|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Односвязанный список | Двухсвязанный список | Дерево | Хэш | Массив (отсортированный) |
| Добавление | 1 | N (сох. Порядок) | Log2(N) | 1 | Log2(N) |
| Удаление | N | N | Log2(N) | 1 | Log2(N) |
| ПОиск | N | N | Log2(N) | 1 | Log2(N) |
| КПД памяти  K – размер данных | K/(K+4) | K/(K+8) | K/(K+8) | K/(K+4) | 1 |

XXXXXXX – массив – последовательность памяти

Люб. Струк. Данных – не последовательность памяти

ОДносвяз. Список характез. Головой (хвост)

Двухсвяз. Список характер. Головой и хвостом

Циклич. Двух. Список характе. Люб. Элементов

Дерево характериз корнем дерева

Хэш – массив структур данных

// \_2011\_03\_29\_1\_1.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

class TNode {

public:

TNode \*next;

TNode \*prev;

int data;

TNode(int \_data, TNode \*\_prev, TNode \*\_next) {

data = \_data;

prev = \_prev;

next = \_next;

}

};

class LinkTwo {

public:

TNode \*head;

TNode \*tail;

LinkTwo() {

head = NULL;

tail = NULL;

}

void clear() {

while (head != NULL) {

if (head->next == NULL) {

delete head;

head = NULL;

} else {

head = head->next;

delete head->prev;

}

}

tail = NULL;

}

void addToHead(int newData) {

if (head == NULL) {

head = new TNode(newData, NULL, NULL);

tail = head;

return;

}

TNode \*newHead = new TNode(newData, NULL, head);

head->prev = newHead;

head = newHead;

}

void addToTail(int newData) {

if (tail == NULL) {

tail = new TNode(newData, NULL, NULL);

head = tail;

return;

}

TNode \*newTail = new TNode(newData, tail, NULL);

tail->next = newTail;

tail = newTail;

}

//удаление первого вхождения

void del(int delData) {

if (head == NULL) {

return;

}

if (head == tail && head->data == delData) {

delete head;

head = NULL;

tail = NULL;

return;

}

if (head->data == delData) {

head = head->next;

delete head->prev;

head->prev = NULL;

return;

}

if (tail->data == delData) {

tail = tail->prev;

delete tail->next;

tail->next = NULL;

return;

}

TNode \*temp = head;

while (temp != NULL) {

if (temp->data == delData) {

break;

}

temp = temp->next;

}

if (temp == NULL) {

return;

}

temp->prev->next = temp->next;

temp->next->prev = temp->prev;

delete temp;

//XXXX

}

void printFromHead() {

TNode \*temp = head;

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->next;

}

}

void printFromTail() {

TNode \*temp = tail;

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->prev;

}

}

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

return 0;

}

Дерево – связанный ориентированный граф, из каждой вершины которого выходит не более 2х связей и входит не более одной связи.

Дерево – корень + левое поддерево + правое поддерево

Левое и/или правое поддерево может быть пустым

// \_2011\_03\_29\_1\_1.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

class TNode {

public:

TNode \*left;

TNode \*right;

int data;

TNode(int \_data, TNode \*\_left, TNode \*\_right) {

data = \_data;

left = \_left;

right = \_right;

}

};

class StackNode {

public:

StackNode \*next;

TNode \*data;

StackNode(TNode \*\_data, StackNode \*\_next) {

data = \_data;

next = \_next;

}

};

class Stack {

public:

StackNode \*head;

Stack() {

head = NULL;

}

void pushValue(TNode \*data) {

head = new StackNode(data, head);

}

TNode \*getValue() {

TNode \*res = head->data;

StackNode \*t = head;

head = head->next;

delete t;

return res;

}

bool isEmpty() {

return (head == NULL);

}

};

class BinaryTree {

public:

TNode \*root;

BinaryTree() {

root = NULL;

}

void addRec(int data, TNode \*&localRoot) {

if (localRoot == NULL) {

localRoot = new TNode(data, NULL, NULL);

return;

}

if (localRoot->data > data) {

addRec(data, localRoot->left);

} else {

addRec(data, localRoot->right);

}

}

void addNode(int data) {

//addRec(data, root);

if (root == NULL) {

root = new TNode(data, NULL, NULL);

return;

}

TNode \*temp = root;

while (temp != NULL) {

if (temp->data > data) {

if (temp->left == NULL) {

temp->left = new TNode(data, NULL, NULL);

return;

}

temp = temp->left;

} else {

if (temp->right == NULL) {

temp->right = new TNode(data, NULL, NULL);

return;

}

temp = temp->right;

}

}

}

void cleatNotRec() {

//обход дерева в ширину (если стек)

//обход дерева в шлубину (если очередь)

Stack \*s = new Stack();

s->pushValue(root);

while (s->isEmpty() == false) {

TNode \*temp = s->getValue();

if (temp == NULL) {

continue;

}

s->pushValue(temp->left);

s->pushValue(temp->right);

delete temp;

}

delete s;

}

bool findNotRec(int searchValue) {

return findRec(searchValue, root);

/\*

if (root == NULL) {

return false;

}

TNode \*temp = root;

while (temp != NULL) {

if (temp->data == searchValue) {

return true;

}

temp = (temp->data > searchValue) ?

temp->left : temp->right;

}

return false;

\*/

}

void clear() {

clearTree(root);

}

void clearTree(TNode \*&localRoot) {

if (localRoot == NULL) {

return;

}

clearTree(localRoot->left);

clearTree(localRoot->right);

delete localRoot;

localRoot = NULL;

}

bool findRec(int searchValue, TNode \*localRoot) {

if (localRoot == NULL) {

return false;

}

if (localRoot->data == searchValue) {

return true;

}

if (localRoot->data > searchValue) {

return findRec(searchValue, localRoot->left);

} else {

return findRec(searchValue, localRoot->right);

}

}

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

return 0;

}