Standard Code Library

Your TeamName

Your School

December 16, 2021

Contents

编译原理实验 1

代码

- 需要 C++11

```cpp
#include<bits/stdc++.h>

using namespace std;

enum {
    Comment = 0, Main, If, Then, While, Do, Static, Int, Double, Struct,
    Break, Else, Long, Switch, Case, Typedef, Char, Return, Const, Float,
    Short, Continue, For, Void, Sizeof, ID, NUM, ADD, SUB, MUL,
    DIV, COLON, DEFINE, LT, NE, LE, GT, GE, EQ, Default,
    Do1, SEMI, LB, RB
};

char name[][10] = {
        "Comment", "Main", "If", "Then", "While", "Do", "Static", "Int", "Double", "Struct",
        "Break", "Else", "Long", "Switch", "Case", "Typedef", "Char", "Return", "Const", "Float",
        "Short", "Continue", "For", "Void", "Sizeof", "ID", "NUM", "ADD", "SUB", "MUL",
        "DIV", "COLON", "DEFINE", "LT", "NE", "LE", "GT", "GE", "EQ", "Default",
        "Do1", "SEMI", "LB", "RB"
};


struct Token {
    int syn = 0;
    string token;
    int val = 0;

    void print() const {
        if (syn == -2) {
            printf("<%d,%s>\n", syn, token.c_str());
            return;
        }
        if (syn == NUM) {
            printf("<%d,", syn);
            for (int i = 31; i >= 0; --i) {
                printf("%d", (val >> i) & 1);
            }
            printf("b>\n");
        }
//        else if (syn == ID)
//            printf("<%d,%s>\n", syn, token.c_str());
        else
            printf("<%d,%s>\n", syn, token.c_str());
    }
};


int identifierNum = RB + 1;
unordered_map<string, int> identifier;

void initIdentifier() {
    identifier["#"] = Comment;
    identifier["main"] = Main;
    identifier["if"] = If;
    identifier["then"] = Then;
    identifier["while"] = While;
    identifier["do"] = Do;
    identifier["static"] = Static;
    identifier["int"] = Int;
    identifier["double"] = Double;
    identifier["struct"] = Struct;
    identifier["break"] = Break;
    identifier["else"] = Else;
    identifier["long"] = Long;
    identifier["switch"] = Switch;
    identifier["case"] = Case;
    identifier["typedef"] = Typedef;
    identifier["char"] = Char;
    identifier["return"] = Return;
    identifier["const"] = Const;
    identifier["float"] = Float;
    identifier["short"] = Short;
    identifier["continue"] = Continue;
    identifier["for"] = For;
    identifier["void"] = Void;
    identifier["sizeof"] = Sizeof;
    identifier["default"] = Default;

}

char *p, *lastp;
int line = 0;


Token *next() {
    auto *tk = new Token;
    tk->token = *p;
    while (*p) {//判断字符串是否结束
        if (*p == '\n') {
            printf("%d: ", line, (int) (p - lastp), lastp);
            while (lastp < p) putchar(*(lastp++));
            printf("\n\n");
            ++p;
            lastp = p;
            line = line + 1;
        } else if (isalpha(*p)) {
            tk->syn = ID;
            tk->token = "";
            while (isalpha(*p) || isdigit(*p)) {
                tk->token += *p;
                ++p;
            }
            if (identifier[tk->token] != 0) {
                if (identifier[tk->token] <= RB) {
                    tk->syn = identifier[tk->token];
                } else {
                    tk->val = identifier[tk->token];
                }
            } else {
                tk->val = identifier[tk->token] = ++identifierNum;
            }
            return tk;
        } else if (*p >= '0' && *p <= '9') {
            tk->syn = NUM;
            tk->val = 0;
            tk->token = "";
            while (*p >= '0' && *p <= '9' || isalpha(*p)) {
                if (isalpha(*p)) {
                    tk->syn = -2;
                } else {
                    tk->val = tk->val * 10 + *p - '0';
                }
                tk->token += *p;
                ++p;
            }
            return tk;
        } else {
            tk->token = *p;
            if (*p == '+') {
                ++p;
                tk->syn = ADD;
                return tk;
            } else if (*p == '-') {
                ++p;
                tk->syn = SUB;
                return tk;
            } else if (*p == '*') {
                ++p;
                tk->syn = MUL;
                return tk;
            } else if (*p == '/') {
                ++p;
                if (*p == '*') {
                    while (true) {
                        ++p;
                        if (!*p) {
                            tk->syn = -1;
                            return tk;
                        }
                        while (*p && *p != '*')
                            ++p;
                        if (*p == '*' && *(p + 1) == '/') {
                            p += 2;
                            break;
                        }
                    }
                    continue;
                } else {
                    tk->syn = DIV;
                    return tk;
                }
            } else if (*p == ':') {
                ++p;
                if (*p == '=') {
                    ++p;
                    tk->token += *p;
                    tk->syn = DEFINE;
                } else {
                    tk->syn = COLON;
                }
                return tk;
            } else if (*p == '<') {
                ++p;
                if (*p == '>') {
                    ++p;
                    tk->token += *p;
                    tk->syn = NE;
                } else if (*p == '=') {
                    ++p;
                    tk->token += *p;
                    tk->syn = LE;
                } else {
                    tk->syn = LT;
                }
                return tk;
            } else if (*p == '>') {
                ++p;
                if (*p == '=') {
                    ++p;
                    tk->token += *p;
                    tk->syn = GE;
                } else {
                    tk->syn = GT;
                }
                return tk;
            } else if (*p == '=') {
                ++p;
                tk->syn = EQ;
                return tk;
            } else if (*p == ';') {
                ++p;
                tk->syn = SEMI;
                return tk;
            } else if (*p == '(') {
                ++p;
                tk->syn = LB;
                return tk;
            } else if (*p == ')') {
                ++p;
                tk->syn = RB;
                return tk;
            } else {
                if (!isblank(*p)) {
                    tk->syn = -2;
                    tk->token = "";
                    if (*p && *p != '\n' && !isblank(*p)) {
                        tk->token += *p;
                        ++p;
                    }
                    return tk;
                }
                ++p;
            }
        }
    }
    printf("%d: ", line);
    while (lastp < p) putchar(*(lastp++));
    tk->syn = -1;
    return tk;
}

char s[1000000];

int main() {
    initIdentifier();
    FILE *f = fopen("1.txt", "r");
    fread(s, 1000000, 1000000, f);
    p = lastp = s;
    Token *tk = next();
    while (tk->syn != -1) {
        tk->print();
        tk = next();
    }
    return 0;
}
```

编译原理实验 2

代码

```
#include <bits/stdc++.h>
using namespace std;

typedef string grammarElement;
typedef pair<char, char> pcc;
map<char, vector<grammarElement>> grammarSet, grammarTmp;
map<char, int> flag;
map<char, set<char>> firstSet;
map<char, set<char>> followSet;
map<pcc, set<string>> M;
char S;
int err = 0;

void dfs(char ch)
{
    // cout << "dfs" << ch << "\n";
    flag[ch] = 2; // working
    for (auto str : grammarSet[ch])
    {
        if (isupper(str[0]))
        {
            if (flag[str[0]] == 1)
            {
                for (auto i : firstSet[str[0]])
                {
                    if (i != '%')
                    {
                        firstSet[ch].insert(i);
```

```cpp
                    }
                }
            }
            else if (flag[str[0]] == 0)
            {
                dfs(str[0]);
                for (auto i : firstSet[str[0]])
                {
                    if (i != '%')
                    {
                        firstSet[ch].insert(i);
                    }
                }
            }
            else
            {
                puts("error: exist left recursion");
                err = 1;
                return;
            }
        }
        else
        {
            firstSet[ch].insert(str[0]);
        }
    }
    flag[ch] = 1; // finish
    // cout << "outdfs" << ch << "\n";
}

char nm[10000];
int e = 0;
vector<int> eg[10000];

void dfs2(string perfix, int x, bool isend)
{
    cout << perfix;
    cout << (isend ? "└──" : "├──");
    cout << nm[x] << endl;
    ;
    for (int i = 0; i < eg[x].size(); ++i)
        dfs2(perfix + (isend ? "    " : "│   "), eg[x][i], i == eg[x].size() - 1);
}

char nt = 'H';

int main()
{
    freopen("ll1test.in", "r", stdin);
    int n;
    cin >> n;
    for (int i = 0; i < n; ++i)
    {
        string x, y;
        cin >> x >> y;
        if (x.length() > 1 || !isupper(x[0]))
        {
            puts("input error: x must be in [A-Z]");
            --i;
            continue;
        }
        if (!grammarSet[x[0]].empty())
        {
            puts("warning: redefining");
        }
        if (i == 0)
        {
            S = x[0];
        }
        err = 0;
        string tmp = "";
        for (auto ch : y)
        {
            if (ch == '|' || ch == '#')
            {
                if (ch == '#')
                {
                    err = 1;
                    grammarSet[x[0]].clear();
                    break;
                }
                grammarSet[x[0]].push_back(tmp);
                tmp = "";
            }
            else
            {
                tmp += ch;
            }
        }
        grammarSet[x[0]].push_back(tmp);
        if (err)
        {
            puts("error: do not use #");
            --i;
            continue;
        }
    }

    for (auto &i : grammarSet)
    {

        for (const auto &j : grammarSet)
        {
            if (j.first == i.first)
            {
                int flag = 0;
                for (int k = 0; k < i.second.size(); ++k)
                {
                    if (i.second[k][0] == i.first)
                    {
                        flag = 1;
                        break;
                    }
                }

                if (!flag)
                    continue;
                cout << "let " << nt << " as " << i.first << "'" << endl;
                auto tmp = i.second;
                i.second.clear();
                for (auto k : tmp)
                {
                    if (k[0] != i.first)
                    {
                        i.second.push_back(k + nt);
                    }
                }
                grammarTmp[nt].push_back("%");
                for (auto k : tmp)
                {
                    if (k[0] == i.first)
                    {
                        grammarTmp[nt].push_back(k.substr(1) + nt);
                    }
                }
                ++nt;
            }
            else if (j.first < i.first)
            {
                for (int k = 0; k < i.second.size();)
                {
                    if (i.second[k][0] == j.first)
                    {
                        for (auto t : j.second)
                            i.second.push_back(t + i.second[k].substr(1));
                        i.second.erase(i.second.begin() + k);
                    }
                    else
                    {
                        ++k;
                    }
                }
                int flag = 0;
                for (int k = 0; k < i.second.size(); ++k)
                {
                    if (i.second[k][0] == i.first)
                    {
                        flag = 1;
                        break;
                    }
                }

                if (!flag)
                    continue;
                cout << "let " << nt << " as " << i.first << "'" << endl;
                auto tmp = i.second;
                i.second.clear();
                for (auto k : tmp)
                {
                    if (k[0] != i.first)
                    {
                        i.second.push_back(k + nt);
                    }
                }
                grammarTmp[nt].push_back("%");
                for (auto k : tmp)
                {
                    if (k[0] == i.first)
                    {
                        grammarTmp[nt].push_back(k.substr(1) + nt);
                    }
                }
                ++nt;
            }
        }
    }
    for (auto i : grammarTmp)
    {
        grammarSet[i.first] = i.second;
    }
    for (auto i : grammarSet)
    {
        cout << i.first << "->";
        for (auto j : i.second)
        {
            cout << j << "|";
        }
        cout << endl;
    }

    for (auto x : grammarSet)
    {
        // cout<<x.first<<endl;
        // for (auto i : grammarSet[x.first])
        // {
        //     cout << x.first << "->" << i << "\n";
        // }
        if (!flag[x.first])
        {
            dfs(x.first);
        }
    }
    for (auto x : firstSet)
    {
        cout << "first[" << x.first << "]={";
        for (auto i : x.second)
        {
            cout << i << ",";
        }
        cout << "}\n";
    }
    followSet[S].insert('#');
    int cflag = 1;
    while (cflag)
    {
        cflag = 0;
        for (auto x : grammarSet)
        {
            for (auto str : x.second)
            {
                for (int i = 0; i < str.size(); ++i)
                {
                    if (isupper(str[i]))
                    {
                        int size = followSet[str[i]].size();
                        if (i + 1 < str.size())
                        {
                            if (isupper(str[i + 1]))
                            {
                                for (auto j : firstSet[str[i + 1]])
                                {
                                    if (j != '%')
                                    {
                                        followSet[str[i]].insert(j);
                                    }
                                }
                                if (firstSet[str[i + 1]].find('%') != firstSet[str[i + 1]].end())
                                {
                                    for (auto j : followSet[str[i + 1]])
                                    {
                                        if (j != '%')
                                        {
                                            followSet[str[i]].insert(j);
                                        }
                                    }
                                }
                            }
                            else
                            {
                                if (str[i + 1] != '%')
                                {

                                    followSet[str[i]].insert(str[i + 1]);
                                }
                            }
                        }
                        else
                        {
                            for (auto j : followSet[x.first])
                            {
                                if (j != '%')
                                {
                                    followSet[str[i]].insert(j);
                                }
                            }
                        }
                        if (followSet[str[i]].size() > size)
                        {
                            cflag = 1;
                        }
                    }
                }
            }
        }
    }
    for (auto x : followSet)
    {
        if (!isupper(x.first))
        {
            continue;
        }
        cout << "follow[" << x.first << "]={";
        for (auto i : x.second)
        {
            cout << i << ",";
        }
        cout << "}\n";
    }

    for (auto x : grammarSet)
    {
        for (auto str : x.second)
        {
            if (isupper(str[0]))
            {
                for (auto i : firstSet[str[0]])
                {
                    if (!isupper(i) && i != '%')
                    {
                        M[pcc(x.first, i)].insert(str);
                    }
                }
            }
            else if (str[0] != '%')
            {
                M[pcc(x.first, str[0])].insert(str);
            }
            if (str[0] == '%' || firstSet[str[0]].find('%') != firstSet[str[0]].end())
            {
                for (auto i : followSet[x.first])
                {
                    M[pcc(x.first, i)].insert("%");
                }
            }
            if (firstSet[x.first].find('%') != firstSet[x.first].end() && followSet[x.first].find('#') !=
 ↪  followSet[x.first].end())
            {
                M[pcc(x.first, '#')].insert("%");
            }
        }
    }
    set<char> row, col;
    for (auto i : M)
    {
        row.insert(i.first.first);
        col.insert(i.first.second);
    }
    cout << setiosflags(ios::left);
    cout << setw(10) << "";
    for (auto i : col)
    {
        cout << setw(10) << i;
    }
    cout << "\n";
    for (auto i : row)
    {
        cout << setw(10) << i;
        for (auto j : col)
        {
            string tmp;
            for (auto k : M[pcc(i, j)])
            {
                if (k.back() == '%' && M[pcc(i, j)].size() > 1)
                {
                    continue;
                }
                tmp += k + ";";
            }
            cout << setw(10) << (tmp.size() ? i + ("->" + tmp) : "");
        }
        cout << "\n";
    }
    string now;
    cin >> now;
    {
        now += "#";
        int it = 0;
        stack<char> stk;
        stk.push(0);
        stk.push(1);
        int flag = 1;
        nm[0] = '#';
        nm[++e] = S;

        while (flag && stk.size())
        {
            auto tmp = stk;
            string out = "";
            int xx = stk.top();
            char x = nm[stk.top()];
            stk.pop();
            while (!tmp.empty())
            {
                out = nm[tmp.top()] + (out);
                tmp.pop();
            }
            cout << setw(30) << out << "\t" << now.substr(it) << endl;
            if (!isupper(x))
            {
                if (it < now.length() && x == now[it])
                {
                    if (it + 1 < now.length())
                        ++it;
                }
                else
                {
                    puts("error 1");
                    return 0;
                }
            }
            else if (x == '#')
            {
                if (now[it] == '#')
                {
                    flag = false;
                }
                else
                {
                    puts("error 2");
                    return 0;
                }
            }
            else if (M[pcc(x, now[it])].size() != 0)
            {
                for (auto i : M[pcc(x, now[it])])
                {
                    if (i != "%")
                    {
                        for (int j = i.length() - 1; j >= 0; --j)
                        {
                            stk.push(++e);
                            nm[e] = i[j];
                            eg[xx].push_back(e);
                        }
                    }
                }
            }
            else
            {
                puts("error 3");
                return 0;
            }
        }
        puts("success");
    }
    dfs2(" ", 1, 1);
    return 0;
}
```

编译原理实验 3

代码

```
class Rule(object):
    def __init__(self):
        self.left = ""
        self.right = []

VT = []
VN = []
Rules = []
FirstVT = []
LastVT = []
rule_list = []
OG = []
og_stack = []

def create_rule_list():
    for i in range(0, len(Rules)):
        Rules[i] = Rules[i].replace(' ', '')
        rule = Rule()
        rule_list.append(rule)
    for j in range(0, len(Rules)):
        arrow_pos = Rules[j].find('-')
        rule_list[j].left = Rules[j][0:arrow_pos]
        rule_list[j].right = list(Rules[j][arrow_pos + 2:])
        while "'" in rule_list[j].right:
            pos = rule_list[j].right.index("'")
            new_sym = "".join(rule_list[j].right[pos - 1: pos + 1])
            del rule_list[j].right[pos]
            del rule_list[j].right[pos - 1]
            if new_sym not in rule_list[j].right:
                rule_list[j].right.append(new_sym)

def identify_vt_and_vn():
    for i in range(0, len(rule_list)):
        if rule_list[i].left not in VN:
            VN.append(rule_list[i].left)
        for j in range(len(rule_list[i].right)):
            if rule_list[i].right[j].isupper():
                if rule_list[i].right[j] not in VN:
                    VN.append(rule_list[i].right[j])
            elif rule_list[i].right[j] != 'ε' and "'" not in rule_list[i].right[j]:
                if rule_list[i].right[j] not in VT:
                    VT.append(rule_list[i].right[j])
            elif "'" in rule_list[i].right[j]:
                if rule_list[i].right[j] not in VN:
                    VN.append(rule_list[i].right[j])
    VT.append('#')

def gen_firstvt(ch):
    for i in range(len(rule_list)):
        if rule_list[i].left == ch:
            # 形如 U -> b... 之类的规则, 将 b 加入 U 的 FirstVT 集
            if rule_list[i].right[0] in VT:
                if rule_list[i].right[0] not in FirstVT[VN.index(ch)]:
```

```python
                    FirstVT[VN.index(ch)].append(rule_list[i].right[0])
            # 形如 U -> Vb... 之类的规则, 将 b 加入 U 的 FirstVT 集
            elif len(rule_list[i].right) > 1 and rule_list[i].right[1] in VT:
                if rule_list[i].right[1] not in FirstVT[VN.index(ch)]:
                    FirstVT[VN.index(ch)].append(rule_list[i].right[1])
            # 形如 U -> V... 的规则, 将 V 的 FirstVT 集里的元素加入 U 的 FirstVT 集
            if rule_list[i].right[0] in VN:
                if not FirstVT[VN.index(rule_list[i].right[0])]:
                    gen_firstvt(rule_list[i].right[0])
                for c in FirstVT[VN.index(rule_list[i].right[0])]:
                    if c not in FirstVT[VN.index(ch)]:
                        FirstVT[VN.index(ch)].append(c)

def gen_lastvt(ch):
    for i in range(len(rule_list)):
        if rule_list[i].left == ch:
            if rule_list[i].right[-1] in VT:
                if rule_list[i].right[-1] not in LastVT[VN.index(ch)]:
                    LastVT[VN.index(ch)].append(rule_list[i].right[-1])
            elif len(rule_list[i].right) > 1 and rule_list[i].right[-2] in VT and rule_list[i].right[-1] in VN:
                if rule_list[i].right[-2] not in LastVT[VN.index(ch)]:
                    LastVT[VN.index(ch)].append(rule_list[i].right[-2])
            if rule_list[i].right[-1] in VN:
                if not LastVT[VN.index(rule_list[i].right[-1])]:
                    gen_lastvt(rule_list[i].right[-1])
                for c in LastVT[VN.index(rule_list[i].right[-1])]:
                    if c not in LastVT[VN.index(ch)]:
                        LastVT[VN.index(ch)].append(c)

def create_og():
    for i in range(len(rule_list)):
        for j in range(0, len(rule_list[i].right) - 1):
            if rule_list[i].right[j] in VT and rule_list[i].right[j+1] in VT:
                OG[VT.index(rule_list[i].right[j])][VT.index(rule_list[i].right[j+1])] = '='
            if j < len(rule_list[i].right) - 2 and rule_list[i].right[j] in VT and rule_list[i].right[j+2] in VT:
                OG[VT.index(rule_list[i].right[j])][VT.index(rule_list[i].right[j+2])] = '='
            if rule_list[i].right[j] in VT and rule_list[i].right[j+1] in VN:
                for c in FirstVT[VN.index(rule_list[i].right[j+1])]:
                    OG[VT.index(rule_list[i].right[j])][VT.index(c)] = '<'
            if rule_list[i].right[j] in VN and rule_list[i].right[j+1] in VT:
                for c in LastVT[VN.index(rule_list[i].right[j])]:
                    OG[VT.index(c)][VT.index(rule_list[i].right[j+1])] = '>'
    for c in FirstVT[VN.index(rule_list[0].left)]:
        OG[VT.index('#')][VT.index(c)] = '<'
    for c in LastVT[VN.index(rule_list[0].left)]:
        OG[VT.index(c)][VT.index('#')] = '>'

def og():
    with open('/home/aquawcac/code/OG/src.txt', 'r', encoding='utf-8') as src_file:
        src = src_file.readlines()
    for i in range(len(src)):
        flag = False
        og_stack = []
        src[i] = src[i].replace('\n', '')
        current = 0
        pos = 1
        og_stack.append('#')
        while current != len(src[i]):
            a = src[i][current]
            s = og_stack[pos-1]
            print('%60s'%og_stack,'%20s'%a,'%20s'%src[i][current:],pos)
            if s in VT:
                j = pos
            else:
                j = pos - 1
            while pos != 2 or a != '#':
                if OG[VT.index(s)][VT.index(a)] == '>':
                    while True:
                        q = s
                        j = j - 1
                        s = og_stack[j-1]
                        if s not in VT:
                            j = j - 1
                            s = og_stack[j-1]
                        if OG[VT.index(s)][VT.index(q)] == '<':
                            pos = j + 1
                            if pos == len(og_stack):
                                og_stack[pos-1] = 'N'
                            else:
                                while pos - 1 != len(og_stack):
                                    og_stack.pop()
                                og_stack.append('N')
                            break
                else:
                    og_stack.append(a)
                    pos = pos + 1
                    current = current + 1
                    break
            if pos == 2:
                if a=='#':
                    flag = True
                    break
        with open('/home/aquawcac/code/OG/output.txt', 'a', encoding='utf-8') as out_file:
            if flag:
                out_file.write('%s 合法\n' % src[i])
            else:
                out_file.write('%s 不合法\n' % src[i])

def print_vt():
    with open('/home/aquawcac/code/OG/set.txt', 'w', encoding='utf-8') as set_file:
        set_file.write("FirstVT\n")
        for k in range(len(VN)):
            set_file.write("%3s:" % VN[k])
            for p in FirstVT[k]:
                set_file.write("%s" % p)
            set_file.write("\n")
        set_file.write("LastVT\n")
        for m in range(len(VN)):
            set_file.write("%3s:" % VN[m])
            for n in LastVT[m]:
                set_file.write("%s" % n)
            set_file.write("\n")

def print_og():
    with open('/home/aquawcac/code/OG/OG.txt', 'w', encoding='utf-8') as chart_write:
        # chart_write.write('生成的优先矩阵如下\n')
        chart_write.write("%3s" % '%')
        for c in VT:
            chart_write.write("%3s" % c)
        chart_write.write("\n")
        for i in range(len(OG)):
            chart_write.write("%3s" % VT[i])
            for j in range(len(OG[i])):
                chart_write.write("%3s" % OG[i][j])
            chart_write.write("\n")

if __name__ == '__main__':
    with open('/home/aquawcac/code/OG/output.txt', 'w', encoding='utf-8') as out_file:
        pass
    with open('/home/aquawcac/code/OG/rule.txt', 'r', encoding='utf-8') as rule_file:
        Rules = rule_file.readlines()
        for i in range(len(Rules)):
            Rules[i] = Rules[i].replace('\n', '')
    create_rule_list()
    identify_vt_and_vn()
    for j in range(len(VN)):
        FirstVT.append([])
        LastVT.append([])
    OG = [[0 for col in range(len(VT))]for row in range(len(VT))]
    for k in range(len(VN)):
        gen_firstvt(VN[k])
    for p in range(len(VN)):
        gen_lastvt(VN[p])
    print_vt()
    create_og()
    print_og()
    og()
```

数据结构

ST 表

- 二维

```cpp
int f[maxn][maxn][10][10];
inline int highbit(int x) { return 31 - __builtin_clz(x); }
inline int calc(int x, int y, int xx, int yy, int p, int q) {
    return max(
        max(f[x][y][p][q], f[xx - (1 << p) + 1][yy - (1 << q) + 1][p][q]),
        max(f[xx - (1 << p) + 1][y][p][q], f[x][yy - (1 << q) + 1][p][q])
    );
}
void init() {
    FOR (x, 0, highbit(n) + 1)
    FOR (y, 0, highbit(m) + 1)
        FOR (i, 0, n - (1 << x) + 1)
        FOR (j, 0, m - (1 << y) + 1) {
            if (!x && !y) { f[i][j][x][y] = a[i][j]; continue; }
            f[i][j][x][y] = calc(
                i, j,
                i + (1 << x) - 1, j + (1 << y) - 1,
                max(x - 1, 0), max(y - 1, 0)
            );
        }
}
inline int get_max(int x, int y, int xx, int yy) {
    return calc(x, y, xx, yy, highbit(xx - x + 1), highbit(yy - y + 1));
}
```

数学

类欧几里得

- $m = \lfloor \frac{an+b}{c} \rfloor$.
- $f(a,b,c,n) = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor$: 当 $a \ge c$ or $b \ge c$ 时，$f(a,b,c,n) = (\frac{a}{c})n(n+1)/2 + (\frac{b}{c})(n+1) + f(a \bmod c, b \bmod c, c, n)$；否则 $f(a,b,c,n) = nm - f(c, c-b-1, a, m-1)$。
- $g(a,b,c,n) = \sum_{i=0}^{n} i \lfloor \frac{ai+b}{c} \rfloor$: 当 $a \ge c$ or $b \ge c$ 时，$g(a,b,c,n) = (\frac{a}{c})n(n+1)(2n+1)/6 + (\frac{b}{c})n(n+1)/2 + g(a \bmod c, b \bmod c, c, n)$；否则 $g(a,b,c,n) = \frac{1}{2}(n(n+1)m - f(c, c-b-1, a, m-1) - h(c, c-b-1, a, m-1))$。
- $h(a,b,c,n) = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor^2$: 当 $a \ge c$ or $b \ge c$ 时，$h(a,b,c,n) = (\frac{a}{c})^2 n(n+1)(2n+1)/6 + (\frac{b}{c})^2(n+1) + (\frac{a}{c})(\frac{b}{c})n(n+1) + h(a \bmod c, b \bmod c, c, n) + 2(\frac{a}{c})g(a \bmod c, b \bmod c, c, n) + 2(\frac{b}{c})f(a \bmod c, b \bmod c, c, n)$；否则 $h(a,b,c,n) = nm(m+1) - 2g(c, c-b-1, a, m-1) - 2f(c, c-b-1, a, m-1) - f(a,b,c,n)$。

图论

LCA

- 倍增

```cpp
void dfs(int u, int fa) {
    pa[u][0] = fa; dep[u] = dep[fa] + 1;
    FOR (i, 1, SP) pa[u][i] = pa[pa[u][i - 1]][i - 1];
    for (int& v: G[u]) {
        if (v == fa) continue;
        dfs(v, u);
    }
}
```

```

int lca(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    int t = dep[u] - dep[v];
    FOR (i, 0, SP) if (t & (1 << i)) u = pa[u][i];
    FORD (i, SP - 1, -1) {
        int uu = pa[u][i], vv = pa[v][i];
        if (uu != vv) { u = uu; v = vv; }
    }
    return u == v ? u : pa[u][0];
}
```

计算几何

二维几何：点与向量

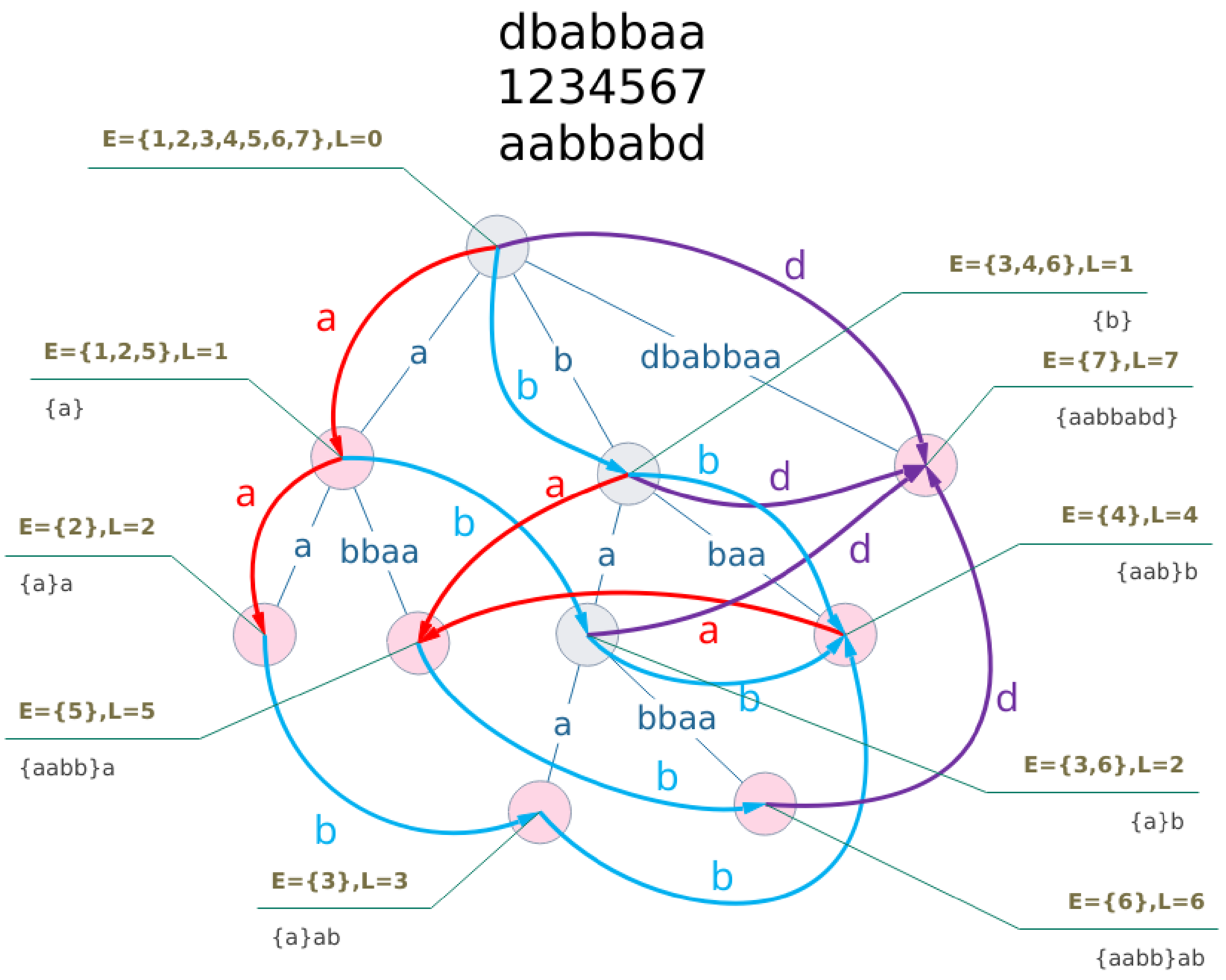
```
#define y1 yy1
#define nxt(i) ((i + 1) % s.size())
typedef double LD;
const LD PI = 3.14159265358979323846;
const LD eps = 1E-10;
int sgn(LD x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }
struct L;
struct P;
typedef P V;
struct P {
    LD x, y;
    explicit P(LD x = 0, LD y = 0): x(x), y(y) {}
    explicit P(const L& l);
};
struct L {
    P s, t;
    L() {}
    L(P s, P t): s(s), t(t) {}
};

P operator + (const P& a, const P& b) { return P(a.x + b.x, a.y + b.y); }
P operator - (const P& a, const P& b) { return P(a.x - b.x, a.y - b.y); }
P operator * (const P& a, LD k) { return P(a.x * k, a.y * k); }
P operator / (const P& a, LD k) { return P(a.x / k, a.y / k); }
inline bool operator < (const P& a, const P& b) {
    return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0);
}
bool operator == (const P& a, const P& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y); }
P::P(const L& l) { *this = l.t - l.s; }
ostream &operator << (ostream &os, const P &p) {
    return (os << "(" << p.x << "," << p.y << ")");
}
istream &operator >> (istream &is, P &p) {
    return (is >> p.x >> p.y);
}

LD dist(const P& p) { return sqrt(p.x * p.x + p.y * p.y); }
LD dot(const V& a, const V& b) { return a.x * b.x + a.y * b.y; }
LD det(const V& a, const V& b) { return a.x * b.y - a.y * b.x; }
LD cross(const P& s, const P& t, const P& o = P()) { return det(s - o, t - o); }
// --------------------------------------------
```

字符串

后缀自动机



杂项

STL

- copy

```
template <class InputIterator, class OutputIterator>
  OutputIterator copy (InputIterator first, InputIterator last, OutputIterator result);
```