ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ­­

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| acc. |  |  |  | С.А. Рогачев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №6 |
| АВЛ - ДЕРЕВЬЯ ПОИСКА |
| по дисциплине: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4631 |  |  |  | Д.В.Килин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2017

**1. Цель работы**

Целью работы является изучение деревьев поиска и получение практических навыков их использования.

* добавлять элементы в сбалансированное дерево поиска;
* удалять элементы из сбалансированного дерева поиска;
* искать элементы в дереве поиска с выводом количества шагов, за которое осуществляется поиск;
* выводить дерево на экран (любым способом доступным для восприятия);
* выводить список, соответствующий обходу вершин, в соответствии с вариантом задания;
* осуществлять операцию, заданную в таблице



**2. Листинг программы**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323 | #include<iostream>  #include <ctime>  using namespace std;  struct node *// структура для представления узлов дерева*  {  int key;  unsigned int height=0;  node\* left=nullptr;  node\* right=nullptr;    };  node\* current = nullptr;  int up = 0;  void print(node\* head, int up) {  if (head)  { print(head->right, up + 1);  for (int i = 0; i< up; i++) cout << " ";  cout << head->key << endl;  print(head->left, up + 1);  }  }  void StraightPrint(node \*head)  {  if (head == NULL) {  return;  }  StraightPrint(head->left);  StraightPrint(head->right);  cout << head->key << " ";  }  unsigned int height(node\*&head) {  *//if (!head) head->height = 0;*  return head ? head->height : 0;  }  void fixheight(node \*&head) {  unsigned int hl, hr;  hl = height(head->left);  hr = height(head->right);  head->height= (hl>hr ? hl : hr) + 1;  }  void LeftRotate(node\* &head) {  node\* p = head->right;  if (p == nullptr)head->right = nullptr;  head->right = p->left;  p->left = head;  fixheight(head);  fixheight(p);  head = p;  }  void RightRotate(node\* &head) {  node\* q = head->left;  if (q == nullptr)head->left = nullptr;  head->left = q->right;  q->right = head;  fixheight(head);  fixheight(q);  head = q;  }  void Balance(node \*&head) {  int bfactor = 0;  if (head == NULL) *// Базовый случай*  {  return;  }  Balance(head->left);  Balance(head->right);  fixheight(head);  if (bfactor == 2) {  if (height(head->right->left) <= height(head->right->right)) {  LeftRotate(head);    return;  }  else {  RightRotate(head->right);    LeftRotate(head);    return;  }  }  if (bfactor == -2) {  if (height(head->left->left) > height(head->left->right)) {  RightRotate(head);    return;  }  else {  LeftRotate(head->left);    RightRotate(head);    return;  }  }  }  void Create(node\*& head,node\*& temp, node\*& current) {  if (head == nullptr) {  head = temp;  current = head;  head->key = temp->key;  return;  }  else {  if (temp->key< current->key) {  if (current->left == nullptr) {  current->left = temp;  Balance(head);  current = head;  return;  }  else {  current = current->left;  Create(head, temp, current);  }  }  else {  if (current->right == nullptr) {  current->right = temp;      *//print(head, up);*    Balance(head);      current = head;    return;  }  else {  current = current->right;  Create(head, temp, current);}  }  }  }  void Generate(node\*& head) {  node\* temp = new node;  temp->key = rand() % 100 + 1;    Create(head, temp, current);    }  void Rec(node\*& Current, int num, bool &b) {  if (num == Current->key) {  cout << "detected" << endl;  b = 1;  }  else {  if (num < Current->key) {  if (Current->left == nullptr) {  cout << "disappeard" << endl;  }  else {  Current = Current->left;  Rec(Current, num, b);  }  }  if (num > Current->key) {  if (Current->right == nullptr) {  cout << "disappeard" << endl;  }  else {  Current = Current->right;  Rec(Current, num, b);}  }}}  node\* FindMin(node\* cur)  {  return p->left ? FindMin(p->left) : p;  }  node\* rmin(node\* cur)  {  if (p->left == 0)return p->right;  p->left = removemin(p->left);  Balance(p);  return p;  }  node\* remove(node\* cur, int k)  {  if (!p) return 0;  if (k < p->key)  p->left = remove(p->left, k);  else if (k > p->key)  p->right = remove(p->right, k);  else  {  node\* left = p->left;  node\* right=p->right;  delete p;  if (right==nullptr) return left;  node\* min = FindMin(right);  min->right = rmin(right);  min->left = right;  return min;  }  return p;  }  void SearchKey(node\* head) {  bool b;  if (head == nullptr) {  cout << "Empty" << endl;  return;  }  int num;  cout << "enter the key" << endl;  cin >> num;  node \* Current = head;  Rec(Current, num,b);  }  void AddKey(node\* head) {  int num;  node\* temp = new node;  cout << "Enter key" << endl;  cin >> num;  temp->key = num;  Create(head, temp, current);  }  void Search(node\* head) {  bool b = 0;  int num = 0;  cin >> num;  node\* Current=head;  Rec(Current,num,b);  cout << Current->key;  cout << endl;  StraightPrint(Current);  }  int main() {  int choice;  srand(time(0));  int level = 0;  node\* head = nullptr;  node\*u;  for (int i = 0; i < 10; i++) {  Generate(head);  }  while (1) {  cout << "Menu" << endl;  cout << "(1)Print\n"  << "(2)Add elem\n"  << "(3)Del elem\n"  << "(4) Search\n"  << "(5) back\n"  << "(6)Exit\n";  cin >> choice;  switch (choice) {  case 1: {  print(head,up);  break;  }  case 2: {  AddKey(head);  break;  }  case 3: {  int num;  cout << "---" << endl;  cin >> num;  u=remove(head,num);  Balance(u);  head = u;  print(head, up);  break;  }  case 4: {  SearchKey(head);  break;  }  case 5: {  Search(head);  break;  }  case 6: {  exit(0);  }  default: {  cout << "incorrect expression\n"  << "try again\n";  }  }  }  system("pause");  return 0;  } |

**3. Пример работы**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

**4. Вывод**

Во время работы выполнения лабораторной работы изучил AVL-деревьями. Изучил методы балансировки деревьев (при помощи поворотов). Реализовал возможность добавления, удаления элементов в дереве. Изучил возможные способы обхода дерева: прямой обратный симметричный.