Họ và tên: Trương Trung Đức

MSSV: 21880291

|  |  |
| --- | --- |
| **Báo cáo** | Bài tập 2 – Chuyên đề tổ chức dữ liệu |
| **Ngày lập báo cáo** | 10/10/2024 |
| **Mức độ hoàn thành** | 100% - hoàn thành các yêu cầu |
| **Mục lục** | [1. Dùng mảng cài đặt hàng đợi số nguyên:](#_Toc179821256)  [2. Dùng danh sách liên kết cài đặt hàng đợi số nguyên:](#_Toc179821257)  [3. So sánh 2 cách cài đặt hàng đợi các số nguyên:](#_Toc179821258)  [4. Viết chương trình tra từ điển hiệu quả.](#_Toc179821259) |

Nội dung:

Yêu cầu 1:

Hàng đợi (queue) là cấu trúc dữ liệu quản lý một tập các đối tượng, hỗ trợ các thao tác:

* Enqueue: thêm đối tượng vào hàng đợi.
* Dequeue: gỡ đối tượng được thêm vào sớm nhất ra khỏi hàng đợi (FIFO – First In First Out).
* Peek: cho biết đối tượng được thêm vào hàng đợi sớm nhất (không gỡ khỏi hàng đợi).

Cài đặt hàng đợi các số nguyên bằng cách dùng mảng và dùng danh sách liên kết.

So sánh 2 cách.

Yêu cầu 2 :

Từ dữ liệu từ điển Tiếng Anh ở trang [https://www.mso.anu.edu.au/%7Eralph/OPTED/.](https://www.mso.anu.edu.au/%7Eralph/OPTED/) Viết chương trình tra từ điển hiệu quả.

1. **Dùng mảng cài đặt hàng đợi số nguyên:**

|  |  |
| --- | --- |
| Mô tả | Mã nguồn |
| Khai báo | #define MAX 100  typedef struct {  int arr[MAX];  int first;  int last;  } Queue; |
| Hàm khởi tạo hàng đợi | void initQueue(Queue\* q) {  q->first = -1; // Đặt first là -1 để biểu thị hàng đợi rỗng  q->last = -1; // Đặt last là -1 để biểu thị hàng đợi rỗng  } |
| Hàm thêm phần tử vào hàng đợi: enqueue | void enqueue(Queue\* q, int value) {  if (isFull(q)) {  printf("Queue is full!\n");  return;  }  if (isEmpty(q)) {  q->first = 0; // Đặt first nếu hàng đợi trước đó rỗng  }    q->arr[++q->last] = value; // Thêm phần tử vào cuối hàng đợi  } |
| Hàm gỡ bỏ và trả về phần tử đầu tiên (phần tử được thêm vào sớm nhất ra khỏi hàng đợi (FIFO)): dequeue | int dequeue(Queue\* q) {  if (isEmpty(q)) {  printf("Queue is empty!\n");  return -1; // Giá trị lỗi  }  int value = q->arr[q->first]; // Lưu giá trị phần tử đầu tiên  if (q->first >= q->last) {  q->first = q->last = -1; // Reset nếu hàng đợi trở về trạng thái rỗng  } else {  q->first++; // Cập nhật chỉ số first  }  return value; // Trả về giá trị  } |
| Hàm xem phần tử đầu tiên mà không gỡ bỏ: peek | int peek(Queue\* q) {  if (isEmpty(q)) {  printf("Queue is empty!\n");  return -1; // Giá trị lỗi  }  return q->arr[q->first]; // Trả về giá trị của phần tử đầu tiên  } |

1. **Dùng danh sách liên kết cài đặt hàng đợi số nguyên:**

|  |  |
| --- | --- |
| Mô tả | Mã nguồn |
| Khai báo nút cho danh sách liên kết và khai báo cấu trúc hàng đợi | typedef struct Node {  int data;  struct Node\* next;  } Node;  typedef struct {  Node\* first;  Node\* last;  } Queue; |
| Hàm khởi tạo hàng đợi | void initQueue(Queue\* q) {  q->first = NULL;  q->last = NULL;  } |
| Hàm thêm phần tử vào hàng đợi: enqueue | void enqueue(Queue\* q, int value) {  Node\* newNode = (Node\*)malloc(sizeof(Node));  newNode->data = value;  newNode->next = NULL;  if (q->last) {  q->last->next = newNode;  } else {  q->first = newNode;  }  q->last = newNode;  } |
| Hàm gỡ bỏ và trả về phần tử đầu tiên (phần tử được thêm vào sớm nhất ra khỏi hàng đợi (FIFO)): dequeue | int dequeue(Queue\* q) {  if (q->first) {  Node\* temp = q->first;  int value = temp->data;  q->first = q->first->next;  if (!q->first) {  q->last = NULL;  }  free(temp);  return value;  } else {  printf("Queue is empty!\n");  return -1;  }  } |
| Hàm xem phần tử đầu tiên mà không gỡ bỏ: peek | int peek(Queue\* q) {  if (q->first) {  return q->first->data;  } else {  printf("Queue is empty!\n");  return -1;  }  } |

1. **So sánh 2 cách cài đặt hàng đợi các số nguyên:**

|  |  |  |
| --- | --- | --- |
|  | **Dùng mảng** | **Dùng danh sách liên kết** |
| Ưu điểm | Dễ cài đặt.  Truy cập nhanh đến phần tử theo chỉ số. | Kích thước linh hoạt, không bị giới hạn.  Thêm và gỡ bỏ nhanh hơn khi số lượng lớn. |
| Nhược điểm | Giới hạn kích thước (khó mở rộng).  Khó khăn khi có nhiều thao tác chèn và gỡ bỏ. | Chiếm nhiều bộ nhớ hơn do cần các con trỏ.  Không có khả năng truy cập phần tử tùy ý (random access) |

1. **Viết chương trình tra từ điển hiệu quả.**

Phân tích:

* Trang: <https://www.mso.anu.edu.au/~ralph/OPTED/> , có 2 dạng dữ liệu có thể tải xuống: OPTED v0.03 by Letter(size) và OPTED v0.03 by Archive



*Hình: trang dữ liệu OPTED*

* Tải xuống dữ liệu **OPTED v0.03 by Letter(size)**.
* Dữ liệu Từ điển được thiết lập thành danh sách từ có định nghĩa, sử dụng đánh dấu HTML tối thiểu. Các thẻ được sử dụng là <P>, <B> và <I> và chúng có tác dụng phân định các từ (giữa <B>), loại từ hoặc loại (giữa <I>) và định nghĩa (Phần còn lại của dòng). Mỗi mục nhập nằm giữa một cặp <P>, </P>.
* Xây dựng **cây tìm kiếm nhị phân:** 
  + Cấu trúc nút trong cây tìm kiếm, gồm từ, nghĩa và con trỏ tới các nút con nhánh trái và nhánh phải.
  + Hàm tạo một nút mới với từ và nghĩa được cung cấp.
  + Hàm thêm một từ vào cây theo quy tắc của cây tìm kiếm nhị phân.
  + Hàm tìm kiếm nghĩa của từ trong cây.
  + Hàm giải phóng bộ nhớ được cấp phát cho cây.
* Hàm để trích xuất từ, nghĩa từ mỗi dòng dữ liệu. Đưa dữ liệu vào cây
* Sử dụng hàng đợi, cài bằng danh sách liên kết để gợi ý các từ khóa từ ký tự nhập vào.
* Cho người dùng nhập từ vào để tra.
* Cây tìm kiếm nhị phân, hàng đợi là file riêng (cho phép sử dụng hàng đợi cài bằng mảng hoặc cài bằng danh sách liên kết)

|  |  |
| --- | --- |
| **Mô tả** | **Mã nguồn** |
|  |  |
| Mỗi nút sẽ có hai con trỏ: một trỏ tới nút con nhánh trái và một trỏ tới nút con nhánh phải. | typedef struct Node {  char word[100];  char meaning[256];  struct Node\* left; // Con trỏ tới nút nhánh trái  struct Node\* right; // Con trỏ tới nút nhánh phải  } Node; |
| Hàm tạo nút mới với từ và nghĩa được thêm vào. | Node\* createNode(const char\* word, const char\* meaning) {  Node\* newNode = (Node\*)malloc(sizeof(Node));  strcpy(newNode->word, word);  strcpy(newNode->meaning, meaning);  newNode->left = NULL;  newNode->right = NULL;  return newNode;  } |
| Dùng đệ quy để thêm một nút mới vào cây.  Hàm sẽ kiểm tra vị trí của từ mới so với từ trong nút hiện tại và quyết định thêm nó vào nhánh trái hoặc nhánh phải. | Node\* insert(Node\* r, const char\* word, const char\* meaning) {  if (r == NULL) {  return createNode(word, meaning); // Tạo nút mới nếu cây trống  }    // So sánh và quyết định thêm vào nhánh trái hoặc nhánh phải  if (strcmp(word, r->word) < 0) {  r->left = insert(r->left, word, meaning); // Đệ quy -thêm vào nhánh trái  } else if (strcmp(word, r->word) > 0) {  r->right = insert(r->right, word, meaning); // Đệ quy -thêm vào nhánh phải  }  return r;  } |
| Dùng đệ quy đuôi.  Hàm tìm kiếm nghĩa của từ. | const char\* search(Node\* r, const char\* word) {  if (r == NULL) {  return NULL; // Không tìm thấy  }    if (strcmp(word, r->word) == 0) {  return r->meaning; // Tìm thấy nghĩa  } else if (strcmp(word, r->word) < 0) {  return search(r->left, word); // Đệ quy -tìm kiếm bên trái  } else {  return search(r->right, word); // Đệ quy -tìm kiếm bên phải  }  } |
| Giải phóng bộ nhớ đã được cấp phát cho từng nút. | void freeTree(Node\* r) {  if (r) {  freeTree(r->left); // Giải phóng nhánh trái  freeTree(r->right); // Giải phóng nhánh phải  free(r); // Giải phóng nút hiện tại  }  } |
| Xử lý dữ liệu và đưa vào cây | void processDictionary(Node\*\* r, const char\* filename) {  FILE\* file = fopen(filename, "r");  if (!file) {  perror("Failed to open file");  return;  }  char line[1024];  while (fgets(line, sizeof(line), file)) {  char word[100], meaning[512];  // Tìm thẻ <P> và phân tích cú pháp  if (strstr(line, "<P>") != NULL) {  while (fgets(line, sizeof(line), file) && strstr(line, "</P>") == NULL) {  if (sscanf(line, "<B>%[^<]</B> (<I>%\*[^<]</I>) %[^\n]", word, meaning) == 2) {  \*r = insert(\*r, word, meaning); // Thêm vào cây  }  }  }  }  fclose(file);  } |
| Đọc dữ liệu từ điển | void processAllDictionaries(Node\*\* root) {  for (char letter = 'A'; letter <= 'Z'; letter++) {  char filename[50];  sprintf(filename, "fileOPTED/%c.html", letter);  processDictionary(root, filename);  }  } |
| Hàm gợi ý từ có chứa ký tự | void printWordsContainingChar(Node\* root, char character) {  if (root == NULL) return;  Queue\* queue = createQueue();  enqueue(queue, root);  while (!isEmpty(queue)) {  Node\* current = dequeue(queue);    if (strchr(current->word, character) != NULL) {  printf("Word: %s, Meaning: %s\n", current->word, current->meaning);  }  if (current->left != NULL) {  enqueue(queue, current->left);  }  if (current->right != NULL) {  enqueue(queue, current->right);  }  }  free(queue);  } |