Assignment 03.02

Võ Lê Ngọc Thịnh

Ngày 11 tháng 4 năm 2025

Mục lục

1 Singly Linked List

1.1 Ứng dụng

- **Quản lý bộ nhớ động (Memory Management):** Hệ thống vận hành và các ứng dụng sử dụng danh sách liên kết đơn để quản lý các khối bộ nhớ tự do. Danh sách này giúp theo dõi các vùng nhớ chưa được sử dụng, cho phép phân bổ và giải phóng bộ nhớ một cách linh hoạt.
- **Biểu diễn đa thức (Polynomial Representation):** Mỗi số hạng của đa thức được đại diện bởi node chứa hệ số và số mũ. Danh sách liên kết rất tiện lợi khi thực hiện các phép toán như cộng, trừ hoặc nhân đa thức do có khả năng thêm/xóa node một cách nhanh chóng.

1.2 Giải thích

- **Quản lý bộ nhớ động:**
  - **Hiệu quả về bộ nhớ:** Vì mỗi node của Singly Linked List chỉ chứa một con trỏ (next) nên không tốn bộ nhớ cho các con trỏ dư thừa như trong doubly linked list.
  - **Thao tác nhanh:** Việc thêm hoặc xóa một node ở đầu danh sách chỉ cần điều chỉnh một vài con trỏ, rất hữu ích khi hệ thống cần quản lý nhiều yêu cầu cấp phát và giải phóng bộ nhớ liên tục.
- **Biểu diễn đa thức:**
  - **Xử lý dữ liệu không đầy đủ:** Nhiều đa thức có phần tử rời rạc do số mũ không liên tục. Sử dụng danh sách liên kết đơn giúp lưu trữ một cách linh hoạt mà không phải dành bộ nhớ cho các số hạng có hệ số bằng 0.
  - **Dễ dàng thao tác:** Thêm, bớt số hạng hoặc cập nhật hệ số đơn giản, thuận tiện cho các thuật toán xử lý đa thức như cộng trừ hay nhân đa thức mà không cần phải dồn dữ liệu vào cấu trúc mảng có kích thước cố định.

1.3 Cấu trúc tương ứng

Quản lý bộ nhớ động

```
struct MemoryBlock{
    int start, size;
    MemoryBlock* next;
    MemoryBlock (int start=0,int size=0){
        this->start = start;
        this->size = size;
```

```
        this->next = nullptr;
    }
};

class FreeList{
    private:
        MemoryBlock* pHead;

    public:
        FreeList(){
            pHead = nullptr;
        }

        void addBlock(int start, int size){
            MemoryBlock* newBlock = new MemoryBlock(start, size);
            newBlock->next = pHead->next;
            pHead = newBlock;
        }

        int allocate(int size){
            if (pHead == nullptr)   return -1;
            MemoryBlock** p = &pHead;
            MemoryBlock* current = pHead;

            while (current != nullptr){
                if (current->size >= size){
                    int result = current->start;
                    current->start += size;
                    current->size -= size;

                    if (current->size == 0){
                        *p = current->next;
                        delete current;
                    }
                    return result;
                }
                p = &(current->next);
                current = current->next;
            }
            return -1;
        }
};
```

Biểu diễn đa thức

```
class Term{
    public:
        int coeff, exponent;
        Term* next;

        Term (int coeff=0, int exponent=0){
            this->coeff = coeff;
            this->exponent = exponent;
            this->next = nullptr;
        }
};

class Polynomial{
    private:
        Term* pHead;

    public:
        Polynomial() : pHead(nullptr) {}

        void addTerm(int coeff, int exponent){
            Term* newTerm = new Term(coeff, exponent);

            if (pHead == nullptr || pHead->exponent < exponent){
                newTerm->next = pHead;
                pHead = newTerm;
                return;
            }

            Term* current = pHead;
            while (current->next && current->next->exponent >=
               exponent){
                current = current->next;
            }

            newTerm->next = current->next;
            current->next = newTerm;
        }

};
```

2 Doubly Linked List

2.1 Ứng dụng

- Quản lý điều hướng "Back" và "Forward" trên trình duyệt web.
- Theo dõi và thực hiện các thao tác "Undo/Redo".

2.2 Giải thích

- **Quản lý điều hướng "Back" và "Forward"**
  - **Dễ di chuyển qua lại:** Khi người dùng truy cập một trang mới hay quay lại trang trước, khả năng di chuyển hai chiều của doubly linked list cho phép dễ dàng chuyển sang trang trước hay trang sau, giúp quản lý lịch sử duyệt web một cách trực quan và hiệu quả.
- **Theo dõi và thực hiện các thao tác "Undo/Redo"**
  - **Theo dõi thao tác hai chiều:** Khi người dùng thực hiện thao tác chỉnh sửa (như thêm, xóa, thay đổi nội dung), mỗi thao tác có thể được lưu dưới dạng một node trong danh sách đôi. Việc chứa con trỏ về cả thao tác trước và sau giúp dễ dàng thực hiện lệnh Undo và Redo.
  - **Hiệu quả và linh hoạt:** Nhờ thiết kế rẽ nhánh theo từng thao tác, người dùng có thể thao tác lại mà không cần lưu trữ toàn bộ lịch sử trong một mảng, tiết kiệm bộ nhớ và thời gian truy xuất.

2.3 Cấu trúc tương ứng

Quản lý điều hướng "Back" và "Forward"

```
class HistoryNode{
    public:
        string url;
        HistoryNode* prev;
        HistoryNode* next;

        HistoryNode(const string &_url){
            this->url = _url;
            this->prev = nullptr;
            this->next = nullptr;
        }
};


```

```cpp
class BrowserHistory{
    private:
        HistoryNode* current_page;

    public:
        BrowserHistory(const string &homepage_url){
            current_page = new HistoryNode(homepage_url);
        }

        void addNewPage(const string &url){
            HistoryNode* newPage = new HistoryNode(url);
            newPage->prev = current_page;
            current_page->next = newPage;
            current_page = newPage;
        }

        bool back(){
            if (current_page->prev != nullptr){
                current_page = current_page->prev;
                return true;
            }
            return false;
        }

        bool forward(){
            if (current_page->next != nullptr){
                current_page = current_page->next;
                return true;
            }
            return false;
        }
};
```

Theo dõi và thực hiện thao tác "Undo/Redo"

```cpp
struct Action {
    string description;
    Action* prev;
    Action* next;
    Action(const string& desc) : description(desc), prev(nullptr)
       , next(nullptr) {}
};

class EditorHistory {
```

```
    private:
        Action* current;
    public:
        EditorHistory() : current(nullptr) {}

        void performAction(const string& desc) {
            Action* newAction = new Action(desc);
            if (current) {
                Action* temp = current->next;
                while(temp) {
                    Action* toDelete = temp;
                    temp = temp->next;
                    delete toDelete;
                }
                current->next = newAction;
                newAction->prev = current;
            }
            current = newAction;
        }

        bool undo() {
            if (current && current->prev){
                current = current->prev;
                return true;
            }
            return false;
        }

        bool redo() {
            if (current && current->next){
                current = current->next;
                return true;
            }
            return false;
        }
};
```

3 Stack

3.1 Ứng dụng

- **Khử đệ quy bằng Stack**
- **Quản lý thao tác "Back/Forward" trên trình duyệt**

3.2 Giải thích

- **Khử đệ quy bằng Stack**
  - **Kiểm soát rõ ràng bộ nhớ:** Stack cho phép ta kiểm soát và quản lý các "frame" gọi hàm một cách trực tiếp. Điều này hữu ích khi muốn tránh việc sử dụng đệ quy quá sâu, từ đó hạn chế rủi ro tràn ngăn xếp (stack overflow).
  - **Mô phỏng rõ cơ chế LIFO:** Đệ quy tự nhiên xử lý theo nguyên tắc gọi hàm mới và quay trở lại theo thứ tự ngược lại (Last In - First Out), nên sử dụng Stack giúp mô phỏng quá trình này một cách trực quan và rõ ràng.
- **Quản lý thao tác "Back/Forward" trên trình duyệt:**
  - **Di chuyển nhanh chóng theo thứ tự LIFO:** Việc quay lại trang trước (Back) và trang sau (Forward) chính xác theo thứ tự LIFO của Stack giúp ứng dụng đảm bảo hiệu năng cao và thao tác phản hồi nhanh.
  - **Đơn giản trong quản lý trạng thái:** Mỗi thao tác (Back hoặc Forward) chỉ cần pop hoặc push trang hiện tại sang Stack tương ứng, đảm bảo việc quản lý lịch sử duyệt web trở nên trực quan và dễ triển khai.

3.3 Cấu trúc tương ứng

Khử đệ quy bằng Stack với bài toán duyệt cây nhị nhân

```
class NodeOfTree{
    public:
        int data;
        NodeOfTree* left;
        NodeOfTree* right;
        NodeOfTree(int _data = 0){
            this->data = _data;
            this->left = nullptr;
            this->right = nullptr;
        }
};

```

```cpp
template<class T>
class NodeOfStack{
    public:
        T data;
        NodeOfStack<T>* next;

        NodeOfStack (T _data=T()){
            this->data = _data;
            this->next = nullptr;
        }
};

template<class T>
class Stack{
    private:
        NodeOfStack<T>* pHead;

    public:
        Stack(){
            this->pHead = nullptr;
        }

        void push(T X){
            NodeOfStack<T>* newNode = new NodeOfStack<T>(X);
            newNode->next = pHead;
            pHead = newNode;
        }

        bool pop(){
            if (pHead == nullptr)   return false;
            NodeOfStack<T>* p = pHead;
            pHead = pHead->next;
            delete p;
            return true;
        }

        bool empty(){
            return (pHead == nullptr);
        }

        T top(){
            return (pHead->data);
        }
```

```
};

void preOrder(NodeOfTree* root) {
    Stack<NodeOfTree*> st;
    st.push(root);

    while (!st.empty()){
        NodeOfTree* current = st.top();
        cout << current->data << ' ';
        st.pop();
        if (current->right != nullptr)
            st.push(current->right);
        if (current->left != nullptr)
            st.push(current->left);
    }
}
```

Quản lý thao tác "Back/Forward" trên trình duyệt

```
class BrowserHistory {
    private:
        stack<string> backStack;
        stack<string> forwardStack;
        string current;

    public:
        BrowserHistory(const string &homepage) : current(homepage
            ) {}

        void visit(const string &url) {
            backStack.push(current);
            current = url;
            while (!forwardStack.empty())
                forwardStack.pop();
        }

        void back() {
            if (backStack.empty())  return;
            forwardStack.push(current);
            current = backStack.top();
            backStack.pop();
        }

        void forward() {
```

```
            if (forwardStack.empty())    return;
            backStack.push(current);
            current = forwardStack.top();
            forwardStack.pop();
        }

};
```

4 Queue

4.1 Ứng dụng

- Duyệt theo chiều rộng (Breath First Search) trên đồ thị, cây
- Lập lịch CPU

4.2 Giải thích

- **Duyệt theo chiều rộng BFS trên đồ thị, cây**
  - Thuật toán BFS sử dụng Queue để đảm bảo rằng các nút được duyệt theo thứ tự "cách nhau" (theo cấp độ) từ nút bắt đầu. Điều này giúp tìm được đường đi ngắn nhất trên đồ thị không trọng số, là ứng dụng hết sức quan trọng trong các hệ thống định tuyến, mạng xã hội, phân tích đồ thị,...
  - Queue hỗ trợ duy trì thứ tự truy cập "đầu vào – đầu ra (FIFO)", đảm bảo tính chính xác của thuật toán khi duyệt theo từng lớp.
- **Lập lịch cho CPU (Round Robin Scheduling)**
  - **Công bằng và phản ứng nhanh:** Round Robin phân bổ thời gian CPU đều cho tất cả các tiến trình, giúp đảm bảo rằng không tiến trình nào bị "chết đói" (starvation) và phản ứng tốt trong môi trường đa nhiệm.
  - **Đơn giản và hiệu quả:** Sử dụng một queue để duy trì thứ tự xử lý của các tiến trình đảm bảo độ phức tạp thấp cho việc thêm và loại bỏ tiến trình khỏi hàng đợi (O(1) cho các thao tác push/pop).

4.3 Cấu trúc tương ứng

Duyệt theo chiều rộng (Breath First Search) trên đồ thị, cây

```
class NodeOfTree{
    public:
        int data;
```

```
        NodeOfTree* left;
        NodeOfTree* right;
        NodeOfTree(int _data = 0){
            this->data = _data;
            this->left = nullptr;
            this->right = nullptr;
        }
};

template<class T>
class NodeOfQueue{
    public:
        T data;
        NodeOfQueue<T>* next;

        NodeOfQueue (T _data=T()){
            this->data = _data;
            this->next = nullptr;
        }
};

template<class T>
class Queue{
    private:
        NodeOfQueue<T>* pHead;
        NodeOfQueue<T>* pTail;

    public:
        Queue(){
            this->pHead = nullptr;
            this->pTail = nullptr;
        }

        void push(T X){
            NodeOfQueue<T>* newNode = new NodeOfQueue<T>(X);
            if (pHead == nullptr)
                pHead = pTail = newNode;
            else{
                pTail->next = newNode;
                pTail = newNode;
            }
        }

```

```
        bool pop(){
            if (pHead == nullptr)   return false;
            NodeOfQueue<T>* p = pHead;
            pHead = pHead->next;
            if (pHead == nullptr)
                pTail = nullptr;
            delete p;
            return true;
        }

        bool empty(){
            return (pHead == nullptr);
        }

        T front(){
            return (pHead->data);
        }
};

void BFS(NodeOfTree* root) {
    Queue<NodeOfTree*> Q;
    Q.push(root);

    while (!Q.empty()) {
        NodeOfTree* current = Q.front();
        Q.pop();
        cout << current->data << " ";
        if (current->left != nullptr)
            Q.push(current->left);
        if (current->right != nullptr)
            Q.push(current->right);
    }
}
```

Lập lịch cho CPU sử dụng thuật toán Round Robin

```
struct Process {
    int id;
    int burstTime;
};

void roundRobinScheduling(queue<Process>& processQueue, int
   timeQuantum) {
    while(!processQueue.empty()) {
```

```
8           Process current = processQueue.front();
9           processQueue.pop();
10
11          if (current.burstTime > timeQuantum) {
12              cout << "Tien trinh P" << current.id << " duoc CPU xu
                    ly trong "<< timeQuantum <<" s.\n";
13              current.burstTime -= timeQuantum;
14              processQueue.push(current);
15          }
16          else {
17              cout << "Tien trinh P" << current.id << " duoc CPU xu
                    ly trong "  << current.burstTime << " s. Hoan
                   thanh.\n";
18          }
19      }
20  }
```