Nội dung

* Cây nhị phân tìm kiếm

Tìm kiếm nhị phân và tuần tự trên mảng/danh sách liên kết, giả sử cần lưu trữ danh sách tìm kiếm (thêm/xóa) phần tử ==> sẽ phải dịch phần tử

VD. Danh sách ban đầu

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 3 | 7 | 12 | 16 | 21 | 34 | 56 |

Nếu lưu trữ bằng mảng + tìm kiếm nhị phân

Giả sử cần thêm 5 vào ==> dịch các phần tử từ vị trí thứ 3 trong mảng sang phải 1 vị trí: thời gian O(n)  
Giả sử xóa 2 ==> phải dịch các phần tử từ vị trí sau 2 sang trái 1 vị trí : O(n)

Nếu lưu trữ dùng sanh sách liên kết: KHÔNg cần dịch phần tử, tuy nhiên chỉ có thể tìm kiếm tuần tự

Có cách nào tốt cho cả thêm xóa lẫn tìm kiếm?

Cây nhị phân tìm kiếm

* Thêm/xóa/tìm kiếm đều mất thời gian cỡ O(logn)
* Là một dạng của cấu trúc liên kết, mỗi nút có tối đa 2 con (con trái/con phải)

Cài đặt cây nhị phân tìm kiếm?

VÍ DỤ 1. Cây nhị phân tìm kiếm với nút chỉ là 1 số nguyên và là khóa

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <stack>    using namespace std;  typedef struct TreeNode  {  int key;  struct TreeNode \*leftChild;  struct TreeNode \*rightChild;  } NODE;    NODE\* search\_loop(NODE \*root, int searchkey)  {  while(root!=NULL && root-> key!= searchkey)  if(searchkey <root->key) root=root->leftChild;  else root=root->rightChild;  return root;  }    // cấp phat bo nho dong cho nut moi  NODE\* makeNewNode(int key)  {  NODE\* newNode = (NODE\*)malloc(sizeof(NODE));  newNode->key = key;  newNode->leftChild = NULL;  newNode->rightChild = NULL;  return newNode;  }    // root phai khac NULL  int insert(NODE\* root, int key)  {  NODE\* pRoot = root; // con tro toi truoc vi tri can chen  while (true) {  if (pRoot->key == key) return -1;//duplicate  if (pRoot->key > key) { // them vao con trai  if (pRoot->leftChild == NULL) {  pRoot->leftChild = makeNewNode(key);  break;  }  else pRoot = pRoot->leftChild;  }  else { // them vao con phai  if (pRoot->rightChild == NULL) {  pRoot->rightChild = makeNewNode(key);  break;  }  else pRoot = pRoot->rightChild;  }  }  return 0;//success  }    int remove\_node(NODE \*&root)  {  if(root == NULL) return -1; //remove null  NODE \*ptr = root; //remember this node for delete later    if(root->leftChild == NULL) root=root->rightChild;  else if(root->rightChild == NULL) root=root->leftChild;  else //find the rightmost node on the left sub tree  {  NODE \*preP = root;  ptr = root->leftChild;  while(ptr->rightChild != NULL)  {  preP = ptr;  ptr = ptr->rightChild;  }  root->key = ptr->key;  if(preP == root) root->leftChild = ptr->leftChild;  else preP->rightChild = ptr->leftChild;  }  free(ptr);  return 0;// remove success  }    int remove(NODE\*& root, int key)  {  if (root == NULL || key == root->key)  return remove\_node(root);  else if (key < root->key)  return remove(root->leftChild, key);  else return remove(root->rightChild, key);  }    void printTree\_inorder\_rec(NODE \*root)  {  if(NULL==root) return;  printTree\_inorder\_rec(root->leftChild);  printf("%d, ",root->key);  printTree\_inorder\_rec(root->rightChild);  }      int main()  {  NODE \* root;  // them lan luot 7,4,8,2,5,10  root = makeNewNode(7);  insert(root,4);  insert(root,8);  insert(root,2);  insert(root,5);  insert(root,10);  printTree\_inorder\_rec(root);  printf("\n");  remove(root,7);  printf("Cay sau khi xoa 7:\n");  printTree\_inorder\_rec(root);  return 0;  } |

**Bài 1**. Mở rộng code để thực hiện thêm việc sau

* Tạo cây với các khóa là số nguyên từ file dạng như sau

|  |
| --- |
| N  2 4 5 6 8 9 5 |

Với n là số lượng khóa và dòng tiếp theo là giá trị các khóa

* Chức năng thêm khóa với khóa được nhập từ bàn phím
* Chức năng tìm và xóa giá trị khóa được nhập từ bàn phím
* Chức năng tìm kiếm khóa trên cây
* Chức năng ghi danh sách hiện tại trên cây ra file (lưu theo dạng như file đọc vào, và duyệt theo thứ tự giữa)

|  |
| --- |
| // cay phai khac NULL  void insertNewKeys(NODE\* root)  {  int key;  char traloi;  do  {  printf("Gia tri muon them vao: ");  scanf("%d",&key);  insert(root,key);  printf("Co muon them khoa tiep(C/K): ");  //fflush(stdin);  //while(getchar()!='\n');  //scanf("%\*[^\n]%\*c");  scanf("%\*[^\n]");  scanf("%\*c");  traloi = getchar();  }  while(traloi!='K' && traloi!='k');  } |
| NODE\* insertNewKeys2(NODE\* root)  {  int key;  char traloi;  do  {  printf("Gia tri muon them vao: ");  scanf("%d",&key);  if(NULL==root) root = makeNewNode(key);  else insert(root,key);  printf("Co muon them khoa tiep(C/K): ");  //fflush(stdin);  //while(getchar()!='\n');  //scanf("%\*[^\n]%\*c");  scanf("%\*[^\n]");  scanf("%\*c");  traloi = getchar();  }  while(traloi!='K' && traloi!='k');    return root;  } |

**Bài 2**. xây dựng cây tìm kiếm nhị phân để lưu trữ thông tin về danh bạ điện thoại

Tìm theo SDT và trả về họ tên người nếu có (khóa tìm kiếm là SDT)

|  |
| --- |
| typedef struct TreeNode  {  char sdt[15];  char hoten[255];  struct TreeNode \*leftChild;  struct TreeNode \*rightChild;  } NODE; |

Xây dựng các hàm

* Tạo 1 nút mới với thông tin là sdt và họ tên: NODE\* makeNewNode(char\* sdt, char\* hoten)
* Hàm thêm 1 nút mới vào cây : int insert(NODE\* root, char\* sdt, char\* hoten)
* Hàm tìm kiếm để tìm và trả về thông tin người có SDT: NODE\* search(NODE \*root, char\* sdt)  
  Nếu tìm thấy thì trả về thông tin, còn không thấy thì trả về NULL
* Hàm xóa 1 SDT trong danh bạ: int remove (NODE \*root, char\* sdt)
* Hàm ghi danh bạ hiện tại ra file
* hàm đọc danh bạ từ file vào cây nhị phân tìm kiếm

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  typedef struct TreeNode  {  char sdt[15];  char hoten[255]; // cap phat luon khi tao nut moi  struct TreeNode \*leftChild;  struct TreeNode \*rightChild;  } NODE;    NODE\* makeNewNode(char\* sdt, char\* hoten)  {  NODE\* newNode = (NODE\*)malloc(sizeof(NODE));  strcpy(newNode->sdt, sdt);  strcpy(newNode->hoten, hoten);  newNode->leftChild = NULL;  newNode->rightChild = NULL;  return newNode;  }    int insert(NODE\* root, char\* sdt, char\* hoten)  {  NODE\* pRoot = root; // con tro toi truoc vi tri can chen  while (true) {  if (strcmp(pRoot->sdt,sdt)==0) return -1;//duplicate  if (strcmp(pRoot->sdt,sdt)>0) { // them vao con trai  if (pRoot->leftChild == NULL) {  pRoot->leftChild = makeNewNode(key,sdt,hoten);  break;  }  else pRoot = pRoot->leftChild;  }  else { // them vao con phai  if (pRoot->rightChild == NULL) {  pRoot->rightChild = makeNewNode(key,sdt,hoten);  break;  }  else pRoot = pRoot->rightChild;  }  }  return 0;//success  }    //viet not cac ham con lai  int main()  {  NODE \* root=NULL;  // them lan luot 7,4,8,2,5,10  /\*  root = makeNewNode(7);  insert(root,4);  insert(root,8);  insert(root,2);  insert(root,5);  insert(root,10);  insertNewKeys(root);  \*/  root = insertNewKeys2(root);    printTree\_inorder\_rec(root);  printf("\n");  remove(root,7);  printf("Cay sau khi xoa 7:\n");  printTree\_inorder\_rec(root);  return 0;  } |

**Bài 3**. xây dựng ứng dụng từ điển

Mỗi từ sẽ gồm 2 phần

* Word: từ (tối đa 30 ky tu): KHÔNG phân biệt chữ hoa/ thường
* Mean: nghĩa của từ này (khong qua 255 ky tu)

Hãy xây dựng ứng dụng dùng cây nhị phân tìm kiếm để lưu trữ từ điển này với các tính năng

1. Thêm từ mới từ bàn phím: người dùng sẽ nhập vào 1 từ mới và nghĩa của nó. Nếu từ này chưa có trong từ điển --> thêm vào
2. Tra cứu nghĩa của từ: người dùng nhập vào 1 từ, chương trình sẽ tìm và trả về nghĩa tương ứng của từ đó (nếu từ đó có trong từ điển)
3. Ghi nội dung từ điển hiện tại ra file
4. Cập nhật nội dung từ điển từ file (người dùng cần nhập vào đường dẫn tới file từ điển). Khi cập nhật :
   1. chỉ thêm những từ chưa có trong từ điển hiện tại,
   2. nếu từ đã có rồi thì chỉ cần cập nhật nghĩa của từ.

Format của file từ điển dạng

|  |
| --- |
| N  $Hello#xin chao, loi chao#  $Hi#xin chao, loi chao than mat#  $take#nhan lay  Cam,nam  Hoac nghia khac# |

Với n là số lượng từ trong từ điển

Dòng tiếp theo là từ và nghĩa của từ được ngăn cách các phần bởi dấu # (vì nghĩa của từ có thể trên nhiều dòng nên bắt buộc phải dùng dấu phân cách) và bắt đầu từ mới sẽ là ký hiệu $