7 9 1 15 6 8 23 45 17

$i = K_i \% M$

1. KẾT NỐI TRỰC TIẾP (KẾT NỐI NGOÀI)

Nguyên tắc: phần tử đụng độ sẽ nối liên kết vào phần tử tại vị trí của nó

M = 6

Chỉ số (Vị trí)	key (Giá trị)
0	6
1	7 -> 1
2	8
3	9 -> 15 -> 45
4	NULL
5	23 -> 17

```
#define M 6
struct nut {
   int key;
   struct nut *tiep;
}
typedef struct nut *Node;
// Node B[M];
```

```
void init(Node B[]) {
   int i;
   for (i == 0; i < M; i++)
      B[i] = NULL;
}
```

```
int search(Node B[], int x) {
   Node tam;
   tam = B[x % M];
   while (tam != NULL) {
      if (tam->key == x) return 1;
      tam = tam->tiep;
   }
   return -1; // không tìm thấy
}
```

```
void add(Node B[], int x) {
   Node tam, new;
   tam = B[x % M];
   new = (Node)malloc(sizeof(Node));
   new->key = x;
   new->tiep = NULL;
   if (tam == NULL) tam = new;
   else { // thêm phần tử cuối danh sách
     while (tam->tiep != NULL) tam = tam->tiep;
     tam->tiep = new;
   }
}
```

7 9 1 15 6 8 23 45 17

$i = K_i \% M$

2. KẾT NỐI HỢP NHẤT (KẾT NỐI TRONG)

Nguyên tắc: phần tử đụng độ sẽ được lưu vào vị trí có chỉ số lớn nhất còn trống

M = 9

Chỉ số (Vị trí)	key (Giá trị)	next (Liên kết)
0	9	3
1	1	-1
2	17	-1
3	45	-1
4	23	2
5	8	4
6	15	8
7	7	-1
8	6	5

```
#define M 9
struct hash {
   int key;
   int next;
}
typedef struct hash HashTable;
// int N = 12;
// HashTable B[N];
// N >= Số lượng phần tử cần
//      xếp vào bảng băm
// N >= M
```

```
void init(Node B[]) {
   int i;
   for (i == 0; i < M; i++) {
     B[i]->key = 0;
     B[i]->next = -1;
   }
}
```

```
int search(Node B[], int x) {
   int i = x % M;
   do {
      if (B[i]->key == x) return 1
      i = B[i]->next;
   } while (i != -1)
   return -1; // không tìm thấy
}
```

```
void add(Node B[], int x) {
   int i = x % M;
   int j = M-1; // Vị trí cuối bảng băm
   if (B[i]->key == 0) B[i]->key = x;
   else {
       while (B[j]->key != 0)
          j--; // tìm vị trí lưu phần tử đụng độ
       B[j]->key = x;
       while (B[i]->next != -1)
          i = B[i]->next; // tìm vị trí lưu liên kết
       B[i]->next = j;
   }
}
```