Федеральное государственное автономное образовательное учреждение
высшего образования
Национальный исследовательский университет
«Высшая школа экономики»

Факультет социально-экономических и компьютерных наук

Самостоятельная работа №1. Задание №2

по дисциплине «Теоретические основы информатики»

Раздел 4 - Сортировка и поиск

Выполнил: студент Яцишин Л.С. (индивидуальная работа), уч. группа ПСАПР-25-2

Пермь, 2025

Содержание

1 Задание

Разработайте процедуры (функции), используя коды на языке C++, разработанные при выполнении задания 1:

1. Построения индекса массива – бинарного дерева – по значениям заданного атрибута записей в массиве (например, по ФИО студента и пр.).
2. Вывода элементов массива (записей) на экран в порядке сортировки в построенном индексе – бинарном дереве: 1) по возрастанию значений ключевого поля; 2) по убыванию значений ключевого поля.
3. Поиска элемента по значению ключевого поля (атрибут) с использованием бинарного дерева – индекса. В случае, если элемент не найден, должно быть выведено соответствующее сообщение. Если элемент с заданным для поиска значением ключа найден в индексе, должны быть выведены значения всех атрибутов из соответствующей записи в массиве введённых данных. Разработайте два варианта процедур поиска по индексу (бинарному дереву): 1) рекурсивную функцию поиска, 2) итерационную функцию поиска.
4. Удаления записи с заданным значением ключевого атрибута из массива данных с соответствующим изменением индекса – бинарного дерева. Удаление может быть выполнено путём внесения пометки об удалении записи в специальное поле (помеченные записи могут быть восстановлены специальной функцией), при этом физическое удаление помеченных для удаления записей выполняется отдельной функцией. По специальной команде записи могут быть удалены физически из массива данных и из индекса (бинарного дерева).
5. Редактирования записей в массиве с возможностью изменения значений атрибутов, по которым выполнена индексация записей массива в бинарном дереве. При этом соответствующие изменения вносятся и в индекс, т. е. может быть перестроено бинарное дерево (индекс), чтобы сохранить порядок сортировки при изменении ключевых значений.

2 Введение

Реализуется продолжение программы, разработанной в задании 1. Предметная область, структура записей, формат файла и базовые операции ввода/вывода сохраняются без изменений и рассматриваются в предыдущем отчёте. Для второго задания изменяется только часть, связанная с индексацией и поиском: вместо индексных массивов используется индекс в виде бинарного дерева по выбранному полю, а также выполняется пересборка этого дерева при изменении данных.

3 Структура данных и форматы

Основной массив Expense (30 элементов) с полями Id, Category, Note, Day, Month, Year, Amount, Deleted.

Формат файла: id;category;note;day;month;year;amount;deleted.

Правила ввода: строки без символов ; и пробелов; дата проверяется по DaysInMonth (Month 1..12, Day в пределах месяца); сумма double > 0; Deleted — логическая метка (0/1).

4 Алгоритмы

Индексное бинарное дерево

- Структура узла: TreeNodeKey, RecIndex, Left, Right, Next, Key — категория, RecIndex — ссылка на запись в массиве.
- Left/Right — указатели на левое и правое поддерево; в левом поддереве хранятся ключи, меньшие текущего, в правом — ключи, большие текущего; Next — цепочка дубликатов ключа.
- Построение: старое дерево освобождается, затем все не удалённые записи из массива вставляются обратно в бинарное дерево поиска.
- Обходы: выполняется симметричный обход (лево–узел–право) для получения категорий по возрастанию и обратный обход для убывания; в каждом узле печатается вся цепочка Next.
- Поиск: реализуются две функции — рекурсивная и итеративная; по найденной цепочке Next печатаются все связанные с ключом записи.

5 Операции

- Ввод/вывод/работа с файлом выполняются так же, как в задании 1; после ввода или чтения файла индексное дерево пересобирается.
- Редактирование по Id: новые значения полей считываются, после изменения записи дерево пересобирается.
- Удаление/восстановление: флаг Deleted устанавливается или снимается по Id или по категории (для всех вхождений); после изменения флага дерево пересобирается.
- Физическое удаление: все не удалённые записи копируются в начало массива, значение RecCount уменьшается до нового числа записей, после чего дерево пересобирается без физически удалённых элементов.

6 Тестирование

Ручные проверки(дополнения к проверкам из первого задания):

1. Построение дерева на sample-data.txt: обход по возрастанию/убыванию выдаёт все 7 записей в отсортированном виде.
2. Поиск по категории books: найдено Id = 303 и в рекурсивной, и в итеративной версиях поиска.
3. Дубликаты ключа: добавлена вторая запись food — обе выводятся через цепочку Next одного узла.
4. Удаление/восстановление категории transport: 302 исчезает из обходов, после восстановления по Id снова появляется; после физического удаления 302 пропадает окончательно.
5. Пересборка после редактирования: смена категории у записи перемещает её в другое место дерева после пересборки.

7 Недостатки

Сохраняются все недостатки, указанные в отчёте к заданию 1 (относительно общей структуры), а также отсутствует балансировка бинарного дерева поиска.

8 Код программы

```
#include <iostream>
#include <string>
#include <fstream>
#include <stdio.h>
using namespace std;

const int MaxRec = 30; // Количество записей в массиве
const int DaysInMonth[13] = { 0,31,28,31,30,31,30,31,31,30,31,30,31 }; // Календарь для проверки дат

struct Expense
{
    unsigned int Id;   // числовой ключ
    string Category;   // краткая категория
    string Note;       // короткое описание
    unsigned int Day;
    unsigned int Month;
    unsigned int Year;
    double Amount;     // сумма > 0
    bool Deleted;      // пометка удаления
};

struct TreeNode
{
    string Key;      // ключ сортировки (категория)
    int RecIndex;    // номер записи в массиве
    TreeNode *Left;
    TreeNode *Right;
    TreeNode *Next;  // цепочка для одинаковых ключей
};

Expense ExpArr[MaxRec];
int RecCount = 0;

TreeNode *Root = NULL;

void ClearInput()
{
    cin.clear();
    while (cin && cin.get() != '\n') {}
    printf("Неверный ввод\n");
}

bool ReadUInt(const char *prompt, unsigned int &val)
{
    string tmp;
    while (true)
    {
        printf("%s (или # - выход): ", prompt);
        if (!(cin >> tmp))
        {
            ClearInput();
```

```
52              continue;
53          }
54          if (tmp == "#")
55          {
56              printf("Отмена\n");
57              return false;
58          }
59          try
60          {
61              val = static_cast<unsigned int>(stoul(tmp));
62              return true;
63          }
64          catch (...)
65          {
66              printf("Неверный ввод\n");
67          }
68      }
69  }
70
71  bool ReadDouble(const char *prompt, double &val)
72  {
73      string tmp;
74      while (true)
75      {
76          printf("%s (или # - выход): ", prompt);
77          if (!(cin >> tmp))
78          {
79              ClearInput();
80              continue;
81          }
82          if (tmp == "#")
83          {
84              printf("Отмена\n");
85              return false;
86          }
87          try
88          {
89              val = stod(tmp);
90              return true;
91          }
92          catch (...)
93          {
94              printf("Неверный ввод\n");
95          }
96      }
97  }
98
99  bool ReadPosDouble(const char *prompt, double &val)
100 {
101     while (true)
102     {
103         if (!ReadDouble(prompt, val)) return false;
104         if (val > 0) return true;
105         printf("Значение должно быть > 0\n");
```

```
106     }
107 }
108
109 bool ReadString(const char *prompt, string &val)
110 {
111     while (true)
112     {
113         printf("%s (или # - выход): ", prompt);
114         getline(cin >> ws, val);
115         if (val == "#")
116         {
117             printf("Отмена\n");
118             return false;
119         }
120         if (val.find(';') != string::npos || val.find(' ') != string::npos)
121         {
122             printf("Символ ';' и пробелы запрещены\n");
123             continue;
124         }
125         return true;
126     }
127 }
128
129 bool ReadDate(unsigned int &y, unsigned int &m, unsigned int &d)
130 {
131     if (!ReadUInt("Год", y)) return false;
132     while (true)
133     {
134         if (!ReadUInt("Месяц", m)) return false;
135         if ((m >= 1) && (m <= 12)) break;
136         printf("Месяц должен быть 1..12\n");
137     }
138     while (true)
139     {
140         if (!ReadUInt("День", d)) return false;
141         int maxd = DaysInMonth[m];
142         if ((d >= 1) && (d <= static_cast<unsigned int>(maxd))) break;
143         printf("В этом месяце %d дней\n", maxd);
144     }
145     return true;
146 }
147
148 // Работа с деревом
149 TreeNode* MakeNode(string key, int recIndex)
150 {
151     TreeNode *p = new TreeNode;
152     p->Key = key;
153     p->RecIndex = recIndex;
154     p->Left = NULL;
155     p->Right = NULL;
156     p->Next = NULL;
157     return p;
158 }
159
```

```
160 void InsertNode(TreeNode* &root, string key, int recIndex)
161 {
162     if (root == NULL)
163     {
164         root = MakeNode(key, recIndex);
165         return;
166     }
167     if (key < root->Key)
168     {
169         InsertNode(root->Left, key, recIndex);
170     }
171     else if (key > root->Key)
172     {
173         InsertNode(root->Right, key, recIndex);
174     }
175     else
176     {
177         TreeNode *t = root;
178         while (t->Next != NULL)
179         {
180             t = t->Next;
181         }
182         t->Next = MakeNode(key, recIndex);
183     }
184 }
185
186 void FreeTree(TreeNode* node)
187 {
188     if (node == NULL) return;
189     FreeTree(node->Left);
190     FreeTree(node->Right);
191     TreeNode *dup = node->Next;
192     while (dup != NULL)
193     {
194         TreeNode *tmp = dup->Next;
195         delete dup;
196         dup = tmp;
197     }
198     delete node;
199 }
200
201 void BuildTree()
202 {
203     FreeTree(Root);
204     Root = NULL;
205     for (int i = 0; i < RecCount; i++)
206     {
207         if (!ExpArr[i].Deleted)
208         {
209             InsertNode(Root, ExpArr[i].Category, i);
210         }
211     }
212 }
213
```

```
214 void PrintOne(Expense E, int Num)
215 {
216     printf("%d) Id: %u; Категория: %s; Описание: %s; Дата: %u.%u.%u; Сумма: %.2f",
217            Num, E.Id, E.Category.c_str(), E.Note.c_str(), E.Day, E.Month, E.Year, E.Amount);
218     if (E.Deleted)
219     {
220         printf(" [удалена]");
221     }
222     printf("\n");
223 }
224
225 void PrintNodeRecords(TreeNode *node)
226 {
227     TreeNode *t = node;
228     while (t != NULL)
229     {
230         int idx = t->RecIndex;
231         if (!ExpArr[idx].Deleted)
232         {
233             PrintOne(ExpArr[idx], idx + 1);
234         }
235         t = t->Next;
236     }
237 }
238
239 void TraverseAsc(TreeNode *node)
240 {
241     if (node == NULL) return;
242     TraverseAsc(node->Left);
243     PrintNodeRecords(node);
244     TraverseAsc(node->Right);
245 }
246
247 void TraverseDesc(TreeNode *node)
248 {
249     if (node == NULL) return;
250     TraverseDesc(node->Right);
251     PrintNodeRecords(node);
252     TraverseDesc(node->Left);
253 }
254
255 TreeNode* SearchRec(TreeNode *node, string key)
256 {
257     if (node == NULL) return NULL;
258     if (key == node->Key) return node;
259     if (key < node->Key) return SearchRec(node->Left, key);
260     return SearchRec(node->Right, key);
261 }
262
263 TreeNode* SearchIter(TreeNode *node, string key)
264 {
265     TreeNode *cur = node;
266     while (cur != NULL)
267     {
```

```
268         if (key == cur->Key) return cur;
269         if (key < cur->Key)
270             cur = cur->Left;
271         else
272             cur = cur->Right;
273     }
274     return NULL;
275 }
276 
277 void InputOne()
278 {
279     if (RecCount >= MaxRec)
280     {
281         printf("Массив заполнен\n");
282         return;
283     }
284     printf("Введите данные расхода номер %d\n", RecCount + 1);
285     printf("Id - только число, строки без пробелов, дата с учётом дней в месяце\n");
286     if (!ReadUInt("Id", ExpArr[RecCount].Id)) return;
287     if (!ReadString("Категория (строка)", ExpArr[RecCount].Category)) return;
288     if (!ReadString("Описание (строка)", ExpArr[RecCount].Note)) return;
289     if (!ReadDate(ExpArr[RecCount].Year, ExpArr[RecCount].Month, ExpArr[RecCount].Day))
        ↪   return;
290     if (!ReadPosDouble("Сумма", ExpArr[RecCount].Amount)) return;
291     ExpArr[RecCount].Deleted = false;
292     RecCount++;
293     printf("Запись добавлена\n");
294     BuildTree();
295 }
296 
297 void PrintAll()
298 {
299     printf("Текущие записи (по порядку ввода):\n");
300     for (int i = 0; i < RecCount; i++)
301     {
302         if (!ExpArr[i].Deleted)
303         {
304             PrintOne(ExpArr[i], i + 1);
305         }
306     }
307     printf("\n");
308 }
309 
310 void SaveToFile()
311 {
312     string FileName;
313     char Mode;
314     string modeStr;
315     printf("Введите имя файла (или # - выход): > ");
316     if (!(cin >> FileName) || FileName == "#")
317     {
318         if (!cin) ClearInput();
319         printf("Отмена\n");
320         return;
```

```
321     }
322     printf("Режим: 1 - новый файл, 2 - добавить в конец (или # - выход): > ");
323     if (!(cin >> modeStr))
324     {
325         ClearInput();
326         return;
327     }
328     if (modeStr.size() != 1)
329     {
330         printf("Неверный режим\n");
331         return;
332     }
333     Mode = modeStr[0];
334     if (Mode == '#')
335     {
336         printf("Отмена\n");
337         return;
338     }
339     if ((Mode != '1') && (Mode != '2'))
340     {
341         printf("Неверный режим\n");
342         return;
343     }
344     ofstream f;
345     if (Mode == '2')
346         f.open(FileName, ios::app);
347     else
348         f.open(FileName, ios::out);
349     if (!f.is_open())
350     {
351         printf("Файл не открыт\n");
352         return;
353     }
354     for (int i = 0; i < RecCount; i++)
355     {
356         f << ExpArr[i].Id << ";" << ExpArr[i].Category << ";" << ExpArr[i].Note << ";";
357         f << ExpArr[i].Day << ";" << ExpArr[i].Month << ";" << ExpArr[i].Year << ";";
358         f << ExpArr[i].Amount << ";" << (ExpArr[i].Deleted ? 1 : 0) << endl;
359     }
360     f.close();
361     printf("Запись в файл выполнена\n");
362 }
363
364 string NextField(string S, size_t &pos)
365 {
366     // Формат строки: id;cat;note;day;month;year;amount;deleted
367     size_t next = S.find(';', pos);
368     if (next == string::npos)
369     {
370         string part = S.substr(pos);
371         pos = S.length();
372         return part;
373     }
374     string part = S.substr(pos, next - pos);
```

```
375     pos = next + 1;
376     return part;
377 }
378 
379 void LoadFromFile()
380 {
381     string FileName;
382     printf("Имя файла для чтения (или # - выход): > ");
383     if (!(cin >> FileName) || FileName == "#")
384     {
385         if (!cin) ClearInput();
386         printf("Отмена\n");
387         return;
388     }
389     ifstream f;
390     f.open(FileName);
391     if (!f.is_open())
392     {
393         printf("Файл не открыт\n");
394         return;
395     }
396     string line;
397     while (!f.eof())
398     {
399         getline(f, line);
400         if (line.length() == 0)
401             continue;
402         if (RecCount >= MaxRec)
403         {
404             printf("Массив заполнен, чтение остановлено\n");
405             break;
406         }
407         size_t pos = 0;
408         string fid = NextField(line, pos);
409         string fcat = NextField(line, pos);
410         string fnote = NextField(line, pos);
411         string fday = NextField(line, pos);
412         string fmonth = NextField(line, pos);
413         string fyear = NextField(line, pos);
414         string famount = NextField(line, pos);
415         string fdel = NextField(line, pos);
416         ExpArr[RecCount].Id = stoi(fid);
417         ExpArr[RecCount].Category = fcat;
418         ExpArr[RecCount].Note = fnote;
419         ExpArr[RecCount].Day = stoi(fday);
420         ExpArr[RecCount].Month = stoi(fmonth);
421         ExpArr[RecCount].Year = stoi(fyear);
422         ExpArr[RecCount].Amount = stod(famount);
423         ExpArr[RecCount].Deleted = (fdel == "1");
424         RecCount++;
425     }
426     f.close();
427     BuildTree();
428     printf("Чтение из файла завершено\n");
```

```
429 }
430
431 void ShowByTree(bool Asc)
432 {
433     if (Root == NULL)
434     {
435         printf("Индекс пуст\n");
436         return;
437     }
438     if (Asc)
439         TraverseAsc(Root);
440     else
441         TraverseDesc(Root);
442     printf("\n");
443 }
444
445 void SearchByCategoryRec()
446 {
447     if (Root == NULL)
448     {
449         printf("Сначала постройте дерево\n");
450         return;
451     }
452     string key;
453     if (!ReadString("Введите категорию для поиска (рекурсивно)", key)) return;
454     TreeNode *node = SearchRec(Root, key);
455     if (node == NULL)
456     {
457         printf("Не найдено\n\n");
458         return;
459     }
460     PrintNodeRecords(node);
461     printf("\n");
462 }
463
464 void SearchByCategoryIter()
465 {
466     if (Root == NULL)
467     {
468         printf("Сначала постройте дерево\n");
469         return;
470     }
471     string key;
472     if (!ReadString("Введите категорию для поиска (итеративно)", key)) return;
473     TreeNode *node = SearchIter(Root, key);
474     if (node == NULL)
475     {
476         printf("Не найдено\n\n");
477         return;
478     }
479     PrintNodeRecords(node);
480     printf("\n");
481 }
482
```

```
483 void EditById()
484 {
485     unsigned int id;
486     printf("Id - только число\n");
487     if (!ReadUInt("Введите Id записи для редактирования", id)) return;
488     for (int i = 0; i < RecCount; i++)
489     {
490         if ((!ExpArr[i].Deleted) && (ExpArr[i].Id == id))
491         {
492             if (!ReadString("Новая категория (строка)", ExpArr[i].Category)) return;
493             if (!ReadString("Новое описание (строка)", ExpArr[i].Note)) return;
494             if (!ReadDate(ExpArr[i].Year, ExpArr[i].Month, ExpArr[i].Day)) return;
495             if (!ReadPosDouble("Новая сумма", ExpArr[i].Amount)) return;
496             printf("Запись изменена\n");
497             BuildTree();
498             return;
499         }
500     }
501     printf("Запись не найдена\n");
502 }
503
504 void DeleteByCategory()
505 {
506     if (Root == NULL)
507     {
508         printf("Сначала постройте дерево\n");
509         return;
510     }
511     string key;
512     if (!ReadString("Категория для удаления", key)) return;
513     bool found = false;
514     for (int i = 0; i < RecCount; i++)
515     {
516         if (!ExpArr[i].Deleted && ExpArr[i].Category == key)
517         {
518             ExpArr[i].Deleted = true;
519             found = true;
520         }
521     }
522     if (found)
523         printf("Все записи категории %s помечены как удалённые\n", key.c_str());
524     else
525         printf("Не найдено\n");
526     BuildTree();
527 }
528
529 void DeleteById()
530 {
531     unsigned int id;
532     printf("Id - только число\n");
533     if (!ReadUInt("Id для удаления", id)) return;
534     bool found = false;
535     for (int i = 0; i < RecCount; i++)
536     {
```

```
537         if (!ExpArr[i].Deleted && ExpArr[i].Id == id)
538         {
539             ExpArr[i].Deleted = true;
540             found = true;
541             printf("Запись с Id %u помечена как удалённая\n", id);
542             break;
543         }
544     }
545     if (!found) printf("Запись с таким Id не найдена\n");
546     BuildTree();
547 }
548 
549 void RestoreById()
550 {
551     unsigned int id;
552     printf("Id - только число\n");
553     if (!ReadUInt("Id записи для восстановления", id)) return;
554     for (int i = 0; i < RecCount; i++)
555     {
556         if (ExpArr[i].Id == id && ExpArr[i].Deleted)
557         {
558             ExpArr[i].Deleted = false;
559             printf("Запись восстановлена\n");
560             BuildTree();
561             return;
562         }
563     }
564     printf("Удалённая запись не найдена\n");
565 }
566 
567 void PackDeleted()
568 {
569     int j = 0;
570     for (int i = 0; i < RecCount; i++)
571     {
572         if (!ExpArr[i].Deleted)
573         {
574             ExpArr[j] = ExpArr[i];
575             j++;
576         }
577     }
578     RecCount = j;
579     BuildTree();
580     printf("Удалённые записи вычищены\n");
581 }
582 
583 void BuildIndexFromMenu()
584 {
585     BuildTree();
586     if (Root == NULL)
587         printf("Индекс пуст\n");
588     else
589         printf("Индекс построен\n");
590 }
```

```
591
592 int main()
593 {
594     char ch = '0';
595     string cmd;
596
597     do
598     {
599         printf("Меню:\n");
600         printf("1 - ввод новой записи\n");
601         printf("2 - вывод записей по вводу\n");
602         printf("3 - запись в файл\n");
603         printf("4 - чтение из файла\n");
604         printf("5 - построить индекс-дерево по категории\n");
605         printf("6 - вывод по дереву (возрастание)\n");
606         printf("7 - вывод по дереву (убывание)\n");
607         printf("8 - поиск по категории (рекурсивно)\n");
608         printf("9 - поиск по категории (итеративно)\n");
609         printf("c - редактировать по Id\n");
610         printf("d - удалить по категории\n");
611         printf("h - удалить по Id\n");
612         printf("e - восстановить по Id\n");
613         printf("f - физическое удаление помеченных\n");
614         printf("0 - выход\n");
615         if (!(cin >> cmd))
616         {
617             ClearInput();
618             continue;
619         }
620         if (cmd.size() != 1)
621         {
622             printf("Команда не распознана\n\n");
623             continue;
624         }
625         ch = cmd[0];
626         printf("\n");
627
628         switch (ch)
629         {
630         case '1':
631             InputOne();
632             break;
633         case '2':
634             PrintAll();
635             break;
636         case '3':
637             SaveToFile();
638             break;
639         case '4':
640             LoadFromFile();
641             break;
642         case '5':
643             BuildIndexFromMenu();
644             break;
```

```
645         case '6':
646             ShowByTree(true);
647             break;
648         case '7':
649             ShowByTree(false);
650             break;
651         case '8':
652             SearchByCategoryRec();
653             break;
654         case '9':
655             SearchByCategoryIter();
656             break;
657         case 'c':
658             EditById();
659             break;
660         case 'd':
661             DeleteByCategory();
662             break;
663         case 'e':
664             RestoreById();
665             break;
666         case 'f':
667             PackDeleted();
668             break;
669         case 'h':
670             DeleteById();
671             break;
672         case '0':
673             break;
674         default:
675             printf("Нет такой команды\n");
676             break;
677         }
678         printf("\n");
679     } while (ch != '0');
680 
681     FreeTree(Root);
682     return 0;
683 }
```