МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра информационных технологий**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1**

**по дисциплине  
 «ПАРАЛЛЕЛЬНОЕ И НИЗКОУРОВНЕВОЕ ПРОГРАММИРОВАНИЕ»**

Выполнил студент группы 25/2                                       Т. Э. Айрапетов

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

Курс    2

Отчет принял доктор физико-математических наук, профессор                                                                                       А.И. Миков

Краснодар

2022 г.

**Задание**: Дано p процессоров, арифметическое выражение (разность и деление). Найти время, за которое выполняется распараллеленная задача.

**Решение**. В программе строится постфиксная запись выражения, а затем выражение заносится в дерево разбора. В процессе построения возможны случаи плохо сбалансированного дерева, поэтому после построения производится балансировка с учётом порядка действий. Затем на каждом уровне дерева считается количество одновременно выполняемых действий и, в зависимости от количества процессоров, вычисляется время работы.

Текст программы:

#include <iostream>

#include <string>

#include <stack>

#include <vector>

#include <map>

using namespace std;

struct Node

{

string data;

Node\* left, \* right;

Node(string data)

{

this->data = data;

left = right = NULL;

};

Node(string data, Node\* left, Node\* right)

{

this->data = data;

this->left = left;

this->right = right;

};

};

vector<string> splitstr(string str, string deli = " "){

int start = 0;

int end = str.find(deli);

vector<string> res;

while (end != -1) {

res.push\_back(str.substr(start, end - start));

start = end + deli.size();

end = str.find(deli, start);

}

res.push\_back(str.substr(start, end - start));

return res;

}

bool isOperator(string c) {

return (c == "-" || c == "/" || c == "+");

}

int Priority(string c)

{

if (c == "/")

return 2;

if (c == "-" || c == "+")

return 1;

else

return 0;

}

// преобразование в постфиксную запись (для дерева)

vector<string> postfix(string exp) {

vector<string> tokens = splitstr(exp);

vector<string> outputList;

stack<string> s;

for (int i = 0; i < tokens.size(); i++){

if (isdigit(tokens[i][0]))

outputList.push\_back(tokens[i]);

if (tokens[i] == "(")

s.push(tokens[i]);

if (tokens[i] == ")"){

while (!s.empty() && s.top() != "("){

outputList.push\_back(s.top());

s.pop();

}

s.pop();

}

if (tokens[i] == "-" || tokens[i] == "/") {

while (!s.empty() && Priority(s.top()) >= Priority(tokens[i])){

outputList.push\_back(s.top());

s.pop();

}

s.push(tokens[i]);

}

}

while (!s.empty()){

outputList.push\_back(s.top());

s.pop();

}

return outputList;

}

// создает дерево разбора

Node\* create(vector<string> postfixExp){

stack<Node\*> s;

for (auto c : postfixExp){

if (isOperator(c)){

auto x = s.top(); s.pop();

auto y = s.top(); s.pop();

auto node = new Node(c, y, x);

s.push(node);

} else {

s.push(new Node(c));

}

}

return s.top();

}

// считает кол-во уровней в дереве разбора

int maxLevel(Node\* t, int l = 0) {

if (t == NULL)

return l - 1;

return max(maxLevel(t->left, l + 1), maxLevel(t->right, l + 1));

}

// добавляет пробелы вокруг скобок

string norm(string exp) {

for (int i = 0; i < exp.size(); i++) {

if (exp[i] == '(' && exp[i + 1] != ' ')

exp.insert(i + 1, " ");

else if (exp[i] == ')' && exp[i - 1] != ' ')

exp.insert(i, " ");

}

return exp;

}

// попытка сбалансировать дерево

void balance(Node\* r) {

auto p1 = r, p2 = r->left;

while (p2->left != NULL && !isdigit(p2->left->data[0]) && abs(maxLevel(r->left) - maxLevel(r->right)) >= 1) {

if (p1->data[0] == '-' && p2->data[0] == '-') {

p1->left = p2->left;

p2->left = p2->right;

p2->right = p1->right;

p1->right = p2;

p2->data = "+";

if (p1 == r) {

p1 = r->left;

p2 = r->left->left;

} else {

p1 = r;

p2 = r->left;

}

}

else {

p1 = p2;

p2 = p2->left;

}

}

}

Node\* root;

// вывод дерева разбора

void Display(int level, Node\* ptr = root)

{

int i;

if (ptr != NULL)

{

Display(level + 1, ptr->right);

cout << endl;

if (ptr == root)

cout << "Root -> ";

for (i = 0; i < level && ptr != root; i++)

cout << " ";

cout << ptr->data;

Display(level + 1, ptr->left);

}

}

// вывод выражения в инфиксной записи

void inorder(Node\* root)

{

if (root == nullptr)

return;

if (isOperator(root->data))

cout << "(";

inorder(root->left);

cout << root->data;

inorder(root->right);

if (isOperator(root->data))

cout << ")";

}

map<int, int> lvs;

void initMap(Node\* r, int l=0) {

if (isdigit(r->data[0]))

return;

if (lvs.find(l) != lvs.end())

lvs[l]++;

else

lvs[l] = 1;

initMap(r->left, l + 1);

initMap(r->right, l + 1);

}

int checkTime(string exp, int p) {

exp = norm(exp);

int w = 0;

for (auto i : exp) {

if (i == '-' || i == '/')

w++;

}

auto r = create(postfix(exp));

inorder(r);

cout << " - Исходное выражение\n";

balance(r);

inorder(r);

cout << " - Распараллеленное выражение\n";

root = r;

Display(0);

auto t = maxLevel(r);

//int a = t + (w - t) / p;

//cout << endl << t << " + (" << w << " - " << t << ") / " << p << " = " << a << endl;

initMap(r);

int res = 0;

for (int i = t - 1; i >= 0; i--) {

if(lvs[i] > p)

res += lvs[i] / p + lvs[i] % p;

else

res ++;

}

return res;

// используется Лемма Брента

}

int main(){

setlocale(LC\_ALL, "rus");

string exp = "(22 - 3 - (3 / 2)) / (4 - 7) - 2 - 4 - 5 / 4";

//string exp = "(2 - 3) - (2 - 3)";

//string exp = "3 - 4 - 5 - 4 - 5 - 4 - 3 - 3";

//string exp = "3 - 3 - 1 - 2 - 4";

int p = 2;

cout << endl << " Answer : " << checkTime(exp, p);

}

Результаты тестов:

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

**Вывод**:

Нужно распараллеливать задачу если это возможно.