Universidade Estadual de Londrina

DC Departamento de Computação

# 5COP093 - Trabalho T3

Utilizando os conhecimentos adquiridos na disciplina até o momento, implemente um programa que realiza a leitura de um arquivo que contém grafos de interferência e tenta alocar registradores através da técnica de coloração de grafos. Neste arquivo cada grafo possui o seguinte formato:

```
Graph 1:
K=21
32 --> 6 7 8 0 1 2
33 --> 6 7 8 0 1
34 --> 7 8 0 6 35 36 37 38 39 40 43 42 41 46 45 44 47
35 --> 34 8 0 7 36 37 38 39 40 43 42 41 46 45 44 47 1
36 --> 34 35 0 8 37 38 39 40 43 42 41 46 45 44 47 1 6
37 --> 34 35 36 0 38 39 40 43 42 41 46 45 44 47 1 6 7
38 --> 34 35 36 37
39 --> 34 35 36 37 40 43 42 41
40 --> 34 35 36 37 39
41 --> 34 35 36 37 39
42 --> 34 35 36 37 39
43 --> 34 35 36 37 39
44 --> 34 35 36 37
45 --> 34 35 36 37
46 --> 34 35 36 37
47 --> 35 36 37 34 1 6 7 8
3<->32 2<->33 1<->34 6<->35 7<->36 8<->37 0<->38 34<->1 35<->6 36<->7 37<->8 47<->0
```

**Descrição do Arquivo de Grafos**

A linha `Graph 1:` indica o início de um grafo de interferência e o seu respectivo número. O arquivo de testes possui um total de 27291 grafos, ou seja o último grafo será iniciado por uma linha contendo o seguinte texto: `Graph 27291:`

A linha `K=21` indica o total de registradores físicos (ou cores) que devem ser considerados no processo de alocação/coloração. Como no exemplo `K` possui o valor 21, isto quer dizer que existem 21 cores, numeradas de `0` até `20` para realizar o processo de alocação/coloração.

As linhas iniciadas por um número seguido do símbolo `-->` indicam um cojunto de interferências. Considere por exemplo a seguinte linha:

```
32 --> 6 7 8 0 1 2
```

Na linha apresentada o registrador virtual `32` interfere com os registradores físicos/virtuais que aparecem após o símbolo `-->`, ou seja, o registrador virtual `32` interfere com os registradores 6, 7, 8, 0, 1, 2. Observe que neste exemplo em particular o registrador virtual `32` interfere com registradores físicos (ou seja, cores), pois o número de todos é menor que 21. Neste exemplo, como `K` é igual a 21, todo registrador que possuir número entre 0 e 20 corresponde a um registrador físico, ou seja, uma cor; e todo registrador que possuir número igual ou maior a 21 corresponde a um registrador virtual e desta forma não possui cor nenhuma associada. Ainda na linha exemplo, como `32` já interfere com alguns registradores físicos (cores), ele só poderá ser alocado aos demais registradores físicos (cores), isto é, os valores: 3, 4, 5, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 e 20.

A última linha do grafo indica operações de `MOVE` (cópia) entre os registradores indicados. É importante observar que nem todo grafo irá possuir essa linha, pois podem haver situações onde não existe nenhuma operação de `MOVE` que possa ser otimizada. Para este trabalho esta linha deve ser **DESCONSIDERADA**, ou seja, caso o grafo possua tal linha, ela deve ser simplesmente descartada.

**Programa Alocador e Saída Esperada**

A sua implementação deve tentar colorir cada um dos grafos do arquivo de entrada com o número de cores que é fornecido na linha que contém o valor de `K`. O processo de alocação possui as seguintes fases:

- **Build:** corresponde a construção do grafo. Nesta etapa basta seguir as interferências indicadas no grafo.
- **Simplify:** corresponde a etapa de remover os nós do grafo. Os nós são removidos do grafo um a um até que todos tenham sido removidos. Lembre-se que sempre deve-se remover um nó cujo grau seja menor que o valor de `K`. Os nós removidos devem ser inseridos em uma pilha.
- **Potencial Spill:** se em algum momento do processo de remoção dos nós do grafo não houver um nó cujo grau seja menor `K`, deve-se escolher um nó para talvez ser enviado para a memória. O seu programa deve escolher sempre o nó que tenha o maior grau no grafo. No caso de haver mais de uma possibilidade de escolha, o seu programa deve selecionar como *potencial spill* o nó que possuir o menor número. Suponha, por exemplo, que durante o processo de remoção dos nós, todos possuem grau maior que `K` e o maior grau do grafo é 25, sendo que os nós `37`, `45` e `88` possuem grau 25; o seu programa deve remover do grafo o nó `37`, pois ele possui o menor número dentre os registradores virtuais a serem escolhidos. Tal estratagema é para garantir um comportamento semelhante entre diferentes implementações do presente trabalho.
- **Assign:** esta etapa ocorre após todos os nós do grafo serem removidos. Ela irá reconstruir o grafo e associar registradores/cores aos nós a medida em que eles são reintroduzidos no grafo, na ordem inversa em que foram removidos. Novamente, de forma a garantir um comportamento uniforme entre diferentes implementações, toda vez que um nó for reintroduzido no grafo, o seu programa deve atribuir ao nó a menor cor possível. Suponha que ao, por exemplo, reintroduzir no grafo o nó `77` e que o mesmo pode receber as cores 10, 15, 19, 20; a escolha deve ser pela cor de número 10 que é a de menor número.
- **Spill:** se durante o processo de atribuição de cores aos nós do grafo, não for possível colorir um determinado nó, ele deve ser enviado para *spill*. Em condições normais isso implica em modificar o programa e reiniciar o processo de coloração, mas no presente caso, como somente os grafos estão disponíveis, você deverá terminar o processo de alocação/coloração indicando que o grafo gerou um *spill*.

A saída do seu programa deve indicar somente se cada um dos grafos foi colorido com sucesso ou se gerou um *spill*. Para grafos coloridos com sucesso o seu programa deve gerar a mensagem `SUCCESS`; para grafos que geraram *spill* a mensagem `SPILL`, ambas após o número do grafo. Como exemplo de saída, considere que o arquivo de testes possui apenas 3 grafos e que os grafos 1 e 3 foram coloridos com sucesso e que o grafo 2 gerou *spill*. A saída esperada é a seguinte:

```
Graph 1: SUCCESS
Graph 2: SPILL
Graph 3: SUCCESS
```

O número de linhas da saída deve ser igual ao número de grafos do arquivo de entrada. Como o arquivo de testes a ser fornecido possui 27291 grafos, a saída deverá conter exatamente 27291 linhas. O seu programa deverá ler os grafos da entrada padrão do sistema e escrever a sua saída na saída padrão do sistema. Em sua implementação você pode usar as ferramentas `flex` e `bison` se desejar.

**IMPORTANTE:** O arquivo de testes possui mais de 40 MegaBytes, desta forma caso você use as ferramentas `flex` e `bison` é necessário planejamento na construção da gramática para que pouca memória seja usada ou o `bison` poderá apresentar erros de sintaxe que não existem e não conseguir ler todo o arquivo.

## Critérios de Correção

A saída gerada pelo seu programa será comparada com uma saída padrão e deve possuir exatamente o mesmo conteúdo para ser considerada correta.

**SUGESTÃO:** Espera-se que cada equipe gere sempre o mesmo resultado de saída, desta forma uma maneira de validar o trabalho é comparar a sua saída com a saída de uma outra equipe. Cada equipe possui a sua implementação, mas todas devem chegar ao mesmo resultado final. Se duas equipes tiverem resultados diferentes, isso implica que uma delas (ou ambas) gerou um resultado incorreto, bastanto analisar em mais detalhes o grafo que gerou a divergência e assim corrigir qualquer problema de implementação.

## Especificações de Entrega

O trabalho deve ser entregue no *moodle* em um arquivo .zip com o nome `regalloc.zip`. Este arquivo .zip deve conter somente os arquivos necessários à compilação, sendo que deve haver um `Makefile` para a geração do executável.

A entrega deve ser feita exclusivamente no *moodle* até a data/hora especificada. Não serão aceitas entregas atrasadas ou por outro meio que não seja o *moodle*.

**Observação:** o arquivo .zip não deve conter pastas, para que quando descompactado, os fontes do trabalho apareçam no mesmo diretório do .zip. O nome do executável gerado pelo `Makefile` deve ser `regalloc`.

**IMPORTANTE:** Arquivos ou programas entregues fora do padrão receberão nota `ZERO`. Entende-se como arquivo fora do padrão aquele que tenha um nome diferente de `regalloc.zip`, que contenha subpastas ou não seja um `.zip`, por exemplo. Entende-se como programa fora do padrão aquele que não contiver um `Makefile`, que apresentar erro de compilação ou o nome do executável for diferente de `regalloc`, por exemplo.