Otimizador

Programas que melhoram outros programas.

Programa otimizado é melhor do que o programa original.

Melhor segundo um critério especificado.

Fácil de fazer mas difícil de garantir que não modifica o funcionamento.

Otimizador

A otimização pode melhorar as características de um programa.

Não é justificativa para programa mal escrito.

É preciso identificar as oportunidades de otimização.

Otimizador

A otimização não altera a **ordem** dos algoritmos, alteram as constantes multiplicativas.

Ou seja:

Se existe algoritmo O(n log n) para o problema

Não se pode esperar que um programa com algoritmo $O(n^2)$ se transforme no algoritmo O(n log n).

Tipo de Otimização

Classificação pela quantidade de informação manipulada

Otimização locais - Em trechos pequenos de programas, por exemplo: trechos sem desvios (trechos em linha reta).

Otimização Intermediária - Apenas em funções, módulos, ou classes (dependendo da linguagem).

Otimização global - Consideram as inter-relações de todas as partes de um programa.

Introdução - Comando Inútil

Exemplo: x=a+b

1 – **Por nunca ser executado**

```
if( 0 )
    x=a+b;
```

2 - **Por repetição**

```
x=a+b;
x=a+b;
```

Otimização Local

Introdução - Comando Inútil

3 – **Por não utilização**

```
int f(int z) {
    int x;
    ...
    x=a+b;
}
```

Otimização Intermediária

Introdução - Comando em Repetição

Retirada de um comando de dentro de repetições

Código não otimizado

```
for (i=0; i<N; i++) {
        a=j+5;
        f(a*i);
}
```

Código otimizado

```
a=j+5;
for (i=0; i<N; i++) {
        f(a*i);
}
```

Introdução - Comando em Repetição

Problema: Se N = 0

Código não otimizado

```
for (i=0; i<N; i++) {
        a=j+5;
        f(a*i);
}
```

O programa foi piorado!

Código otimizado

```
a=j+5;
for (i=0; i<N; i++) {
        f(a*i);
}
```

Definições

Bloco básico – Trecho de código em linha reta.

Condições:

1 – Primeiro comando inicia um bloco básico.

2 - Qualquer comando rotulado (label), ou que seja alvo de comando de desvio inicia um novo bloco básico.

3 - Qualquer comando de desvio termina um bloco básico.

Loop – Repetição da execução de um bloco básico.

Oportunidades de Otimização

Eliminação de sub-expressões comuns

Mesma expressão ocorre mais de uma vez em um trecho.

Condição: Se variáveis que ocorrem na expressão não tem valores alterados entre as ocorrências.

```
...
x=a+b;
...
y=a+b;
...
```

```
...
t1=a+b;
x=t1;
...
y=t1;
...
```

```
...
x=a+b;
...
y=x;
...
```

Oportunidades de Otimização

Eliminação de código morto

Código que **NÃO** pode ser alcançado durante execução de programa.

```
goto x;
i=3; /* este código não será executado */
...
x: ...
```

Oportunidades de Otimização

```
#define DEBUG 0
...
if(DEBUG) {
... /* este código não será executado */
}
```

```
int f(int x) {
        return x++; /* incremento não será executado */
}
```

Oportunidades de Otimização

Renomeação de variáveis temporárias

Eliminar variáveis temporárias desnecessárias.

```
x=(a+b)*(c+d);
y=(e*f)+(g*h);
```

```
t1=a+b
t2=c+d
t3=t1*t2
x=t3
t4=e*f
t5=g*h
t6=t4+t5
y=t6
```

```
t1=a+b
t2=c+d
x=t1*t2

t4=e*f
t5=g*h
y=t4+t5
```

Oportunidades de Otimização

Renomeação de variáveis temporárias

Eliminar variáveis temporárias desnecessárias.

```
x=(a+b)*(c+d);
y=(e*f)+(g*h);
```

```
t1=a+b
t2=c+d
x=t1*t2

t4=e*f
t5=g*h
y=t4+t5
```

t2 t1 t5 t4

```
t1=a+b
t2=c+d
x=t1*t2
t1=e*f
t2=g*h
y=t1+t2
```

Oportunidades de Otimização

Transformações algébricas

Aplicar transformações baseadas em propriedades algébricas.

Exemplos

- Comutatividade
- Associatividade
- Identidade

Oportunidades de Otimização

Transformações algébricas

```
x=(a+b)+(c+d)+(e+f);

    t1=a+b;
    t2=c+d;
    t3=e+f
    t4=t1+t2;
    x=t4+t3
```

```
x=(((((a+b)+c)+d)+e)+f);

    t1=a+b;
    t2=t1+c;
    t1=t2+d;
    t2=t1+e
    x=t2+f
```

Oportunidades de Otimização

Dobramento de constantes

Expressões ou sub-expressões com valores constantes podem ser avaliadas em tempo de compilação (*dobradas).*

```
#define N 100
…
while ( I < N-1 ) {
…
}
```

```
#define N 100
#define N_1 99
…
 while ( I < N_1 ) {
…
}
```

Oportunidades de Otimização

Redução de força

Operações caras podem ser substituídas por operações baratas.

strlen(strcat(s1, s2))

strlen(s1) + strlen(s2)

Oportunidades de Otimização

Eliminar variáveis de indução

Variável de indução: A cada ciclo de uma repetição tem seu valor alterado de forma constante.

```
for(j = 0; a[j*2] < v; j++) {
    v = b + c;
}
```

```
for(t1 = 0; a[t1] < N; t1+=2){
    v = b + c;
}
```

Oportunidades de Otimização

Expansão de ciclos

Efetuar múltiplas execuções do código em cada ciclo.

Reduzir overhead da avaliação de novo ciclo.

```
for(i = 0; i < N; i++) {
    v = b + c;
}
```

```
for(i = 0; i < N; i+=2){
        v = b + c;
        v = b + c;
}
```

Oportunidades de Otimização

Inlining de funções

Substituir a chamada de uma função pelo seu código.

Reduzir overhead da chamada da função.