编译原理第七次作业

秦沁 qqin@mail.ustc.edu.cn

中国科学技术大学
University of Science and Technology of China

(1.1)

- 针对以下C程序片段，直接在源程序上进行循环优化（循环不变计算外提，强度消弱与复写传播优化等）

```
int a[100][100], b[100][100], c[100][100];


int i,j,k;


for(i=0;i<100;i++)
        for(j=0;j<100;j++)
                for(k=0;k<100;k++)
                        c[i][j] = c[i][j] + a[i][k] * b[k][j];
//int 占4个字节
```

(1.1) 循环不变代码外提

```
int a[100][100], b[100][100], c[100][100];
int i,j,k;
for(i=0;i<100;i++)
        for(j=0;j<100;j++)
                for(k=0;k<100;k++)
                        c[i][j] = c[i][j] + a[i][k] * b[k][j];
```

k 循环中的循环不变式外提。

(1.1) 循环不变代码外提

```
int a[100][100], b[100][100], c[100][100];
int i,j,k;
for(i=0;i<100;i++)
        for(j=0;j<100;j++) {
                t1=addr(c[i][j]);
                t2=addr(a[i]);
                for(k=0;k<100;k++)
                        *t1 = *t1 + t2[k] * b[k][j];
        }
```

蓝字表示上一步发生改变的代码，红字表示这一步即将发生改变的代码。下同。

j 循环中的循环不变式外提。

(1.1) 复写传播

```
int a[100][100], b[100][100], c[100][100];
int i,j,k;
for(i=0;i<100;i++){
        t3=addr(c[i]);
        t4=addr(a[i]);
        for(j=0;j<100;j++){
                t1=addr(t3[j]);
                t2=t4;
                for(k=0;k<100;k++)
                        *t1 = *t1 + t2[k] * b[k][j];
        }
}
```

复写传播：删除 t2。

(1.1) 强度消弱

```
int a[100][100], b[100][100], c[100][100];
int i,j,k;
for(i=0;i<100;i++){
        t3=addr(c[i]);
        t4=addr(a[i]);
        for(j=0;j<100;j++){
                t1=addr(t3[j]);
                for(k=0;k<100;k++)
                        *t1 = *t1 + t4[k] * b[k][j];
        }
}
```

把数组写成 *(开始地址+偏移)的形式，以寻找可进行强度消弱的归纳变量。

(1.1) 强度消弱

```
int a[100][100], b[100][100], c[100][100];
int i,j,k;
for(i=0;i<100;i++){
        t3 = c + i * 400;
        t4 = a + i * 400;
        for(j=0;j<100;j++){
                t1 = t3 + j * 4;
                for(k=0;k<100;k++)
                        *t1 = *t1 + *(t4 + k * 4) * *(b + k * 400 + j * 4);
        }
}
```

对 k 循环中表达式进行强度消弱。

(1.1) 强度消弱

```
int a[100][100], b[100][100], c[100][100];
int i,j,k;
for(i=0;i<100;i++){
        t3 = c + i * 400;
        t4 = a + i * 400;
        for(j=0;j<100;j++) {
                t1 = t3 + j * 4;
                t5 = t4; t6 = b + j * 4;
                for(k=0;k<100;k++){
                        *t1 = *t1 + (*t5) * (*t6);
                        t5 = t5 + 4; t6 = t6 + 400;
                }
        }
}
```

对 j 循环中表达式进行强度消弱。

(1.1) 强度消弱

```
int a[100][100], b[100][100], c[100][100];
int i,j,k;
for(i=0;i<100;i++){
        t3 = c + i * 400;
        t4 = a + i * 400;
        t7 = t3; t8 = b;
        for(j=0;j<100;j++){
                t1 = t7;
                t5 = t4; t6 = t8;
                for(k=0;k<100;k++){
                        *t1 = *t1 + (*t5) * (*t6);
                        t5 = t5 + 4; t6 = t6 + 400;
                }
                t7 = t7 + 4; t8 = t8 + 4;
        }
}
```

对 i 循环中表达式进行强度消弱。

(1.1) 复写传播

```
int a[100][100], b[100][100], c[100][100];
int i,j,k;
t9 = c; t10 = a;
for(i=0;i<100;i++){
        t3 = t9;
        t4 = t10;
        t7 = t3; t8 = b;
        for(j=0;j<100;j++) {
                t1 = t7;
                t5 = t4; t6 = t8;
                for(k=0;k<100;k++) {
                        *t1 = *t1 + (*t5) * (*t6);
                        t5 = t5 + 4; t6 = t6 + 400;
                }
                t7 = t7 + 4; t8 = t8 + 4;
        }
        t9 = t9 + 400; t10 = t10 + 400;
}
```

复写传播：删除 t3, t4, t1。

(1.1) 复写传播

```
int a[100][100], b[100][100], c[100][100];
int i,j,k;
t9 = c; t10 = a;
for(i=0;i<100;i++){
        t7 = t9; t8 = b;
        for(j=0;j<100;j++){
                t5 = t10; t6 = t8;
                for(k=0;k<100;k++){
                        *t7 = *t7 + (*t5) * (*t6);
                        t5 = t5 + 4; t6 = t6 + 400;
                }
                t7 = t7 + 4; t8 = t8 + 4;
        }
        t9 = t9 + 400; t10 = t10 + 400;
}
```

最终得到的优化后代码。

(1.2)

- 给出相应的三地址中间代码，并标出其中所有的基本块入口代码
- 直接给出在源程序上进行循环优化后的结果（公共子表达式删除、循环不变计算外提、强度消弱、复写传播等优化）。

```
int from,to,id,i,j,n,sum; //max是常量
int g[max][max],eq[max],queue[max],used[max];
for(i = from; i < to; i++){
        sum += g[eq[queue[i]]][queue[i]];
        if ( i != from ){
                used[queue[i]] = 1;
                for(j = 1;j <= n; j++) if(!used[j]){
                        if(g[queue[i]][j] < g[id][j])
                                g[id][j] = g[queue[i]][j];
                }
        }
} //int 占4个字节
```

(1.2) 三地址码

```
(1)      i:=from              //基本块入口代码
L1:
(2)      if i>=to goto L2     //基本块入口代码
(3)      t1:=i*4              //基本块入口代码
(4)      t2:=queue[t1]
(5)      t3:=t2*4
(6)      t4:=eq[t3]
(7)      t5:=t4*max
(8)      t6:=i*4
(9)      t7:=queue[t6]
(10)     t8:=t5+t7
(11)     t9:=t8*4
(12)     t10:=g[t9]
(13)     sum:=sum+t10
(14)     if i==from goto L3
(15)     t11:=i*4             //基本块入口代码
(16)     t12:=queue[t11]
(17)     t13:=t12*4
(18)     used[t13]:=1
(19)     j:=1
```

queue[i]

queue[i]

queue[i]

基本块的划分依据：控制流从基本块的开始进入，从它的末尾离开。（课本P248）

sum += g[eq[queue[i]]][queue[i]];

易错点：多维数组的翻译见课件lec9_1，P60-61的示例。

(1.2) 三地址码

```
L4:
(20)    if j>n goto L5                  //基本块入口代码
(21)    t14:=j*4                        //基本块入口代码
(22)    t15:=used[t4]
(23)    if used[t15]!=0 goto L6
(24)    t16:=i*4                        //基本块入口代码
(25)    t17:=queue[t16]
(26)    t18:=t17*max
(27)    t19:=t18+j
(28)    t20:=t19*4
(29)    t21:=g[t20]
(30)    t22:=id*max
(31)    t23:=t22+j
(32)    t24:=t23*4
(33)    t25:=g[t24]
(34)    if t21>=t25 goto L7
```

queue[i]

g[queue[i]][j]

addr(g[id][j])

(1.2) 三地址码

```
(35)    t26:=id*max      //基本块入口代码
(36)    t27:=t26+j       addr(g[id][j])
(37)    t28:=t27*4
(38)    t29:=i*4
(39)    t30:=queue[t29]  queue[i]
(40)    t31:=t30*max
(41)    t32:=t31+j       g[queue[i]][j]
(42)    t33:=t32*4
(43)    t34:=g[t33]
(44)    g[t28]:=t34
L7:
L6:
(45)    j:=j+1           //基本块入口代码
(46)    goto L4
L5:
L3:
(47)    i:=i+1           //基本块入口代码
(48)    goto L1
L2:
```

(1.2) 公共子表达式删除

```
int from,to,id,i,j,n,sum; //max是常量
int g[max][max],eq[max],queue[max],used[max];
for(i = from; i < to; i++){
        sum += g[eq[queue[i]]][queue[i]];
        if ( i != from ){
                used[queue[i]] = 1;
                for(j = 1;j <= n; j++) if(!used[j]){
                        if(g[queue[i]][j] < g[id][j])
                                g[id][j] = g[queue[i]][j];
                }
        }
}
```

根据刚才写出的三地址码，把一些公共子表达式对应的c语言表达式复用。

(1.2) 循环不变计算外提

```
int from,to,id,i,j,n,sum; //max是常量
int g[max][max],eq[max],queue[max],used[max];
for(i = from; i < to; i++){
        t1 = queue[i];
        sum += g[eq[t1]][t1];
        if ( i != from ){
                used[t1] = 1;
                for(j = 1;j <= n; j++)
                        if(!used[j]){
                                t2 = g[t1][j];
                                t3 = addr(g[id][j]);
                                if(t2 < *t3) *t3 = t2;
                        }
        }
}
```

j 循环中的循环不变式外提。

(1.2) 循环不变计算外提

```
int from,to,id,i,j,n,sum; //max是常量
int g[max][max],eq[max],queue[max],used[max];
for(i = from; i < to; i++){
        t1 = queue[i];
        sum += g[eq[t1]][t1];
        if ( i != from ){
                used[t1] = 1;
                t4 = addr(g[t1]);
                t5 = addr(g[id]);
                for(j = 1;j <= n; j++)
                        if(!used[j]){
                                t2 = t4[j];
                                t3 = addr(t5[j]);
                                if(t2 < *t3) *t3 = t2;
                        }
        }
}
```

i 循环中的循环不变式外提。

(1.2) 复写传播

```
int from,to,id,i,j,n,sum; //max是常量
int g[max][max],eq[max],queue[max],used[max];
t6 = addr(g[id]);
for(i = from; i < to; i++){
        t1 = queue[i];
        sum += g[eq[t1]][t1];
        if ( i != from ){
                used[t1] = 1;
                t4 = addr(g[t1]);
                t5 = t6;
                for(j = 1;j <= n; j++)
                        if(!used[j]){
                                t2 = t4[j];
                                t3 = addr(t5[j]);
                                if(t2 < *t3) *t3 = t2;
                        }
        }
}
```

复写传播：删除 t5。

(1.2) 强度消弱

```
int from,to,id,i,j,n,sum; //max是常量
int g[max][max],eq[max],queue[max],used[max];
t6 = addr(g[id]);
for(i = from; i < to; i++){
        t1 = queue[i];
        sum += g[eq[t1]][t1];
        if ( i != from ){
                used[t1] = 1;
                t4 = addr(g[t1]);
                for(j = 1;j <= n; j++)
                        if(!used[j]){
                                t2 = t4[j];
                                t3 = addr(t6[j]);
                                if(t2 < *t3) *t3 = t2;
                        }
        }
}
```

把数组写成 *(开始地址+偏移)的形式，以寻找可进行强度消弱的归纳变量。

(1.2) 强度消弱

```
int from,to,id,i,j,n,sum; //max是常量
int g[max][max],eq[max],queue[max],used[max];
t6 = g + id * 4;
for(i = from; i < to; i++){
        t1 = *(queue + i * 4);
        sum += *(g + (*(eq + t1 * 4) * max + t1) * 4);
        if ( i != from ){
                *(used + t1 * 4) = 1;
                t4 = g + t1 * 4;
                for(j = 1;j <= n; j++)
                        if(! *(used + j * 4)){
                                t2 = *(t4 + j * 4);
                                t3 = t6 + j * 4;
                                if(t2 < *t3) *t3 = t2;
                }
        }
}
```

对 j 循环中表达式进行强度消弱。

(1.2) 强度消弱

```
int from,to,id,i,j,n,sum; //max是常量
int g[max][max],eq[max],queue[max],used[max];
t6 = g + id * 4;
for(i = from; i < to; i++){
        t1 = *(queue + i * 4);
        sum += *(g + (*(eq + t1 * 4) * max + t1) * 4);
        if ( i != from ){
                *(used + t1 * 4) = 1;
                t4 = g + t1 * 4;
                t7 = used + 4; t8 = t4 + 4; t9 = t6 + 4;
                for(j = 1;j <= n; j++) {
                        if(! *(t7)){
                                t2 = *(t8);
                                t3 = t9;
                                if(t2 < *t3) *t3 = t2;
                        }
                        t7 = t7 + 4; t8 = t8 + 4; t9 = t9 + 4;
                }
        }
}
```

对 i 循环中表达式进行强度消弱。

(1.2) 复写传播

```
int from,to,id,i,j,n,sum; //max是常量
int g[max][max],eq[max],queue[max],used[max];
t6 = g + id * 4;
t10 = queue + from * 4;
for(i = from; i < to; i++){
        t1 = *(t10);
        sum += *(g + (*(eq + t1 * 4) * max + t1) * 4);
        if ( i != from ){
                *(used + t1 * 4) = 1;
                t4 = g + t1 * 4;
                t7 = used + 4; t8 = t4 + 4; t9 = t6 + 4;
                for(j = 1;j <= n; j++) {
                        if(! *(t7)){
                                t2 = *(t8);
                                t3 = t9;
                                if(t2 < *t3) *t3 = t2;
                        }
                        t7 = t7 + 4; t8 = t8 + 4; t9 = t9 + 4;
                }
        } t10 = t10 + 4;
}
```

复写传播：删除 t3。

(1.2) 复写传播

最终得到的优化后代码。

```
int from,to,id,i,j,n,sum; //max是常量
int g[max][max],eq[max],queue[max],used[max];
t6 = g + id * 4;
t10 = queue + from * 4;
for(i = from; i < to; i++){
        t1 = *(t10);
        sum += *(g + (*(eq + t1 * 4) * max + t1) * 4);
        if ( i != from ){
                *(used + t1 * 4) = 1;
                t4 = g + t1 * 4;
                t7 = used + 4; t8 = t4 + 4; t9 = t6 + 4;
                for(j = 1;j <= n; j++) {
                        if(! *(t7)){
                                t2 = *(t8);
                                if(t2 < *t9) *t9 = t2;
                        }
                        t7 = t7 + 4; t8 = t8 + 4; t9 = t9 + 4;
                }
        }
        t10 = t10 + 4;
}
```

(2)

- 针对Homework 6的（1）中的C函数，在其三地址码基础上，给出流图，回边和自然循环。

```
#define N 32
int a[N],b[N];
int arr[N+1][N+1];
void lcs() {
    for (i = 1; i <= length1; ++i) {
        for (j = 1; j <= length2; ++j) {
            if (a[i - 1] == b[j - 1]) { // 0
                arr[i][j] = arr[i - 1][j - 1] + 1;
            }
            else {
            arr[i][j] = arr[i - 1][j] > arr[i][j - 1] ? arr[i - 1][j] : arr[i][j - 1];
            }
        }
    }
} // end of lcs()
```

(2) 三地址码

```
(1)     i:=1                          //基本块入口代码
L1:
(2)     if i>length1 goto L2          //基本块入口代码
(3)     j:=1                          //基本块入口代码
L3:
(4)     if j>length2 goto L4          //基本块入口代码
(5)     t1:=i-1                       //基本块入口代码
(6)     t2:=t1*4
(7)     t3:=a[t2]
(8)     t4:=j-1
(9)     t5:=t4*4
(10)    t6:=b[t5]
(11)    if t3==t6 goto L5
(12)    goto L6                       //基本块入口代码
L5:
(13)    t7:=i*33                      //基本块入口代码
(14)    t8:=t7+j
(15)    t9:=t8*4
```

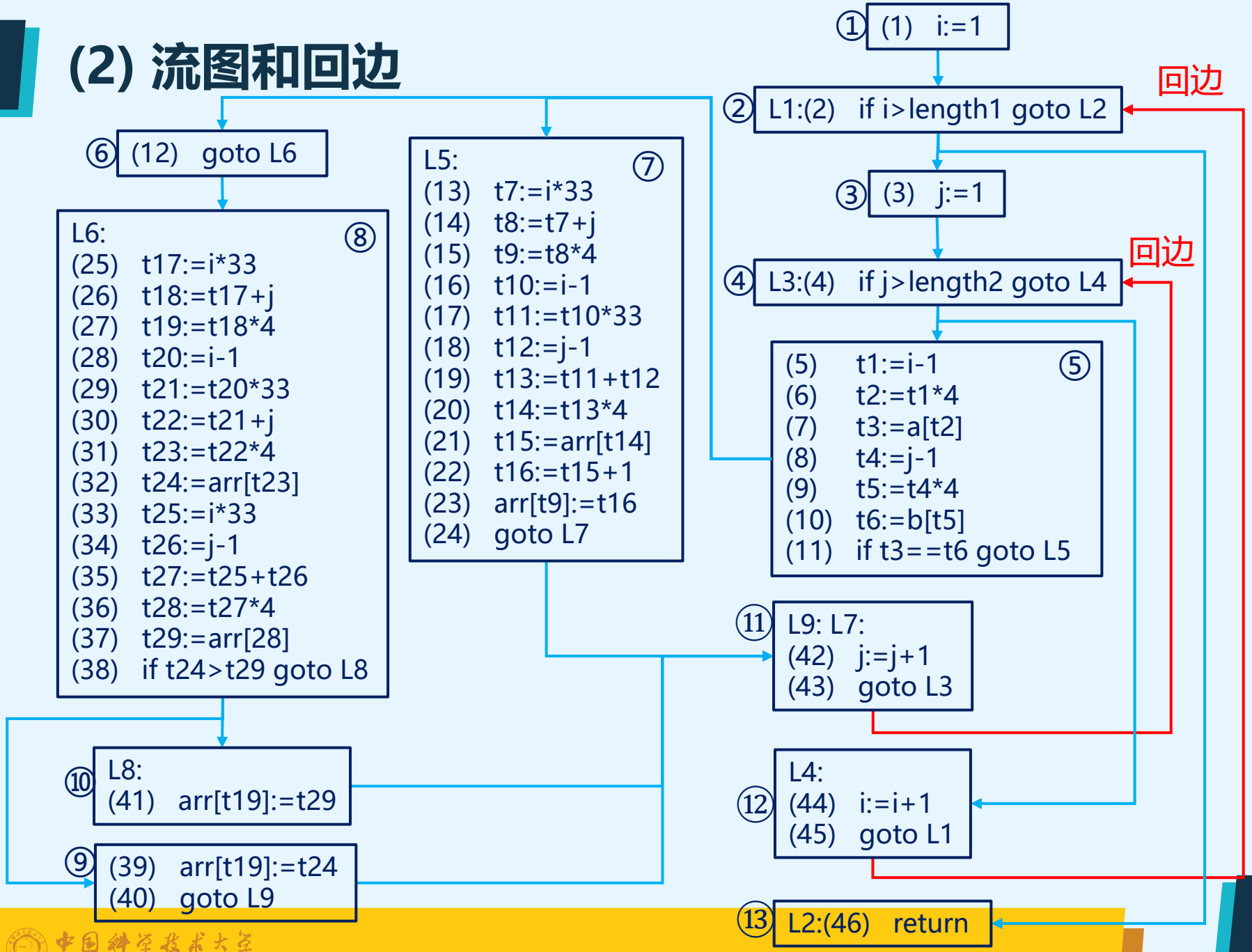
(2) 三地址码

```
(16)    t10:=i-1
(17)    t11:=t10*33
(18)    t12:=j-1
(19)    t13:=t11+t12
(20)    t14:=t13*4
(21)    t15:=arr[t14]
(22)    t16:=t15+1
(23)    arr[t9]:=t16
(24)    goto L7
L6:
(25)    t17:=i*33          //基本块入口代码
(26)    t18:=t17+j
(27)    t19:=t18*4
(28)    t20:=i-1
(29)    t21:=t20*33
(30)    t22:=t21+j
(31)    t23:=t22*4
(32)    t24:=arr[t23]
```

(2) 三地址码

```
(33)    t25:=i*33
(34)    t26:=j-1
(35)    t27:=t25+t26
(36)    t28:=t27*4
(37)    t29:=arr[28]
(38)    if t24>t29 goto L8
(39)    arr[t19]:=t24    //基本块入口代码
(40)    goto L9
L8:
(41)    arr[t19]:=t29    //基本块入口代码
L9:
L7:
(42)    j:=j+1           //基本块入口代码
(43)    goto L3
L4:
(44)    i:=i+1           //基本块入口代码
(45)    goto L1
L2:
(46)    return           //基本块入口代码
```

(2) 流图和回边


① (1) i:=1
② L1:(2) if i>length1 goto L2
③ (3) j:=1
④ L3:(4) if j>length2 goto L4
⑤ (5) t1:=i-1
(6) t2:=t1*4
(7) t3:=a[t2]
(8) t4:=j-1
(9) t5:=t4*4
(10) t6:=b[t5]
(11) if t3==t6 goto L5
⑥ (12) goto L6
⑦ L5:
(13) t7:=i*33
(14) t8:=t7+j
(15) t9:=t8*4
(16) t10:=i-1
(17) t11:=t10*33
(18) t12:=j-1
(19) t13:=t11+t12
(20) t14:=t13*4
(21) t15:=arr[t14]
(22) t16:=t15+1
(23) arr[t9]:=t16
(24) goto L7
⑧ L6:
(25) t17:=i*33
(26) t18:=t17+j
(27) t19:=t18*4
(28) t20:=i-1
(29) t21:=t20*33
(30) t22:=t21+j
(31) t23:=t22*4
(32) t24:=arr[t23]
(33) t25:=i*33
(34) t26:=j-1
(35) t27:=t25+t26
(36) t28:=t27*4
(37) t29:=arr[28]
(38) if t24>t29 goto L8
⑨ (39) arr[t19]:=t24
(40) goto L9
⑩ L8:
(41) arr[t19]:=t29
⑪ L9: L7:
(42) j:=j+1
(43) goto L3
⑫ L4:
(44) i:=i+1
(45) goto L1
⑬ L2:(46) return
回边
回边

(2) 回边的计算

此页PPT只为让大家理解回边的含义，考试时可直接写出回边，无需写这么详细的步骤。

- (支配集和回边的定义见课本P311-313)
- 计算出每个结点的支配集

D(1)={1}
D(2)={1, 2}
D(3)={1, 2, 3}
D(4)={1, 2, 3, 4}
D(5)={1, 2, 3, 4, 5}
D(6)={1, 2, 3, 4, 5, 6}
D(7)={1, 2, 3, 4, 5, 7}
D(8)={1, 2, 3, 4, 5, 6, 8}
D(9)={1, 2, 3, 4, 5, 6, 8, 9}
D(10)={1, 2, 3, 4, 5, 6, 8, 10}
D(11)={1, 2, 3, 4, 5, 11}
D(12)={1, 2, 3, 4, 12}
D(13)={1, 2, 13}

- 存在11->4和12->2两条边，而4和2分别在11和12的支配集里面，所以这两条边是回边。

(2) 自然循环

- 回边12->2确定的自然循环：
  - {2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
- 回边11->4确定的自然循环：
  - {4, 5, 6, 7, 8, 9, 10, 11}

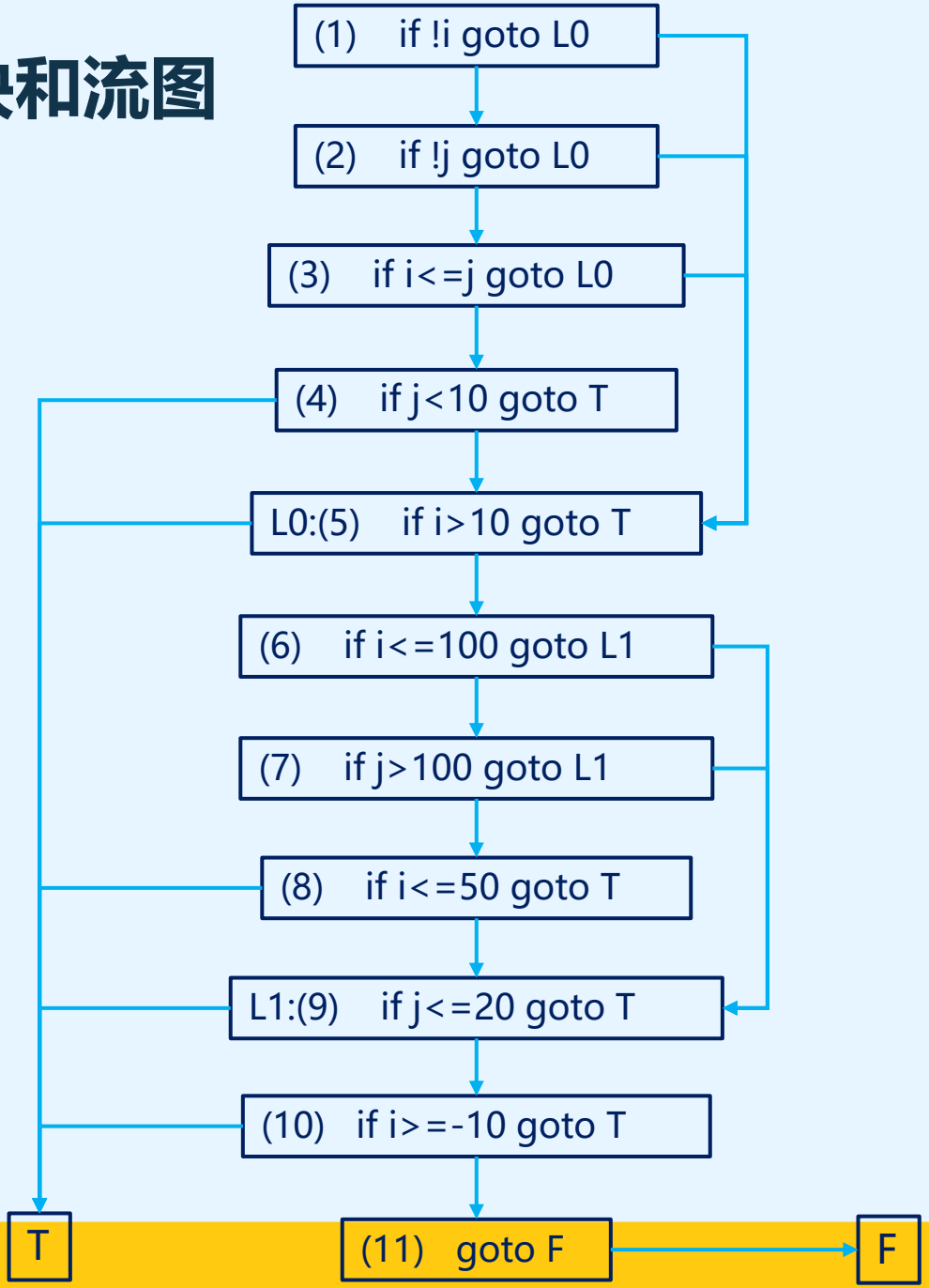
(3)

- 针对Homework 6的（2.2）中（b），在其三地址码基础上，给出基本块和流图。

```
i && j && i > j  && j < 10  || (i>10)  || !(i <= 100)
&& ( j <= 100)  && !( i>50 )  || !(j > 20 && i < -10)
```

```
(1)   if !i goto L0
(2)   if !j goto L0
(3)   if i<=j goto L0
(4)   if j<10 goto T
L0:
(5)   if i>10 goto T
(6)   if i<=100 goto L1
(7)   if j>100 goto L1
(8)   if i<=50 goto T
L1:
(9)   if j<=20 goto T
(10)  if i>=-10 goto T
(11)  goto F
```

(3) 基本块和流图


(1) if !i goto L0
(2) if !j goto L0
(3) if i<=j goto L0
(4) if j<10 goto T
L0:(5) if i>10 goto T
(6) if i<=100 goto L1
(7) if j>100 goto L1
(8) if i<=50 goto T
L1:(9) if j<=20 goto T
(10) if i>=-10 goto T
T
(11) goto F
F

(4)

- 有C/C++程序及其汇编代码如下：

```
int a[24];
int (&fra())[24] { return a; }
int(&(&fr())())[24] { return fra; }
int main() { return fr()()[20]; }
//int 占4个字节
```

```
a:
    .zero   96
fra():
    push rbp
    mov  rbp, rsp
    _______①_______
    pop  rbp
    ret
fr():
    push rbp
    mov  rbp, rsp
    _______②_______
    pop  rbp
    ret
main:
    push rbp
    mov  rbp, rsp
    _______③_______
    pop     rbp
    ret
```

(4.1)

```
int a[24];
int (&fra())[24] { return a; }
int(&(&fr())())[24] { return fra; }
int main() { return fr()()[20]; }
//int 占4个字节
```

解析：fr是一个函数，它没有参数，返回值类型是一个函数的引用。这个函数没有参数，返回值类型是一个数组的引用，数组的大小是24，数组的每个元素都是int型。

- (4.1) 给出该C/C++程序中名字fr的类型表达式。引用类型可用refer(T)形式来描述。
  - ()->refer(()->refer(array(24, int)))
  - (类型表达式的格式参考课件lec12 p9).

(4.2)

```
int a[24];
int (&fra())[24] { return a; }
int(&(&fr())())[24] { return fra; }
int main() { return fr()()[20]; }
//int 占4个字节
```

- (4.2) 补全三个空白下划线处的汇编代码（每处可能不止一条）。
  1. lea rax, [a]
     //把a的地址加载到rax寄存器。
  2. lea rax, [fra()]
     //把fra()的地址加载到rax寄存器。
  3. call fr() // 调用fr()函数。
     call rax
     //调用fr()的返回值，即fra()函数的引用。
     mov rax DWORD PTR [rax+80]
     //从fra()返回的数组中取出元素，放入rax寄存器。

```
a:
    .zero   96
fra():
    push rbp
    mov  rbp, rsp
    _______①_______
    pop  rbp
    ret
fr():
    push rbp
    mov  rbp, rsp
    _______②_______
    pop  rbp
    ret
main:
    push rbp
    mov  rbp, rsp
    _______③_______
    pop     rbp
    ret
```

(4.2)

- 其他合理的答案也是正确的。

```
1   a:
2     .zero 96
3   fra():
4     push rbp
5     mov rbp, rsp
6     mov eax, OFFSET FLAT:a
7     pop rbp
8     ret
9   fr():
10    push rbp
11    mov rbp, rsp
12    mov eax, OFFSET FLAT:fra()
13    pop rbp
14    ret
15  main:
16    push rbp
17    mov rbp, rsp
18    call fr()
19    call rax
20    mov eax, DWORD PTR [rax+80]
21    pop rbp
22    ret
```

在汇编语言中，`mov eax, OFFSET FLAT:a` 表示将符号 `a` 的地址（偏移量）加载到寄存器 `eax` 中。

具体解释如下：

1. `mov` **指令**
   `mov` 是一个基本的汇编指令，用于将数据从源操作数传送到目标操作数。在这里，`eax` 是目标，`OFFSET FLAT:a` 是源。
2. `OFFSET` **操作符**
   `OFFSET` 是一个汇编语言中的操作符，用于获取符号的地址。
   - 如果符号 `a` 是一个变量或数据标号，`OFFSET a` 就会返回 `a` 在内存中的偏移地址。
   - 这个地址通常是相对于段的起始地址。
3. `FLAT` **内存模型**
   在现代 x86 汇编中，`FLAT` 是一种内存模型，表示整个内存空间是线性的，段选择器的值通常被设置为 0。这在 32 位和 64 位模式下很常见，因为它简化了段寄存器的使用。
   - `FLAT` 模型下，`OFFSET FLAT:a` 通常可以直接理解为符号 `a` 的线性地址。
4. `a`
   `a` 是一个符号，通常是一个全局变量、函数或某个内存位置的标号。具体含义需要根据上下文来判断。

(5.1)

- 有C程序如下：

```
int main(){
    int(*p)[20],i,j;
    *(*p++ +j) = i + j;
    return 0;
} //int 占4个字节
```

- (5.1)给出该C程序中变量p的类型表达式：
  - pointer(array(20, int))

(5.2)

内存寻址方式（课件lec11 P15）：
(%ecx,%edx,4) 相当于%ecx + %edx * 4

```
int main(){
   int(*p)[20],i,j;
   *(*p++ +j) = i + j;
   return 0;
} //int 占4个字节
```

```
movl  -4(%ebp), %ecx
... //待补全
movl  %eax, (%ecx,%edx,4)
... // 待补全
```

（5.2）补全下划线处C语句对应的linux汇编代码：(答案不唯一)

```
movl  -4(%ebp), %ecx          // 把 p 的值存入 %ecx 寄存器
movl  (%ecx), %ecx            // 把 *p 的值存入 %ecx 寄存器
movl  -12(%ebp), %edx         // 把 j 的值存入 %edx 寄存器
movl  -8(%ebp), %eax          // 把 i 的值存入 %eax 寄存器
addl -12(%ebp), %eax          // 计算 i + j，存入 %eax 寄存器
movl  %eax, (%ecx,%edx,4)     // 把 i + j 存入 (*p + j * 4) 的位置
addl   $80, -4(%ebp)          // p++
```

(6)

- 有C程序片段如下：

（6.1）给出相应的三地址中间代码。其中，采用短路计算方式来翻译布尔表达式，且每一个关系表达式仅对应一条跳转代码。

（6.2）根据上述给出的三地址代码，划分基本块，画流图且找出其中的自然循环。

```
int i, j, v;
i = 1;
while (i < N) {
        if (a[i] < a[i-1]) {
                v = a[i];
                a[i] = a[i-1];
                j = i - 2;
                while (j >=0 && a[j] > v) {
                        a[j+1] = a[j];
                        j = j - 1;
                }
                a[j+1] = v;
        }
        i = i + 1;
} //int 占4个字节
```

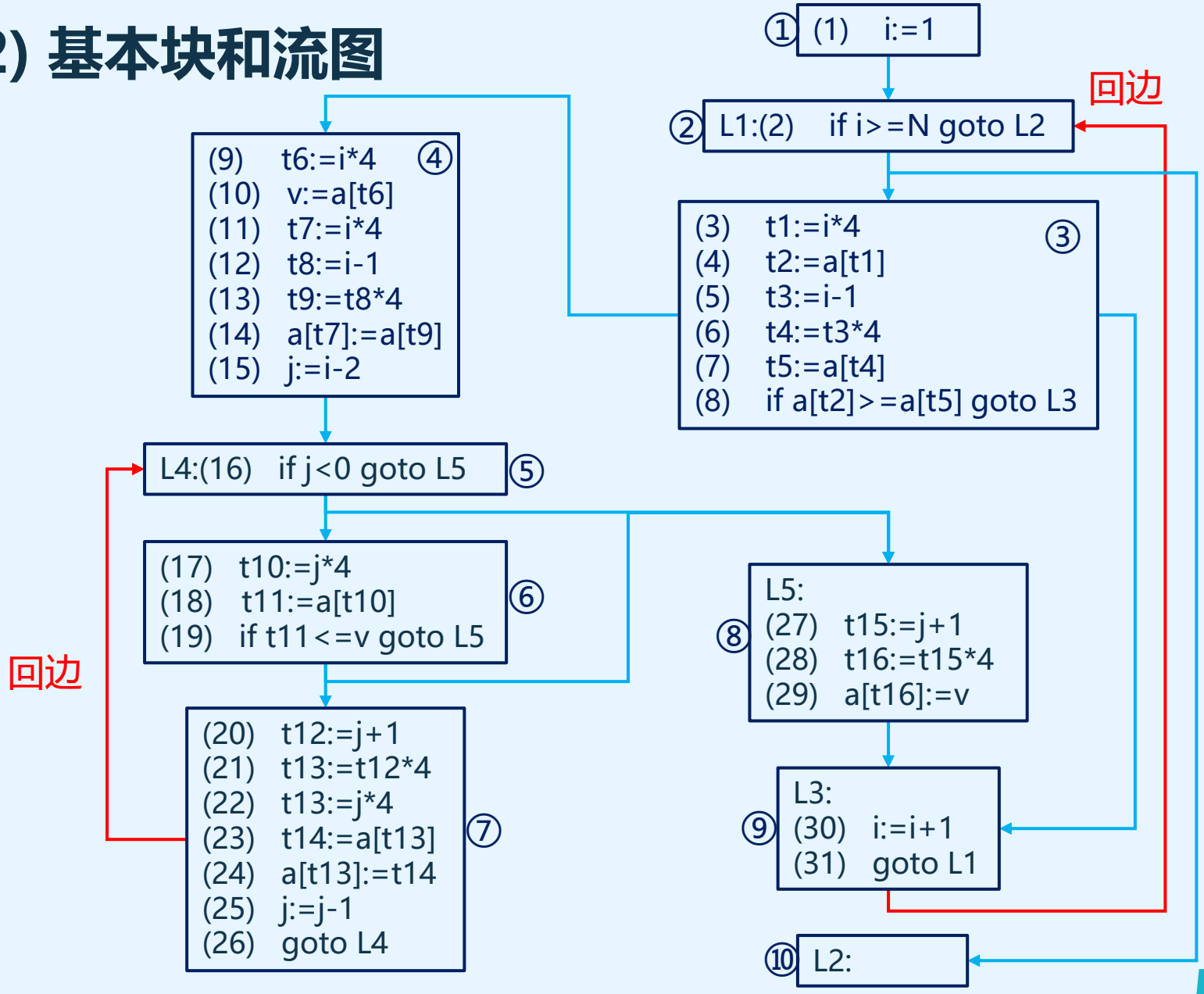
(6.1) 三地址码

```
(1)    i:=1                    //基本块入口代码
L1:                            while (i < N)
(2)    if i>=N goto L2         //基本块入口代码
(3)    t1:=i*4                 //基本块入口代码
(4)    t2:=a[t1]
(5)    t3:=i-1
(6)    t4:=t3*4
(7)    t5:=a[t4]
(8)    if a[t2]>=a[t5] goto L3   if (a[i] < a[i-1])
(9)    t6:=i*4                 //基本块入口代码
(10)   v:=a[t6]
(11)   t7:=i*4
(12)   t8:=i-1
(13)   t9:=t8*4
(14)   a[t7]:=a[t9]
(15)   j:=i-2
```

(6.1) 三地址码

```
L4:
(16)  if j<0 goto L5          //基本块入口代码
(17)  t10:=j*4                //基本块入口代码
(18)  t11:=a[t10]                while (j >=0 && a[j] > v)
(19)  if t11<=v goto L5
(20)  t12:=j+1                //基本块入口代码
(21)  t13:=t12*4
(22)  t13:=j*4
(23)  t14:=a[t13]
(24)  a[t13]:=t14
(25)  j:=j-1
(26)  goto L4
L5:
(27)  t15:=j+1                //基本块入口代码
(28)  t16:=t15*4
(29)  a[t16]:=v
L3:
(30)  i:=i+1                  //基本块入口代码
(31)  goto L1
L2:                           //基本块入口代码
```

(6.2) 基本块和流图


①
(1) i:=1
②
L1:(2) if i>=N goto L2
③
(3) t1:=i*4
(4) t2:=a[t1]
(5) t3:=i-1
(6) t4:=t3*4
(7) t5:=a[t4]
(8) if a[t2]>=a[t5] goto L3
④
(9) t6:=i*4
(10) v:=a[t6]
(11) t7:=i*4
(12) t8:=i-1
(13) t9:=t8*4
(14) a[t7]:=a[t9]
(15) j:=i-2
⑤
L4:(16) if j<0 goto L5
⑥
(17) t10:=j*4
(18) t11:=a[t10]
(19) if t11<=v goto L5
⑦
(20) t12:=j+1
(21) t13:=t12*4
(22) t13:=j*4
(23) t14:=a[t13]
(24) a[t13]:=t14
(25) j:=j-1
(26) goto L4
⑧
L5:
(27) t15:=j+1
(28) t16:=t15*4
(29) a[t16]:=v
⑨
L3:
(30) i:=i+1
(31) goto L1
⑩
L2:
回边
回边

(6.2) 自然循环

- 回边9->2确定的自然循环:
  - {2, 3, 4, 5, 6 ,7, 8, 9}
- 回边7->5确定的自然循环:
  - {5, 6, 7}

参考资料

- 2023秋季学期课程习题答案
- https://github.com/Wloner0809/ustc-course-resource/tree/main/%E7%BC%96%E8%AF%91%E5%8E%9F%E7%90%86
- 大家的作业

若此答案有错误或不足，欢迎大家批评指正！

谢谢！

祝大家元旦快乐

天天开心~