Departamento de Ciência da Computação - IME - USP

MAC0328 Algoritmos em Grafos

Tarefa 3 Entrega: até 30 de abril de 2008

Este projeto foi originalmente elaborado
pelo professor Francisco Reverbel

Make

OK, let's be straight about it,
the syntax of make is really stupid.
If you use spaces where you're supposed to use tabs or vice versa,
your makefile blows up. And the error messages are really confusing.

Fonte: Running Linux, Matt Welsh and Lar Kaufman

Objetivo

1. Manipulação da representação de um digrafo através de **vetor de listas de adjacência**.

Descrição

Introdução

Neste exercício-programa você fará um programa em `C` que se chamara `Make.c`. O programa utilizará as **estruturas de dados** das **notas de aulas** do professor **Paulo Feofiloff** para a representação de um "digrafo de dependência" através de **vetor de listas de adjacência**, como descrito mais adiante.

O seu programa deverá chamar-se `Make`. O `M' maiúsculo em `Make` é para diferenciar o seu programa do utilitário **`make`** do `UNIX`. Da mesma maneira que o `make`, o seu programa, ao ser chamado na linha de comando sem opção alguma, lerá um arquivo de nome `MakeFile` contendo informações de dependência e comandos para reconstrução (*rebuilding commands*) de objetos. O `F` maiúsculo em `MakeFile` server para não confundirmos a entrada do `Make` com o arquivo `Makefile`; que vocês muito provavelmente estarão utilizando.

Formato de um `MakeFile`

Um `MakeFile` consiste de uma seqüência de "regras" com o seguinte formato:

```
<target> :  <dependências>
        <comando1>
        <comando2>
        ...
```

Um `target` é usualmente o nome de um arquivo que é gerado por um programa; exemplos de targets são arquivos executáveis (bin ou .o). Um target também pode ser o nome de uma ação, tal como "`clean`".

Uma `dependência` é um arquivo que é usado para criar o `target`. Um `target` pode depender de vários arquivos.

Um `comando` é uma ação que `Make` pode mandar que seja executada. Um regra pode ter mais que um comando, cada um em uma linha. **Deve existir** um caractere `TAB ('\t')` no início de cada linha que possui um comando.

Exemplo de um `MakeFile`

A seguir está um exemplo típico de um arquivo `MakeFile` que seu programa `Make` deve ser capaz de tratar (os números das linhas foram colocada apenas para efeito de referência, eles não fazem parte do `MakeFile`):

```
1   meuprog: meuprog.o  fila.o  etc.o
2         gcc meuprog.o fila.o etc.o -o meuprog
3
4   meuprog.o:  meuprog.c  minhasdefs.h  fila.h   etc.h
5         gcc -c meuprog.c
6
7   fila.o:  fila.c minhasdefs.h  fila.h
8         gcc -c fila.c
9
10  etc.o: etc.c  minhasdefs.h  etc.h
11        gcc -c etc.c
12
13  clean:
14        rm -f meuprog.o fila.o etc.o
15        cp meuprog.c ../ultimo/meuprog-salvo.c
16        cp fila.c  ../ultimo/fila-sava.c
```

Este exemplo ilustra o formato do arquivo `MakeFile` que o seu programa deve tratar:

Linhas 1, 4, 7, 10 e 13 fornecem informações de dependência. Em uma linha de dependência o nome de um `target file` aparece primeiro, seguido por dois pontos (':'). Depois dos dois pontos segue-se uma lista (possivelmente vazia) de arquivos dos quais o `target` depende. A Linha 7, por exemplo, diz que o `target` file `fila.o` depende dos arquivos `fila.c, minhasdefs.h` e `fila.h`.

Uma linha de dependência é seguida por uma ou mais linhas com comandos (`rebuild commands`) que criam ou atualizam o target file especificado na linha de dependência. O primeiro caractere de um linha contendo um comando **deve** ser um `TAB` (`\t'`); uma, aparentemente idêntica, seqüência de espaços **não** pode ser usada no lugar do `TAB`. Li nhas 2, 5, 8, 11 e 14 contêm comandos. A Linha 8, por exemplo, diz que o comando

```
meu_prompt> gcc -c fila.c
```

deverá ser executado para reconstruir o target `fila.o` a partir dos arquivos `fila.c, minhasdefs.h` e `fila.h`. Neste particular `MakeFile`, cada linha de dependência é seguida por apenas uma linha de comando. Seu programa, entretanto, deve ser capaz de tratar o caso em que várias linhas de comando seguem uma linha de dependência.

As linhas em branco (linhas 3, 6, 9 e 12) são ignoradas. Estas linhas podem estar presente apenas para facilitar a legibilidade do `MakeFile`, mas elas não são necessária. Um `Makefile` similar, mas sem linhas em branco, também deve ser aceito pelo seu `Make`.

Digrafo de dependências

As linhas de dependências em um `MakeFile` (ou `Makefile`) definem um *digrafo de dependências*: um digrafo onde os vértices correspondem a `targets` e arquivos e os arcos representam a dependência entre estes `targets` e arquivos. Sempre que um arquivo $\texttt{arquivo}_1$ depende de um arquivo $\texttt{arquivo}_2$ o correspondente digrafo de dependências deve conter um arco do vértice de nome $\texttt{arquivo}_1$ ao vértice de nome $\texttt{arquivo}_2$. A Linha 7 do nosso exemplo de `MakeFile` acima diz que o respectivo digrafo de dependências contém arcos do vértice de nome `fila.o` ao vértice de nome `fila.c`, ao vértice de nome `minhasdefs.h`, e ao vértice `fila.h`.

Um ***ciclo*** em um digrafo é caminho fechado. Mais precisamente, um ciclo é uma seqüência de vértices $v_1\text{-}v_2\text{-}\ldots\text{-}v_k$ tal que

> existe um arco de v_i a v_{i+1} ($i=1,\ldots,k-1$) e um arco de v_k a v_1.

Um digrafo de depêndencias **não** deve conter ciclos. Isto, entretanto, não é obstácute para que um digrafo correspondente a um arquivo `MakeFile` tenha circuitos. Isto apenas significa que o `Make` deverá ser capaz de verificar a presença ou não de ciclos no digrafo. A existência de ciclos deve ser considerada pelo `Make` como um erro na especificação do `MakeFile`. Detecção de ciclos é um dos objetos de estudo da Tarefa 4. Por enquanto não se preocupe com isto.

Representação do digrafo

O seu programa deve usar a **estruturas de dados** e **vetor de listas de adjacência** das **notas de aulas**do professor **Paulo Feofiloff** para a representação do digrafo de dependências. Considere, além de outras estrutura comuns aos programa que temos visto, as seguintes declarações:

```
static char  *nome[maxV], *comandos[maxV];
```

Para cada vértice v o seu programa deve manter em:

- `nome[v]` o ponteiro para um string que contém o nome do vértice. Exemplos de nomes de vértices do digrafo de dependências correspondente ao `Makefile` do exemplo anterior "`meuprog.c`", "`fila.h`" e "`clean`".
- `G->adj[v]` um ponteiro para uma estrutura do tipo `link` que é o início da lista ligada de vértices que são ponta final de algum arco com ponta inicial em `v`. Há um arco `v-w` para cada arquivo `nome[w]` do qual `nome[v]` depende.
- `comandos[v]` é um ponteiro para um string. Este string deverá conter todos os comandos associados ao `target` `nome[v]`, uma após o outro, separados por um um ';' (ponto-e-vírgula).

Por enquanto acho que é só isto que precisamos de um `Vertex`. No Tarefa 4 precisaremos guardar mais informações associadas a cada vértice.

Comportamento do `Make`

O programa `Make` será chamado em uma linha de comando do shell da seguinte maneira:

```
meu_prompt> Make
```

Chamado desta maneira `Make` deverá procura no diretório corrente um arquivo de nome `MakeFile` e processá-lo da seguinte forma:

1. (Contrói o digrafo de dependências) `Make` lê o arquivo `MakeFile`, que tem o formato descrito acima, e contrói uma representação do digrafo de dependência através de vetor de listas de adjacência
2. (Salva o digrafo) Após a criação do digrafo de dependência na representação descrita, `Make` percorre o digrafo e cria no diretório corrente um arquivo de nome `MakeFile.dg`. Este arquivo `MakeFile.dg` é parecido com o exemplo de `MakeFile`, possivelmente sem as linhas em branco ou trocando alguma das regras de lugar; esta ordem é irrelevante, ela depende de como o seu `Make` gerou o grafo.

Entrega

1. Este exercício-programa vale 10 pontos.
2. Como sempre, somente entregue seu programa se o mesmo não apresentar erros de compilação.
3. Junto com o seu programa você deve entregar um Makefile de tal forma que

   ```
   meu_prompt> make Make
   ```

   produza o executável de nome `Make` correspondente a sua Tarefa 3.
4. O Paca não costuma aceitar a entrega de vários arquivos. Por isto, você deve depositar um arquivo `tarefa3.tgz` com todo o seu trabalho. Espera-se que

```
meu_prompt>tar -xvf tarefa3.tgz
```

crie um diretório que tenha o seu `login` na rede Linux como nome. Neste diretório devem estar todos os arquivos da sua Tarefa 3, inclusive o Makefile que você usou. Se você achar necessário coloque junto um arquivo `00-leia-me`, onde você descreve algo que achar necessário: como limitações e bugs no seu programa

---

Last modified: Mon Apr 28 12:15:48 BRT 2008