

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

Trường: ĐH CNTP TP.HCM Khoa: Công nghệ thông tin Bộ môn: Công nghệ phần mềm. Môn học: TH Cấu trúc dữ liệu & giải thuật	BÀI 8. CÂY NHIỀU NHÁNH (B-Tree)	
---	--	--

A. MỤC TIÊU:

- Hiểu được cấu trúc dữ liệu động.
- Lập trình và vận dụng được cấu trúc dữ liệu cây nhiều nhánh B-Tree vào từng bài toán cụ thể.
- Làm được các bài tập áp dụng cây nhiều nhánh B-Tree.

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	1	

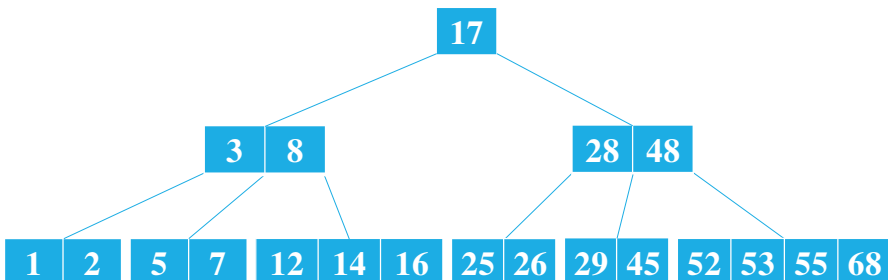
C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

1. Khái niệm cây nhiều nhánh

Cây nhiều nhánh M-Phân là một cây tìm kiếm tự cân bằng thỏa các điều kiện sau:

- Mỗi node có tối đa M node con (*cây con*).
- Một cây M-Phân đầy đủ có chiều cao $\log_M N$.
- Mỗi node có tối đa **M** cây con, và có **M-1** khóa.
- Các khóa trong mỗi node (cây con) được sắp xếp tăng.
- Các khóa trong cây con thứ i đều nhỏ hơn khóa i.
- Các khóa trong cây con thứ (i+1) đều lớn hơn khóa i.



Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

Cây M-Phân tìm kiếm có các tính chất sau:

- Tất cả node lá có cùng mức.
- Tất cả các node trung gian (*trừ node gốc*) có nhiều nhất M cây con và có ít nhất $M/2$ cây con (khác rỗng).
- Mỗi node hoặc là node lá hoặc có $k+1$ cây con (k là số khoá của node này).
- Node gốc có nhiều nhất M cây con hoặc có thể có 2 cây con (Node gốc có 1 khoá và không phải là node lá) hoặc không chứa cây con nào (Node gốc có 1 khoá và cũng là node lá).
- Tại thời điểm chèn một nút đầy đủ, cây chia thành hai phần và khóa có giá trị trung bình được chèn tại nút cha.
- Hoạt động hợp nhất diễn ra khi các nút bị xóa.

2. Cấu trúc của một nút



Mỗi nút của cây nhiều nhánh M-phân gồm ba thành phần:

- Số lượng khóa có trong nút: **numTree**.
- Một mảng chứa các khóa: **Keys**.
- Một mảng có M con trỏ đến các nút con (cây con): **Branch**.

3. Các thao tác trên cây nhiều nhánh

a. Khai báo

```
#define M 5 /* Cây M-Phân */
typedef int ItemType;
struct BNode {
    int numTree; /* numTree < M (Số khóa trong nút
sẽ luôn ít hơn M của cây B) */
    ItemType Keys[M - 1]; /*Mảng chứa các khóa*/
    BNode* Branch[M]; /* numTree + 1 (M con trỏ sẽ
được sử dụng) */
};
struct BTree {
    BNode* Root; /* Con trỏ quản lý node gốc */
```

```
};  
typedef BNode* NodePtr; /* Kiểu dữ liệu con trỏ node  
*/
```

b. Khởi tạo cây rỗng

```
/* Initialize BTree */  
void initBTree(BTree &bt)  
{  
    bt.Root = NULL;  
}
```

c. Duyệt cây để xem nội dung

```
void displayBTree(NodePtr pRoot, int blanks) {  
    if (pRoot) {  
        int i;  
        for (i = 1; i <= blanks; i++)  
            printf(" ");  
        for (i = 0; i < pRoot->numTree; i++)  
            printf("%d ", pRoot->Keys[i]);  
        printf("\n");  
        for (i = 0; i <= pRoot->numTree; i++)  
            displayBTree(pRoot->Branch[i], blanks+10);  
    } /*End of if*/  
} /*End of displayBTree()*/
```

d. Tìm vị trí của một nút có chứa giá trị đã cho

```
int searchPosition(ItemType key, ItemType *keyArray,  
int numTree) {  
    int pos = 0;  
    while (pos < numTree && key > keyArray[pos])  
        pos++;  
    return pos;  
} /*End of searchPosition()*/
```

e. Thêm nút vào cây

```
void insert(NodePtr &root, ItemType key) {  
    NodePtr newNode;  
    ItemType upKey;
```

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
KeyStatus result;
result = insertNode(root, key, &upKey, &newNode);
if (result == Duplicate)
    printf("Key already available\n");
if (result == InsertIt)
{
    NodePtr upRoot = root;
    //root = (BNode*)malloc(sizeof(BNode));
    root = new BNode;
    root->numTree = 1;
    root->Keys[0] = upKey;
    root->Branch[0] = upRoot;
    root->Branch[1] = newNode;
}/*End of if */
}/*End of insert()*/
```

```
KeyStatus insertNode(NodePtr pCurrent, ItemType key,
ItemType *upKey, NodePtr* newNode) {
    NodePtr newPtr, lastPtr;
    int pos, i, numTree, splitPos;
    ItemType newKey, lastKey;
    KeyStatus result;
    if (pCurrent == NULL)
    {
        *newNode = NULL;
        *upKey = key;
        return InsertIt;
    }
    numTree = pCurrent->numTree;
    pos = searchPosition(key, pCurrent->Keys, numTree);
    if (pos < numTree && key == pCurrent->Keys[pos])
        return Duplicate;
    result = insertNode(pCurrent->Branch[pos], key,
    &newKey, &newPtr);
    if (result != InsertIt)
        return result;
```

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
/*If Keys in node is less than M-1 where M is order of
B tree*/
if (numTree < M - 1)
{
    pos = searchPosition(newKey, pCurrent->Keys,
numTree);
    /*Shifting the key and pointer right for inserting
the new key*/
    for (i = numTree; i > pos; i--)
    {
        pCurrent->Keys[i] = pCurrent->Keys[i - 1];
        pCurrent->Branch[i + 1] = pCurrent->Branch[i];
    }
    /*Key is inserted at exact location*/
    pCurrent->Keys[pos] = newKey;
    pCurrent->Branch[pos + 1] = newPtr;
    ++pCurrent->numTree; /*incrementing the number of
Keys in node*/
    return Success;
}/*End of if */
/*If Keys in nodes are maximum and position of node to
be inserted is last*/
if (pos == M - 1)
{
    lastKey = newKey;
    lastPtr = newPtr;
}
else /*If Keys in node are maximum and position of
node to be inserted is not last*/
{
    lastKey = pCurrent->Keys[M - 2];
    lastPtr = pCurrent->Branch[M - 1];
    for (i = M - 2; i > pos; i--)
    {
        pCurrent->Keys[i] = pCurrent->Keys[i - 1];
        pCurrent->Branch[i + 1] = pCurrent->Branch[i];
    }
    pCurrent->Keys[pos] = newKey;
    pCurrent->Branch[pos + 1] = newPtr;
```

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
    }
    splitPos = (M - 1) / 2;
    (*upKey) = pCurrent->Keys[splitPos];

    (*newNode) = new BNode; /*Right node after split*/
    pCurrent->numTree = splitPos; /*No. of Keys for left
splitted node*/
    (*newNode)->numTree = M - 1 - splitPos; /*No. of Keys
for right splitted node*/
    for (i = 0; i < (*newNode)->numTree; i++)
    {
        (*newNode)->Branch[i] = pCurrent->Branch[i +
splitPos + 1];
        if (i < (*newNode)->numTree - 1)
            (*newNode)->Keys[i] = pCurrent->Keys[i +
splitPos + 1];
        else
            (*newNode)->Keys[i] = lastKey;
    }
    (*newNode)->Branch[(*newNode)->numTree] = lastPtr;
    return InsertIt;
} /*End of insertNode()*/
```

f. Xóa một nút bất kỳ

```
void deleteNode(NodePtr &root, ItemType key) {
    NodePtr upRoot;
    KeyStatus result;
    result = remove(root, root, key);
    switch (result)
    {
        case SearchFailure:
            printf("Key %d is not available\n", key);
            break;
        case LessKeys:
            upRoot = root;
            root = root->Branch[0];
            free(upRoot);
    }
}
```

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
        printf("\nBtree after removing the %d
value:\n", key);
        displayBTree(root, 0);
        break;
    case Success:
        printf("\nBtree after removing the %d
value:\n", key);
        displayBTree(root, 0);
        break;
    }/*End of switch*/
}/*End of delnode()*/

KeyStatus remove(NodePtr &root, NodePtr pCurrent,
ItemType key) {
    int pos, i, pivot, numTree, min;
    ItemType *keyArray;
    KeyStatus result;
    NodePtr *Branch, leftPtr, rightPtr;

    if (pCurrent == NULL)
        return SearchFailure;
    /*Assigns values of node*/
    numTree = pCurrent->numTree;
    keyArray = pCurrent->Keys;
    Branch = pCurrent->Branch;
    min = (M - 1) / 2; /*Minimum number of Keys*/

    pos = searchPosition(key, keyArray, numTree);
    if (Branch[0] == NULL)
    {
        if (pos == numTree || key < keyArray[pos])
            return SearchFailure;
        /*Shift Keys and pointers left*/
        for (i = pos + 1; i < numTree; i++)
        {
            keyArray[i - 1] = keyArray[i];
            Branch[i] = Branch[i + 1];
        }
    }
}
```

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
    }
    return --pCurrent->numTree >= (pCurrent == root ?
1 : min) ? Success : LessKeys;
}/*End of if */

if (pos < numTree && key == keyArray[pos])
{
    NodePtr qp = Branch[pos], qp1;
    ItemType nkey;
    while (1)
    {
        nkey = qp->numTree;
        qp1 = qp->Branch[nkey];
        if (qp1 == NULL)
            break;
        qp = qp1;
    }/*End of while*/
    keyArray[pos] = qp->Keys[nkey - 1];
    qp->Keys[nkey - 1] = key;
}/*End of if */
result = remove(root, Branch[pos], key);
if (result != LessKeys)
    return result;

if (pos > 0 && Branch[pos - 1]->numTree > min)
{
    pivot = pos - 1; /*pivot for left and right node*/
    leftPtr = Branch[pivot];
    rightPtr = Branch[pos];
    /*Assigns values for right node*/
    rightPtr->Branch[rightPtr->numTree + 1] =
rightPtr->Branch[rightPtr->numTree];
    for (i = rightPtr->numTree; i > 0; i--)
    {
        rightPtr->Keys[i] = rightPtr->Keys[i - 1];
        rightPtr->Branch[i] = rightPtr->Branch[i - 1];
    }
    rightPtr->numTree++;
}
```


Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
rightPtr->Keys[0] = keyArray[pivot];
rightPtr->Branch[0] = leftPtr->Branch[leftPtr->numTree];
keyArray[pivot] = leftPtr->Keys[--leftPtr->numTree];
return Success;
}/*End of if */
if (pos < numTree && Branch[pos + 1]->numTree > min)
{
    pivot = pos; /*pivot for left and right node*/
    leftPtr = Branch[pivot];
    rightPtr = Branch[pivot + 1];
    /*Assigns values for left node*/
    leftPtr->Keys[leftPtr->numTree] = keyArray[pivot];
    leftPtr->Branch[leftPtr->numTree + 1] = rightPtr->Branch[0];
    keyArray[pivot] = rightPtr->Keys[0];
    leftPtr->numTree++;
    rightPtr->numTree--;
    for (i = 0; i < rightPtr->numTree; i++)
    {
        rightPtr->Keys[i] = rightPtr->Keys[i + 1];
        rightPtr->Branch[i] = rightPtr->Branch[i + 1];
    }/*End of for*/
    rightPtr->Branch[rightPtr->numTree] = rightPtr->Branch[rightPtr->numTree + 1];
    return Success;
}/*End of if */

if (pos == numTree)
    pivot = pos - 1;
else
    pivot = pos;

leftPtr = Branch[pivot];
rightPtr = Branch[pivot + 1];
/*merge right node with left node*/
leftPtr->Keys[leftPtr->numTree] = keyArray[pivot];
```

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
    leftPtr->Branch[leftPtr->numTree + 1] = rightPtr->Branch[0];
    for (i = 0; i < rightPtr->numTree; i++)
    {
        leftPtr->Keys[leftPtr->numTree + 1 + i] =
            rightPtr->Keys[i];
        leftPtr->Branch[leftPtr->numTree + 2 + i] =
            rightPtr->Branch[i + 1];
    }
    leftPtr->numTree = leftPtr->numTree + rightPtr->numTree + 1;
    free(rightPtr); /*Remove right node*/
    for (i = pos + 1; i < numTree; i++)
    {
        keyArray[i - 1] = keyArray[i];
        Branch[i] = Branch[i + 1];
    }
    return --pCurrent->numTree >= (pCurrent == root ? 1 :
min) ? Success : LessKeys;
}/*End of remove()*/
```

g. Tìm kiếm một nút có chứa giá trị đã cho

```
NodePtr searchNode(NodePtr root, ItemType key) {
    int pos, numTree;
    NodePtr pCurrent = root;
    while (pCurrent)
    {
        numTree = pCurrent->numTree;
        pos = searchPosition(key, pCurrent->Keys,
numTree);
        if (pos < numTree && key == pCurrent->Keys[pos])
        {
            return pCurrent;
        }
        pCurrent = pCurrent->Branch[pos];
    }
    return NULL;
}
```

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
 }/*End of searchNode()*/

void search(NodePtr root, ItemType key)
{
    int pos, i, numTree;
    NodePtr pCurrent = root;
    printf("Search path:\n");
    while (pCurrent)
    {
        numTree = pCurrent->numTree;
        for (i = 0; i < pCurrent->numTree; i++)
            printf(" %d", pCurrent->Keys[i]);
        printf("\n");
        pos = searchPosition(key, pCurrent->Keys,
numTree);
        if (pos < numTree && key == pCurrent-
>Keys[pos])
        {
            printf("Key %d found in position %d of
last dispalyed node\n", key, pos);
            return;
        }
        pCurrent = pCurrent->Branch[pos];
    }
    printf("Key %d is not available\n", key);
}/*End of search()*/
```

II. Bài tập hướng dẫn mẫu

Bài 1. Viết chương trình quản lý các số nguyên bằng Cây nhiều nhánh (*Cây M-Phân*)?

- **Bước 1:** Tạo một Project mới.
- **Bước 2:** Add vào project 2 file sau: **BTree.h**, **BTree.cpp**
- **Bước 3:** Chạy thử chương trình và kiểm tra kết quả thực hiện.

III. Bài tập ở lớp

Bài 2. Dựa vào Bài tập mẫu 1. Hãy hoàn thiện chương trình với

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

những chức năng sau:

- a. Bổ sung dạng menu cho chương trình.
- b. Tạo lại cây từ một mảng **a** có ít nhất 50 số nguyên.
- c. Tạo lại cây bằng cách đọc dữ liệu từ 1 file text.
- d. Thêm một vài phần tử là các số nguyên bất kỳ, và quan sát sự thay đổi của cây (*tách và gộp các nút*).
- e. Xóa một vài phần tử bất kỳ của cây, và quan sát sự thay đổi của cây (*tách và gộp các nút*).
- f. Tìm kiếm một vài giá trị để kiểm tra có tồn tại trên cây hay không?
- g. Xuất ra các nút chứa ít giá trị nhất.
- h. Xuất ra các nút chứa nhiều giá trị nhất.
- i. Đếm số nút chứa toàn giá trị là các số nguyên tố.
- j. Tính tổng giá trị các nút trên cây.

Bài 3. Cho cây M-Phân mà mỗi nút chứa M-1 phân số. Hãy viết chương trình để thực hiện những chức năng sau:

- a. Tạo menu của chương trình.
- b. Tạo cây từ một mảng **a** có ít nhất 30 phân số.
- c. Duyệt cây và quan sát kết quả.
- d. Thêm 1 vài phân số bất kỳ vào cây.
- e. Tìm kiếm 1 phân số **p** có trên cây hay không?
- f. Xóa một phân số **p** trên cây (*phân số **p** nhập từ bàn phím*).
- g. Xóa những phân số có giá trị >2 .
- h. Xóa những phân số có mẫu số là số nguyên tố.
- i. Đếm số lượng phân số có tử số lớn hơn mẫu số.
- j. Xóa toàn bộ cây.

VI. Bài tập về nhà

Bài 4. Tiếp theo Bài tập 3. Hãy bổ sung thêm những chức năng sau:

- a. Liệt kê các phân số có tử số nhỏ hơn mẫu số.
- b. Đếm có bao nhiêu phân số có mẫu số là số chẵn.

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

- c. Liệt kê các phân số có tử số và mẫu số đồng thời là số hoàn thiện.

Bài 5. Cây M-Phân lưu trữ dữ liệu là một từ điển Anh-Việt (Mỗi phần tử trong nút có dữ liệu gồm 2 trường: **word** là khóa chứa một từ tiếng anh, **mean** là nghĩa tiếng Việt). Hãy xây dựng cây từ điển M-Phân với những chức năng sau:

- a. Xây dựng chương trình dạng menu.
- b. Tạo cây từ 1 file text lưu từ điển Anh-Việt.
- c. Duyệt cây để xem nội dung.
- d. Tra cứu nghĩa của 1 từ bất kỳ.
- e. Thêm một từ bất kỳ vào cây, duyệt lại cây để xem kết quả.
- f. Xóa một từ bất kỳ khỏi cây, duyệt lại cây để xem kết quả.
- g. Bổ sung hay chỉnh sửa nghĩa của 1 từ bất kỳ.
- h. Cho biết số lượng từ của từ điển.
- i. Xóa toàn bộ cây.

-- HẾT --