Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Дискретный анализ»

Студент: А. О. Дубинин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2018

Лабораторная работа №2

Задача: Необходимо создать программную библиотеку, реализующую указанную структуру данных, на основе которой разработать программу-словарь. В словаре каждому ключу, представляющему из себя регистронезависимую последовательность букв английского алфавита длиной не более 256 символов, поставлен в соответствие некоторый номер, от 0 до $2^{64}-1$. Разным словам может быть поставлен в соответствие один и тот же номер.

Вариант структуры: АВЛ-дерево.

1 Описание

Согласно [2] АВЛ-дерево — это двоичное дерево поиска, особенностью которого является то, что оно является сбалансированным в следующем смысле: для любого узла дерева высота его правого поддерева отличается от высоты левого поддерева не более чем на единицу. Доказано, что этого свойства достаточно для того, чтобы высота дерева логарифмически зависела от числа его узлов: высота h АВЛ-дерева с n ключами лежит в диапазоне от $log_2(n+1)$ до $1.44 * log_2(n+2) - 0.328$. А так как основные операции над двоичными деревьями поиска (поиск, вставка и удаление узлов) линейно зависят от его высоты, то получаем гарантированную логарифмическую зависимость времени работы этих алгоритмов от числа ключей, хранимых в дереве.

В процессе добавления или удаления узлов в АВЛ-дереве возможно возникновение ситуации, когда balance factor некоторых узлов оказывается равными 2 или -2, т.е. возникает расбалансировка поддерева. Для выправления ситуации применяются хорошо нам известные повороты вокруг тех или иных узлов дерева.

Сохранения и запись словаря осуществляются при помощи индексации вершин, и последовательной записи их в бинарном компактном представлении.

2 Исходный код

Словарь должен хранит пару ключ-значение,поэтому создадим новую структуру TData, в которой будем хранить ссылку на ключ и значение в целочисленном виде.

main.cpp	
int main()	Основная функция. В ней происходит ввод данных и вызов методов дерева
char ToLower(char ch)	Возращает символ строки в нижнем регистре
bool IsLetter(char ch)	Возращает true, если символ, переданный функции, является буквой английского алфавита, иначе false.
struct TNode	
uint8_t height	Высота узла
TData *data	Данные узла
TNode *left	Ссылка на левого сына.
TNode *right	Ссылка на правого сына.
class TTree	
TTree()	Конструктор по умолчания, обеспечивает пустое дерево.
void Insert(TData data)	Вставка элемента в дерево.
void Delete(TData data)	Удаление элемента из дерева.
bool Search(TData Возвращает true, если элемент был найден в дереве, иначе false.	data)
bool SearchSub(TNode * data)	node, TData Функция помощник, для поиска элемента в дереве, возвращает true, если элемент был найден в дереве, иначе false.
bool Save(const char* filename)	Сохранение дерева на диск, возвращает true, если успешно, иначе false.
bool SaveSub(FILE *f, TNode *node)	Функция помощник для сохранение дерева на диск, возвращает true, если успешно, иначе false.
bool Load(const char* filename)	Загрузка дерева с диска, возвращает true, если успешно, иначе false.
bool LoadSub(FILE *f, TNode *	node)

	Функция помощник для загрузка дерева с диска, возвращает true, если успешно, иначе false.
void Erase()	Удаление дерева.
TTree()	Деструктор вызывающий удаление дерева.
TNode *root	Ссылка на корень дерева.
TNode *CreateNode(TData data)	Создание листа с данными.
void DeleteNode(TNode *	node)
Удаление узла дерева.	
TNode *DeleteSub(TNode *node, TData data)	Функция помощник, для удаление узла дерева.
TNode *InsertSub(TNode *node, TData data)	Функция помощник, для вставки элемента.
TNode *Balance(TNode *node)	Функция балансировки узла, возращает ссылку на сбалансированный узел.
TNode *RotateLeft(TNode *q)	Поворот дерева налево.
TNode *RotateRight(TNode *q)	Поворот дерева направо.
void SetHeight(TNode *node)	Установить или обновить высоту узла.
TNode *Min(TNode *node)	Вернуть минимальный узел из переданого узла(дерева)
TNode *DelMin(TNode *node)	Удалить минимальный узел в дереве и вернуть измененное дерево.
int BalFact(TNode *node)	Вернуть баланс фактору узла.
int GetHeight(TNode *node)	Вернуть высоту узла.

3 Профилирование

Данные gprof на тесте. Тест был взят на 10000 строк.

Функции выполняющиеся дольше всего

```
Flat profile:

Each sample counts as 0.01 seconds.
time   name
42.88  TTree::LoadSub(_IO_FILE*,TNode*&)
28.59  TTree::SaveSub(_IO_FILE*,TNode*)
14.29  TTree::CreateNode(TData)
7.15   ToLower(char)
7.15   TTree::DeleteNode(TNode*&)
```

Количество вызовов функций

```
Each sample counts as 0.01 seconds.
calls      us/call  us/call  name
1880241     0.01     0.01  ToLower(char)
1877709     0.00     0.00  IsLetter(char)
845161      0.05     0.05  TTree::CreateNode(TData)
144730      0.00     0.00  TTree::GetHeight(TNode*)
46128       0.00     0.00  TTree::BalFact(TNode*)
26237       0.00     0.00  TTree::SetHeight(TNode*)
22791       0.00     0.00  TTree::Balance(TNode*)
7491        0.00     0.00  TTree::Search(TData&)
7482        0.00     0.00  TTree::SearchSub(TNode*&,TData&)
2505        0.00     0.05  TTree::Insert(TData)
2504        0.00     0.05  TTree::InsertSub(TNode*,TData)
1290        7.76     7.76  TTree::DeleteNode(TNode*&)
1289       15.52    46.48  TTree::LoadSub(_IO_FILE*,TNode*&)
1289        0.00    54.24  TTree::Load(char const*)
1218       57.50    57.50  TTree::SaveSub(_IO_FILE*,TNode*)
1218        0.00    57.50  TTree::Save(char const*)
882         0.00     0.00  TTree::RotateLeft(TNode*)
841         0.00     0.00  TTree::RotateRight(TNode*)
```

Исходя из этих данных наиболее приоритетными для оптимизации являются операции работы с диском, как самая трудоёмкая часть программы.

4 Тест производительности

Реализованный алгоритм сравнивается с $std::map$, в котором ключом будет $std::string$, а значением $unsigned\ long\ long$.

Для добавления в $std::map$ используется метод $insert(std::make\_pair(string, value))$. Он возвращает пару значений: итератор на добавленный элемент и булевское значение – истина, если вставка прошла успешно, и ложь, если такой ключ уже присутствует.

Для удаления из $std::map$ используется метод $erase(string)$, который возвращает количество удалённых элементов.

Для удаления из $std::map$ используется метод $find(string)$. Он возвращает итератор на найденный элемент, либо итератор, указывающий на конец контейнера – $end()$.

1000 команд

```
art@mars:~/study/semester_3/DA/lab_2$ g++ -std=c++11 benchmark_stdmap.cpp -o
benchmark
art@mars:~/study/semester_3/DA/lab_2$ g++ -std=c++11 lab2.cpp TTree.cpp -o
lab2
art@mars:~/study/semester_3/DA/lab_2$ ./benchmark <tests/01.t >b_log
art@mars:~/study/semester_3/DA/lab_2$ ./lab2 <tests/01.t  >l_log
art@mars:~/study/semester_3/DA/lab_2$ diff l_log b_log
974,975c974,975
<Container: avl-tree
<runtime = 41.504
---
>Container: map
>runtime = 66.88
```

10000 команд

```
art@mars:~/study/semester_3/DA/lab_2$ g++ -std=c++11 benchmark_stdmap.cpp -o
benchmark
art@mars:~/study/semester_3/DA/lab_2$ g++ -std=c++11 lab2.cpp TTree.cpp -o
lab2
art@mars:~/study/semester_3/DA/lab_2$ ./benchmark <tests/01.t >b_log
art@mars:~/study/semester_3/DA/lab_2$ ./lab2 <tests/01.t  >l_log
art@mars:~/study/semester_3/DA/lab_2$ diff l_log b_log
9824,9825c9824,9825
<Container: avl-tree
```

```
<runtime = 1456.72
---
>Container: map
>runtime = 3155.48
```

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я изучил различные структуры вида дерево, осознал их преимущества и недостатки, написал программную библиотеку, реализующую сложный тип данных, тщательно изучил и отладил её работу. Я опробовал на практике бинарный ввод и вывод данных в C++, работу с файлами, функции $fread$, $fwrite$, $fopen$.

Список литературы

[1] Роберт Седжвик, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Фундаментальные алгоритмы на C++* — Издательство «DiaSoft», 2001. — 688 с. (ISBN 966-7393-89-5)

[2] *АВЛ-деревья - Habr*
URL: `https://habr.com/post/150732/`

[3] *std::map - cppreference.com*
URL: `http://en.cppreference.com/w/cpp/container/map`