S. B. JAIN INSTITUTE OF TECHNOLOGY, MANAGEMENT & RESEARCH, NAGPUR.

Practical No. 7

Aim: Develop a program to detect common subexpression in three address code using data structure DAG.

Name of Student: Shrutika Pradeep Bagdi

Roll No.: CS22130

Semester/Year: 6th Semester/3rd Year

Academic Session: 2024-25

Date of Performance:

Date of Submission:

AIM: Develop a program to detect common sub-expression in three address code using data structure DAG.

OBJECTIVE / EXPECTED LEARNING OUTCOME:

The objectives and expected learning outcome of this practical are:

- To illustrate the use of data structure Directed Acyclic Graph.
- To use DAG for code optimization of common subexpression elimination.

HARDWARE AND SOFTWARE REQUIRMENTS:

Hardware Requirement:

- Processor: Dual Core
- RAM: 1GB
- Hard Disk Drive: > 80 GB

Software Requirement:

- GNU C compiler

THEORY:

1) Introduction of DAG

2) Common Subexpression in Code optimization

3) Use of DAG in detecting common subexpression with an example

4) Limitations of DAG in detecting common subexpression elimination

CODE:

csc15@linux-p2-1272il: ~/CS22130

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_INSTRUCTIONS 100
#define MAX_OPERANDS 3
#define MAX_OPERAND_SIZE 20

typedef struct DAGNode {
    char operation[MAX_OPERAND_SIZE];
    char operands[MAX_OPERANDS][MAX_OPERAND_SIZE];
    struct DAGNode *left;
    struct DAGNode *right;
    int id;
} DAGNode;

DAGNode* dagNodes[MAX_INSTRUCTIONS];
int dagNodeCount = 0;

typedef struct Instruction {
    char result[MAX_OPERAND_SIZE];
    char operand1[MAX_OPERAND_SIZE];
    char operand2[MAX_OPERAND_SIZE];
    char operation[MAX_OPERAND_SIZE];
} Instruction;

int findInDAG(char* op, char* operand1, char* operand2) {
    for (int i = 0; i < dagNodeCount; i++) {
        if (strcmp(dagNodes[i]->operation, op) == 0 &&
            strcmp(dagNodes[i]->operands[0], operand1) == 0 &&
            strcmp(dagNodes[i]->operands[1], operand2) == 0) {
            return dagNodes[i]->id;
        }
    }
    return -1;
}
```

```
int createDAGNode(char* op, char* operand1, char* operand2) {
    DAGNode* newNode = (DAGNode*)malloc(sizeof(DAGNode));
    strcpy(newNode->operation, op);
    strcpy(newNode->operands[0], operand1);
    strcpy(newNode->operands[1], operand2);
    newNode->id = dagNodeCount++;
    dagNodes[dagNodeCount - 1] = newNode;
    return newNode->id;
}

void processTACInstruction(Instruction* tacInstr) {
    int existingNodeId = findInDAG(tacInstr->operation, tacInstr->operand1, tacInstr->operand2);

    if (existingNodeId != -1) {
        printf("Common subexpression found: %s = %s %s %s (Reuse node ID: %d)\n",
            tacInstr->result, tacInstr->operand1, tacInstr->operation, tacInstr->operand2, existingNodeId);
    } else {
        int newNodeId = createDAGNode(tacInstr->operation, tacInstr->operand1, tacInstr->operand2);
        printf("No common subexpression: %s = %s %s %s (New node ID: %d)\n",
            tacInstr->result, tacInstr->operand1, tacInstr->operation, tacInstr->operand2, newNodeId);
    }
}
```

```
int main() {
    int numInstructions;

    printf("Enter number of TAC instructions: ");
    scanf("%d", &numInstructions);

    Instruction tacInstructions[numInstructions];

    for (int i = 0; i < numInstructions; i++) {
        printf("Enter instruction %d (format: result operand1 operation operand2):\n", i + 1);
        scanf("%s %s %s %s", tacInstructions[i].result, tacInstructions[i].operand1, tacInstructions[i].operation, tacInstructions[i].operand2);
    }

    for (int i = 0; i < numInstructions; i++) {
        processTACInstruction(&tacInstructions[i]);
    }

    return 0;
}
```

OUTPUT:

```
csc15@linux-p2-1272il:~/CS22130$ vi Practical7.c
csc15@linux-p2-1272il:~/CS22130$ cc Practical7.c
csc15@linux-p2-1272il:~/CS22130$ ./a.out
Enter number of TAC instructions: 5
Enter instruction 1 (format: result operand1 operation operand2):
t1 x + y
Enter instruction 2 (format: result operand1 operation operand2):
t2 p + q
Enter instruction 3 (format: result operand1 operation operand2):
t3 t1 + p
Enter instruction 4 (format: result operand1 operation operand2):
t4 x + y
Enter instruction 5 (format: result operand1 operation operand2):
t5 t2 * t3
No common subexpression: t1 = x + y (New node ID: 0)
No common subexpression: t2 = p + q (New node ID: 1)
No common subexpression: t3 = t1 + p (New node ID: 2)
Common subexpression found: t4 = x + y (Reuse node ID: 0)
No common subexpression: t5 = t2 * t3 (New node ID: 3)
```

CONCLUSION:

DISCUSSION AND VIVA VOCE:

Q 1: What is DAG?

Q 2: How common subexpression is eliminated?

Q 3: What is difference between local data analysis and global data analysis?

Q 4: Is there any specific situation where DAG will not be able to capture common subexpression?

REFERENCE:

- Lab Manual of Compiler Design (Institute of Aeronautical Engineering, Dundigal, Hyderabad)
- Alfred Aho, Ravi Sethi, Jeffrey D Ullman, "Compilers Principles Techniques and Tools", Pearson. Education,