные символы
3. <read> Введенная строка с названием файла некорректная (введенная строка пустая или выходит за размер STR_SIZE + 1 (STR_SIZE = 256))
4. <read> Данные файла некорректны
5. <read> Файл пустой
6. <read> Файл с указанным названием не найден
7. <show> Таблица пустая
8. <add> Введенный индекс некорректный (содержит иные символы или выходит за диапазон значений от 1 до <размера таблицы + 1>)
9. <add> Введённый адрес квартиры некорректный (введенная строка пустая или выходит за размер STR_SIZE + 1 (STR_SIZE = 256))
10. <add> Введенная площадь некорректна (содержит иные символы или <= 0)
11. <add> Введенное кол-во комнат некорректное (содержит иные символы или <= 0)
12. <add> Введенная цена за квадратный метр некорректная (содержит иные символы или < 0)
13. <add> Введенный тип квартиры некорректный (значение не равно **primary** или **secondary**)
14. <add> Введенная отделка некорректная (значение не равно **yes** или **no**)
15. <add> Введенный год постройки некорректный (содержит иные символы или < 0)
16. <add> Введенное кол-во предыдущих собственников некорректное (содержит иные символы или < 0)
17. <add> Введенное наличие животных некорректное (значение не равно **yes** или **no**)
18. <del> Введенный индекс некорректный (содержит иные символы или квартиры под этим индексом нет)
19. <del> Таблица пустая
20. <sort> Таблица пустая
21. <sort> Введенный индекс сортировки некорректный (содержит иные символы или выходит за отрезок [1, 2])
22. <sort> Введенный индекс ключа некорректный (содержит иные символы или выходит за представленный в программе диапазон значений [1, 4])
23. <byprice> Введенное минимальное значение некорректное (содержит иные символы)
24. <byprice> Введенное максимальное значение некорректное (содержит иные символы или меньше минимального значения)
25. <byprice> В таблице нет подходящих квартир
26. <byprice> Таблица пустая
27. <statistics> Таблица пустая
28. <keys_output_by_aparts> Таблица пустая
29. <keys_output_by_aparts> Таблицы ключей нет (не производился запуск <statistics>)

30. <keys_output> - Ключей нет
31. <save> Таблица пустая
32. <save> Введенная строка с названием файла некорректная (введенная строка пустая или выходит за размер STR_SIZE + 1 (STR_SIZE = 256))

Аварийные ситуации

1. Программа не смогла выделить необходимую память для работы

Описание внутренних СД

Информация о квартире:

```
typedef struct
{
    char address[STR_SIZE + 1];
    double square;
    int room_count;
    double quad_meter_cost;

    bool is_primary;
    apart_info_t info;
} apart_t;
```

Информация о квартире представляет собой структуру на языке Си, состоящую из:

address - Адрес квартиры (строка длиной STR_SIZE + 1 (STR_SIZE == 256))
square - Площадь квартиры
room_count - Кол-во комнат
quad_meter_cost - Цена за квадратный метр
is_primary - Является ли квартира первичной
info - Информация о первичной/вторичной квартире

Информация о первичной/вторичной квартире:

```
typedef union
{
    primary_t primary;
    secondary_t secondary;
} apart_info_t;
```

Информация о первичной/вторичной квартире представляет собой объединение на языке Си, состоящее из:

primary - Информация о первичной квартире
secondary - Информация о вторичной квартире

Информация о первичной квартире:

```
typedef struct
{
    bool is_with_decoration;
} primary_t;
```

Информация о первичной квартире представляет собой структуру на языке Си, состоящую из:

is_with_decoration - Есть ли отделка (true - да, false - нет)

Информация о вторичной квартире:

```
typedef struct
{
    int build_year;
    int owner_count;
    bool were_there_animals;
} secondary_t;
```

Информация о первичной квартире представляет собой структуру на языке Си, состоящую из:

build_year - Год постройки
owner_count - Кол-во предыдущих владельцев
were_there_animals - Были ли животные (true - да, false - нет)

Информация о ключе:

```
typedef struct
{
    size_t index;
    int room_count;
} keystat_t;
```

Информация о ключе представляет собой структуру на языке Си, состоящую из:

index - Индекс структуры, которой принадлежит данный ключ
room_count - Кол-во комнат (сортировка с ключами проводится исключительно по кол-ву комнат)

Строка данных:

```
char address[STR_SIZE + 1];
```

Строка данных представляет из себя массив типа char, состоящий из **STR_SIZE + 1** элементов (**STR_SIZE** = 256)

Описание алгоритма

Для начала считываются код действия. Далее в зависимости от выбранного действия:

<read> (Чтение таблицы из файла):

```
int aparts_read(apart_t **aparts, size_t *size)
```

- Вводится название файла, из которого нужно будет прочитать таблицу
- Открывает файл на чтение
- Файл проверяется на корректность: Считывается каждая квартира, параллельно получая кол-во квартир. Получается размер **size**

```
bool file_is_incorrect(FILE *f, size_t *size)
```

- Считывается каждая квартира, пока не считается **size** квартир. Считанная квартира записывается в соответствующую область памяти.
- Файл закрывается.

<show> (Вывод таблицы)

```
void aparts_output(apart_t *aparts, size_t size)
```

- По порядку считываются квартиры из выделенной памяти
- Выводится индекс квартиры
- Считанная квартира выводится по шаблону

<add>

```
int aparts_add(apart_t **aparts, size_t *size, size_t i)
```

- Вводится индекс, на месте которого должна стоять квартира (или индекс, больший максимального на 1 для добавления в конец).
- Вводятся данные по квартире
- Выделяется новая память, большая старой на величину размера записи информации о квартире

- Из старой памяти копируются все квартиры с индексом, меньшим индекса записи.
- Оставшиеся квартиры копируются в область памяти, идущей ЗА областью памяти для квартиры с введенным индексом.
- В нужную область памяти копируется информация о новой квартире
- Старая память освобождается

<del>

```
int aparts_delete(apart_t **aparts, size_t *size, size_t i)
```

- Вводится индекс, по которому должна быть удалена информация о квартире
- Выделяется новая память, меньшая старой на величину размера записи информации о квартире
- Данные из старой памяти, не считая данные под индексом, копируются в новую память
- Старая память освобождается

<sort>

- Вводится индекс предложенной сортировки
- Вводится индекс предложенного ключа сортировки
- Выполняется алгоритм выбранной сортировки:

Insertion sort (сортировка вставками):

```
int insertion_sort(void *base, size_t num, size_t size, int (*compare)(const void *, const void *, void *(*)(apart_t *)), void *(*key)(apart_t *))
```

1. Начало: Начинаем со второй записи таблицы (индекс 1 (для пользователя это индекс 2)), так как первый элемент уже считается отсортированным.
2. Выбор элемента: Выбираем текущую запись (**key**) для вставки.
3. Сравнение: Сравниваем **key** с записями до него слева от него (отсортированной части таблицы).
4. Сдвиг: Если элемент слева больше, сдвигаем его вправо.
5. Вставка: Когда находим запись, который меньше или равен **key**, вставляем **key** на его позицию.
6. Повторение: Переходим к следующей записи (индекс +1) и повторяем шаги 2-5, пока не дойдем до конца таблицы.
7. Конец: Таблица отсортирована.

Gnome sort (гномья сортировка):

```
int gnome_sort(void *base, size_t num, size_t size, int (*compare)(const void *, const void *, void *(*)(apart_t *)), void *(*key)(apart_t *))
```

1. Начало: Начинаем с первого элемента (индекс 0 (для пользователя это индекс 1)).
2. Сравнение: Сравниваем текущий элемент с предыдущим.
3. Порядок:
   - Если текущий элемент меньше или равен предыдущему, двигаемся вперед (индекс +1).
   - Если текущий элемент больше предыдущего, меняем их местами и возвращаемся на один элемент назад (индекс -1).
4. Границы: Если индекс текущего становится равен 0, перемещаемся на следующий элемент (индекс +1).
5. Повторение: Продолжаем процесс, пока не пройдем через все записи.
6. Конец: Таблица отсортирована.

<byprice>

```
int aparts_output_by_price(apart_t *aparts, size_t size, double min_price, double max_price)
```

- Вводится минимальная цена диапазона
- Вводится максимальная цена диапазона
- Программа проходится по таблице: если квартира вторичная, имеет 2 комнаты и в ней не было животных, программа выводит эту запись

<statistics>

```
int statistics_get(apart_t *aparts, size_t size, keystat_t *keys)
```

- Выделяется память под ключи

```
int keys_create(apart_t *aparts, size_t size, keystat_t **keys)
```

- Программа циклом проходится по таблице: считывает запись и заполняет соответствующий ключ информацией записи: индексом и кол-вом комнат
- Выделяется память под копию таблицы и копию ключей
- Далее алгоритм **ITER_COUNT** раз (**ITER_COUNT** == 100) по порядку считывает время (в мкс) для: Insertion sort для таблицы, insertion sort для массива ключей, gnome sort для таблицы, gnome sort для массива ключей
    - В выделенную память копируются исходные данные таблицы/ключей
    - Считывается начальное время
    - Выполняется сортировка (см. алгоритм для <sort>)
    - Считывается конечное время
    - К соответствующей переменной прибавляется разница между начальным и конечным временами (в мкс)

- Вычисляется среднее значение времён, необходимых для сортировки разными способами и разными алгоритмами.
- Полученные значения выводятся на экран

<keys_output_by_aparts>

```
void aparts_output_by_keys(apart_t *aparts, size_t size, keystat_t *keys)
```

- Программа циклом проходится по каждому из ключей
- По каждому из ключей сразу же выводится запись с информацией о квартире по индексу в структуре с ключом

<keys_output>

```
void output_keys(keystat_t *keys, size_t size)
```

- Программа проходится по каждому из ключей
- Данные ключей выводятся на экран

<save>

```
int aparts_save(apart_t *aparts, size_t size)
```

- Вводится название файла, в котором нужно сохранить данные
- Программа создает/открывает файл на запись (если файл уже существовал - на перезапись)
- Далее циклом проходится по таблице: считывается запись и записывается в файл
- Файл закрывается

<exit>

```
case CODE_EXIT:
    flag = false;

    if (aparts != NULL && size != 0)
        free(aparts);

    keys_free(&keys);

    break;
```

- Освобождается память, выделенная под таблицу и массив ключей (при наличии)
- Программа завершает свою работу

Набор тестов

В ходе выполнения лабораторной работы была написана тестовая система, при помощи которой проверялись данные ситуации

Позитивные тесты

Номер теста	Входные данные	Ожидаемые выходные данные
01 - <read> Чтение файла	1 data.txt	DATA WAS READ SUCCESSFULLY!

02 - <show> Показ информации по файлу	1 data.txt 2	ALL APARTS:
03 - <add> Добавление элемента в конец	1 data.txt 3 337 Victory Avenue, 12 70.000000 3 22000.000000 primary no	Index: 337 Address: Victory Avenue, 12 Square: 70.000000 Room count: 3 Quad meter cost: 22000.000000 PRIMARY Is with decoration: No
04 - <add> Добавление элемента при пустой таблице	3 1 Victory Avenue, 12 70.000000 3 22000.000000 primary no 2	Index: 1 Address: Victory Avenue, 12 Square: 70.000000 Room count: 3 Quad meter cost: 22000.000000 PRIMARY Is with decoration: No
05 - <del> Удаление единственной записи	3 1 Victory Avenue, 12 70.000000 3 22000.000000 primary no 4 1 2	NO DATA
06 - <sort> Сортировка 5-и записей (insertion sort, key = address)	1 table_5.txt 1 1 2	Index: 1 Address: Garden Street, 20 Square: 55.500000 Room count: 2 Quad meter cost: 29000.000000 PRIMARY Is with decoration: Yes Index: 2 Address: Krylatskaya Street, 3 Square: 40.000000

		Room count: 1 Quad meter cost: 36000.000000 PRIMARY Is with decoration: No Index: 3 Address: Lenin Street, 10 Square: 45.500000 Room count: 2 Quad meter cost: 30000.000000 PRIMARY Is with decoration: Yes Index: 4 Address: Mir Avenue, 25 Square: 60.000000 Room count: 3 Quad meter cost: 25000.000000 PRIMARY Is with decoration: No Index: 5 Address: Pushkin Street, 15 Square: 35.000000 Room count: 1 Quad meter cost: 35000.000000 PRIMARY Is with decoration: Yes
07 - <sort> Сортировка 5-и записей (gnome sort, key = quad_meter_cost)	1 table_5.txt 2 4 2	Index: 1 Address: Mir Avenue, 25 Square: 60.000000 Room count: 3 Quad meter cost: 25000.000000 PRIMARY Is with decoration: No Index: 2 Address: Garden Street, 20 Square: 55.500000 Room count: 2 Quad meter cost: 29000.000000 PRIMARY Is with decoration: Yes Index: 3

		Address: Lenin Street, 10 Square: 45.500000 Room count: 2 Quad meter cost: 30000.000000 PRIMARY Is with decoration: Yes Index: 4 Address: Pushkin Street, 15 Square: 35.000000 Room count: 1 Quad meter cost: 35000.000000 PRIMARY Is with decoration: Yes Index: 5 Address: Krylatskaya Street, 3 Square: 40.000000 Room count: 1 Quad meter cost: 36000.000000 PRIMARY Is with decoration: No
08 - <sort> Сортировка таблицы из единственной записи (gnome sort, key = room_count)	3 1 Victory Avenue, 12 70.000000 3 22000.000000 primary no 5 2 3 2	Index: 1 Address: Victory Avenue, 12 Square: 70.000000 Room count: 3 Quad meter cost: 22000.000000 PRIMARY Is with decoration: No
09 - <byprice> Получение вторичных 2-х комнатных квартир без животных стоимостью от 0 до 10000000 единиц	1 data.txt 6 0 10000000	Index: 336 Address: Krylatskaya Street, 3 Square: 40.000000 Room count: 2 Quad meter cost: 36000.000000 SECONDARY Build year: 2005 Owner count: 23 Were there animals: No

		APARTS FOUNDED!
10 - <byprice> Получение вторичных 2-х комнатных квартир без животных стоимостью от 1440000 до 1440000 единиц	1 data.txt 6 1440000 1440000	Index: 336 Address: Krylatskaya Street, 3 Square: 40.000000 Room count: 2 Quad meter cost: 36000.000000 SECONDARY Build year: 2005 Owner count: 23 Were there animals: No APARTS FOUNDED!
11 - <statistics> Получение данных по сортировке	1 data.txt 7	Time insertion (aparts): Time insertion (keys): Time gnome (aparts): Time gnome (keys):
12 - <keys_output_by_aparts> Вывод информации по ключам после статистики сортировок (<statistics>)	1 table_5.txt 7 8	Index: 2 Address: Krylatskaya Street, 3 Square: 40.000000 Room count: 1 Quad meter cost: 36000.000000 PRIMARY Is with decoration: No Index: 5 Address: Pushkin Street, 15 Square: 35.000000 Room count: 1 Quad meter cost: 35000.000000 PRIMARY Is with decoration: Yes Index: 1 Address: Garden Street, 20 Square: 55.500000 Room count: 2 Quad meter cost: 29000.000000 PRIMARY Is with decoration: Yes

		Index: 3 Address: Lenin Street, 10 Square: 45.500000 Room count: 2 Quad meter cost: 30000.000000 PRIMARY Is with decoration: Yes Index: 4 Address: Mir Avenue, 25 Square: 60.000000 Room count: 3 Quad meter cost: 25000.000000 PRIMARY Is with decoration: No
13 - <save> Сохранение таблицы в файл	1 table_5.txt 9 table_5.txt 1 table_5.txt 2	Index: 1 Address: Garden Street, 20 Square: 55.500000 Room count: 2 Quad meter cost: 29000.000000 PRIMARY Is with decoration: Yes Index: 2 Address: Krylatskaya Street, 3 Square: 40.000000 Room count: 1 Quad meter cost: 36000.000000 PRIMARY Is with decoration: No Index: 3 Address: Lenin Street, 10 Square: 45.500000 Room count: 2 Quad meter cost: 30000.000000 PRIMARY Is with decoration: Yes Index: 4 Address: Mir Avenue, 25 Square: 60.000000 Room count: 3 Quad meter cost: 25000.000000

		PRIMARY Is with decoration: No Index: 5 Address: Pushkin Street, 15 Square: 35.000000 Room count: 1 Quad meter cost: 35000.000000 PRIMARY Is with decoration: Yes
14 - <exit> Завершение программы	10	Have a nice day!

Негативные тесты

Номер теста	Входные данные	Выходные данные
01 - Введён неверный код	11	INVALID CODE
02 - Код с иными символами	1.1	NVALID CODE
03 - <read> Файл не существует	1 grep.txt	THERE IS NO FILE WITH THIS FILENAME
04 - <read> Файл пустой	1 empty_file.txt	NO DATA IN FILE
05 - <read> Файл некорректный	1 incor.txt	INVALID FILE WORKING
06 - <add> Некорректное поле (room_count)	3 1 street, 4 e	ENTERED DATA IS INVALID!
07 - <add> Поле с кол-вом животных имеет отличное от **yes** или **no** значение	3 1 street, 4 40 2 20000 secondary 2005	ENTERED DATA IS INVALID!

	2 meemememe	
08 - <del> Введенный индекс выходит за пределы файла	1 data.txt 4 350	INVALID INDEX
09 - <del> таблица пустая	4	NO DATA
10 - <sort> Индекс сортировки выходит за пределы	1 data.txt 5 3	INVALID SORT CODE!
11 - <sort> Индекс ключа выходит за пределы	1 data.txt 5 1 5	INVALID KEY CODE!
12 - <sort> Таблица пустая	5	NO DATA
13 - <byprice> Некорректная минимальная цена	1 data.txt 6 -1	INVALID MIN VALUE
14 - <byprice> Некорректная максимальная цена	1 data.txt 6 1000 1ghh	INVALID MAX VALUE
15 - <byprice> Минимальная цена больше, чем максимальная	1 data.txt 6 1000 1	MAX PRICE CAN'T BE LOWER THAN MIN PRICE
16 - <byprice> Нет вторичных квартир	1 table_5.txt 6 1 11	NO SECONDARY APARTS WITHOUT ANIMALS WITH 2 ROOMS IN FILE
17 - <byprice> Нет подходящих квартир	1 data.txt	NO DATA TO OUTPUT

	6 2000000 3000000	
18 - <statistics> Таблица пустая	7	NO DATA
19 - <keys_output_by_aparts> Таблица пустая	8	NO DATA
20 - <keys_output_by_aparts> Нет ключей для вывода данных	1 table_5.txt 8	NO STATISTICS! PLEASE, GET STATISTICS FOR OUTPUT BY KEYS
21 - <save> Таблица пустая	9	NO DATA
22 - <save> Некорректное название файла	1 data.txt 1234567891234567891234 5678912345678912345678 9123456789123456789123 4567891234567891234567 8912345678912345678912 3456789123456789123456 7891234567891234567891 2345678912345678912345 6789123456789123456789 1234567891234567891234 5678912345678912345678 9123456789123456789123 4567891234567891234567 8912345678912345678912 3456789123456789123456 7891234567891234567891 2345678912345678912345 6789123456789123456789 1234567891234567891234 5678912345678912345678 9123456789123456789123 4567891234567891234567 8912345678912345678912 3456789123456789123456 7891234567891234567891 2345678912345678912345 6789123456789123456789 1234567891234567891234 5678912345678912345678 9123456789123456789123 4567891234567891234567 8912345678912345678912 3456789123456789123456	INVALID FILENAME

	789123456789.txt	

Оценка эффективности сортировок между собой и сортировок, используя массивом записей и массивом ключей

При помощи программы удалось сравнить сортировки между собой, а также сравнить результаты сортировки, используя массив записей и массив ключей для файла, имеющего 336, 140, 47, 19 и 11 записей.
Программа выполнила замеры времени/памяти и составила статистику по эффективности и ресурсозатратности.

336 записей

```
TIME DATA (in microseconds):
```

```
Time insertion sort (aparts): 7731
Time insertion sort (keys): 663
Time gnome sort (aparts): 10865
Time gnome sort (keys): 1010

SORTS COMPARING (using aparts):
Insertion sort is 40.538093% faster, than Gnome sort

SORTS COMPARING (using keys):
Insertion sort is 52.337858% faster, than Gnome sort

INSERTION SORT COMPARING (using aparts and keys):
Insertion sort is 1066.063348% faster, if we using keys

GNOME SORT COMPARING (using aparts and keys):
Gnome sort is 975.742574% faster, if we using keys

MEMORY (Insertion sort)
Keys requires 4.966422% bigger memory, than aparts sort

MEMORY (Gnome sort)
Keys requires 5.263158% bigger memory, than aparts sort
```

140 записей

```
TIME DATA (in microseconds):
Time insertion sort (aparts): 1445
Time insertion sort (keys): 150
Time gnome sort (aparts): 1848
Time gnome sort (keys): 170

SORTS COMPARING (using aparts):
Insertion sort is 27.889273% faster, than Gnome sort

SORTS COMPARING (using keys):
Insertion sort is 13.333333% faster, than Gnome sort

INSERTION SORT COMPARING (using aparts and keys):
Insertion sort is 863.333333% faster, if we using keys

GNOME SORT COMPARING (using aparts and keys):
Gnome sort is 987.058824% faster, if we using keys

MEMORY (Insertion sort)
Keys requires 4.553938% bigger memory, than aparts sort

MEMORY (Gnome sort)
Keys requires 5.263158% bigger memory, than aparts sort
```

47 записей

```
TIME DATA (in microseconds):
```

```
Time insertion sort (aparts): 178
Time insertion sort (keys): 15
Time gnome sort (aparts): 205
Time gnome sort (keys): 20

SORTS COMPARING (using aparts):
Insertion sort is 15.168539% faster, than Gnome sort

SORTS COMPARING (using keys):
Insertion sort is 33.333333% faster, than Gnome sort

INSERTION SORT COMPARING (using aparts and keys):
Insertion sort is 1086.666667% faster, if we using keys

GNOME SORT COMPARING (using aparts and keys):
Gnome sort is 925.000000% faster, if we using keys

MEMORY (Insertion sort)
Keys requires 3.179825% bigger memory, than aparts sort

MEMORY (Gnome sort)
Keys requires 5.263158% bigger memory, than aparts sort
```

19 записей

```
TIME DATA (in microseconds):
```

```
Time insertion sort (aparts): 49
Time insertion sort (keys): 4
Time gnome sort (aparts): 48
Time gnome sort (keys): 5

SORTS COMPARING (using aparts):
Gnome sort is 2.083333% faster, than Insertion sort

SORTS COMPARING (using keys):
Insertion sort is 25.000000% faster, than Gnome sort

INSERTION SORT COMPARING (using aparts and keys):
Insertion sort is 1125.000000% faster, if we using keys

GNOME SORT COMPARING (using aparts and keys):
Gnome sort is 860.000000% faster, if we using keys

MEMORY (Insertion sort)
Keys requires 0.263158% bigger memory, than aparts sort

MEMORY (Gnome sort)
Keys requires 5.263158% bigger memory, than aparts sort
```

11 записей

```
TIME DATA (in microseconds):
Time insertion sort (aparts): 28
Time insertion sort (keys): 2
Time gnome sort (aparts): 17
Time gnome sort (keys): 2

SORTS COMPARING (using aparts):
Gnome sort is 64.705882% faster, than Insertion sort

SORTS COMPARING (using keys):
Gnome sort and Insertion sort are equal

INSERTION SORT COMPARING (using aparts and keys):
Insertion sort is 1300.000000% faster, if we using keys

GNOME SORT COMPARING (using aparts and keys):
Gnome sort is 750.000000% faster, if we using keys

MEMORY (Insertion sort)
Aparts requires 3.167421% bigger memory, than keys sort

MEMORY (Gnome sort)
Keys requires 5.263158% bigger memory, than aparts sort
```

Таблица результатов:

Кол-во элементов	Insertion sort (aparts), мксек	Insertion sort (keys), мксек	Gnome sort (aparts), мксек	Gnome sort (keys), мксек
336	7731	663	10865	1010
140	1445	150	1848	170
47	178	15	205	20
19	49	4	48	5
11	28	2	17	2

Память:
Сортировка с использование массива ключей требует на 5,263158% больше памяти, чем сортировка непосредственно таблицы.

На основе полученных данных можно сделать следующие заключения:

Сравнение Insertion sort и Gnome sort, используя массив записей (таблицу) (SORTS COMPARING (using aparts))
Сортировка вставками эффективнее гномьей сортировки на большом кол-ве записей, но если размер таблицы небольшой (в окрестности 19 записей и меньше), гномья сортировка становится эффективнее.

Сравнение Insertion sort и Gnome sort, используя массив ключей (SORTS COMPARING (using keys))
Сортировка вставками эффективнее гномьей сортировки и при небольшом кол-ве записей, но с уменьшением размера до совсем маленьких значений (в окрестности 11 и ниже) сортировка вставками становится менее эффективной и при 11 записях имеет примерное одинаковое время с гномьей сортировкой.

Сравнение Insertion sort, используя массив записей (таблицу) и массив ключей (INSERTION SORT COMPARING (using aparts and keys))
В данном случае эффективность сортировки с использованием массива ключей в среднем превышает сортировку с использованием таблицы в 11,882127 раз во всех случаях.

Сравнение Gnome sort, используя массив записей (таблицу) и массив ключей (GNOME SORT COMPARING (using aparts and keys))
В данном случае эффективность сортировки с использованием массива ключей в среднем превышает сортировку с использованием таблицы в 9,995603 раз во всех случаях.

Сравнение памяти, затраченной для Insertion sort с использованием массива записей (таблицу) и массива ключей (MEMORY (Insertion sort))
При большом кол-ве записей затраченная память при использовании массива ключей примерно на 5% больше, чем при использовании таблицы, но при уменьшении записей эта разница уменьшается: при размере записей ниже примерно 19-18 записей массив ключей становится эффективнее по памяти.

Сравнение памяти, затраченной для Gnome sort с использованием массива записей (таблицу) и массива ключей (MEMORY (Gnome sort))
В данном случае кол-во затраченной памяти - константное значение, примерно равное 5.263158% (примерное, так как вещественное число выведено до 6 знаков после точки).

На основе заключений можно сформулировать плюсы и минусы того или иного способа

Сортировка с использованием таблицы:

ПЛЮСЫ
- По памяти эффективнее

МИНУСЫ
- Скорость медленнее, чем у массива ключей
- При малом кол-ве записей эффективность по памяти МОЖЕТ пропасть (в зависимости от сортировки)

Сортировка с использованием массива ключей:

ПЛЮСЫ
- Быстрая скорость выполнения
- При малом кол-ве записей по памяти может быть эффективнее (в зависимости от сортировки)

МИНУСЫ
- При большом кол-ве записей требует больше памяти для реализации, чем сортировка с использованием таблицы

Выводы по проделанной работе

Для представления таблиц в компьютере следует использовать тип данных “запись”. При наличии вариационной части в данных следует использовать тип данных “объединение” для экономии памяти, так как объем памяти, необходимый для записи с вариантами складывается из длин полей фиксированной части и максимального по длине поля вариантной части.. Если таблицу необходимо быстро отсортировать и перед нами не стоит вопрос ресурсозатратности, можно использовать так называемый “массив ключей”, сортировка которого в разы быстрее сортировки обычной таблицы, однако массив ключей требует больше памяти для реализации.

Контрольные вопросы

1. **Как выделяется память под вариантную часть записи?**

Под вариантную часть записи память выделяется на основе наибольшего по ресурсозатратности поля вариантной части. (При отсутствии упаковки) производится выравнивание данных

2. **Что будет, если в вариантную часть ввести данные, не соответствующие описанным?**

В таком случае значение любого из полей вариантного поля не определено

3. **Кто должен следить за правильностью выполнения операций с вариантной частью записи?**

Программист

4. **Что представляет собой таблица ключей, зачем она нужна?**

Таблица ключей - таблица, содержащий идентификатор записи в исходной таблице и значение его поля. Она необходима для ускорения сортировки данных, поиска данных при большом размере таблицы

5. **В каких случаях эффективнее обрабатывать данные в самой таблице, а когда – использовать таблицу ключей?**

Эффективнее обрабатывать данные в самой таблице, если перед нами стоит вопрос ресурсозатратности процесса или таблица не такая большая. В остальных случаях эффективнее использовать таблицу ключей.

6. **Какие способы сортировки предпочтительнее для обработки таблиц и почему?**

Для сортировки предпочтительнее использовать массив ключей, так как при довольно небольшой разнице в памяти (памяти под массив ключей необходимо БОЛЬШЕ, но не всегда) сортировка массива ключей в разы быстрее сортировки непосредственно исходной таблицы.