HOMEWORK

Homework#1: Interpolation Search

Input:

- Tham số dòng lệnh: `file_thực_thi Số_cần_tìm Dãy_số` (cách nhau bở khoảng trắng).
- **VD**: a.exe 3 1 3 5 7 9

Output:

- Vị trí của số cần tìm (Nếu không tìm thấy, hãy in ra -1) - Số vòng lặp để tìm thấy số cần tìm.
- **VD**: 2 - 1

Homework#2: So khớp chuỗi

Input:

- Tham số dòng lệnh: `file_thực_thi File_text Chuỗi_cần_tìm Thuật_toán` (cách nhau bở khoảng trắng).

  Các thuật toán bao gồm: `BF` (brute-force), `RK` (Rabin-Karp), `KMP` (Knuth-Morris-Patt), `BM` (Boyer-Moore)
- **VD**: a.exe input.txt AABA BM
- input.txt: AABAACAADAABAABA

Output:

- Số lần xuất hiện của chuỗi cần tìm kiếm và thời gian thực thi (ms).
- **VD**: 3 - 0.69

Homework#3: Sparse Table

Yêu cầu:

1. Tạo bảng: (đồng thời xóa bảng cũ trùng tên). Tối đa 5 bảng.
   - Tham số dòng lệnh: `file_thực_thi make Tên_bảng Loại_bảng Dãy_số`
     Trong đó:
     – Loại bảng: MIN / MAX / GCD
   - **VD:** a.exe make A MIN 1 2 7 4 6 5
   - In bảng ra màn hình.
2. Tra cứu bảng:
   - Tham số dòng lệnh: `file_thực_thi query Tên_bảng Chặn_dưới Chặn_trên`
   - **VD:** a.exe query A 1 4
   - In giá trị cần tìm ra màn hình

Homework#4: Circular Linkedlist

Yêu cầu:

Định nghĩa cấu trúc dữ liệu của một DSLK vòng và cài đặt các hàm `addHead`, `addTail`, `removeHead`, `removeTail`, `addAfter`, `removeAfter`, `printList` tương ứng. *(Tham khảo Lab 1)*

Homework#5: Priority Queue

Thông tin của một đối tượng trong một Priority Queue được định nghĩa như sau:

- **ID**: Chuỗi kí tự, thể hiện thông tin của đối tượng.
- **Order**: Số nguyên dương, thứ tự thực theo thời gian của đối tượng.
- **Priority**: Số nguyên dương, độ ưu tiên của đối tượng.

Yêu cầu:

Định nghĩa các cấu trúc dữ liệu của hàng đợi ưu tiên bằng **DSLK** đơn và **Min-heap** thông qua các hàm sau:

- `isEmpty()`: Kiểm tra xem hàng đợi có rỗng hay không.
- `Insert()`: Thêm một đối tượng vào hàng đợi có sẵn.
- `Extract()`: Trích xuất một đối tượng (theo độ ưu tiên) ra khỏi hàng đợi có sẵn.
- `Remove()`: Xóa một đối tượng với **ID** cho trước ra khỏi hàng đợi có sẵn.
- `changePriority()`: Thay đổi độ ưu tiên của một phần tử với **ID** cho trước. Cập nhật hàng đợi.
- Và các hàm hỗ trợ nếu cần thiết.

Homework#6: Danh sách đa liên kết

Cài đặt cấu trúc danh sách đa liên kết tương ứng để giải các bài toán sau:

1. **Sắp xếp Topo:**

   Input: File *"input.txt"* có định dạng $(x_1, x_2), (x_2, x_3), ...$ thể hiện các quan hệ $x_1 \prec x_2, x_2 \prec x_3, ...$ với $x_1, x_2, x_3, ...$ là các số nguyên dương thể hiện tên công việc.

   - **VD:** (1,2)(2,3)(1,4) *<Không có khoảng trắng>*

   Output: Một thứ tự topo thích hợp. Các công việc cách nhau bởi khoảng trắng.

   - **VD:** 1 2 3 4

2. **Radix Sort:**

   Input: File *"input.txt"* có định dạng:

   – Dòng thứ 1: Số `k` thể hiện khoảng chia yêu cầu và số `n` thể hiện số phần tử cần được sắp xếp, cách nhau bởi khoảng trắng.

   – Dòng thứ 2: `n` số nguyên dương cần sắp xếp, cách nhau bởi khoảng trắng.

   - **VD:**

     2 5

     1234 5678 678 123 234

   Output: Các số đã được sắp xếp, cách nhau bởi khoảng trắng.

3. **VD:** 123 234 678 1234 5678

Homework#7: Cây Trie

Cấu trúc của một cây Trie được định nghĩa như sau:

```
struct TrieNode{
    int ID;
    TrieNode* next[26];
};
```

Yêu cầu:

Hãy cài đặt các hàm sau đây

- `void Insert(TrieNode* &Dic, string word, int ID, ...)`

  Mô tả: Thêm một từ vào cây Trie có sẵn.

- `void createTrie(TrieNode* &Dic, string DicFile, ...)`

  Mô tả: Tạo một cây Trie bằng cách đọc một file từ điển cho trước.

- `int lookUp(TrieNode* Dic, string word, ...)`

  Mô tả: Tìm và trả về giá trị của một từ trên cây Trie cho trước.

- `vector <string> lookUpPrefix(TrieNode* Dic, string prefix)`

  Mô tả: Tìm và trả về các từ có cùng tiền tố trên cây Trie cho trước.

- `void Remove(TrieNode* Dic, string word, ...)`

  Mô tả: Tìm và xóa một từ trên cây Trie cho trước

Homework#8: Cây Đỏ Đen

```
struct RBNode{
    int key;
    bool color; //Black = 0
    RBNode* parent;
    RBNode* left;
    RBNode* right
};
```

Yêu cầu:

Hãy cài đặt các hàm (và các hàm phụ trợ nếu cần thiết) sau đây

- `void Insert(RBNode* &pRoot, int key, ...)`

  Mô tả: Thêm một giá trị vào cây đỏ đen cho trước.

- `RBNode* createTree(int a[], int n, ...)`

  Mô tả: Xây dựng một cây đỏ đen từ mảng cho trước.

- `RBNode* lookUp(RBNode* pRoot, int key, ...)`

  Mô tả: Tìm và trả về một node với giá trị cho trước. Nếu node không tồn tại, trả về `NIL`.

- `int Height(RBNode* pRoot)`

  Mô tả: Tìm và trả về chiều cao của một cây đỏ đen cho trước.

- `int BlackHeight(RBNode* pRoot)`

  Mô tả: Tìm và trả về chiều cao đen của một cây đỏ đen cho trước.

- `int Remove(RBNode* pRoot, int key, ...)`

  Mô tả: Tìm và xóa một giá trị trên cây đỏ đen cho trước

Homework#9: Nén LZW

Input 1:

- Thao tác nén
- Tham số dòng lệnh: `file_thực_thi -c Dãy_ký_tự` (cách nhau bởi khoảng trắng).
- **VD**: a.exe -c WYS*WYGWYS*WYSWYSG

Output 1:

- Dãy số tương ứng (thập phân)
- Dãy số tương ứng (nhị phân)
- Tỉ lệ nén = $\frac{N-n}{N} * 100$ (n: số bit sau khi nén, N: số bit trước khi nén)
- **VD**: 87 89 83 42 256 71 256 258 262 262 71

  001010111 001011001 001010011 000101010 100000000 001000111 100000000

  100000010 100000110 100000110 001000111

  31.25

Input 2:

- Thao tác nén
- Tham số dòng lệnh: `file_thực_thi -e Dãy_số` (cách nhau bởi khoảng trắng).
- **VD**: a.exe -e 87 89 83 42 256 71 256 258 262 262 71

Output 2:

- Dãy ký tự tương ứng
- Tỉ lệ nén = $\frac{N-n}{N} * 100$ (n: số bit sau khi nén, N: số bit trước khi nén)
- **VD**: WYS*WYGWYS*WYSWYSG

  31.25