МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Бинарные деревья

Студент гр. 9303 _____ Муратов Р.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Познакомиться с такой часто используемой на практике, особенно при решении задач кодирования и поиска, нелинейной структурой данных, как бинарное дерево, способами её представления и реализации, получить навыки решения задач обработки бинарных деревьев.

Задание.

Вариант 8(в)

Для заданного бинарного дерева с произвольным типом элементов:

- получить лес, естественно представленный этим бинарным деревом;
- вывести изображение бинарного дерева и леса;
- перечислить элементы леса в горизонтальном порядке (в ширину).

Основные теоретические положения.

Деревом называется конечное множество T, состоящее из одного или более узлов, таких, что

1. имеется один специально обозначенный узел, называемый корнем данного дерева;
2. остальные узлы (исключая корень) содержатся в $m \geq 0$ попарно не пересекающихся множествах T1, T2, .., Tm, каждое из которых, в свою очередь,является деревом (поддеревом дерева T).

Это рекурсивное определение дерева.

Лесом называется множество (обычно упорядоченное), состоящее из некоторого (быть может, равного нулю) числа непересекающихся деревьев.

Тогда дернво можнорассматривать как структуру, состоящую из корня и леса поддеревьев.

Дерево T = (корень, лес поддеревьев)

Скобочное представление дерева и леса:

< лес > ::= пусто | < дерево > < лес >,

< дерево > ::= (< корень > < лес >).

Бинарное дерево - это конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом.

Примером может служить структура на рисунке 1.

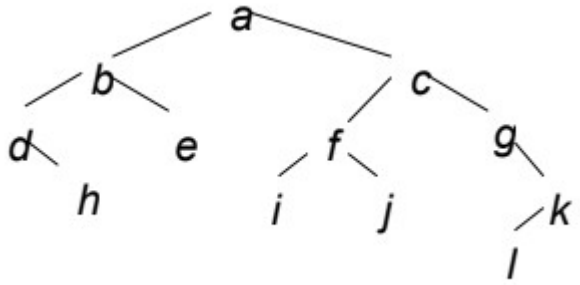

Рисунок 1 – Пример бинарного дерева

Скобочное представление бинарного дерева (БД):

< БД > ::= < пусто > | < непустое БД >,

< пусто > ::= Λ,

< непустое БД > ::= (< корень > < БД > < БД >).

Выполнение работы.

Для решения поставленной задачи были реализованы шаблонная структура узла `Node` и два шаблонных класса: `BinTree` (бинарное дерево) и `Forest` (лес).

Поля структуры `Node`:

- `size_t left` — индекс массива, по которому находится левый «сын» текущего узла;
- `size_t right` — индекс массива, по которому находится правый «сын» текущего узла;
- `T data` — данные, хранящиеся в узле.

Поля класса `BinTree`:

- `Node<T>* arr_` – указатель на массив, в котором хранятся узлы дерева;
- `size_t capacity_` – текущая вместимость массива (используется при увеличении размера массива при исчерпании свободного места);
- `size_t size_` – текущий размер массива, то есть количество элементов, хранящихся в массиве;
- `size_t root_` – индекс массива, по которому хранится корень бинарного дерева.

Для чтения бинарного дерева из входного потока перегружен `operator>>` и реализованы два метода класса:

- `void read(std::istream&)` - вспомогательный метод, который начинает записывать первый узел в ячейку массива с индексом root_;
- `void readBT(std::istream, size_t)` — основной метод, который считывает скобочное представление бинарного дерева из входного потока и сохраняет его.

Для вывода бинарного дерева в выходной поток перегружен `operator<<` и реализован метод:

- `void print(std::ostream&, const Node<T>&, std::string)` — выводит дерево в выходной поток. Объект класса `string` используется в качестве префикса для «красивого» вывода.

Дополнительные методы:

- `void passSpaces(T&, std::istream&)` - вспомогательный метод, чтобы пропускать лишние пробелы из входного потока;
- `void resize()` - увеличивает вместимость массива, если в нем закончилось свободное место;
- `void update(size_t)` — обновляет ячейки массива, которые указывают на свободное место, то есть поддерживает актуальность данных.

- `Forest::Forest<T>* toForest() const` — данный метод возвращает лес, естественно представленный данным бинарным деревом.

Поля класса Forest:

- `Node<T>* arr_` – указатель на массив, в котором хранятся узлы бинарного дерева, которому соответствует лес;
- `size_t capacity_` – текущая вместимость массива (используется при увеличении размера массива при исчерпании свободного места);
- `size_t size_` – текущий размер массива, то есть количество элементов, хранящихся в массиве;
- `size_t root_` – индекс массива, по которому хранится корень бинарного дерева.

Для вывода бинарного дерева в выходной поток перегружен `operator<<` и реализованы четыре метода:

- `void printForest(std::ostream&, const Node<T>&)` – вспомогательный метод, который выводит корень первого дерева;
- `void printTree(std::ostream&, const Node<T>&, std::string)` — основной метод, который выводит все деревья леса. Объект класса `string` выступает в качестве префикса;
- `void printWide(std::ostream&)` – вспомогательный метод для вывода леса «в ширину»;
- `void printWideRec(std::ostream&)` – основной метод для вывода леса «в ширину».

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

Познакомился с такой часто используемой на практике, особенно при решении задач кодирования и поиска, нелинейной структурой данных, как бинарное дерево, способами её представления и реализации, получил навыки решения задач обработки бинарных деревьев.

Основная сложность была связана с корректным хранением бинарного дерева в массиве и поддержанием актуальности данных. Также много времени заняла реализация «красивого» вывода.

Разработана программа, которая считывает с входного потока скобочное представление бинарного дерева, конвертирует его в лес, выводит изображения бинарного дерева и леса и перечисляет элементы леса в горизонтальном порядке (в ширину).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <memory>
#include <cstring>
#include <queue>
#include <string>
#include <fstream>

template<class T>
struct Node {
  size_t left = 0;
  size_t right = 0;
  T data;
};

template<class T>
class Forest;

template<class T>
class BinTree {
  template<class U>
     friend  std::ostream&  operator<<(std::ostream&  os,  const
BinTree<U>& binT);
  template<class U>
   friend std::istream& operator>>(std::istream& is, BinTree<U>&
binT);
public:
  BinTree(size_t capacity, size_t root)
    : capacity_(capacity), size_(0), root_(root) {
    arr_ = new Node<T>[capacity];
    for (size_t i = 0; i < capacity_ - 1; ++i) {
      arr_[i].left = i + 1;
    }
    arr_[capacity_ - 1].left = 0;
  }

  ~BinTree() {
    delete [] arr_;
  }

  operator bool() const {
    return (size_ != 0);
  }

  size_t size() const { return size_; }
  size_t capacity() const { return capacity_; }
  size_t root() const { return root_; }
  const Node<T>* array() const { return arr_; }

  void readBT(std::istream& is, size_t index) {
    T x;
```

```
      passSpaces(x, is);

      arr_[index].data = x;
      ++size_;

      update(index);

      passSpaces(x, is);

      if (x == '(') {
        arr_[index].left = arr_[0].left;
        readBT(is, arr_[0].left);
        passSpaces(x, is);
        if (x == '(') {
          arr_[index].right = arr_[0].left;
          readBT(is, arr_[0].left);
          passSpaces(x, is);
        }
      } else if (x == '#') {
        passSpaces(x, is);
        arr_[index].right = arr_[0].left;
        readBT(is, arr_[0].left);
        passSpaces(x, is);
      }
    }

    void read(std::istream& is) {
      T x;
      passSpaces(x, is);
      if (x == '(') {
        readBT(is, root_);
      } else {
        std::cerr << "Error: unexpected symbol" << std::endl;
      }
    }


    void passSpaces(T& x, std::istream& is) {
      do {
        is >> x;
      } while (x == ' ');
    }

    Forest<T>* toForest() const {
      return new Forest<T>(*this);
    }

     void print(std::ostream& os, const Node<T>& node, std::string
prefix) const {
      os << prefix << "->" << node.data << std::endl;
      if (!node.left && !node.right)
        return;
      if (prefix.back() == '+') {
        prefix.pop_back();
        prefix += " ";
      }
```

```
      if (node.left) {
        if (!node.right) {
          print(os, arr_[node.left], prefix + "  +");
          return;
        }
        else
          print(os, arr_[node.left], prefix + "  |");
      } else {
        os << prefix << "  |->#" << std::endl;
      }
      if (node.right) {
        print(os, arr_[node.right], prefix + "  +");
      } else {
        os << prefix << "  +->#" << std::endl;
      }
    }

    void resize() {
      size_t newCapacity = capacity_ + capacity_ / 2;
      Node<T>* newArr = new Node<T>[newCapacity];

        std::memcpy((void*)newArr, (void*)arr_, sizeof (Node<T>) *
capacity_);

      newArr[0].left = capacity_;
      for (size_t i = capacity_; i < newCapacity - 1; ++i) {
        newArr[i].left = i + 1;
      }
      newArr[newCapacity - 1].left = 0;

      delete [] arr_;
      arr_ = newArr;
      capacity_ = newCapacity;
    }

    void update(size_t indexChange) {
      size_t temp = 0;
      while (1) {
        if (arr_[temp].left == indexChange) {
          arr_[temp].left = arr_[indexChange].left;
          arr_[indexChange].left = 0;
          if (arr_[temp].left == temp) {
            resize();
          }
          break;
        } else {
          temp = arr_[temp].left;
        }
      }
    }

  private:
    Node<T>* arr_;
    size_t capacity_;
    size_t size_;
    size_t root_;
```

```
};

template<class U>
std::ostream& operator<<(std::ostream& os, const BinTree<U>&
binT) {
  if (binT) {
    binT.print(os, binT.arr_[binT.root_], "");
  } else {
    os << "Binary tree is empty!" << std::endl;
  }
  return os;
}

template<class U>
std::istream& operator>>(std::istream& is, BinTree<U>& binT) {
  binT.read(is);
  return is;
}

template<class T>
class Forest {
  template<class U>
    friend std::ostream& operator<<(std::ostream& is, const
Forest<U>& frst);
public:
  Forest(const BinTree<T>& bt)
    : capacity_(bt.capacity()),
      size_(bt.size()),
      root_(bt.root()) {
    arr_ = new Node<T>[capacity_];
     std::memcpy((void*)arr_, (void*)bt.array(), sizeof (Node<T>)
* capacity_);
  }

  ~Forest() {
    delete [] arr_;
  }

  operator bool() const {
    return (size_ != 0);
  }

  void printWide(std::ostream& os) const {
    if (*this) {
      std::queue<Node<T>> q;
      q.push(arr_[root_]);
      printWideRec(os, q);
    } else {
      os << "Forest is empty!" << std::endl;
    }
  }

    void printWideRec(std::ostream& os, std::queue<Node<T>>&
nodeQueue) const {
    if (nodeQueue.empty()) {
      os << "\n";
```

```
            return;
          }
          Node<T> curNode = nodeQueue.front();
          os << curNode.data << " ";
          nodeQueue.pop();

          if (curNode.left)
            nodeQueue.push(arr_[curNode.left]);
          if (curNode.right)
            nodeQueue.push(arr_[curNode.right]);
          printWideRec(os, nodeQueue);
        }

        void printForest(std::ostream& os, const Node<T>& node) const {
          os << "\nTree:" << std::endl;
          os << "->" << node.data << std::endl;
          if (node.left) {
            printTree(os, arr_[node.left], "  ");
          }
          if (node.right) {
            printForest(os, arr_[node.right]);
          }
        }

           void printTree(std::ostream& os, const Node<T>& node,
std::string prefix) const {
          prefix += (node.right == 0) ? '+' : '|';
          os << prefix << "->" << node.data << std::endl;
          if (node.left) {
            if (prefix.back() == '+') {
              prefix.pop_back();
              prefix += " ";
            }
            printTree(os, arr_[node.left], prefix + "  ");
          }
          if (node.right) {
            prefix.pop_back();
            printTree(os, arr_[node.right], prefix);
          }
        }

      private:
        Node<T>* arr_ = nullptr;
        size_t capacity_ = 0;
        size_t size_ = 0;
        size_t root_ = 0;
      };

      template<class U>
      std::ostream& operator<<(std::ostream& os, const Forest<U>& frst)
{
        if (frst) {
          frst.printForest(os, frst.arr_[frst.root_]);
        } else {
          os << "Forest is empty!" << std::endl;
        }
```

```
    return os;
  }

  int main() {
    size_t startCap = 0;
    size_t root = 0;

    std::cout << "Enter start capacity (> 1): " << std::endl;
    std::cin >> startCap;
     std::cout << "Enter root index (0 < index < capacity): " <<
std::endl;
    std::cin >> root;

    if ((root < 1) || (root >= startCap)) {
      std::cout << "Incorrect input" << std::endl;
      return 1;
    }

    std::string fileName;
    std::cout << "Enter file name: ";
    std::cin >> fileName;

    std::ifstream file(fileName);
    if (!file.is_open()) {
      std::cout << "Can't open a file\n";
      return 1;
    }

    std::ofstream resFile("./result.txt");
    std::cout << "Result will be saved in result.txt" << std::endl;

    BinTree<char> bt(startCap, root);
    file >> bt;

    resFile << "1st task check:\n";
    std::shared_ptr<Forest<char>> fptr(bt.toForest());
    resFile << *fptr;

    resFile << "2nd task check:\n";
    resFile << bt;


    resFile << "3rd task check:\n";
    fptr->printWide(resFile);

    file.close();
    resFile.close();
    return 0;
  }
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии

1.	(a(b)(c#(d(e))))	1st task check:
		
		Tree:
		->a
		  +->b
		
		Tree:
		->c
		
		Tree:
		->d
		  +->e
		2nd task check:
		->a
		 |->b
		 +->c
		  |->#
		  +->d
		  +->e
		3rd task check:
		a b c d e	

2.	(a (b (j (o (p)) (m)) (c (k) (l))) (d # (e (f # (g # (h (i) (n)))))))	1st task check: Tree: ->a |->b | |->j | | +->o | | +->p	

		| +->m |->c | +->k +->l Tree: ->d Tree: ->e |->f |->g |->h | +->i +->n 2nd task check: ->a |->b | |->j | | |->o | | | +->p | | +->m | +->c | |->k | +->l +->d |-># +->e +->f |-># +->g |-># +->h	

		|->i +->n 3rd task check: a b d j c e o m k l f p g h i n	
3.	(a)	1st task check: Tree: ->a 2nd task check: ->a 3rd task check: a	
4.		1st task check: Forest is empty! 2nd task check: Binary tree is empty! 3rd task check: Forest is empty!	Если входной файл пуст, то программа корректно работает.