UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
http://www.icmc.usp.br
DEPARTAMENTO DE CIÊNCIAS DE COMPUTAÇÂO

SCC0216 – Modelagem Computacional em Grafos
Prof.ª Rosane Minghim

TRABALHO 2 – RESOLUÇÃO DE DEPENDÊNCIAS

Introdução

Para quem trabalha com Linux (ou outros sistemas operacionais tipo *UNIX*), os chamados "sistemas de gerenciamento de pacotes" – como *yum*, *rpm* e, no caso do *Ubuntu* e outros sistemas *Debian* (e objeto de estudo deste trabalho), *apt* – são ferramentas importantes para a administração do sistema. Entre as funcionalidades oferecidas por essas ferramentas, duas se destacam: a) o *download* automático de pacotes; e b) a **resolução automática de dependências**. É nesta segunda funcionalidade que focaremos este trabalho.

Descrição do Trabalho

Você irá simular o processo de resolução de dependências do *apt* – *Advanced Packaging System* – para instalação e desinstalação de pacotes. A entrada consistirá em uma lista de pacotes e suas dependências, seguida de uma sequência de comandos de instalação e desinstalação a serem executados.

Considerando que uma dependência deve ser instalada sempre **antes** do pacote dependente, a sua primeira tarefa é: ao final de todos os comandos de instalação e desinstalação, fornecer a lista de todos os pacotes presentes no sistema, na sequencia em que foram instalados.

Ao saber que os estudantes da USP realizariam este trabalho, a equipe de desenvolvimento do *Debian* pediu ajuda para resolver um problema sério com seus repositórios de pacotes – as dependências cíclicas. Se um pacote A depende de B, B depende de C e C depende de A, existe uma dependência cíclica (note que, entre A e C, poderia existir um número qualquer de pacotes). Portanto, a sua segunda tarefa é: encontrar todos os grupos de pacotes que apresentam dependências cíclicas (no exemplo anterior, o grupo seria composto dos pacotes A, B e C).

Entrada

O exemplo a seguir ilustra o formato de entrada:

```
pkg_A
  Depends: pkg_B
pkg_B
  Depends: pkg_C
pkg_C
  Depends: pkg_A
pkg_D
  Depends: pkg_E
pkg_E
---
install pkg_D
install pkg_A
remove pkg_D
```

A lista de pacotes e dependências aparece primeiro, no seguinte formato:

- O nome do pacote *P* (dependente), sozinho, em uma linha. O nome terá no máximo 30 caracteres e serão apresentados no máximo 2500 pacotes.
- O nome de cada dependência de *P* nas próximas linhas, precedido por dois espaços e a expressão "*Depends:*".

Para separar os pacotes e os comandos, será introduzida uma linha com três hífens. A partir desta separação, cada linha contém um comando:

- Os comandos podem ser "*install*" ou "*remove*".
- Cada comando é seguido pelo nome de um dos pacotes especificados anteriormente.

A seção ***observações*** abaixo oferece algumas dicas para facilitar a leitura da entrada.

Saída

A saída deverá apresentar a solução para os dois problemas propostos:

1. Todos os pacotes instalados no sistema ao final da execução de todos os comandos, na ordem cronológica em que foram instalados (do mais antigo ao mais recente).
2. Cada pacote deverá estar acompanhado do identificador do seu "grupo de dependências cíclicas". As dependências cíclicas devem ser detectadas durante a instalação dos pacotes. Cada grupo deve receber um número inteiro em ordem crescente a partir de 1, de acordo com a ordem em que foi detectado. Se um

pacote não faz parte de nenhum grupo de dependências cíclicas, deve ser impresso o número 0.

Considerando o exemplo de entrada da seção anterior, a saída esperada seria:

```
pkg_E 0
pkg_C 1
pkg_B 1
pkg_A 1
```

Note que os pacotes A, B e C fazem parte do grupo de dependências cíclicas #1, detectado durante o comando de instalação do pacote A, que exigiu B, que exigiu C, que exigiu A – portanto configurando o ciclo. O pacote D não aparece pois foi removido por um comando; no entanto, sua dependência E foi mantida (leia o restante do enunciado para entender exatamente por quê).

Detalhes sobre comandos

1. Quando um pacote é instalado, todas suas dependências também são instaladas, recursivamente (ou seja, dependências de dependências).
    a. Note que uma dependência deve ser sempre instalada **antes** do pacote dependente.
    b. Se uma dependência cíclica for encontrada, como no exemplo anterior, evite *loops* infinitos resolvendo da seguinte forma: *se um pacote é dependente de outro pacote que foi* ***marcado para instalação***, *considere essa dependência como resolvida.* No exemplo anterior, *C* foi instalado primeiro (pois *A* já estava marcado para instalação), depois *B*, depois *A*.
    c. Se dois pacotes podem ser instalados em qualquer ordem, então deve ser respeitada a ordem em que os pacotes foram apresentados na entrada (para mais detalhes, ver a seção ***observações*** abaixo*)*. Considere o exemplo:

        ```
        pkg_1
          Depends: pkg_2
          Depends: pkg_3
        pkg_3
        pkg_2
        ```

        Neste caso, *pkg_2* e *pkg_3* podem ser instalados em qualquer ordem, portanto a ordem de instalação deve seguir a entrada: *pkg_2, pkg_3*, *pkg_1*.
2. Se um pacote já estiver instalado e um comando de instalação for recebido, nada deve ser feito (ou seja, o *timestamp* da instalação não deve ser atualizado).
3. Quando um pacote é removido com “remove”, por padrão nenhuma de suas dependências é removida. No entanto, todos seus dependentes devem ser removidos, ou a estabilidade do sistema estaria comprometida.

Observações

1. A restrição da ordem de saída dos pacotes que podem ser instalados em qualquer ordem é necessária para que o sistema possa corrigir o problema sem ambiguidade. Para conseguir esse resultado, sempre que for realizar uma travessia no grafo, ou seja, percorrendo as arestas de cada vértice, respeite a ordem em que os vértices foram apresentados na entrada. Quando um vértice aparece como dependência antes de ser definido formalmente, sua primeira aparição (como dependência) é a que conta. Se você utiliza uma implementação de grafo baseada em matriz de adjacências, esse resultado é natural; caso você use listas de adjacências, mantenha as arestas ordenadas pelo índice do vértice.

2. Este trabalho utiliza dois conceitos importantes da disciplina: *Ordenação Topológica* e detecção de *Componentes Fortemente Conexos.* Se estiver na dúvida sobre a implementação, releia a aula e as referências relativas a estes dois conceitos.

3. Cada caso de teste do sistema irá verificar uma pequena parte da implementação, começando com exemplos simples e continuando até os mais complexos. O comando "*install*", por exemplo, será testado primeiro separadamente e depois em conjunto com "*remove*". Além disso, os casos de teste iniciais não apresentarão dependências cíclicas, ou seja, todos os pacotes estarão acompanhados do número 0. **Se você conseguir implementar o trabalho apenas parcialmente, submeta seu programa mesmo assim; a avaliação será em cima de cada caso de teste.**

4. Não se assuste com o formato da entrada. Lembre-se que:

    - Um **scanf** sempre ignora quaisquer espaços (ou *newlines*) antes da próxima leitura; por exemplo, o comando a seguir sempre retorna a próxima string da entrada, sem espaços, não importa qual seja:

      ```
      scanf("%s", buffer);
      ```

    - Quando **scanf** chega ao final da entrada, seu retorno é **EOF**; por exemplo, o seguinte comando faz a leitura de todas as strings da entrada em sequencia:

      ```
      while (scanf("%s", buffer) != EOF) { ... }
      ```

5. A entrada, os dados e os algoritmos utilizados neste exercício são todos baseados em dados reais de sistemas *Debian*, com poucas alterações. Ao resolver este exercício você estará, na prática, resolvendo um problema real de uma ferramenta utilizada diariamente por milhares de administradores de sistema no mundo.