CHƯƠNG 4:
TẬP HỢP

Nguyễn Văn Linh

Khoa Công nghệ Thông tin - Truyền thông

Đại học Cần Thơ

NỘI DUNG

- Khái niệm tập hợp
- Các phép toán trên tập hợp
- Cài đặt tập hợp
- Từ điển
- Bảng băm

KHÁI NIỆM TẬP HỢP

- Tập hợp các thành viên (members) hoặc phần tử (elements) như khái niệm toán học.
- Các phần tử của tập hợp phải khác nhau
- Tập hợp có thứ tự hoặc không có thứ tự.
- Ở đây ta sẽ xét tập hợp có thứ tự, tức là trên tập hợp S có các quan hệ < thỏa mãn:
- Với mọi a, b trong S thì a<b hoặc b<a
- Với mọi a, b, c trong S, nếu a<b và b<c thì a<c

CANTHO UNIVERSITY

BIỂU DIỄN TẬP HỢP (1)

- Liệt kê các phần tử trong cặp dấu ngoặc {}
  - $x \in S$: x là một phần tử của tập hợp S
  - $x \notin S$: x không là một phần tử của tập hợp S
  - $\varnothing$: tập hợp rỗng, không có thành viên
  - VD: A={1,2} B= {1,2,3}

BIỂU DIỄN TẬP HỢP (2)

- Cho hai tập hợp A và B:
  - A là 1 bộ phận của B, kí hiệu $A \subseteq B$: nếu mọi phần tử của A đều là phần tử của B
    - VD: $A \subseteq B$
  - Tập hợp A và B bằng nhau, kí hiệu $A = B$:nếu $A \subseteq B$ và $B \subseteq A$
  - Hợp của hai tập hợp: $A \cup B = \{x | x \in A \text{ hoặc } x \in B\}$
  - Giao của hai tập hợp: $A \cap B = \{x | x \in A \text{ và } x \in B\}$
  - Hiệu của hai tập hợp: $A \setminus B = \{x | x \in A \text{ và } x \notin B\}$

CÁC PHÉP TOÁN TRÊN TẬP HỢP

Phép toán	Diễn giải
Make_Null_Set(S)	Tạo tập hợp S rỗng
Empty_Set(S)	Kiểm tra xem tập hợp S có rỗng?
Member(X,S)	Kiểm tra xem X có thuộc S?
Insert_Set(X,S)	Thêm phần tử X vào tập hợp S
Delete_Set(X,S)	Xoá phần tử X trong tập hợp S
Union(A, B,C)	$C=A \cup B$
Intersection(A, B, C)	$C=A \cap B$
Difference(A,B,C)	$C=A \setminus B$

CANTHO UNIVERSITY

CÀI ĐẶT TẬP HỢP

- CÀI ĐẶT BẰNG VECTƠ BIT
- CÀI ĐẶT BẰNG DANH SÁCH LIÊN KẾT

CÀI ĐẶT TẬP HỢP BẰNG VECTƠ BIT (1)

- Thường được dùng khi tập hợp của ta là 1 tập con của tập số nguyên, có giá trị từ 0..n-1. Khi đó ta sẽ dùng 1 mảng các bit có kích thước n để lưu trữ tập hợp
- Nếu i thuộc tập hợp thì phần tử thứ i của mảng có giá trị 1
- VD: muốn lưu trữ các tập có giá trị phần tử từ 0..9. Ta dùng mảng có tối đa 10 phần tử.
- Tập hợp A={2,5,7,9} sẽ được biểu diễn bởi:

0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	1	0	1	0	1

CÀI TẬP HỢP ĐẶT BẰNG VECTƠ BIT (2)

- **Khai báo**

```
#define Max_Length  100 // giá trị phần tử lớn nhất
typedef int Set [Max_Length];
```

- **Tạo tập hợp rỗng:**

```
void Make_Null_Set (Set &a){
      int i;
     for(i=0;i<Max_Length;i++) a[i]=0;
}
```

KIỂM TRA TẬP HỢP RỖNG

```
int Empty_Set (Set a){
    int i= 0, Empty= 1;
    while ((i<Max_Length) && Empty)
        if (a[i]==1) Empty=0;
        else i++;
    return Empty;
}
```

TẠO HỢP CỦA 2 TẬP HỢP

```
void Set_Union (Set a,Set b,Set &c){
    for (int i=0; i<Max_Length; i++)
        if((a[i]==1)||(b[i]==1)) c[i]=1;
        else c[i]=0;
    }
```

TẠO GIAO CỦA 2 TẬP HỢP

```
void Set_Intersection(Set a,Set b,Set &c){
    for (int i=0; i<Max_Length; i++)
        if  ((a[i]==1)&&(b[i]==1))  c[i]=1;
        else c[i]=0;
    }
```

TẠO HIỆU CỦA 2 TẬP HỢP

```
void Set_Difference(Set a,Set b,Set &c){
    for (int i=0; i< Max_Length; i++)
        if  ((a[i]==1)&& (b[i]==0)) c[i]=1;
        else c[i]=0;
    }
```

Xét xem một phần tử có thuộc một tập hợp hay không?

```
int Member (int i, Set a) {
    if (i<0) || (i>Max_Length-1) return 0;
    return a[i]==1;
}
```

Cài đặt Hợp của 2 tập hợp bằng cách dùng hàm member

```
void Set_Union (Set a,Set b,Set &c){
    int i;
    for (i=0;i<Max_Length;i++)
      if (Member(i,a) || Member(i,b)) c[i]=1;
      else c[i]=0;
    }
```

Xen một phần tử i vào trong tập hợp a

```
void Insert_Set (int i, Set &a){
    if ((i<0) || (i>Max_Length-1))
        printf("Gia tri khong hop le\n");
    else a[i]=1;
}
```

Xoá một phần tử i trong tập hợp a

```
void Delete_Set (int i, Set &a){
    if ((i<0) || (i>Max_Length-1))
        printf("Gia tri khong hop le\n");
    else a[i]=0;
}
```

ĐÁNH GIÁ PHƯƠNG PHÁP CÀI ĐẶT TẬP HỢP BẰNG VECTƠ BIT

- Ưu điểm:
  - Dễ cài đặt.
  - Thực hiện nhanh
- Nhược điểm:
  - Không thể biểu diễn cho các tập hợp có số lượng phần tử lớn bất kỳ.
  - Sử dụng bộ nhớ không tối ưu

CÀI ĐẶT TẬP HỢP BẰNG DSLK

- Khai báo

```
typedef int Element_Type;
typedef struct Node
    {
      Element_Type Data;
      Node * Next;
    };
typedef Node * Position;
typedef Position Set;
```

TẠO TẬP HỢP RỖNG

```
void Make_Null_Set(Set &S){
    S= (Node*) malloc(sizeof(Node));
    S->Next = NULL;
}
```

KIỂM TRA TẬP HỢP RỖNG

```
int Empty_Set(Set S){
Return S->Next==NULL;
}
```

Kiểm tra một phần tử có thuộc tập không?

```
int Member(Element_Type X, Set S){
        Position P = S;
        int Found = 0;
        while ((P->Next != NULL) && (!Found))
                if (P->Next->Data == X) Found = 1;
                else P = P->Next;
        return Found;
}
```

Thêm một phần tử vào tập

- Tìm kiếm xem có GT cần xen chưa?
- Nếu chưa có mới xen
- Xen bằng cách:
  - Tạo một ô mới T
  - Đặt T->Data = X
  - Xen T vào đầu/cuối DS S

Thêm một phần tử vào tập

```
void Insert_Set(Element_Type X, Set &S){
        Position T, P = S;
        int  Finish=0,  Found=0;
        while ((P->Next!=NULL)&&(!Finish) &&(!Found))
              if (P->Next->Data == X) Found = 1;
                    else  if (P->Next->Data<X) P=P->Next;
                          else  Finish=1;
         // Neu X khong ton tai trong S thi xen X vao vi tri P
         if(!Found) {
               T=(Node*)malloc(sizeof(Node));
              T->Data=X;
              T->Next=P->Next;
              P->Next=T;
          }
   }
```

Xóa một phần tử khỏi tập

- Tìm giá trị cần xóa
- Nếu tìm thấy thì giải phóng vị trí đó

Xóa một phần tử khỏi tập

```
void Delete_Set(Element_Type X, Set &S){
      Position T, P=S;
      int  Found=0;
       while ((P->Next!=NULL)&& (!Found))
                  if  (P->Next->Data==X) Found =1;
                  else P=P->Next;
       if (Found){
                  T=P->Next;
                   P->Next=T->Next;
                 // P->Next=P-Next->Next;
                   free(T);
       }
}
```

Tạo tập C là HỢP của 2 tập A và B

- Tạo tập hợp C rỗng
- Xen tất cả các phần tử của A vào C
- Xét tất cả các phần tử của B, nếu phần tử đó không thuộc A thì xen vao C

Tạo tập C là HỢP của 2 tập A và B

```
void Set_Union(Set A, Set B, Set &C){
    Position P = A;
    Make_Null_Set(C);
    while (P->Next!=NULL) {
            Insert_Set (P->Next->Data,C);
            P=P->Next;
    }
    P=B;
     while (P->Next != NULL){
          if (!Member(P->Next->Data, A))
                  Insert_Set (P->Next->Data, C);
           P=P->Next;
    }
}
```

Tạo tập C là GIAO của 2 tập A và B

- Tạo tập C rỗng
- Xét tất cả các phần tử của A, nếu phần tử nào thuộc B thì xen phần tử đó vào C

Tạo tập C là GIAO của 2 tập A và B

```
void Set_Intersection(Set A, Set B, Set &C) {
    Position P = A;
    Make_Null_Set(C);
    while (P ->Next!=NULL){
        if (Member(P ->Next ->Data, B))
            Insert_Set (P ->Next ->Data,C);
        P=P->Next;
    }
}
```

Tạo tập C là HiỆU của 2 tập A và B

- Tạo tập C rỗng
- Xét tất cả các phần tử của A, nếu phần tử nào không thuộc B thì xen phần tử đó vào C

Tạo tập C là HIỆU của 2 tập A và B

```
void Set_Difference (Set A, Set B, Set &C) {
    Position P = A;
    Make_Null_Set(C);
    while (P ->Next!=NULL){
        if (! Member(P ->Next ->Data, B))
            Insert_Set (P ->Next ->Data,C);
        P=P->Next;
    }
}
```

ĐÁNH GIÁ PHƯƠNG PHÁP CÀI ĐẶT TẬP HỢP BẰNG DSLK

- Ưu điểm: Có thể biểu diễn cho một tập hợp bất kỳ, số lượng phần tử không hạn chế. Sử dụng bộ nhớ tối ưu
- Nhược điểm: Tốc độ thực hiện chậm

BÀI TẬP

- Viết các khai báo cấu trúc dữ liệu và các thủ tục/hàm cho các phép toán trên tập hợp để cài đặt tập hợp kí tự (256 kí tự ASCII) bằng vectơ bit.

TỪ ĐIỂN

- Khái niệm: là một tập hợp mà ta không quan tâm đến các phép toán hợp, giao và hiệu của 2 tập hợp
- Do trong một số ứng dụng, chỉ sử dụng các phép toán Insert_Set, Delete_Set và Member
- Có thể cài đặt từ điển bằng:
  - Véct-bít
  - Danh sách liên kết có thứ tự hoặc không thứ tự
  - Mảng có kích thước cố định với con nháy chỉ đến vị trí cuối cùng (Tương tự cài đặt danh sách bằng mảng)
  - **Bảng băm**

CÀI ĐẶT TỪ ĐIỂN BẰNG MẢNG (1)

- Khai báo

```
#define Max_Length ...   //So phan tu toi da
typedef ... Element_Type;   //Kieu du lieu
typedef int Position;
typedef struct{
  Element_Type Data[Max_Length];
  Position Last;
} Dictionary;
```

CÀI ĐẶT TỪ ĐIỂN BẰNG MẢNG (2)

- Khởi tạo từ điển rỗng

```
void Make_Null_Dictionary(Dictionary &D){
    D.Last=0;
}
```

- Kiểm tra từ điển rỗng

```
int Empty_Dictionary(Dictionary D){
    return (D.Last==0);
}
```

- Kiểm tra từ điển đầy

```
int Full_Dictionary(Dictionary D){
    return (D.Last==Max_Length);
}
```

CÀI ĐẶT TỪ ĐIỂN BẰNG MẢNG (3)

- Hàm kiểm tra 1 phần tử có trong từ điển không

```
int Member(Element_Type X, Dictionary D){
    Position P=1;
    int Found=0;
    while ((P <= D.Last) && (!Found))
        if (D.Data[P-1] == X) Found = 1;
        else P++;
    return Found;
}
```

CÀI ĐẶT TỪ ĐIỂN BẰNG MẢNG (4)

- Thêm 1 phần tử vào từ điển:

```
void Insert_Dictionary(Element_Type X, Dictionary &D){
        if (!Member(X , D)) {
            D.Last++;
            D.Data[D.Last-1]=X;
        }else
            printf("\nPhan tu da ton tai trong tu dien");
    }
```

CÀI ĐẶT TỪ ĐIỂN BẰNG MẢNG (5)

- Xóa 1 phần tử khỏi từ điển:

```
void Delete_Dictionary(Element_Type X, Dictionary &D) {
        Position Q=1;
        int Found =0;
        while((Q <= D.Last)&&( !Found)
            if ( D.Data[Q-1] == X) Found =1 else Q++;
        if (Found) {
            /* for (int i=Q; i < D.Last;  i++)
                  D.Data[i-1] = D.Data[i];
            */
             D.Data[Q-1]=D.Data[D.Last -1];
             D.Last--;
        }
        else
            printf(« Khong ton tai trong Tu dien!");
}
```

ĐÁNH GIÁ PHƯƠNG PHÁP CÀI ĐẶT TỪ ĐIỂN BẰNG MẢNG

- Ưu điểm:
  - Tương tự cài đặt danh sách bằng mảng nên dễ dàng thực hiện
- Khuyết điểm:
  - Kích thước không thể lớn tùy ý
  - Tìm một phần tử chậm
  - Dùng bộ nhớ không hiệu quả

CÀI ĐẶT TỪ ĐIỂN BẰNG BẢNG BĂM

- BĂM ĐÓNG
- BĂM MỞ

- Cho phép tìm kiếm nhanh

BĂM ĐÓNG (1)

- Định nghĩa: Là mảng một chiều có B phần tử. Mỗi phần tử được gọi là một bucket (lô). Người ta lưu trữ dữ liệu trong các bucket.
- Để phân phối dữ liệu vào trong các bucket, người ta dùng Hàm băm.
- Có nhiều cách để xác định hàm băm, cách thông dụng nhất là h(x) = x % B.
- Giá trị dữ liệu x sẽ được lưu trong T[h(x)]

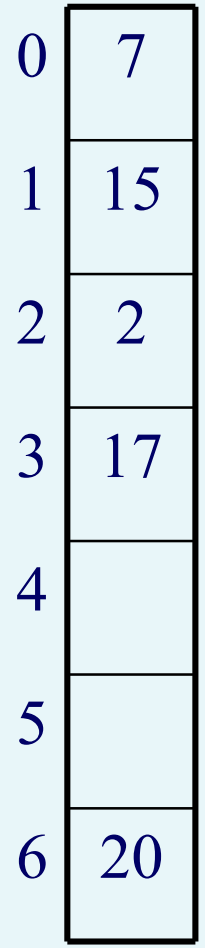
BĂM ĐÓNG (2)

Ví dụ: Ta cần lưu trữ các số nguyên 15, 7, 17, 20, 2 vào trong bảng băm có số bucket B = 7 và sử dụng hàm băm h(x) = x %7

0	7
1	15
2	2
3	17
4	
5	
6	20

BĂM ĐÓNG (3)

- Sự đụng độ: Có nhiều hơn một giá trị cần lưu trữ trong cùng một bucket.
- Ví dụ trong bảng băm trên ta đưa thêm giá trị 22, h(22) = 22%7 = 1 đụng độ với giá trị 15



BĂM ĐÓNG (4)

- Giải quyết đụng độ: Bằng chiến lược băm lại tuyến tính.
- Tính lại hi(x) = (h(x)+i) % B, với i = 1, 2, 3, … cho đến khi ta tìm thấy một bucket còn trống.
- Ví dụ

  h1(22) = (h(22)+1)%7 = 2

  h2(22) = (h(22)+2)%7 = 3

  h3(22) = (h(22)+3)%7 = 4

  Đưa 22 vào bucket 4

0	7
1	15
2	2
3	17
4	22
5	
6	20

CÀI ĐẶT TỪ ĐIỂN BẰNG BĂM ĐÓNG (1)

- Khai báo

```
#define B 100
#define Deleted –1000
 //Gia dinh gia tri cho o da bi xoa
#define Empty 1000
//Gia dinh gia tri cho o chua su dung
typedef int Element_Type;
typedef Element_Type Dictionary[B];
```

CÀI ĐẶT TỪ ĐIỂN BẰNG BĂM ĐÓNG (2)

- **Tạo từ điển rỗng**

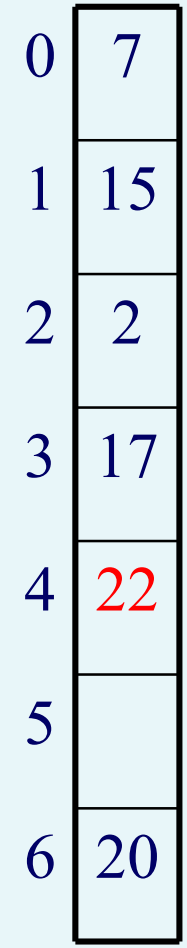
```
void Make_Null_Dictionary(Dictionary &D) {
        for (int i=0 ; i< B; i++) D[i]=Empty;
}
```

- **Hàm băm**

```
int H (Element_Type X){
    return X%B;
}
```

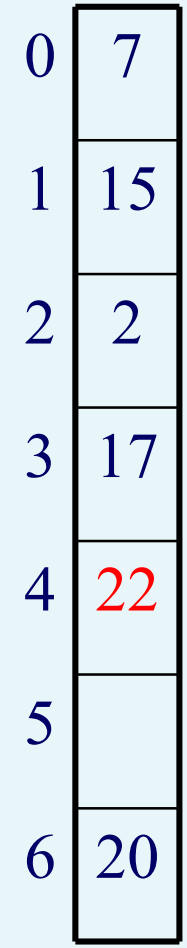
CÀI ĐẶT TỪ ĐIỂN BẰNG BĂM ĐÓNG
Hàm Member

- **Giải thuật: Xét dãy các bucket h(x), h1(x), h2(x), ... cho đến khi tìm thấy x hoặc gặp một vị trí trống**



CÀI ĐẶT TỪ ĐIỂN BẰNG BĂM ĐÓNG
Hàm Member

```
int Member(Element_Type X, Dictionary D) {
    int P = H(X);
    int Found = 0;
   int i=0;
    while ((i< B)) && (D[(P+i)%B]!=Empty) &&
    (!Found))
          if (D[(P+i)%B]) == X) Found = 1;
          else i++;
 return Found;
}
```



CÀI ĐẶT TỪ ĐIỂN BẰNG BĂM ĐÓNG

Thêm phần tử vào từ điển

```
void Insert_Dictionary(Element_Type X, Dictionary &D){
   int i=0, init;
   if (!Member(X,D)){
      init = H(X);
      while (i<B)&&(D[(i+init)%B]!=Empty)&&(D[(i+init)%B] != Deleted )
                  i++;
       D[(i+init)%B]=X;
    }else
       printf("\nPhan tu da ton tai");
}
```

0	7
1	15
2	2
3	
4	
5	
6	20

CÀI ĐẶT TỪ ĐIỂN BẰNG BĂM ĐÓNG
Xóa phần tử

```
void Delete_Dictionary(ElementType X, Dictionary &D){
    // tìm X và nếu tìm thấy mới xoá (đặt phần tử = Deleted)
    int Found =0;
    int i=0, init =H(X);
    while ((i<=B-1)&&(D[(i+init)%B]!=Empty) && (!Found))
        if (D[(i+init)%B]==X) Found =1;
        else i++;
    if (Found)
        D[(i+init)%B]=Deleted;
     else printf("\nKhong ton tai!");
    }
```

0	7
1	15
2	2
3	17
4	22
5	
6	20

ĐÁNH GIÁ BẢNG BĂM ĐÓNG

- Ưu điểm
  - Tìm kiếm nhanh
  - Dễ dàng cài đặt
- Nhược điểm
  - Sử dụng bộ nhớ không hiệu quả
  - Đụng độ

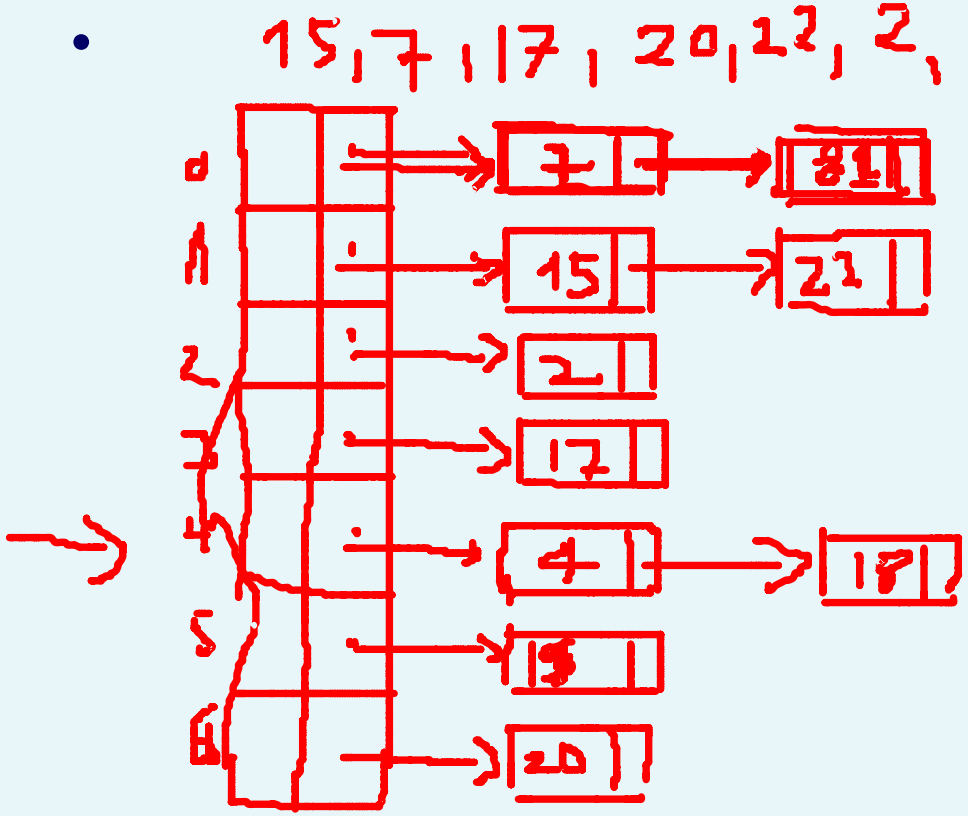
Bài tập

- Cài đặt hàm Full_Dictionary và Empty_Dictionary
- Cài đặt hàm kiểm tra từ điển rỗng

Kiểm tra từ điển rỗng

```
int Empty_Dictionary(Dictionary D){
   int Is_Empty = 1, i=0;
   while (i<B) && Is_Empty
        if ((B[i] != Empty) && (B[i] != Deleted))
             Is_Empty = 0;
        else i++;
   return Is_Empty;
}
```

BẢNG BĂM MỞ

- Định nghĩa: Là một mảng một chiều có B phần tử. Mỗi phần tử là một con trỏ, trỏ đến một danh sách liên kết lưu trữ dữ liệu. Mỗi danh sách liên kết được gọi là một lô (bucket).
- 

CÀI ĐẶT TỪ ĐIỂN BẰNG BĂM MỞ
Khai báo

```
#define B ...
typedef ... Element_Type;
typedef struct Node {
   Element_Type Data;
   Node* Next;
};
typedef Node* Position;
typedef Position Dictionary[B];
```

CÀI ĐẶT TỪ ĐIỂN BẰNG BĂM MỞ
Khởi tạo từ điển rỗng

```
void Make_Null_Dictionary(Dictionary &D) {
   for(int i=0;i<B;i++)
       D[i]->Next=NULL;
}
```

CÀI ĐẶT TỪ ĐIỂN BẰNG BĂM MỞ
Hàm Member

```
int Member(Element_Type X, Dictionary D) {
    Position P=D[H(X)];
    int Found=0;
    //Duyet tren ds duoc tro boi D[H(X)]
    while((P->Next!=NULL) && (!Found))
        if (P->Next->Data==X) Found=1;
        else P=P->Next;
    return Found;
}
```

Thêm một phần tử vào từ điển (Xen vào đầu lô)

```
void Insert_Dictionary(Element_Type X, Dictionary &D){
    int Bucket;
    Position Temp;
    if (!Member(X,D)) {
        Bucket=H(X);
        Temp=(Node*)malloc(sizeof(Node));
        Temp->Data=X;
        Temp->Next=D[Bucket]->Next;
        D[Bucket]->Next=Temp;
    }
}
```

Thêm một phần tử vào từ điển (Xen vào cuối lô)

```
void Insert_Dictionary(Element_Type X, Dictionary &D){
     Position Temp, P=D[H(X)];
      int Found = 0;
     while ((P->Next !=NULL)  && (!Found))
           if (P->Next->Data == X) Found =1;
           else P= P->Next;
       if (!Found) {
            Temp=(Node*)malloc(sizeof(Node));
            Temp->Data=X;
            Temp->Next=NULL;
            P->Next=Temp;
        }
}
```

CÀI ĐẶT TỪ ĐIỂN BẰNG BĂM MỞ
Xóa phần tử có giá trị X

```
void Delete_Dictionary(Element_Type X, Dictionary &D){
      int Found = 0;
      Position Temp, P = D[H(X)];
      while ((P->Next!=NULL) && (!Found))
            if (P->Next->Data==X) Found=1;
            else P = P->Next;
     if (Found){  //Xoá tại vị trí P
            Temp=P->Next;
             P->Next=Temp->Next;
            free(Temp);
      }
}
```

ĐÁNH GIÁ BẢNG BĂM MỞ

- Ưu điểm
  - Tìm kiếm nhanh
  - Không bị hạn chế số phần tử
  - Không bị đụng độ
  - Sử dụng bộ nhớ tối ưu.

CANTHO UNIVERSITY

Bài tập: Từ điển mà các phần tử trong các lô được sắp thứ tự

- Viết các hàm member, insert, delete

Thank you