

BHT Đoàn khoa MMT&TT – Training cuối kì II K16

— TRAINING — CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

Trainers:



Trần Công Thành – ATCL2021
Nguyễn Văn Long – ATTT2021
Lê Khắc Trung Nam – ATCL2021
Phạm Thái Bảo – ATCL2021

Nội Dung



1. Các thuật toán sắp xếp

2. Cây nhị phân tìm kiếm & B-tree



3. Bảng băm(hash table)



4. Lý thuyết đồ thị





Đặt câu hỏi tại đây



CÁC THUẬT TOÁN SẮP XẾP

1. Interchange Sort:

Ý tưởng:

- Xuất phát từ đầu dãy, tìm tất các cặp nghịch thế chứa phần tử này, triệt tiêu chúng bằng cách đổi chỗ 2 phần tử trong cặp nghịch thế. Lặp lại xử lý trên với phần tử kế trong dãy.

Độ phức tạp: $O(n^2)$

- Tốt nhất: so sánh $n(n-1)/2$, hoán vị 0
- Xấu nhất: so sánh $n(n-1)/2$, hoán vị $n(n-1)/2$



1. Interchange Sort:

Cài đặt:



```
1 void interchangesort(int a[], int n){  
2     for (int i = 0; i < n-1; i++)  
3         for (int j = i+1; j < n; j++)  
4             if (a[j] < a[i])  
5                 swap(a[j], a[i]);  
6 }  
7
```



2. Selection Sort:

Ý tưởng:

- Tìm kiếm phần tử nhỏ nhất trong dãy có N phần tử, đưa nó về đầu dãy, tiếp tục với N-1 phần tử còn lại. Vòng lặp kết thúc khi dãy hiện hành chỉ còn một phần tử.

Độ phức tạp: $O(n^2)$

- Tốt nhất: so sánh $n(n-1)/2$, gán 0
- Xấu nhất: so sánh $n(n-1)/2$, gán $3n(n-1)/2$



2. Selection Sort:

Cài đặt:



```
11 void seclectionsort (int a[], int n){  
12     for (int i = 0; i < n-1; i++){  
13         int min = i;  
14         for (int j = i+1; j < n; j++){  
15             if (a[j] < a[min]) min = j;  
16         }  
17         swap(a[i], a[min]);  
18     }  
19 }
```



3. Bubble Sort:

Ý tưởng:

- Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn trong cặp phần tử đó về vị trí đúng đầu dãy hiện hành, sau đó sẽ không xét đến nó ở bước tiếp theo, do vậy ở lần xử lý thứ i sẽ có vị trí đầu dãy là i .
- Lặp lại xử lý trên cho đến khi không còn cặp phần tử nào để xét.

Độ phức tạp: $O(n^2)$

- Tốt nhất: so sánh $n(n-1)/2$, hoán vị 0
- Xấu nhất: so sánh $n(n-1)/2$, hoán vị $n(n-1)/2$



3. Bubble Sort:

Cài đặt:



```
1 void bubblesort(int a[], int n){  
2     for (int i = 0; i < n-1; i++){  
3         for (int j = n-1 ; j > i; j--){  
4             if (a[j] <= a[j-1])  
5                 swap(a[j], a[j-1]);  
6         }  
7     }  
8 }
```



4. Insertion Sort:

Ý tưởng:

- Giả sử ta có dãy $a[0], a[1], \dots, a[i-1]$ đã được sắp xếp, ta cần chèn $a[i]$ vào đúng vị trí của nó để $a[0] \dots a[i]$ theo thứ tự. Vị trí cần chèn j cần thỏa điều kiện $a[j] \leq a[i]$. Thuật toán lặp đúng $n - 1$ lần để n phần tử được sắp xếp.

Độ phức tạp: $O(n^2)$

- Tốt nhất: so sánh $n-1$, gán $2(n-1)$
- Xấu nhất: so sánh $n(n-1)/2$, gán $n(n+1)/2 - 1$



4. Insertion Sort:

Cài đặt:

```
1 void insertionsort(int a[], int n){
2     for (int i = 1; i < n; i++){
3         int pos= i-1;
4         int x= a[i];
5         while (pos >=0 && a[pos] > x){
6             a[pos+1] = a[pos];
7             pos--;
8         }
9         a[pos+1] = x;
10    }
11 }
```



5. Quick Sort:

Ý tưởng:

- Dựa trên thuật toán chia để trị (Divide and Conquer), giải thuật Quick Sort phân hoạch dãy a_1, a_2, \dots, a_n thành 2 phần:

- Phần 1: Gồm các phần tử có giá trị bé hơn hoặc bằng pivot
- Phần 2: Gồm các phần tử có giá trị lớn hơn hoặc bằng pivot

Tiếp tục công việc với mỗi mảng con (chọn pivot, phân đoạn) cho tới khi mảng được sắp xếp.

Độ phức tạp:

- Tốt nhất: $O(n \cdot \log(n))$
- Trung bình: $O(n \cdot \log(n))$
- Xấu nhất: $O(n^2)$



5. Quick Sort:

Cài đặt:

```
1 void quicksort(int a[], int left, int right){
2     int pivot = a[(left+right)/2], l=left, r=right;
3     while (l <= r){
4         while (a[l] < pivot ) l++;
5         while (pivot < a[r]) r--;
6         if (l <= r){
7             swap(a[r], a[l]);
8             l++;
9             r--;
10        }
11    }
12    if (left < r){
13        quicksort(a, left, r);
14    }
15    if (l < right){
16        quicksort(a, l, right);
17    }
18 }
```



6. Heap Sort:

Ý tưởng:

Chuyển dãy cần sắp xếp thành cấu trúc heap

- Đặc điểm: tại nút thứ i tương ứng với phần tử thứ i trong mảng sẽ có con trái là $2*i+1$ và con phải là $2*i+2$ (nếu $2*i+1$ và $2*i+2 < n$)

Lần lượt chuyển nút gốc của heap đưa về vị trí thích hợp (về phía cuối)

Độ phức tạp: $O(n*\log(n))$



6. Heap Sort:

Cài đặt:



```
1 void heapify(int a[], int n , int i){
2     int largest = i, left = 2*i +1 , right = 2*i +2;
3     if (left < n && a[largest] < a[left]) largest = left;
4     if (right < n && a[largest] < a[right]) largest = right;
5     if (largest != i){
6         swap (a[largest], a[i]);
7         heapify(a,n , largest);
8     }
9 }
10 void heapsort(int a[], int n){
11     for (int i = n/2 -1; i >=0 ; i--){
12         heapify(a,n, i);
13     }
14     for (int i = n-1; i >=0 ; i--){
15         swap(a[i], a[0]);
16         heapify(a,i, 0);
17     }
18 }
```





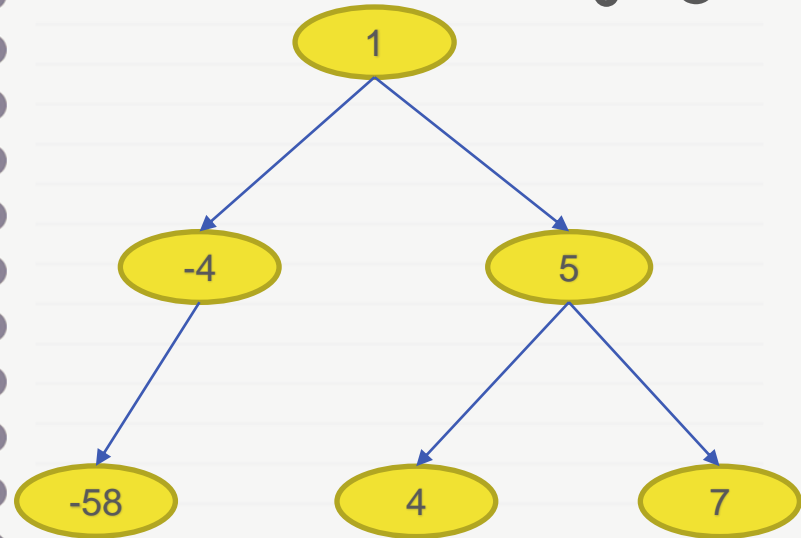
CÂY NHỊ PHÂN TÌM KIẾM

- **Định nghĩa**
- **Các thao tác**

Định nghĩa



- Cây nhị phân
- Bảo đảm nguyên tắc bố trí khoá tại mỗi nút:
 - Các nút trong cây trái nhỏ hơn nút hiện hành
 - Các nút trong cây phải lớn hơn nút hiện hành



Cấu trúc dữ liệu của 1 nút

```
typedef struct tagTNode {  
    int Key;  
    struct tagTNode *pLeft;  
    struct tagTNode *pRight;  
}TNode;
```



Cấu trúc dữ liệu của cây

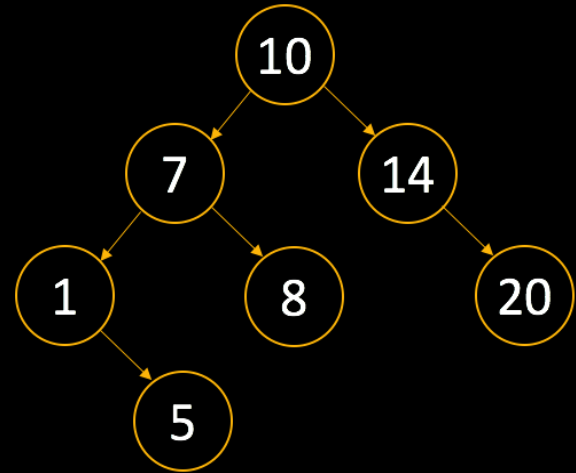
```
typedef TNode *TREE;
```

Các thao tác trên cây nhị phân tìm kiếm

- Tạo 1 cây rỗng
- Tạo 1 nút có Key bằng x
- Thêm 1 nút
- Xoá 1 nút có Key bằng x trên cây
- Tìm 1 nút có Key bằng x trên cây



[10, 7, 14, 20, 1, 5, 8]



Tạo cây rỗng

```
void CreateTree(TREE &T) {  
    T=NULL;  
}
```



Tạo 1 nút có Key bằng x

```
TNode *CreateTNode(int x) {  
    TNode *p;  
    p = new TNode;  
    if(p == NULL)  
        exit(1);  
    p->key = x;  
    p->pLeft = NULL;  
    p->pRight = NULL;  
    return p;  
}
```

Thêm một nút x

```
int insertNode(TREE &T, Data X) {  
    if(T)  
    {  
        if(T->Key == X) return 0;  
        if(T->Key > X)  
            return insertNode(T->pLeft, X);  
        else  
            return insertNode(T->pRight, X);  
    }  
    T = new TNode;  
    if(T == NULL)  
        return -1;  
    T->Key = X;  
    T->pLeft = T->pRight = NULL;  
    return 1;  
}
```

Ví dụ:

Thêm các nút có giá trị 21, 28, 14, 32, 25, 18, 11, 30, 19, 15 vào cây Binary Search Tree rỗng

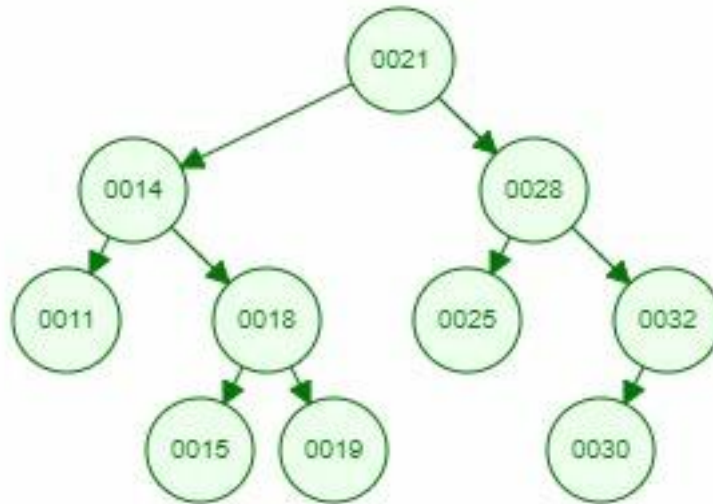
Tìm nút có khoá bằng x

```
TNode * searchNode(TREE Root, Data x) {  
    Node *p = Root;  
    while (p != NULL)  
    {  
        if(x == p->Key)  
            return p;  
        else if(x < p->Key)  
            p = p->pLeft;  
        else  
            p = p->pRight;  
    }  
    return NULL;  
}
```



```
TNode *SearchTNode(TREE T, int x) {  
    if(T!=NULL)  
    {  
        if(T->key==x)  
            return T;  
        else if(x>T->key)  
            return SearchTNode(T->pRight,x);  
        else  
            return SearchTNode(T->pLeft,x);  
    }  
    return NULL;  
}
```

Tìm nút có khoá bằng x



Hình họa tìm các giá trị: 18,15 và 30

Hủy 1 nút có khoá bằng X trên cây

- Hủy 1 phần tử trên cây **phải đảm bảo điều kiện ràng buộc của Cây nhị phân tìm kiếm**
- Có 3 trường hợp khi hủy 1 nút trên cây
 - TH1: X là nút lá
 - TH2: X chỉ có 1 cây con (cây con trái hoặc cây con phải)



- TH3: X có đầy đủ 2 cây con
 - TH1: Ta xoá nút lá mà không ảnh hưởng đến các nút khác trên cây
 - TH2: Trước khi xoá x ta móc nối cha của X với con duy nhất của X.
 - TH3: Ta dùng cách xoá gián tiếp

Hủy 1 nút có 2 cây con

- Ta dùng cách hủy gián tiếp, do X có 2 cây con
- Thay vì hủy X ta tìm **phần tử thế mạng Y**.
- Thông tin lưu tại nút Y sẽ được chuyển lên lưu tại X.
- Ta tiến hành xoá hủy nút Y



- Cách tìm nút thế mạng Y cho X:
Có 2 cách

- C1: Nút Y là nút có khoá nhỏ nhất (trái nhất) bên cây con phải X
- C2: Nút Y là nút có khoá lớn nhất (phải nhất) bên cây con trái của X

Hủy 1 nút có khoá bằng X trên cây

```
void DeleteNodeX1(TREE &T,int x){  
    if(T!=NULL){  
        if(T->Key<x)  
            DeleteNodeX1(T->KeyRight,x);  
        else{  
  
            if(T->Key>x)  
                DeleteNodeX1(T->Left,x);  
  
            else{  
  
                TNode *p = T;  
  
                if (T->Left==NULL)  
  
                    T = T->Right;
```



```
        else{  
  
            if(T->Right==NULL)  
  
                T=T->Left;  
  
            else  
  
                ThayThe1(p, T->Right);  
  
        }  
  
        delete p;  
  
    } } }  
  
    else printf("Khong tim thay phan can xoa  
tu");}
```

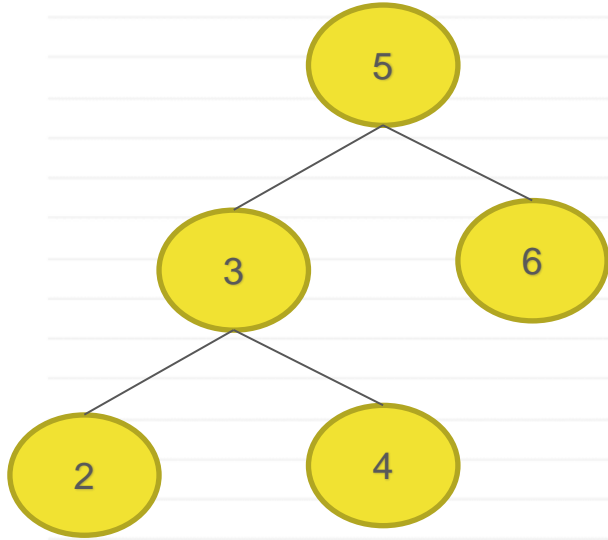
Hủy 1 nút có khoá bằng X trên cây

```
void ThayThe1(TREE &p, TREE &T) {  
    if(T->Left!=NULL)  
        ThayThe1(p,T->Left);  
    else {  
        p->Key = T->Key;  
        p=T;  
        T=T->Right;  
    }  
}
```



```
void ThayThe2(TREE &p, TREE &T) {  
    if(T->Right!=NULL)  
        ThayThe2(p,T->Right);  
    else {  
        p->Key = T->Key;  
        p=T;  
        T=T->Left;  
    }  
}
```

Các kiểu duyệt cây



- Depth First Traversals:

- Inorder (Left, Root, Right) : 2 3 4 5 6
- Preorder (Root, Left, Right) : 5 3 2 4 6
- Postorder (Left, Right, Root) : 2 4 3 6 5

- Breadth-First or Level Order Traversal: 5 3 6 2 4



Các kiểu duyệt cây

Algorithm Inorder(tree)

1. Traverse the left subtree, i.e., call
Inorder(left-subtree)

2. Visit the root.

3. Traverse the right subtree, i.e.,
call Inorder(right-subtree)

```
void printInorder(struct Node* node)
{
    if (node == NULL)
        return;

    /* first recur on left child */
    printInorder(node->left);

    /* then print the data of node */
    cout << node->data << " ";

    /* now recur on right child */
    printInorder(node->right);
}
```

Các kiểu duyệt cây

Algorithm Preorder(tree)

1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder(left-subtree)
3. Traverse the right subtree, i.e., call Preorder(right-subtree)

```
void printPreorder(struct Node* node)
{
    if (node == NULL)
        return;

    /* first print data of node */
    cout << node->data << " ";

    /* then recur on left subtree */
    printPreorder(node->left);

    /* now recur on right subtree */
    printPreorder(node->right);
}
```

Các kiểu duyệt cây

Algorithm Postorder(tree)

1. Traverse the left subtree, i.e., call Postorder(left-subtree)

2. Traverse the right subtree, i.e., call Postorder(right-subtree)

3. Visit the root.



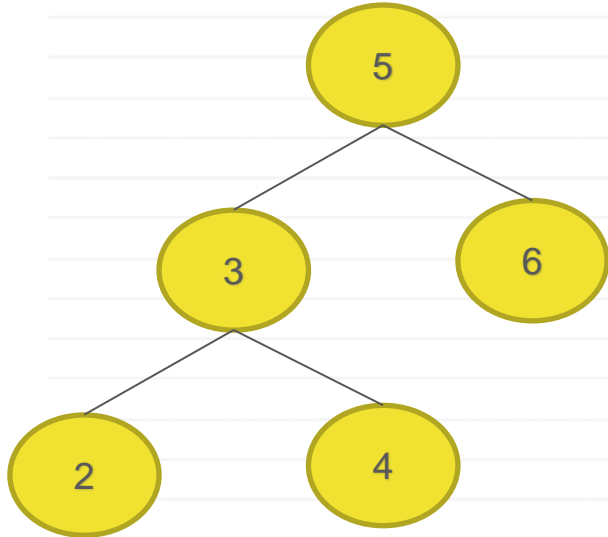
```
void printPostorder(struct Node* node)
{
    if (node == NULL)
        return;

    // first recur on left subtree
    printPostorder(node->left);

    // then recur on right subtree
    printPostorder(node->right);

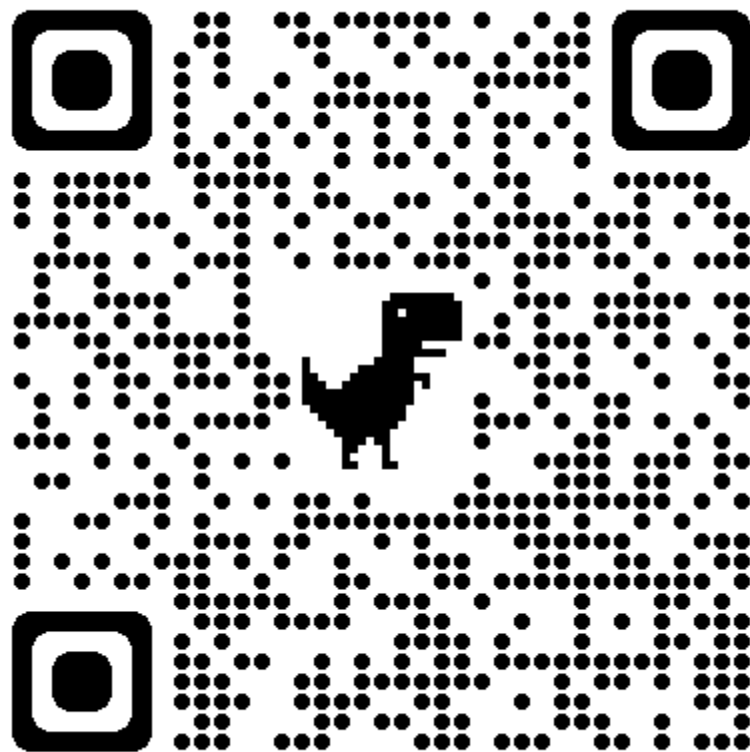
    // now deal with the node
    cout << node->data << " ";
}
```


Một số yêu cầu khác



Viết hàm đếm số lượng nút lá có trên cây

```
int CountLeaf(const Tree& T){  
    if(T == NULL)  
        return 0;  
    if(T->Left == NULL && T->Right == NULL)  
        return 1;  
    else  
        return CountLeaf(T->Left)+  
               CountLeaf(T->Right);  
}
```



B-Tree

- Định nghĩa
- Chèn
- Xóa

Định nghĩa

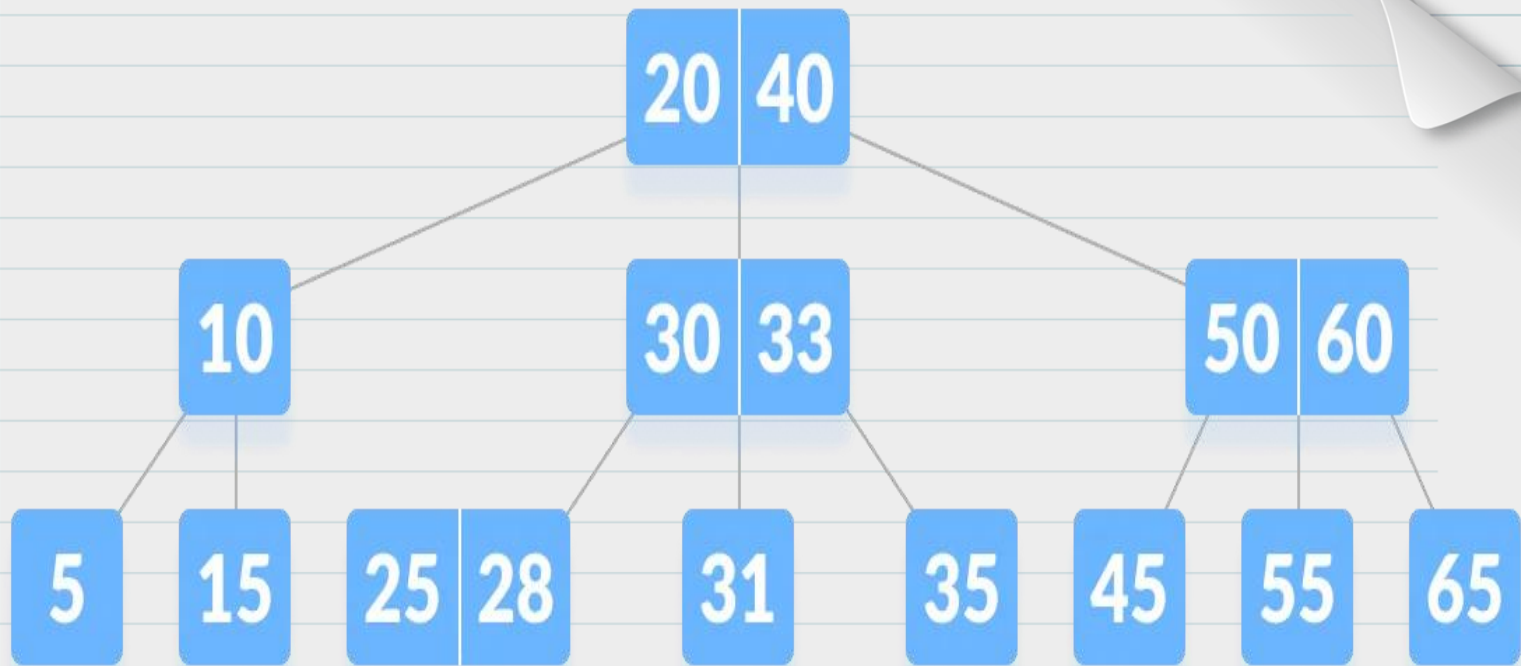


Cho số tự nhiên $k > 0$, B-Tree bậc m (với $m \geq k$) là một cây nhiều nhánh tìm kiếm thỏa mãn các tính chất

1. Tất cả các node lá ở cùng một mức.
2. Tất cả các node trung gian trừ node gốc có tối đa m cây con ($m-1$ khóa) và tối thiểu $m/2 + 1$ cây con khác rỗng ($m/2$ khóa).



- 3. Mỗi node hoặc là node lá hoặc node có k khóa thì có $k+1$ cây con và phân chia các khóa trong các nhánh con theo cách của cây tìm kiếm.
- 4. Node gốc có nhiều nhất m cây con hoặc ít nhất có 2 cây con khi node gốc không là node lá hoặc không có cây con nào khi cây chỉ có 1 node gốc.

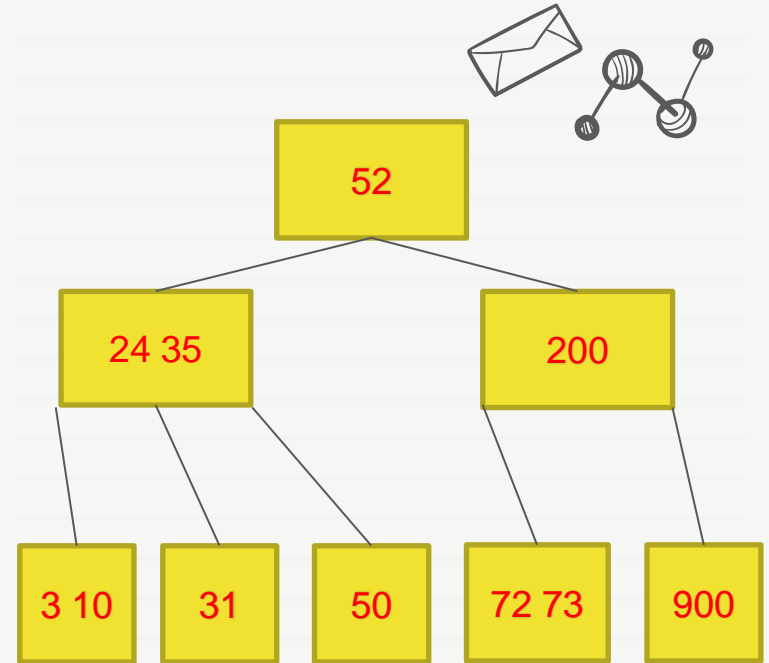


Ví dụ



B-tree of order 3:

- $m=3$
- Max number of child = $m = 3$
- Min number of child $(m \div 2) = 2$
- Max number of key = $m-1 = 2$
- Min number of key = $(m \div 2) - 1 = 1$



Insert



Thêm dãy số: 27, 19, 23, 9, 1, 3, 11, 21, 5, 13, và 17 vào cây B-Tree **bậc 3** theo thứ tự từ trái sang phải



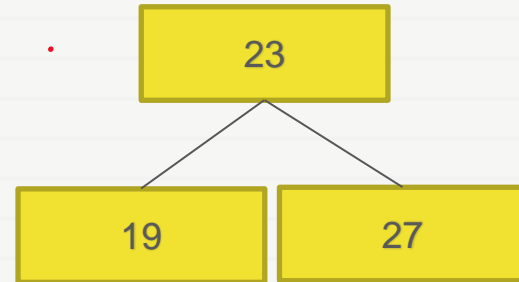
Thêm 19,27:

19 27

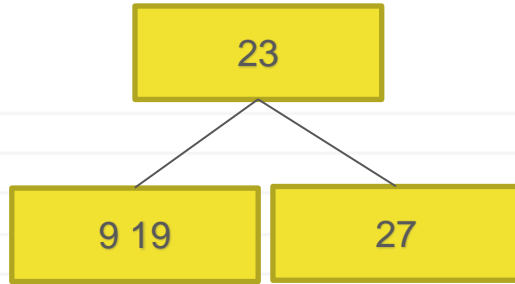


Thêm 23:

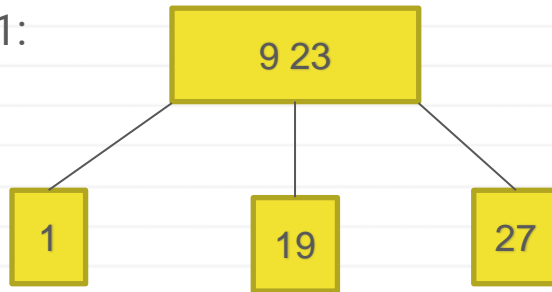
19 23 27



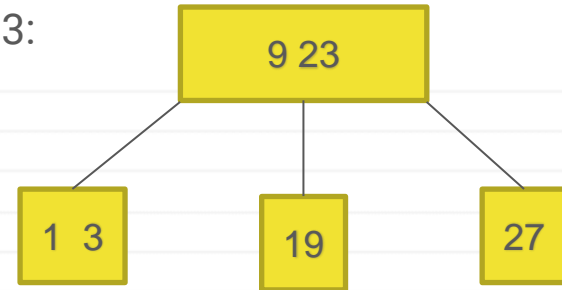
Thêm 9:



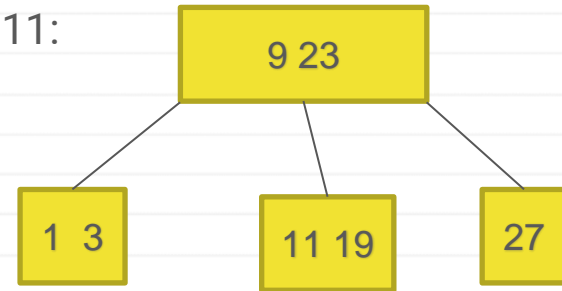
Thêm 1:



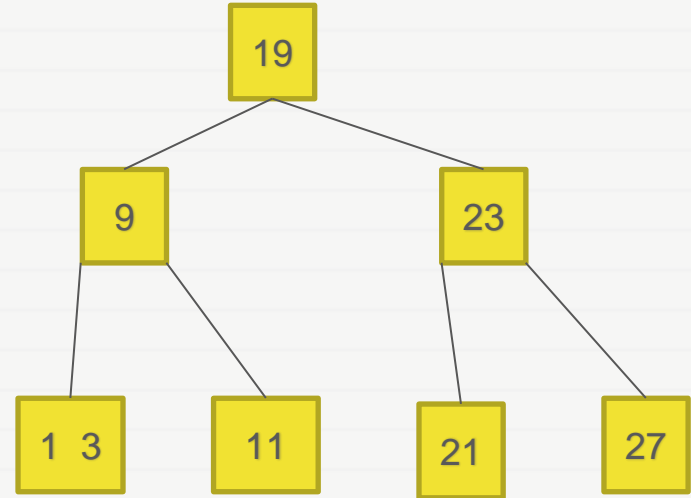
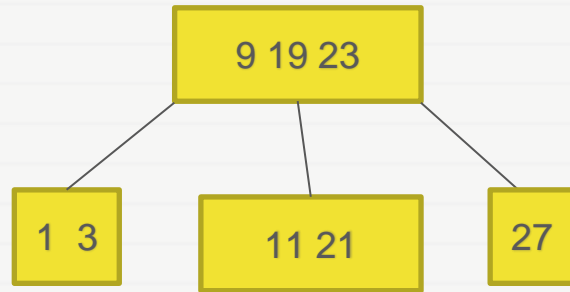
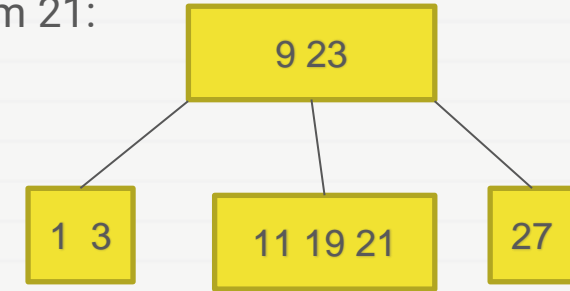
Thêm 3:



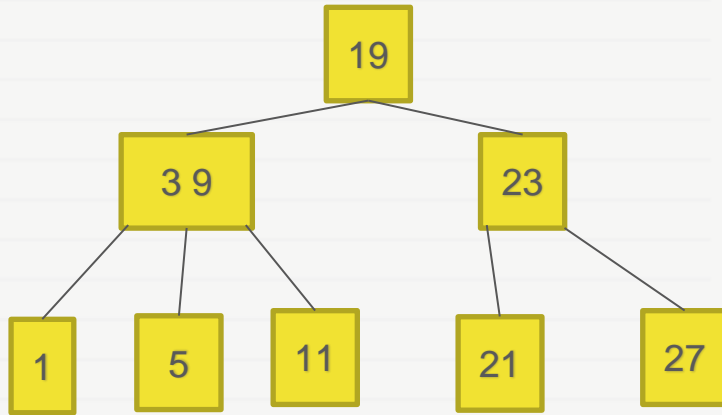
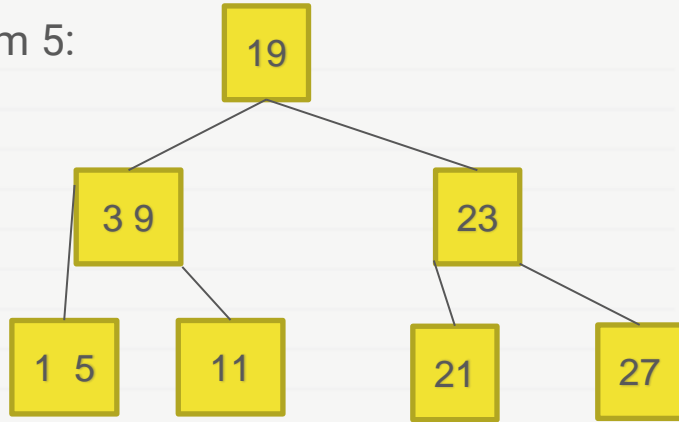
Thêm 11:



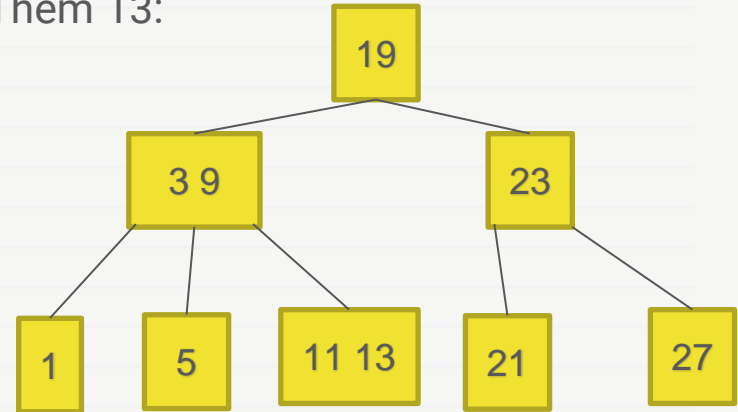
Thêm 21:



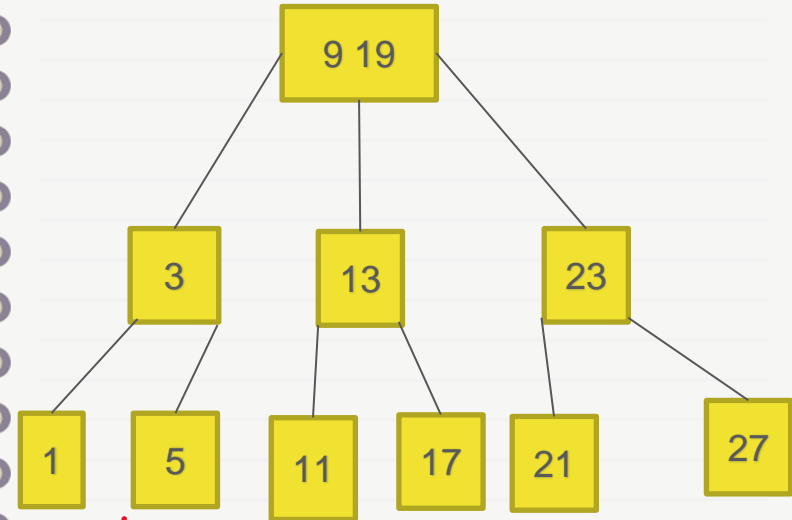
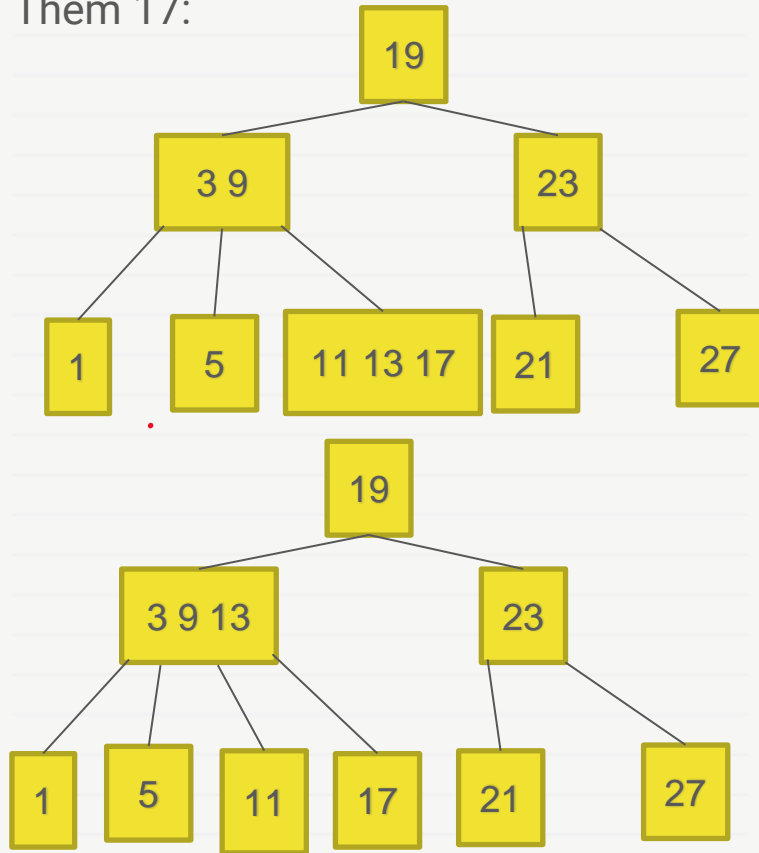
Thêm 5:



Thêm 13:



Thêm 17:

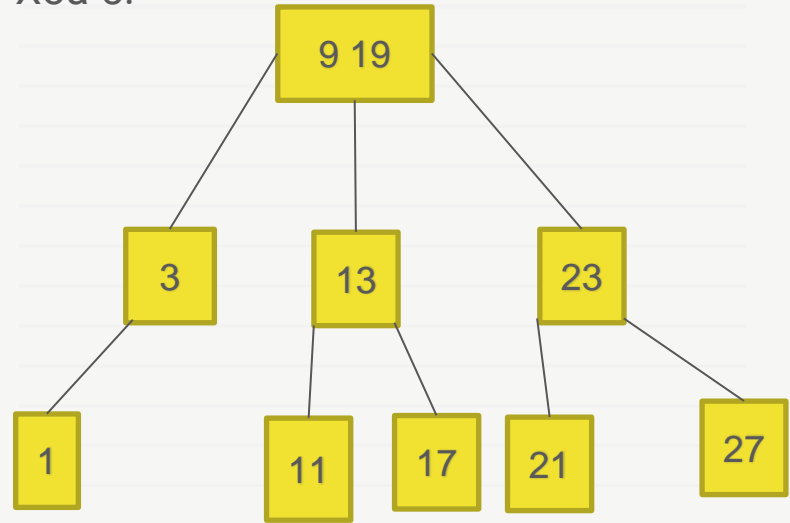


Delete

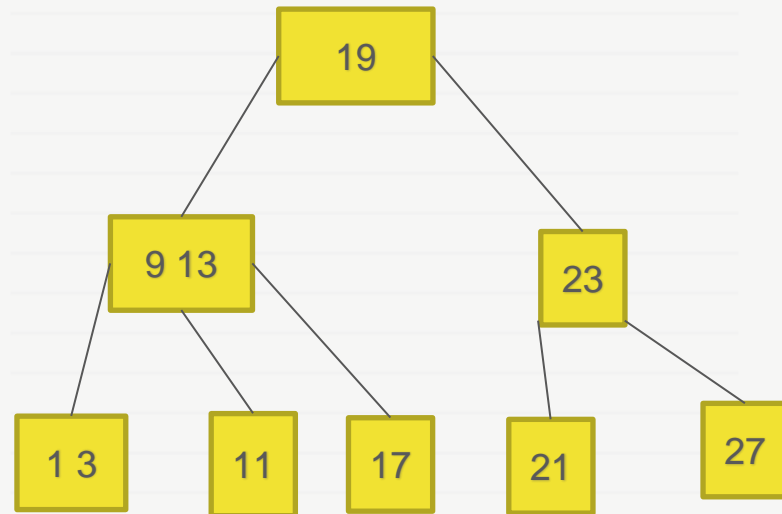
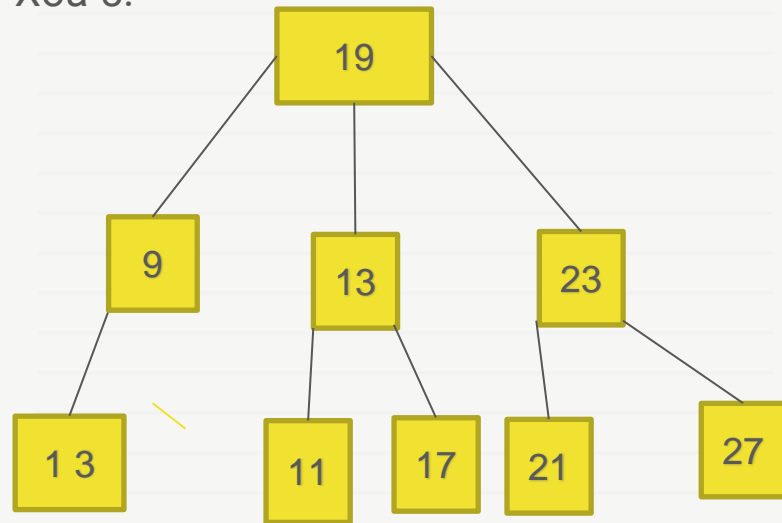


Xóa các giá trị 5, 13 và 19 cây B-Tree
bậc 3 vừa tạo theo thứ tự từ trái sang
phải

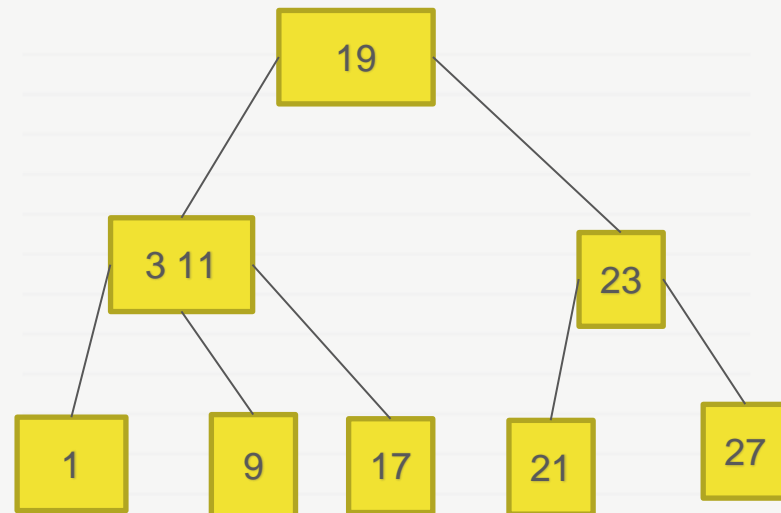
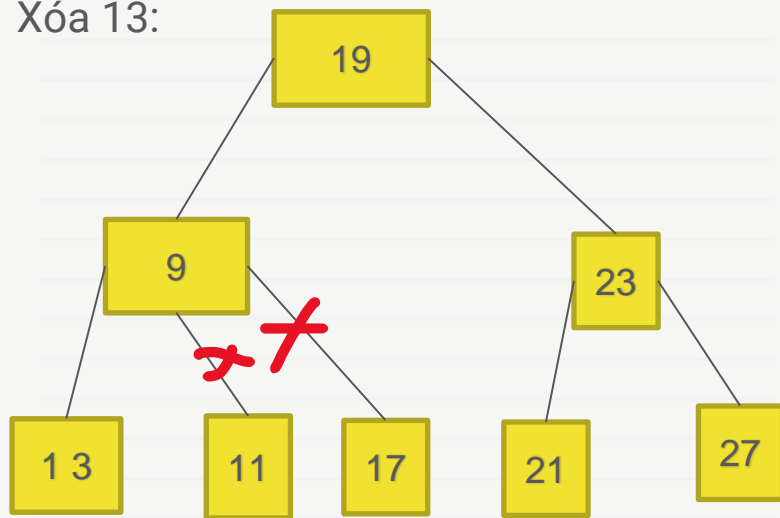
Xóa 5:



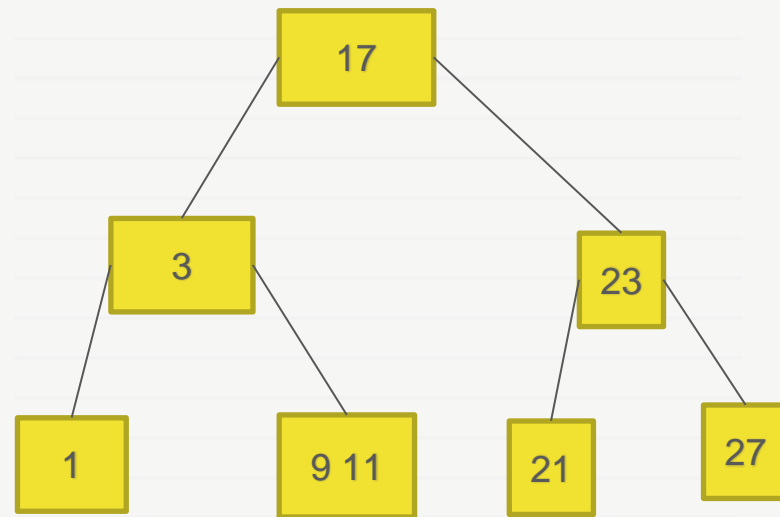
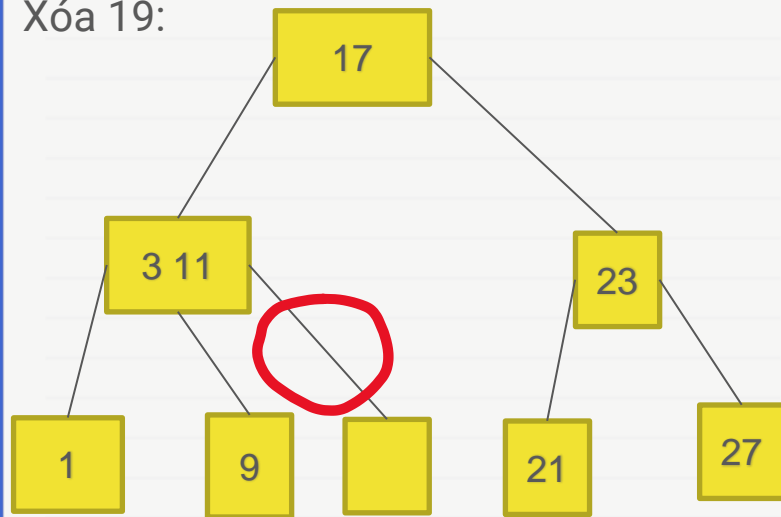
Xóa 5:



Xóa 13:



Xóa 19:



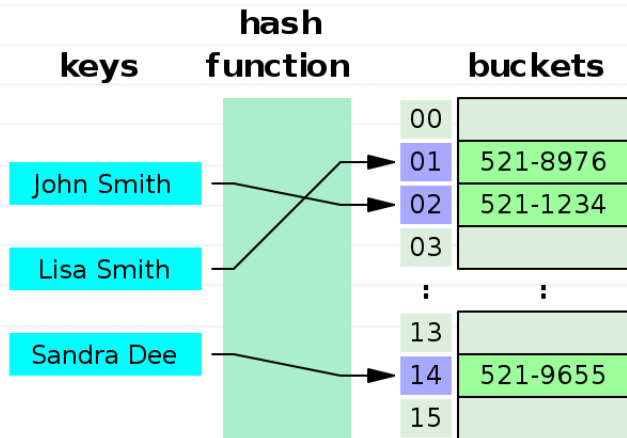


BẢNG BĂM (HASH TABLE)

Định nghĩa

Bảng băm

Bảng băm là một cấu trúc dữ liệu sử dụng hàm băm để ánh xạ các khóa (key) đến một dãy các số nguyên và dùng các số này để truy xuất.



Hàm băm

Các khóa có thể là dạng số hoặc chuỗi

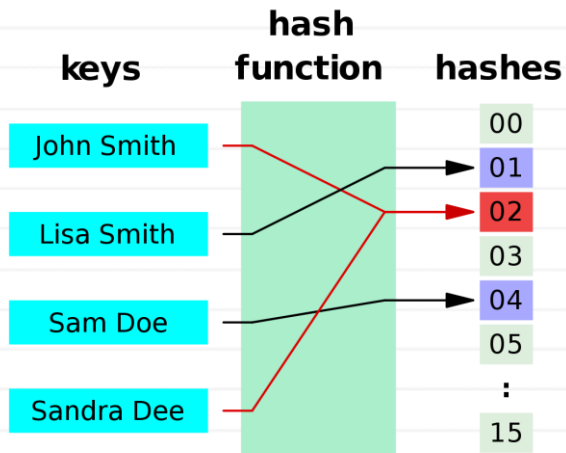
Hàm băm dùng để đổi các khóa thành chỉ số trong mảng.

Một hàm băm tốt sẽ thỏa mãn các yêu cầu sau:

- Dễ tính toán và bản thân nó không phải là thuật toán
- Phân bố đồng đều
- Ít đụng độ

Sự đụng độ

Đụng độ là hiện tượng các khóa khác nhau nhưng khi băm lại có cùng địa chỉ



Xử lí đụng độ



Phương pháp nối kết

- Nối kết trực tiếp
- Nối kết hợp nhất

Phương pháp băm lại

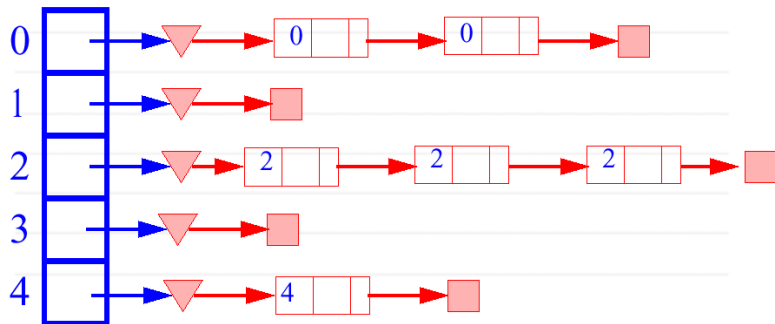
- Dò tuyến tính
- Dò bậc 2
- Băm kép

Phương pháp nối kết

Nối kết trực tiếp

Các phần tử được băm thành danh sách liên kết.

Các phần tử được băm cùng địa chỉ sẽ gom lại thành một danh sách liên kết.



Nối kết hợp nhất

Bảng băm sẽ sử dụng mảng danh sách liên kết gồm 2 trường **key** và **next** chứa **M** phần tử

key : chứa phần tử

Next : là con trỏ chỉ đến node tiếp theo nếu xảy ra đụng độ.

Khi khởi động, tất cả **key** sẽ là **NULL** và **next** là -1.

Khi xảy ra đụng độ, mình sẽ đi ngược từ cuối bảng lên và tìm vị trí còn trống để thêm phần tử. Và cập nhật trường **next** sao cho các thành phần bị xung đột tạo thành một danh sách liên kết

Nối kết hợp nhất



Cho dãy số : 8, 2, 5, 1, 16, 15

$M = 7$

$f(\text{key}) = \text{key} \% 7$

Đầu tiên, cho tất cả **key = NULL** và **next = -1**

Thêm **8** vào bảng:

$f(8) = 8 \% 7 = 1$

	KEY	NEXT
0	NULL	-1
1	NULL	-1
2	NULL	-1
3	NULL	-1
4	NULL	-1
5	NULL	-1
6	NULL	-1

$f(\text{key}) = \text{key} \% 7$

Nối kết hợp nhất



Cho dãy số : 8, 2, 5, 1, 16, 15

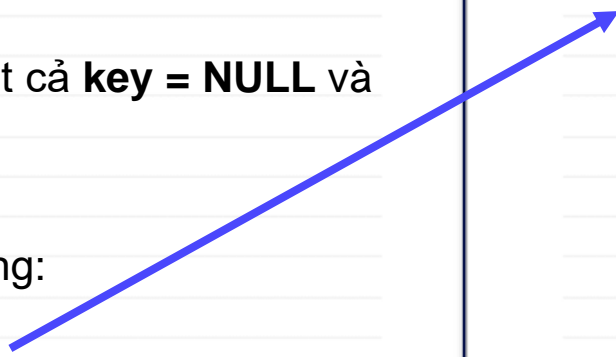
$M = 7$

$f(\text{key}) = \text{key} \% 7$

Đầu tiên, cho tất cả **key = NULL** và **next = -1**

Thêm 8 vào bảng:

$f(8) = 8 \% 7 = 1$



	KEY	NEXT
0	NULL	-1
1	8	-1
2	NULL	-1
3	NULL	-1
4	NULL	-1
5	NULL	-1
6	NULL	-1

$f(\text{key}) = \text{key} \% 7$

Nối kết hợp nhất



Cho dãy số : **8, 2, 5, 1, 16, 15**

$M = 7$

$f(\text{key}) = \text{key} \% 7$

Thêm **2** vào bảng:

$f(2) = 2 \% 7 = 2$

	KEY	NEXT
0	NULL	-1
1	8	-1
2	NULL	-1
3	NULL	-1
4	NULL	-1
5	NULL	-1
6	NULL	-1

$f(\text{key}) = \text{key} \% 7$

Nối kết hợp nhất



Cho dãy số : **8, 2, 5, 1, 16, 15**

$M = 7$

$f(\text{key}) = \text{key} \% 7$

Thêm **2** vào bảng:

$f(2) = 2 \% 7 = 2$



	KEY	NEXT
0	NULL	-1
1	8	-1
2	2	-1
3	NULL	-1
4	NULL	-1
5	NULL	-1
6	NULL	-1

$f(\text{key}) = \text{key} \% 7$

Nối kết hợp nhất



Cho dãy số : **8, 2, 5, 1, 16, 15**

$M = 7$

$f(\text{key}) = \text{key} \% 7$

Thêm **5** vào bảng:

$f(5) = 5 \% 7 = 5$

	KEY	NEXT
0	NULL	-1
1	8	-1
2	2	-1
3	NULL	-1
4	NULL	-1
5	NULL	-1
6	NULL	-1

$f(\text{key}) = \text{key} \% 7$

Nối kết hợp nhứt



Cho dãy số : 8, 2, 5, 1, 16, 15

$M = 7$

$f(\text{key}) = \text{key} \% 7$

Thêm 5 vào bảng:

$f(5) = 5 \% 7 = 5$



	KEY	NEXT
0	NULL	-1
1	8	-1
2	2	-1
3	NULL	-1
4	NULL	-1
5	5	-1
6	NULL	-1

$f(\text{key}) = \text{key} \% 7$

Nối kết hợp nhất



Cho dãy số : 8, 2, 5, 1, 16, 15

$M = 7$


$f(\text{key}) = \text{key} \% 7$

Thêm 1 vào bảng:

$f(1) = 1 \% 7 = 1$

=> Đụng độ

Duyệt từ cuối bảng, tìm vị trí trống



	KEY	NEXT
0	NULL	-1
1	8	-1
2	2	-1
3	NULL	-1
4	NULL	-1
5	5	-1
6	NULL	-1

$f(\text{key}) = \text{key} \% 7$

Nối kết hợp nhất



Cho dãy số : 8, 2, 5, 1, 16, 15

$M = 7$

$f(\text{key}) = \text{key} \% 7$

Thêm 1 vào bảng:

$f(1) = 1 \% 7 = 1$

=> Đụng độ

Duyệt từ cuối bảng, tìm vị trí trống

Cập nhật **next** tại vị trí 1 với vị trí mới của **key = 1**

	KEY	NEXT
0	NULL	-1
1	8	-1
2	2	-1
3	NULL	-1
4	NULL	-1
5	5	-1
6	1	-1

$f(\text{key}) = \text{key} \% 7$

Nối kết hợp nhất



Cho dãy số : 8, 2, 5, 1, 16, 15

$M = 7$

$f(\text{key}) = \text{key} \% 7$

Thêm 1 vào bảng:

$f(1) = 1 \% 7 = 1$

=> Đụng độ

Duyệt từ cuối bảng, tìm vị trí trống

Cập nhật **next** tại vị trí 1 với vị trí mới của **key = 1**

	KEY	NEXT
0	NULL	-1
1	8	6
2	2	-1
3	NULL	-1
4	NULL	-1
5	5	-1
6	1	-1

$f(\text{key}) = \text{key} \% 7$

Nối kết hợp nhất



Cho dãy số : **8, 2, 5, 1, 16, 15**

$M = 7$

$f(\text{key}) = \text{key} \% 7$

Thêm **16** vào bảng:

$f(16) = 16 \% 7 = 2$

=> Đụng độ

Duyệt từ cuối bảng, tìm vị trí trống

	KEY	NEXT
0	NULL	-1
1	8	6
2	2	-1
3	NULL	-1
4	NULL	-1
5	5	-1
6	1	-1

$f(\text{key}) = \text{key} \% 7$

Nối kết hợp nhất



Cho dãy số : **8, 2, 5, 1, 16, 15**

$M = 7$

$f(\text{key}) = \text{key} \% 7$

Thêm **16** vào bảng:

$f(16) = 16 \% 7 = 2$

=> Đụng độ

Duyệt từ cuối bảng, tìm vị trí trống

	KEY	NEXT
0	NULL	-1
1	8	6
2	2	-1
3	NULL	-1
4	16	-1
5	5	-1
6	1	-1

$f(\text{key}) = \text{key} \% 7$

Nối kết hợp nhất



Cho dãy số : **8, 2, 5, 1, 16, 15**

$M = 7$

$f(\text{key}) = \text{key} \% 7$

Thêm **16** vào bảng:

$f(16) = 16 \% 7 = 2$

=> Đụng độ

Duyệt từ cuối bảng, tìm vị trí trống

Cập nhật **next** tại vị trí **2** với vị trí mới của **key = 16**

	KEY	NEXT
0	NULL	-1
1	8	6
2	2	4
3	NULL	-1
4	16	-1
5	5	-1
6	1	-1

$f(\text{key}) = \text{key} \% 7$

Nối kết hợp nhất



Cho dãy số : **8, 2, 5, 1, 16, 15**

$M = 7$


$f(\text{key}) = \text{key} \% 7$

Thêm **15** vào bảng:

$f(15) = 15 \% 7 = 1$

=> Đụng độ

Duyệt từ cuối bảng, tìm vị trí trống



	KEY	NEXT
0	NULL	-1
1	8	6
2	2	4
3	NULL	-1
4	16	-1
5	5	-1
6	1	-1

$f(\text{key}) = \text{key} \% 7$

Nối kết hợp nhất

Cho dãy số : 8, 2, 5, 1, 16, 15

$M = 7$

$f(\text{key}) = \text{key} \% 7$

Thêm **15** vào bảng:

$f(15) = 15 \% 7 = 1$

=> Đụng độ

Duyệt từ cuối bảng, tìm vị trí trống

Cập nhật **next** tại vị trí **6** với vị trí mới của **key = 15**



	KEY	NEXT
0	NULL	-1
1	8	6
2	2	4
3	15	-1
4	16	-1
5	5	-1
6	1	3

$f(\text{key}) = \text{key} \% 7$

Phương pháp băm lại

Dò tuyến tính

Nếu không đựng độ thì thêm vào vị trí của hàm băm, nếu không xét địa chỉ kế tiếp.

$$H(\text{key}) = (H(\text{key}) + i) \% M$$

Dò bậc hai

Vì dò tuyến tính sẽ gây ra việc rải các phần tử không đều. Dò bậc hai giúp cho việc rải các phần tử đều hơn.

$$H(\text{key}) = (H(\text{key}) + i^2) \% M$$

Băm kép

Dùng hai hàm băm khác nhau để rải đều các phần tử.

Ta dùng hai hàm băm bất kì. VD:

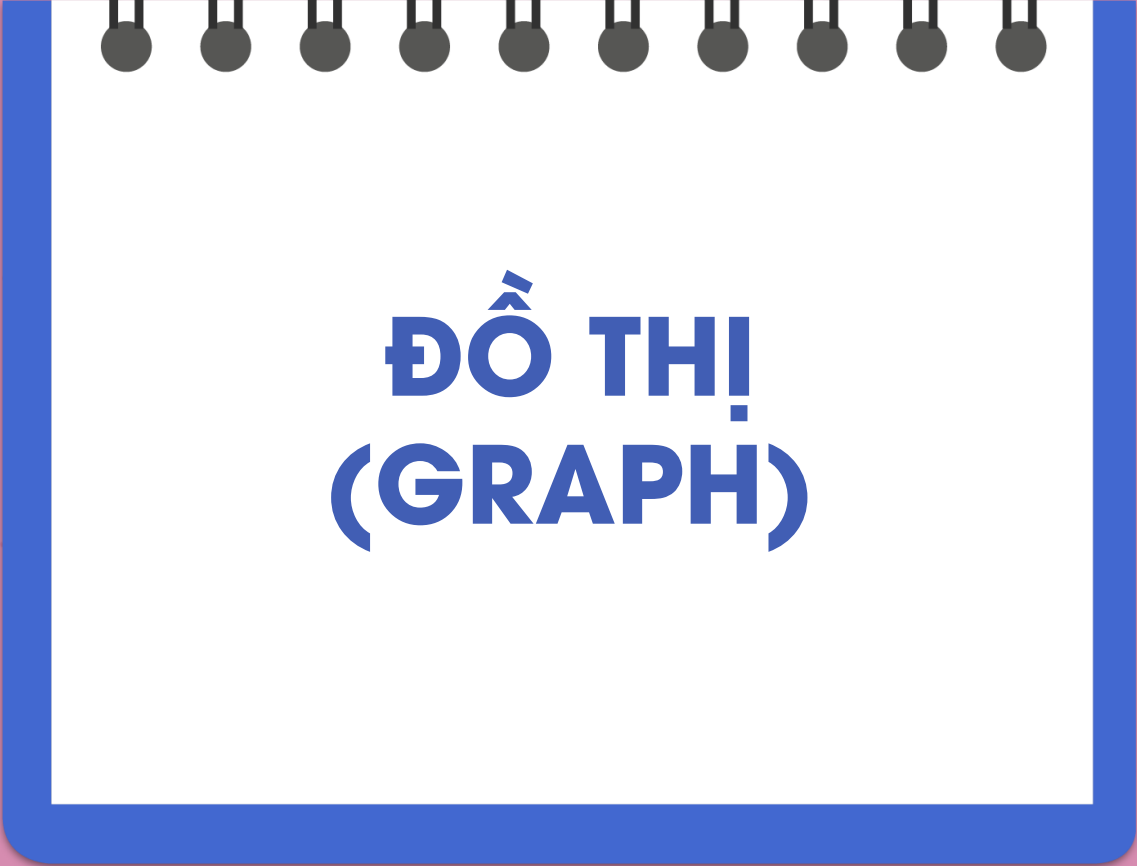
$$H1(\text{key}) = (H(\text{key}) + i) \% M$$

$$H2(\text{key}) = (M - 2) - \text{key} \% (M - 2)$$

$$H(\text{key}) = (H1(\text{key}) + i * H2(\text{key})) \% M$$

M : kích thước của bảng

i : số lần đựng độ

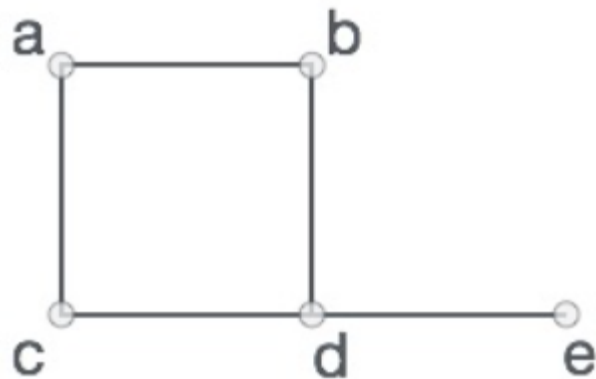


ĐỒ THỊ (GRAPH)

Định nghĩa

Một số khái niệm về đồ thị

- **Một đồ thị** là một dạng biểu diễn hình ảnh của một **tập các đối tượng**, trong đó các **cặp đối tượng** được kết nối bởi các **liên kết**.
- **Các đối tượng** được nối liền nhau được biểu diễn bởi các điểm được gọi là các **đỉnh (vertices)**, và các **liên kết** mà kết nối các **đỉnh** với nhau được gọi là các **cạnh (edges)**.



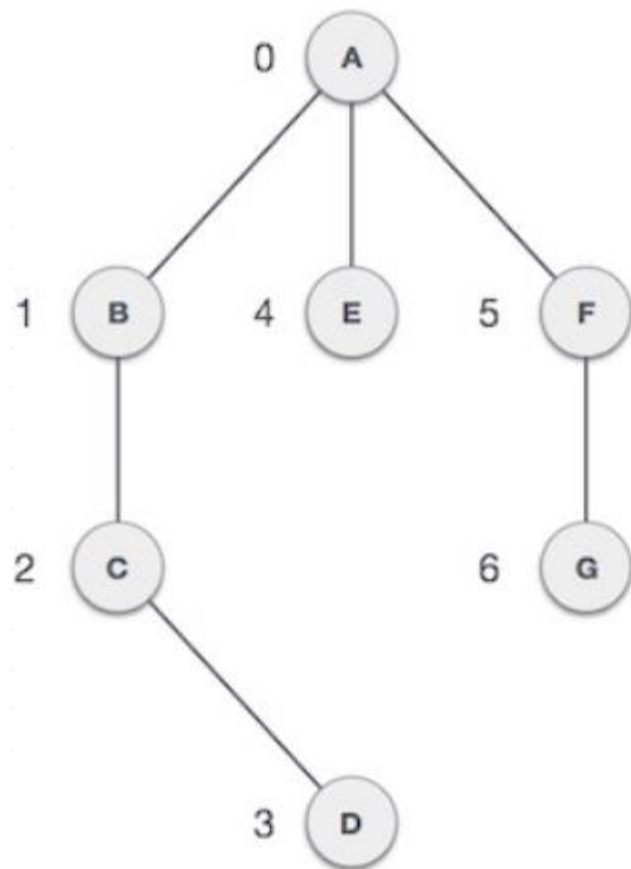
Trong đồ thị trên:

$$V = \{a, b, c, d, e\}$$

$$E = \{ab, ac, bd, cd, de\}$$

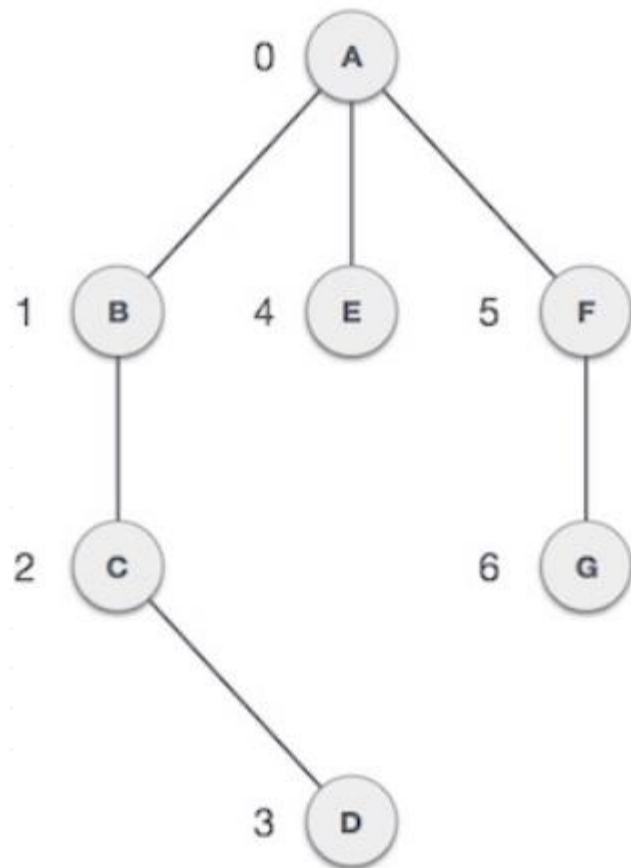
Một số khái niệm về đồ thị

- **Đỉnh (Vertex):** Mỗi nút của hình được biểu diễn như là một đỉnh. **Trong ví dụ bên: các hình tròn** biểu diễn các **đỉnh**. Do đó, các điểm từ A tới G là các **đỉnh**. Chúng ta có thể biểu diễn các **đỉnh** này bởi sử dụng một *mảng*, trong đó đỉnh A có thể được nhận diện bởi chỉ mục 0, điểm B là chỉ mục 1, ... như hình bên.
- **Cạnh (Edge):** Cạnh biểu diễn một đường nối hai đỉnh. **Trong hình bên: các đường nối** A và B, B và C, ... là các **cạnh**. Chúng ta có thể sử dụng một *mảng hai chiều* để biểu diễn các **cạnh** này. Trong ví dụ bên, AB có thể được biểu diễn như là 1 tại hàng 0; BC là 1 tại hàng 1, cột 2, ...



Một số khái niệm về đồ thị

- **Kề nhau:** Hai đỉnh là kề nhau nếu chúng được **kết nối** với nhau thông qua một **cạnh**. Trong hình bên, B là **kề** với A; C là **kề** với B, ...
- **Đường:** Đường biểu diễn một dãy các **cạnh** giữa **hai đỉnh**. Trong hình bên, ABCD biểu diễn một **đường** từ A tới D.



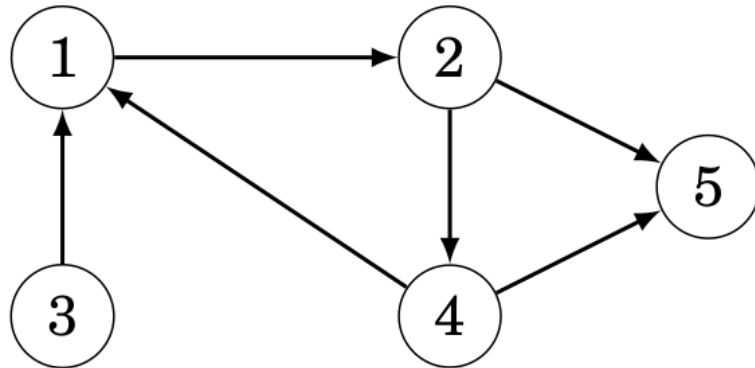
Định nghĩa

Có hướng

Đồ thị có hướng G là một cặp có thứ tự $G := (V, A)$, trong đó

- V , tập các **đỉnh** hoặc **nút**,
- A , tập các cặp có thứ tự chứa các đỉnh, được gọi là các **cạnh có hướng** hoặc **cung**. Một cạnh $e = (x, y)$ được coi là có hướng **từ** x **tới** y ; x được gọi là **điểm đầu/gốc** và y được gọi là **điểm cuối/ngọn** của cạnh.

Ví dụ:



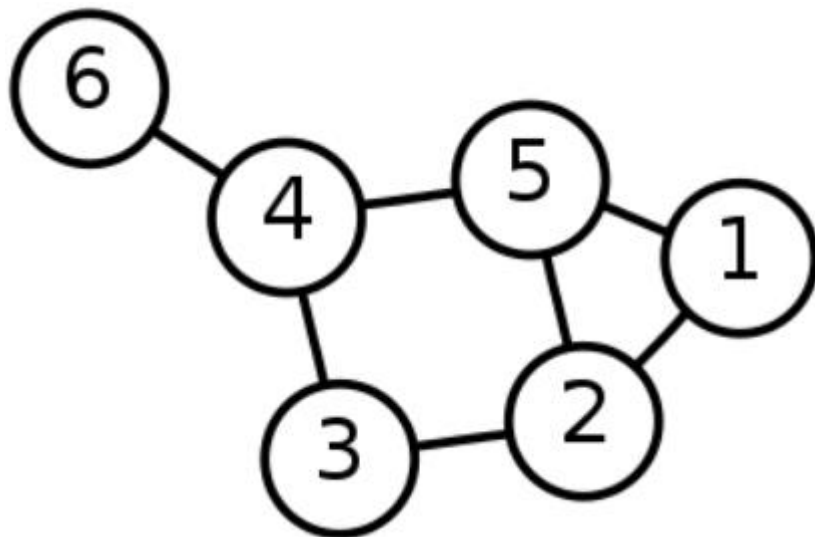
Định nghĩa

Vô hướng

Đồ thị vô hướng hoặc **đồ thị** G là một cặp không có thứ tự (*unordered pair*) $G:=(V, E)$, trong đó

- V , tập các **đỉnh** hoặc **nút**,
- E , tập các cặp không thứ tự chứa các đỉnh phân biệt, được gọi là **cạnh**. Hai đỉnh thuộc một cạnh được gọi là các **đỉnh đầu cuối** của cạnh đó

Ví dụ:

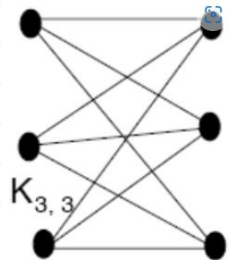
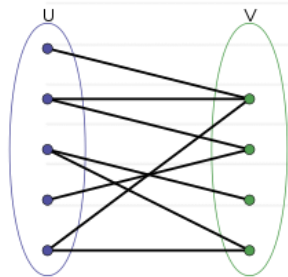
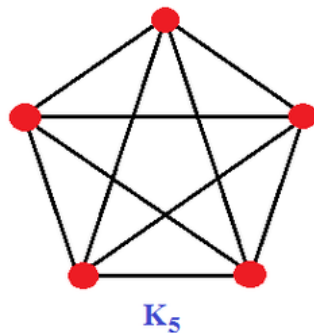


Một đồ thị vô hướng với 6 đỉnh (nút) và 7 cạnh.

Các Dạng Đồ Thị

- Đồ thị **ĐỦ**: đồ thị vô hướng, đơn, giữa hai đỉnh bất kỳ đều có đúng một cạnh:
 - ❖ Đồ thị đủ N đỉnh ký hiệu là K_N .
 - ❖ K_N có $N(N - 1)/2$ cạnh.
- Đồ thị **LƯỠNG PHÂN**: đồ thị $G = (X, E)$ được gọi là đồ thị lưỡng phân nếu tập X được chia thành hai tập X_1 và X_2 thỏa:
 - ❖ X_1 và X_2 phân hoạch;
 - ❖ Cạnh chỉ nối giữa X_1 và X_2 .
- Đồ thị **LƯỠNG PHÂN ĐỦ**: là đồ thị lưỡng phân đơn, vô hướng thỏa với $\forall i, j : i \in X_1$ và $j \in X_2$ có đúng một cạnh i và j .

Ví dụ:



Bậc của đỉnh trong đồ thị

Vô hướng

Xét đồ thị vô hướng G :

- Bậc của đỉnh x trong đồ thị G là số các cạnh kề với đỉnh x , mỗi khuyên được tính hai lần
- Ký hiệu: $d_G(x)$ (hay $d(x)$) nếu đang xét một đồ thị nào đó.

Có hướng

Xét đồ thị có hướng G :

- Nửa bậc ngoài của đỉnh x là số các cạnh đi ra khỏi đỉnh x , ký hiệu $d^+(x)$.
- Nửa bậc trong của đỉnh x là số các cạnh đi vào đỉnh x , ký hiệu $d^-(x)$.
- Bậc của đỉnh x : $d_x = d^+(x) + d^-(x)$

Dây chuyền, chu trình

- Một dây chuyền trong $G=(X, U)$ là một đồ thị con $C=(V, E)$ của G với:
 - ❖ $V = \{x_1, x_2, \dots, x_M\}$
 - ❖ $E = \{u_1, u_2, \dots, u_{M-1}\}$ với $u_1=x_1x_2$, $u_2=x_2x_3, \dots, u_{M-1}=x_{M-1}x_M$; liên kết x_i và x_{i+1} không phân biệt thứ tự.
- Khi đó, x_1 và x_M được nối với nhau bằng dây chuyền C . x_1 là đỉnh đầu và x_M là đỉnh cuối của C .
- Số cạnh của C được gọi là độ dài của C .
- Khi các cạnh hoàn toàn xác định bởi cặp đỉnh kề, dây chuyền có thể viết gọn (x_1, x_2, \dots, x_M)

Thành phần liên thông

- Một thành phần liên thông của đồ thị là một lớp tương đương được xác định bởi quan hệ LIÊN KẾT \sim ;
- Số thành phần liên thông của đồ thị là số lượng các lớp tương đương;
- Đồ thị liên thông là đồ thị chỉ có một thành phần liên thông.
- Khi một đồ G gồm p thành phần liên thông G_1, G_2, \dots, G_p thì các đồ thị G_i cũng là các đồ thị con của G và $d_G(x) = d_{G_i}(x), \forall x$ của G_i .

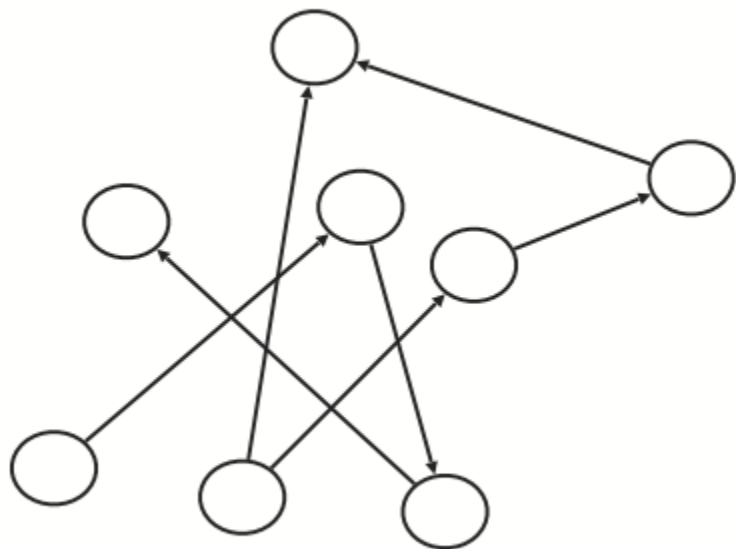
BIỂU DIỄN ĐỒ THỊ BẰNG MA TRẬN

Cách 1

- Xét đồ thị $G = (X, U)$, giả sử tập X gồm N đỉnh và được sắp thứ tự $X = \{x_1, x_2, \dots, x_N\}$, tập U gồm M cạnh và được sắp thứ tự $U = \{u_1, u_2, \dots, u_M\}$.
- Ma trận kề của đồ thị G , ký hiệu $B(G)$, là một ma trận nhị phân cấp $N \times N$: $B = (B_{ij})$ với B_{ij} được định nghĩa:
 - ❖ $B_{ij} = 1$ nếu có cạnh nối x_i tới x_j ,
 - ❖ $B_{ij} = 0$ trong trường hợp ngược lại.

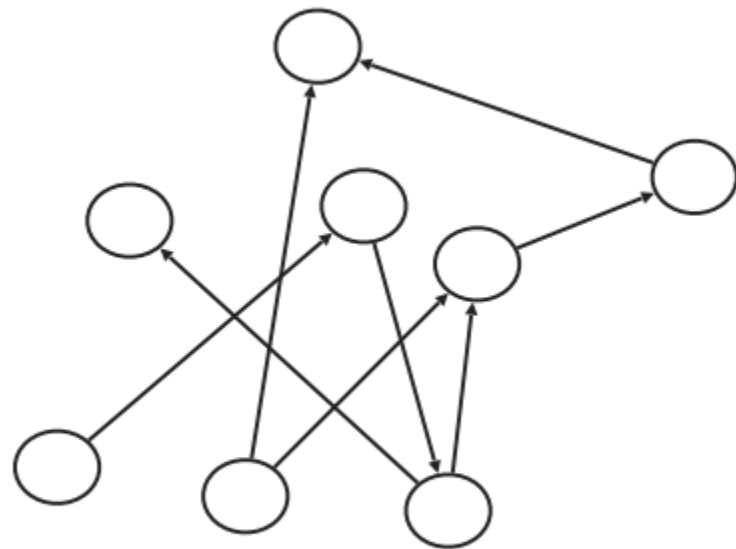
Cách 2

- Xét đồ thị $G = (X, U)$ vô hướng, giả sử tập X gồm N đỉnh và được sắp thứ tự $X = \{x_1, x_2, \dots, x_N\}$, tập U gồm M cạnh và được sắp thứ tự $U = \{u_1, u_2, \dots, u_M\}$.
- Ma trận của G , ký hiệu $A(G)$, là ma trận nhị phân $N \times M$: $A = (A_{ij})$ với A_{ij} được định nghĩa:
 - ❖ $A_{ij} = 1$ nếu đỉnh x_i kề với cạnh u_j ,
 - ❖ $A_{ij} = 0$ nếu ngược lại.



G

G gồm 2 thành phần liên thông

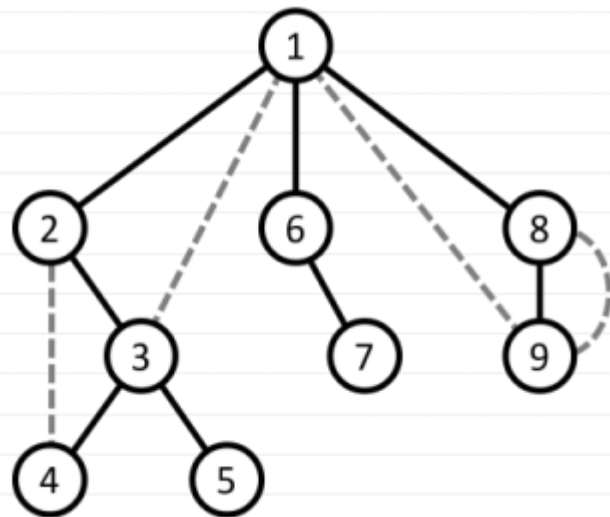


H

H là đồ thị liên thông

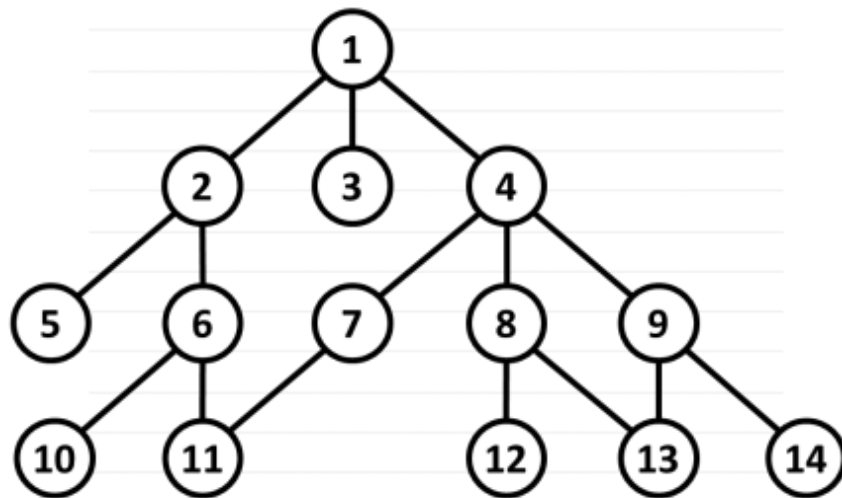
Thuật toán duyệt theo chiều sâu (Deep-First Search-DFS)

- Cho $G = (V, E)$ là đồ thị có tập các đỉnh V và tập các cạnh E . v là một đỉnh trong V và u là đỉnh kề của v , sao cho u cũng thuộc V .
- Khi đó ta dán nhãn cho tất cả các đỉnh của đồ thị là 0. Chọn một đỉnh v thuộc tập V để bắt đầu duyệt. Gán nhãn đỉnh v này là 1: v đã được duyệt.
- Chọn đỉnh u trong tập V kề với đỉnh v mà nhãn là 0. Duyệt qua đỉnh u và gán nhãn u là 1. Tiếp tục quá trình duyệt đến khi tất cả các đỉnh đồ thị có nhãn là 1



Thuật toán duyệt theo chiều rộng (Breadth-First Search-BFS)

- Sử dụng một cấu trúc dữ liệu hàng đợi để lưu trữ thông tin trung gian thu được trong quá trình tìm kiếm:
- Chèn đỉnh gốc vào hàng đợi (đang hướng tới)
- Lấy ra đỉnh đầu tiên trong hàng đợi và quan sát nó, chèn tất cả các đỉnh kề với đỉnh vừa thăm nhưng chưa được quan sát trước đó vào hàng đợi.
- Nếu hàng đợi là rỗng, thì tất cả các đỉnh có thể đến được đều đã được quan sát – dừng việc duyệt.
- Nếu hàng đợi không rỗng thì quay về bước 2.



```

void DFS(int v)
{
    ChuaXet[v] = 1;
    for ( int u=0; u < V; u++)
        if ( A[v][u]!=0 )
            if ( ChuaXet[u]==0 )
                DFS( u);
}

```

```

void BFS(int v)
{
    int QUEUE[MaxV], topQ=0,
    bottomQ=0;
    QUEUE[topQ++] = v;
    ChuaXet[v] = 1;
    while ( bottomQ >= topQ )
    {
        v = QUEUE[bottomQ++];
        for (int u=0; u < V; ++u)
            if ( A[v][u] != 0 )
                if (ChuaXet[u]==0)
                {
                    QUEUE[topQ++] = u;
                    ChuaXet[u] = 1;
                }
    }
}

```


Ví dụ:

Có n thành phố. Một số trong số chúng được kết nối, trong khi một số thì không. Nếu thành phố a được kết nối trực tiếp với thành phố b và thành phố b được kết nối trực tiếp với thành phố c , thì thành phố a được kết nối gián tiếp với thành phố c .

Tỉnh là một nhóm các thành phố được kết nối trực tiếp hoặc gián tiếp và không có thành phố nào khác ngoài nhóm.

Bạn được cung cấp một ma trận $n \times n$ `isConnected` trong đó `isConnected[i][j] = 1` nếu thành phố thứ i và thành phố thứ j được kết nối trực tiếp và `isConnected[i][j] = 0` nếu không.

Tính số lượng các tỉnh.

Bài giải chi tiết:



The End



Thank you for listening

Form điểm danh

