ЛАБОРАТОРНАЯ РАБОТА №1

По дисциплине: Усовершенствованные методы разработки алгоритмов и сложные структуры данных
Тема занятия: Динамическое программирование: оптимальные бинарные деревья поиска
Цель занятия: научить сохранять данные в таблицах для исключения повторных вычислений
Количество часов: 4

Содержание работы

1. Подключение кода для работы с бинарным деревом поиска.
2. Добавление в описание структуры данных полей ***leaf***, ***pq***, ***depth***.
3. Реализация хранения данных: массивов ***p[n]***, ***q[n]***, ***e[n+1][n]***, ***w[n+1][n]***, ***root[n][n]***.
4. Написание основного алгоритма – функции ***optimal_BST()***.
5. Проверка таблиц ***e***, ***w***, ***root*** – функция ***print_arrays()***.
6. Написание рекурсивной функции ***build()*** для заполнения дерева из таблицы ***root***.
7. Изменение основных функций: ***insert()***, ***print()***.
8. Тестирование и отладка кода.

Методические указания по выполнению

1. Подключение кода для работы с бинарным деревом поиска.

Из лабораторной работы №10 курса «Алгоритмы и структуры данных» необходимо использовать код для построения бинарного дерева поиска. Основные функции – ***insert()***, ***print()***, остальные функции необходимы для реализации полноценного приложения, работающего методом динамического программирования.

2. Добавление в описание структуры данных полей ***value***, ***leaf***, ***pq***, ***depth***.

leaf – информации об узле – это ключ или фиктивный ключ;
pq – вероятность ключа (инициализируется значением p[i] для ключа, и q[i] – для фиктивного);
depth – глубина узла в дереве (для подсчета математического ожидания стоимости поиска OBST).

3. Реализация хранения данных: массивов ***p[n]***, ***q[n]***, ***e[n+1][n]***, ***w[n+1][n]***, ***root[n][n]***.

p[n] – вероятности ключей;
q[n] – вероятности фиктивных ключей;
e[n+1][n] – математическое ожидание стоимости поиска в OBST;
w[n+1][n] – сумма вероятностей;
root[n][n] – корни поддеревьев, содержащих единственный оптимальный ключ.

Для задачи из источника [1] рекомендуется взять ***n = 6***. Массивы ***p[n]*** и ***q[n]*** – заполняются из таблицы исходных значений, например, следующими:

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Рекомендуется реализовать отдельную функцию для их заполнения, например, ***init (p, q)***.

Массивы ***e[n+1][n]***, ***w[n+1][n]*** и ***root[n][n]*** – двумерные динамические массивы, создать можно по следующему примеру:

```
double **e = new double*[n + 1];        // n+1 строка в массиве
for (int i = 0; i < n + 1; i++)
  e[i] = new double[n];                 // и n столбцов
```

4. Написание основного алгоритма – функции ***optimal_BST()***.

Исходя из главной рекурсивной формулы для построения OBST:

$$e[i,j] = \begin{cases} q_{i-1}\ , & \text{если } j = i-1\ , \\ \min\limits_{i \le r \le j} \{e[i,r-1] + e[r+1,j] + w(i,j)\}\ , & \text{если } i \le j\ . \end{cases}$$

алгоритм используется следующий:

```
optimal_BST (p, q, n, e, w, root)
    for (i = 1 to n + 1)
        e[i][i - 1] = q[i - 1];
        w[i][i - 1] = q[i - 1];
    for (l = 1 to n)
        for (i = 1 to n - l + 1)
              j = i + l - 1;
              e[i][j] = INT_MAX;
              w[i][j] = w[i][j - 1] + p[j] + q[j];
              for (r = i to j)
                    t = e[i][r - 1] + e[r + 1][j] + w[i][j];
                    if t < e[i][j]
                          e[i][j] = t;
                          root[i][j] = r;
```

Массивы передаются в функцию следующим образом: одномерные – ***double *p***, а двумерные – ***double **e***. Обратите внимание, что таблицы заполняются только до побочной диагонали, включая её. Для использования ***INT_MAX*** необходимо подключить заголовочный файл ***LIMITS.H***.

5. Проверка таблиц ***e***, ***w***, ***root*** – функция ***print_arrays()***.

После вызова функции ***optimal_BST()*** необходимо проверить полученные таблицы ***e***, ***w***, ***root***. Для этого напишите функцию ***print_arrays (e, w, root)***.

Двумерный массив можно вывести следующим образом:

```
cout << "\nTable E\n";
for (int i = 1; i <= n; i++) {
    for (int j = 0; j < i - 1; j++)
        cout << "    \t";                // сдвиг для вывода аналогично рисунку
    for (int j = i - 1; j < n; j++)
        cout << e[i][j] << '\t';
    cout << endl;
}
```

6. Написание рекурсивной функции ***build()*** для заполнения дерева из таблицы ***root***.

```
build (T, root, i, j, p, q)
    if i <= j
        k = root[i][j];
        insert(T, k, 0, p, q);
        build(T, root, i, k - 1, p, q);
        build(T, root, k + 1, j, p, q);
    else
        insert(T, j, 1, p, q);
```

Рассмотрим код.

Согласно основной рекурсивной формуле для построения OBST видно, что если ***j=i-1*** или ***i>j***, то имеется всего один фиктивный ключ, который вставляется в дерево в ветке ***else***:

```
        insert(T, j, 1, p, q);
```

здесь ***T*** – корень дерева, ***j*** – номер фиктивного ключа, чтобы взять соответствующую вероятность из таблицы ***q***, ***1*** – информация о том, что это фиктивный ключ или лист. Таблицы ***p*** и ***q*** необходимы для заполнения узла соответствующей вероятностью.

Ветка ***if***: если ***i<=j***, то сначала из таблицы root определяется ***k*** – номер вставляемого ключа (единственного оптимального на промежутке ***[i, j]***). И затем в дерево вставляется данный ключ. ***0*** – информация о том, что это ключ не фиктивный (не лист). Также здесь происходит два рекурсивных вызова для промежутков ***[i, k-1]*** и ***[k+1, j]***.

7. Изменение основных функций: ***insert()***, ***print()***.

В функции ***insert()*** необходимо выполнить следующие изменения:

- расширить список параметров:
  ```
  void insert(tree *&t, int v, bool l, double*p, double*q)
  ```
  здесь первые два параметра не изменяются: ***t*** – ссылка на корень дерева, ***v*** – номер ключа, ***l*** – информация о том, является ли ключ листом, ***p*** и ***q*** – таблицы вероятностей ключей;
- проинициализировать новое поле ***z–>leaf*** значением параметра ***l***;
- проинициализировать новое поле ***z–>pq*** значением из таблицы ***p***, если это ключ, и значением из таблицы ***q***, если это фиктивный ключ;
- проинициализировать новое поле ***z–>d*** значением глубины родительского узла ***y***, увеличенным на единицу;

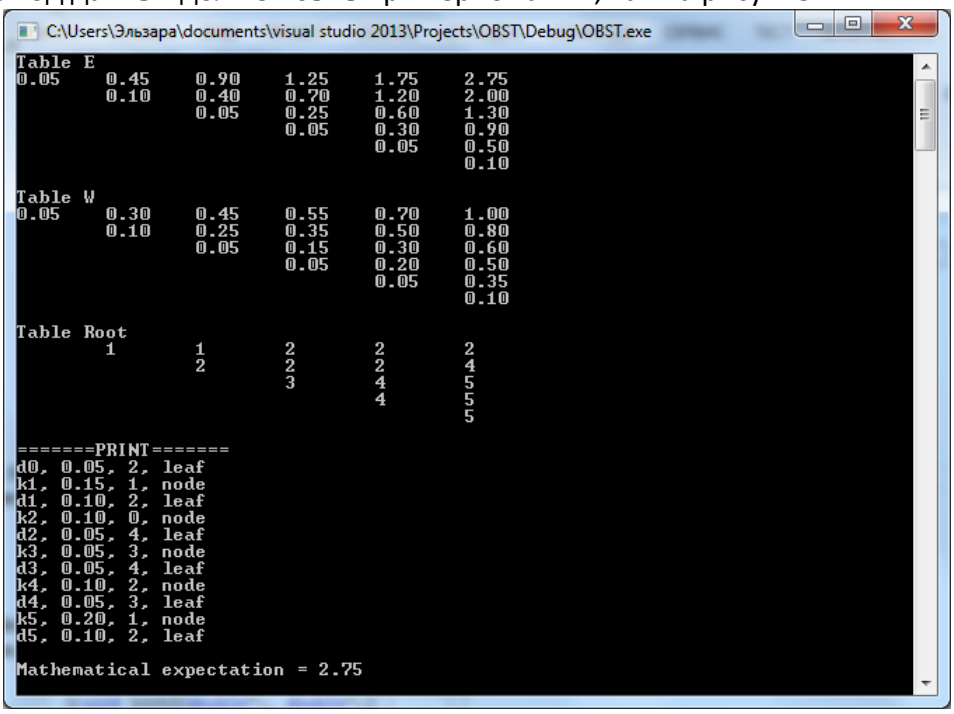
Узел	Глубина	Вероятность	Вклад
k1	1	0,15	0,30
k2	0	0,10	0,10
k3	3	0,05	0,20
k4	2	0,10	0,30
k5	1	0,20	0,40
d0	2	0,05	0,15
d1	2	0,10	0,30
d2	4	0,05	0,25
d3	4	0,05	0,25
d4	3	0,05	0,20
d5	2	0,10	0,30
Всего			2,75

В функции ***print()*** необходимо выполнить следующие изменения:

- вывести поля: ключ, вероятность данного узла, глубину, информацию о том, это ключ или фиктивный ключ;
- добавить вычисление математического ожидания стоимости поиска OBST по формуле (глубина узла+1)*вероятность:

8. Тестирование и отладка кода.

В результате вывод данных должен быть примерно таким, как на рисунке.



Задание:

1. Запустите полученный алгоритм для следующих входных данных:

i	0	1	2	3	4	5	6	7
p_i		0.04	0.06	0.08	0.02	0.10	0.12	0.14
q_i	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05

Пособия и инструменты:

1. Microsoft Visual Studio

Литература:

1. Кормен Т.Х. Алгоритмы: построение и анализ, 3-е издание : Пер. с англ. / Т.Х. Кормен, Ч.И. Лейзерсон, Р.Л. Ривест, К. Штайн. – М.: Издательский дом «Вильямс», 2013. – 1328 с.