**­­­TRƯỜNG ĐẠI HỌC**

**CÔNG NGHỆ THÔNG TIN**

🙣🕮🙡

Cấu Trúc Dữ Liệu Và Giải Thuật

**ĐỒ ÁN MÔN HỌC**

Đề tài: Cây biểu thức toán học

(Đã báo cáo)

*GVGD: Huỳnh Thị Thanh Thương*

• Danh sách thành viên:

Lương Quang Huy (nhóm trưởng)

Lê Quang Huy

Nguyễn Tấn Huệ

Nguyễn Khoa Hiếu

 » *Phân công công việc và phần trăm đóng góp của từng thành viên:*

Phân công công việc:

+ Code chính: Lương Quang Huy

+ Code phụ: Nguyễn Tấn Huệ, Nguyễn Khoa Hiếu, Lê Quang Huy

+ Viết tiểu luận: Nguyễn Tấn Huệ, Nguyễn Khoa Hiếu, Lê Quang Huy

+ Soạn slide: Nguyễn Khoa Hiếu, Nguyễn Tấn Huệ, Lê Quang Huy, Lương Quang Huy

+ Tìm kiếm nội dung: Nguyễn Khoa Hiếu, Nguyễn Tấn Huệ, Lê Quang Huy, Lương Quang Huy

+ Trao đổi vấn đề nghiên cứu, tổng hợp nội dung : Nguyễn Khoa Hiếu, Lê Quang Huy, Lương Quang Huy, Nguyễn Tấn Huệ

*-*Mức độ % đóng góp của từng thành viên:

Lê Quang Huy: 100%

Lương Quang Huy: 100%

Nguyễn Tấn Huệ: 100%

Nguyễn Khoa Hiếu: 100%

 ❄***Khái quát Nội dung slide:***

Nêu ra vấn đề và cách giải quyết bài toán, ưu điểm của cách giải quyết này

1. Chuyển biểu thức sang dạng hậu tố

2. Đưa hậu tố vào Cây (BSTree)

3. Tính kết quả biểu thức

4. Ví Dụ minh họa

5.Tổng kết

**Bài toán đặt ra :** **{[r(32^3)-10\*10%13]+6!}**

* Câu hỏi được đặt ra là làm sao để tính toán một bài toán gồm nhiều phép tính và độ ưu tiên phức tạp trên máy ?

**Cách giải quyết:**

B1: Chuyển biểu thức đã cho ban đầu thành dạng Hậu tố (postfix)

B2: Đưa dạng Hậu tố -> BSTree

B3: Tính toán ra kết quả

* Có 2 cách tính toán ra KQ:
* Sử dụng stack
* Sử dụng cấu trúc Cây

**Ưu điểm**

1. Các nodes đều được liên kết với nhau bằng các con trỏ.

🡪Cài đặt sử dụng các con trỏ của các cấu trúc dữ liệu Trees không hề có giới hạn cận trên nào về số lượng các nodes.

2. Lưu trữ các phép toán dưới dạng một hệ thống phân cấp tự nhiên.

🡪 Giúp tìm kiếm và tính toán nhanh hơn.

1. **Chuyển Biểu thức sang hậu tố**

**1.1 *Thứ tự ưu tiên trong biểu thức:***

- Toán tử

- Dấu ngoặc

- Toán hạng

Đầu tiên ta sẽ xét toán tử trước

Có các toán tử bao gồm: + , - , \* , / , % , sqrt , ^ và !

Xét về độ ưu tiên các toán tử:

*“ ^ , sqrt , !* ” > *“ \* , / , %* ” > “ *+ , -* ”

Ta có phương thức lấy độ ưu tiên toán tử như sau:

bool isOperator(char x) //Kiểm tra nó là toán tử

{

return x == '+' || x == '-' || x == '\*' || x == '/' || x == '%' || x == '^' || x == 'r' || x == '!’;

}

int getPriority(char x) //Kiểm tra thứ tự ưu tiên của toán tử

{

if (x == '+' || x == '-') return 1;

if (x == '\*' || x == '/' || x == '%') return 2;

if (x == '^' || x == 'r' || x == '!') return 3;

return 0;

}

Tiếp theo ta sẽ xét tiếp phần dấu ngoặc

Nếu biểu thức có các dấu ngoặc :

ngoặc tròn ( ), ngoặc vuông [ ], ngoặc nhọn { }

Ta thực hiện phép tính theo thứ tự :

( ) → [ ] → { }

Và cuối cùng là toán hạng

Xét các số nguyên dương : {0 1 2 3 4 5 6 7 8 9} Và các số thập phân.

Hàm kiểm tra Toán hạng

bool isOperand(char x) //Kiểm tra nó là toán hạng

{

if ((x >= '0' && x <= ‘9’) || x == ‘.’ )

return true;

return false;

}

* 1. ***Cách chạy của bài toán***

*Khởi tạo:*

+ Result(string): chứa kết quả là biểu thức hậu tố.

+ OperatorStack: làm các bước đệm cho các thao tác xử lý trung gian.

* For each char x in string(Trung tố)

1. IF (X = toán hạng) -> push X vào Result
2. IF (X = ‘( { [ ‘) -> push vào OperatorStack
3. IF (X = ‘ ) } ] ‘ -> lấy các phần tử trong OperatorStack ra và push vào Result đến khi gặp ” ( { [ “ thì dừng.

Xét theo mức độ ưu tiên dấu ngoặc

( ) → [ ] → { }

Sau đó, xóa các ký tự ” ( { [ “ đó khỏi OperatorStack

4. IF (X = toán tử)

A. OperatorStack rỗng -> push X vào trong OperatorStack

B. OperatorStack có phần tử:

a.Đỉnh OperatorStack != Toán Tử -> push X vào OperatorStack

b. Đỉnh OperatorStack == Toán Tử 🡪kiểm tra độ ưu tiên:

-IF (Toán Tử đỉnh OperatorStack < X) -> push X vào OperatorStack

-IF (toán tử đầu OperatorStack > = X)

Bước 1: Lấy phần tử đỉnh OperatorStack push vào Result.

Bước 2: Lặp lại bước 1:

+ OperatorStack rỗng -> dừng lại.

+ Phần tử đỉnh OperatorStack hiện tại != Toán Tử -> dừng, sau đó push X vào OperatorStack.

5. Khi kết thúc duyệt, OperatorStack vẫn còn phần tử -> push hết vào Result.

Ta có biểu thức *{[r(32^3)-10\*10%13]+6!}*

Để chuyển biểu thức trung tố sang hậu tố ta thực hiện các bước như bảng sau:

|  |  |  |
| --- | --- | --- |
| **Duyệt** | **OperatorStack** | **Result** |
| { | { |  |
| [ | { [ |  |
| r | { [ r | 2 |
| ( | { [ r ( | 2 |
| 32 | { [ r ( | 2 32 |
| ^ | { [ r ( ^ | 2 32 |
| 3 | { [ r ( ^ | 2 32 3 |
| ) | { [ r | 2 32 3 ^ |
| - | { [ - | 2 32 3 ^ r |
| 10 | { [ - | 2 32 3 ^ r 10 |
| \* | { [ - \* | 2 32 3 ^ r 10 |

|  |  |  |
| --- | --- | --- |
| **Duyệt** | **OperatorStack** | **Result** |
| 10 | { [ - \* | 2 32 3 ^ r 10 10 |
| % | { [ - % | 2 32 3 ^ r 10 10 \* |
| 13 | { [ - % | 2 32 3 ^ r 10 10 \* 13 |
| ] | { | 2 32 3 ^ r 10 10 \* 13 % - |
| + | { + | 2 32 3 ^ r 10 10 \* 13 % - |
| 6 | { + | 2 32 3 ^ r 10 10 \* 13 % - 6 |
| ! | { + ! | 2 32 3 ^ r 10 10 \* 13 % - 6 1 |
| } |  | 2 32 3 ^ r 10 10 \* 13 % - 6 1 ! + |

Thấy căn (r) , giai thừa (!) thì push căn, giai thừa vào Operatorstack, thêm giả lập số 2( tương ứng với r ) và số 1 (tương ứng với !) vào Result

**2. Đưa hậu tố vào Cây BSTree.**

**2.1. Cài đặt:**

-Tên hàm: kết quả cuối cùng là TNode gốc của cây, ta sử dụng R để lưu trữ, string str là biểu thức ban đầu được nhập vào.

void PostfixtoTree(TNode\*& R, string str)

-Thân hàm: stack để chứa các thay đổi khi liên tục tạo TNode và thay thế TNode.

stack<TNode\*> stackNode

**2.2. Các bước để thực hiện:**

**Bước 1: Chuyển xâu nhập vào từ bàn phím từ dạng Trung tố sang Hậu tố.**

**Bước 2: Chạy duyệt xâu Hậu tố :**

**\*Kiểm tra từng phần tử trong xâu:**

+ Nếu phần tử được xét đến là toán hạng -> Tạo 1 TNode để chứa toán hạng đó, sau đó lưu TNode vừa được tạo này vào trong stackNode.

+ Nếu phần tử được xét đến là toán tử -> Tạo 1 TNode để chứa toán tử đó, Nối TNode toán tử đó với 2 TNode khác ở đỉnh của stackNode theo thứ tự đỉnh đầu tiên bên Phải và đỉnh tiếp theo bên Trái của TNode toán tử. Sau khi thực hiện tuần tự các bước, xóa 2 phần tử ở đỉnh stackNode đó đi. Cuối cùng, đưa TNode toán tử đó vào stackNode.

**Bước 3: Phần tử cuối cùng của stackNode là gốc của Cây biểu thức.**

**3. Tính kết quả biểu thức:**

**3.1. Tính toán biểu thức bằng cây BSTree:**

*Bước 1: Chạy duyệt đệ qui cây BSTree theo thứ tự Node-Left-Right.*

*Bước 2: Kiểm tra các TNode:*

+Nếu phần tử được xét đến là toán hạng -> chuyển phần tử từ dạng xâu sang dạng số sau đó trả về giá trị.

+Nếu phần tử được xét đến là toán tử -> kiểm tra toán tử, cứ mỗi toán tử sẽ ứng với trả về giá trị phép tính của Nhánh trái và Nhánh phải của phần tử.

*Bước 3: Khi chạy đến phần tử cuối cùng, là gốc của cây ta được kết quả cuối cùng là kết quả của phép toán.*

**3.2. Tính toán biểu thức bằng Stack:**

**Bước 1: Chạy duyệt xâu Hậu tố.**

stack<double> stackNumber: chứa các giá trị số và kết quả tính toán**.**

**Bước 2: Kiểm tra các phần tử trong xâu:**

+Nếu phần tử được xét đến là toán hạng -> chuyển phần tử từ dạng xâu sang dạng số sau đó đưa vào stackNumber.

+Nếu phần tử được xét đến là toán tử -> kiểm tra toán tử, tương ứng với phép tính ta thực hiện phép toán của nó với 2 phần tử ở đỉnh stackNumber theo thứ tự, phần tử đỉnh đầu tiên nằm phía bên phải phép tính và phần tử đỉnh kế tiếp nằm phía bên trái phép tính. Sauk hi thực hiện sau các bước tính tiến hành xóa 2 phần tử ở đỉnh stackNumber đi và thêm kết quả mới tính được vào stackNumber.

**Bước 3: Giá trị cuối cùng của stackNumber chính là kết quả của bài toán.**