Universidade Estadual de Londrina

DC Departamento de Computação

1COP029 - Trabalho T1: Alocador de Registradores

Utilizando os conhecimentos adquiridos na disciplina até o momento, implemente um programa que realiza a leitura de arquivos que contêm grafos de interferência e tenta alocar registradores através da técnica de coloração de grafos. Cada arquivo possui um único grafo no seguinte formato:

```
Grafo 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
```

Descrição do Arquivo de Grafos

A linha `Grafo 1:` indica o início de um grafo de interferência e o seu respectivo número.

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
```

Versão deste texto: V. 2023.01

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.

Programa Alocador e Saída Esperada

A sua implementação deve tentar colorir cada um dos grafos dos arquivos de entrada com o número de cores que é fornecido na linha que contém o valor de `K` até o valor 2 de forma decrescente. Assim, se o valor de `K` for 21, o seu programa deve tentar colorir o grafo utilizando 21 cores; depois deve tentar colorir o grafo com 20 cores; depois com 19 cores e assim sucessivamente até que apenas 2 cores sejam utilizadas.

Observe que ao tentar colorir o grafo com:

- `K` cores: você poderá utilizar as cores (registradores físicos) de `0` até `K-1`.
- `K-1` cores: você poderá utilizar as cores (registradores físicos) de `0` até `K-2`.
- `K-2` cores: você poderá utilizar as cores (registradores físicos) de `0` até `K-3`.
- ...
- `4` cores: você poderá utilizar as cores (registradores físicos) de `0` até `3`.
- `3` cores: você poderá utilizar as cores (registradores físicos) de `0` até `2`.
- `2` cores: você poderá utilizar as cores (registradores físicos) de `0` até `1`.

Caso o arquivo de testes contenha, por exemplo, o valor de `K = 10` e você estiver tentando colorir o grafo com 2 cores (registradores `0` e `1`), tenha em mente que os registradores físicos de `2` até `9` podem aparecer no grafo e ainda estar interferindo com registradores virtuais; você apenas não pode utilizar os mesmos na sua alocação.

De forma que o resultado seja padronizado, algumas regras devem ser seguidas durante a alocação. As fases do processo de alocação e suas regras são as seguintes:

- **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`. Nesta etapa, você deve sempre remover o nó que contiver o menor grau. Se o seu valor de `K` for 4 e no grafo houver um nó com grau 3 e outro com grau 2, você deve escolher o nó de grau 2. Se por acaso dois ou mais nós diferentes possuirem um grau que seja igual ao mínimo, você deve escolher o nó de menor número do registrador virtual. Como exemplo, considere o caso onde o valor de `K` é igual a 4 e no grafo o menor grau é o valor 2, o qual ocorre nos nós `77` e `99`,

você deve enviar para a pilha o nó `77` pois ele possui o menor número de registrador virtual. Os nós removidos devem ser inseridos em uma pilha. A ação de empilhar um nó deve ser indicada na saída do seu programa.

- **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. A ação de selecionar um nó como *potencial spill* deve ser indicada na saída do seu programa.

- **Select/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. 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, por exemplo, ao reintroduzir no grafo o nó `77` e que o mesmo possa receber as cores 10, 15, 19, 20; a escolha deve ser pela cor de número 10 que é a de menor número. A ação de desempilhar um nó da pilha e atribuir uma cor ao mesmo deve ser indicada na saída do seu programa.

- **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 se cada um dos grafos foi colorido com sucesso ou se gerou um *spill*. Como exemplo, considere que o seu programa recebeu como entrada um arquivo que contém o seguinte grafo:

```
Grafo 666:
K=4
12 --> 13 14 15 20 21
13 --> 12 21
14 --> 12 20 19 21
15 --> 16 19 12 21
16 --> 19 15 21
17 --> 19 18 20
18 --> 19 17
19 --> 17 18 14 20 15 16
20 --> 19 17 14 12
21 --> 14 13 12 15 16
```

O grafo possui o número `666` e possui 4 registradores físicos. Você deve tentar colorir o grafo utilizando primeiramente 4 cores (`0` até `3`), depois 3 cores (`0` até `2`) e por final 2 cores (`0` e `1`). Para o Grafo 666 apresentado como exemplo, o seu programa deve gerar a seguinte saída:

```
Graph 666 -> Physical Registers: 4
---------------------------------------
---------------------------------------
K = 4

Push: 13
Push: 18
Push: 17
Push: 16
Push: 15
Push: 19
Push: 20
Push: 12
Push: 14
Push: 21
Pop: 21 -> 0
Pop: 14 -> 1
Pop: 12 -> 2
Pop: 20 -> 0
Pop: 19 -> 2
Pop: 15 -> 1
Pop: 16 -> 3
Pop: 17 -> 1
Pop: 18 -> 0
Pop: 13 -> 1
---------------------------------------
K = 3

Push: 13
Push: 18
Push: 17
Push: 12 *
Push: 20
Push: 14
Push: 19
Push: 15
Push: 16
Push: 21
Pop: 21 -> 0
Pop: 16 -> 1
Pop: 15 -> 2
Pop: 19 -> 0
Pop: 14 -> 1
Pop: 20 -> 2
Pop: 12 -> NO COLOR AVAILABLE
---------------------------------------
K = 2
```

```
Push: 19 *
Push: 18
Push: 17
Push: 12 *
Push: 13
Push: 20
Push: 14
Push: 15 *
Push: 16
Push: 21
Pop: 21 -> 0
Pop: 16 -> 1
Pop: 15 -> NO COLOR AVAILABLE
---------------------------------------
---------------------------------------
Graph 666 -> K = 4: Successful Allocation
Graph 666 -> K = 3: SPILL
Graph 666 -> K = 2: SPILL
```

Na saída apresentada existe uma linha que indica o número do grafo e o total de registradores físicos disponíveis (valor de `K`), que no arquivo de entrada é 4. A seguir a saída apresenta as tentativas de se colorir os grafo com 4 cores, depois 3 cores e finalmente 2 cores. A cada tentativa é indicado o valor de `K` sendo utilizado.

Observe que cada vez que um nó é removido do grafo e enviado para a pilha, é gerada uma linha contendo a mensagem `Push:` seguida do número do nó empilhado. A linha que contém o texto `Push:  12` indica que o nó de número 12 foi removido do grafo e enviado para a pilha.

Quando um nó é selecionado como *potential spill* ele é marcado com um `*`. Na saída apresentada ao se utilizar 3 cores, foi necessário escolher o nó `12` como *potential spill*. Tal indicação pode ser observada na linha que contém o texto `Push:  12 *`, onde o `*` após o valor 12 faz a marcação de *potential spill*.

Quando todos os nós do grafo estiverem na pilha, se inicia o processo de coloração que consiste em desempilhar os nós e lhes atribuir uma cor. No arquivo de saída tal ação é indicada pela mensagem `Pop:`. Considere a linha que contém o texto `Pop:  21 -> 0`; ela indica que o nó 21 foi desempilhado e recebeu a cor/registrador de número 0. A linha `Pop:  14 -> 1` indica que o nó 14 foi desempilhado e que recebeu a cor/registrador de número 1.

Se todos os nós forem coloríveis, a saída irá indicar a cor de cada um deles. Se em algum momento do processo de coloração não for possível atribuir uma cor a um nó, a mensagem `NO COLOR AVAILABLE` deve ser emitida, como ocorreu na linha que contém o texto `Pop:  12 -> NO COLOR AVAILABLE`, indicando que quando o registrador virtual 12 foi removido da pilha e reintroduzido no grafo, nenhuma cor estava disponível para o mesmo, o que indica que é necessário realizar *spill*.

Ao final de todas as tentativas, a saída contém um pequeno resumo de todo o processo de coloração. No exemplo de saída apresentado este resumo contém o seguinte texto:

```
Graph 666 -> K = 4: Successful Allocation
Graph 666 -> K = 3: SPILL
```

```
Graph 666 -> K = 2: SPILL
```

Neste resumo é mostrado o número do grafo e o resultado da alocação para cada valor de `K`. Quando a alocação é bem sucessida a mensagem `Successful Allocation` é mostrada, caso contrário a mensagem `SPILL` é mostrada.

Implementação e Critérios de Correção

O seu programa deve ser implementado exclusivamente em `C` ou `C++`.

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.

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

Recomendações

Por favor, evite escrever código da seguinte forma:

```
for(int i = 0; ...)
{
    ...
}
```

onde uma variável local está sendo declarada dentro de um comando. Se ainda sim quiser utilizar tal estilo de programação, não se esqueça de colocar no `Makefile` as devidas opções para que o compilador aceite tal construção, pois nem todos os compiladores a aceitam por padrão.

Em geral a opção `-std=c99` é o suficiente para que tal construção seja aceita e compilada sem maiores problemas. Sua utilização, em geral, é da seguinte forma:

```
$gcc -std=c99 teste.c -o teste
```

Se você programar utilizando `C++ 11`, por favor, utilize também a opção adequada do compilador para habilitar a compilação de tal versão do `C++`.

IMPORTANTE: Se ficou com alguma dúvida em relação a qualquer item deste texto, não hesite em falar com o professor da disciplina, pois ele está à disposição para sanar eventuais dúvidas, além do que, isso faz parte do trabalho dele.

SUGESTÃO: Espera-se que cada aluno gere sempre o mesmo resultado de saída, desta forma uma maneira de validar o trabalho é comparar a sua saída com a saída de um outro colega. Cada aluno possui a sua implementação, mas todas devem chegar ao mesmo resultado final. Se dois alunos tiverem resultados diferentes, isso implica que um deles (ou ambos) gerou um resultado incorreto, bastando 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 *AVA* 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 *AVA* até a data/hora especificada. Não serão aceitas entregas atrasadas ou por outro meio que não seja o *AVA*.

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`.

O programa gerado deve ler as suas entradas da entrada padrão do sistema e imprimir as saídas na saída padrão do sistema. Um exemplo de execução para uma entrada chamada `grafo.txt` seria a seguinte:

```
$./regalloc < grafo.txt
```

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 que não seja um `.zip` por exemplo. Entende-se como programa fora do padrão aquele que não contiver um `Makefile`, que apresentar erro de compilação, que não ler da entrada padrão, não imprimir na saída padrão ou o nome do executável for diferente de `regalloc`, por exemplo. Uma forma de verificar se seu arquivo ou programa está dentro das especificações é testar o mesmo com o `script` de testes que é fornecido no *AVA*. Se o seu arquivo/programa **não** funcionar com o `script`, significa que ele está **fora** das especificações e, portanto, receberá nota **ZERO**.