MAC 328 — Algoritmos em Grafos

Tarefa 2 — 4/5/2012 — entrega 31/5/2012

1 Sincronização

Um *autômato* (o termo mais preciso é *semiautômato*, mas aqui vamos usar o termo mais curto) $\mathcal{A}$ com k letras é um digrafo em que de cada vértice saem k arcos, cada um rotulado com uma letra diferente; neste caso, é permitido ter mais de um arco com mesmo início e fim, e *também são permitidos laços*. Dado um caminho em $\mathcal{A}$, seu *rótulo* é a *palavra* (string) formada pela sequência de rótulos nos arcos; note que, dado um vértice inicial v e uma palavra s, existe um único passeio começando em v e rótulo s.

Uma palavra s *sincroniza* $\mathcal{A}$ se todos os passeios com rótulo s terminam no mesmo vértice, e neste caso se diz que $\mathcal{A}$ é *sincronizado*.

O objetivo aqui é construir um programa que leia um autômato e ou

- informe que ele não é sincronizado, ou
- dê uma palavra sincronizadora curta ou
- desista por cansaço (haverá limite de tempo para rodar).

Existem alguns problemas por trás disso:

Existe algoritmo polinomial para determinar se existe palavra sincronizadora, e dá para mostrar que, se $\mathcal{A}$ tem n vértices e admite uma cadeia sincronizadora, então existe uma de tamanho $\leq n^3$ (mais abaixo aparece uma indicação de como provar isso). Por outro lado, o problema *dado um autômato, determinar o comprimento da menor palavra sincronizadora para ele*, é NP-difícil.

Por outro lado, palavras de tamanho n^3 parecem compridas demais. Há quase 50 anos foi formulada a, ainda não provada,

Conjectura de Cerny(1964): *Se um autômato com n vértices é sincronizado, então existe uma palavra sincronizadora com comprimento $\leq (n-1)^2$.*

Assim, vamos adotar a seguinte definição de *palavra curta*: palavra de comprimento $\leq (n-1)^2$.

2 O programa

Mais precisamente, seu programa deve ler da entrada padrão um arquivo descrevendo um autômato, no seguinte formato:

> PRIMEIRA LINHA: $n \quad k$ (número de vértices $n < 65536$, número de letras $k \leq 26$)
>
> n LINHAS: $i: \quad v_1 \quad v_2 \quad \ldots \quad v_k$
> indicando, para o vértice i, para onde apontam os arcos correspondendo às letras $1, 2, \ldots, k$.

Os vértices estarão numerados de 0 a $n-1$. As linhas podem não estar ordenadas.

Ele deve devolver na saída padrão uma única linha, que deve ter uma das formas seguintes:

0	—	se o autômato não for sincronizado.
$m \quad w$	—	se ele é sincronizado, w é uma palavra sincronizadora curta de comprimento m (utilize as letras minúsculas do alfabeto, em sequência, para imprimir w).
-1	—	se parou sem decidir, seguido de uma mensagem explicativa.

O programa deve ler na linha de comando o parâmetro `tempmax`: Tempo máximo em segundos para rodar o programa (default: ilimitado). Se o processamento passar desse tempo, o programa deve desistir, explicando que foi esse o caso. Também pode desistir se estourar a memória.

O programa deve ser escrito, de preferência, em C. Outras linguagens podem ser aceitas, mediante consulta prévia. O uso de bibliotecas prontas implementando EDs que são parte do programa não é proibido, mas também mediante consulta prévia.

O programa fonte deve ser entregue em um arquivo tgz, cujo nome deve identificar o autor; coloque também no início de cada arquivo fonte um cabeçalho que identifique claramente o autor. Escreva seu programa de forma que seja possível ler o código - coloque comentários adequados, capriche na aparência.

3 Grafos implícitos

No decorrer do curso, temos visto algoritmos que recebem um grafo representado por uma estrutura de dados, em um certo sentido, completa. Em muitas

situações, os grafos são **gerados** a partir de outros dados, e nem sempre vale a pena ou é possível guardar na memória a matriz de adjacência ou as listas de adjacência.

Nesse caso, vale a pena pensar em grafos (e digrafos) como um tipo abstrato de dados (no limite, uma classe, no sentido de POO). Nesse sentido, um digrafo define uma série de operações associadas a arcos e vértices:

- Dado um arco, determinar seu início e seu fim.
- Dado um vértice, iterar por todos os arcos que saem dele.
- Dado um arco e uma ponta, devolver a outra ponta.

Tudo isso é trivial com as EDs que usamos. No caso de grafos gerados, normalmente cada vértice e arco tem um *nome*, que não é, a princípio, um conveniente inteiro entre 0 e `V-1`. Nesses casos, em vez de uma ED na memória, as operações acima são realizadas por algoritmos. Ainda assim, muito do que vimos pode ser aplicado, com uma dificuldade a mais: marcação de vértices ou arcos, que era uma operação trivial, aqui pode se tornar complexa, já que é preciso associar a marca ao nome, e o conjunto de todos os nomes é ilimitado ou muito grande.

Exemplos dessa situação são os grafos associados aos processadores i-M-e da prova 1, e $\mathcal{P}(\mathcal{A})$ descrito em seguida.

4 Um pouco de teoria, ligada a sincronização

Um tico de notação. Dado $\mathcal{A}$, para cada vértice v e palavra w, $v \cdot w$ denota o vértice onde termina o caminho de rótulo w com início v. Em particular, para cada letra σ, existe um arco de v a $v \cdot \sigma$, com rótulo σ.

O *grafão* $\mathcal{P}(\mathcal{A})$ do autômato $\mathcal{A}$ é o autômato cujos vértices são todos os subconjuntos não vazios de V, e, para cada conjunto Q e cada letra σ, o arco com rótulo σ começando em Q termina em $Q \cdot \sigma = \{v \cdot \sigma \mid v \in Q\}$. É fácil provar que para conjunto Q e cada palavra w, $Q \cdot w = \{v \cdot w \mid v \in Q\}$, ou seja, é o conjunto de vértices de $\mathcal{A}$ onde se chega a partir de Q. com rótulo w.

Por conta dessas propriedades, vale o seguinte: w sincroniza $\mathcal{A}$ se e só se $V \cdot w$ tem cardinalidade 1. Assim, uma palavra sincronizadora mínima pode ser obtida com uma busca em largura em $\mathcal{P}(\mathcal{A})$. Pena que esse grafão seja enorme.

HEURÍSTICA 1: Faça uma busca em largura em $\mathcal{P}(\mathcal{A})$, tentando encontrar um conjunto unitário. Se encontrar, use informação da busca para recuperar a palavra sincronizadora correspondente, que será mínima. Se a busca terminar sem conjunto unitário, o autômato não é sincronizado. O problema é que no meio do caminho o custo de memória/tempo pode passar dos limites. Note que não é necessário gerar o grafo$\mathcal{P}(\mathcal{A})$; basta ir gerando seus vértices, conforme a busca. O que dá trabalho é descobrir se um subconjuto recém-gerado já apareceu.

Se essa heurística acima deu certo, beleza. Senão, considere a

HEURÍSTICA 2: Ela usa um subgrafo do grafão, que vou denotar por $\mathcal{P}_2(\mathcal{A})$: consiste só dos conjuntos de tamanho 2 e 1. Usando busca nesse grafo, é fácil determinar, dados dois vértices u, v uma palavra w mínima tal que $u \cdot w = v \cdot w$ (se não existir, $\mathcal{A}$ não é sincronizado). Mantenha um conjunto Q, inicialmente valendo V, e uma palavra z, inicialmente vazia. Atualização: escolha $u, v \in Q$, produza w como acima, faça $Q \leftarrow Q \cdot w, z \leftarrow zw$ (concatenação). Termine quando $|Q| = 1$.

Não é difícil mostrar que cada w tem comprimento $\leq \frac{n(n-1)}{2}$ e que ocorrem no máximo $n-1$ atualizações, logo o z na saída tem tamanho $\leq \frac{n(n-1)^2}{2}$. Com sorte, sai um z curto.

Essas duas heurísticas podem ser combinadas com uma decomposição, se o grafo não for fortemente conexo:

HEURÍSTICA 3: Suponha que é possível particionar $V = S \cup T$, de forma que nenhum arco vai de T para S. Considere os dois autômatos:

$\mathcal{A}_T$ É simplesmente o subgrafo de $\mathcal{A}$ induzido por T.

$\mathcal{A}_S$ Consiste do subgrafo de $\mathcal{A}$ induzido por S mais um vértice novo t. Transforme cada arco ligando um vértice de S a T em um ligando o mesmo vértice a t, como mesmo rótulo. Ajunte também laços em t com todos os rótulos.

Fácil ver que $\mathcal{A}$ é sincronizado, se e só se $\mathcal{A}_S$ e $\mathcal{A}_T$ o são. Como parte da prova, mostra-se que se z sincroniza $\mathcal{A}_S$ e w sincroniza $\mathcal{A}_T$, então zw sincroniza $\mathcal{A}$. E, como diz a propaganda, tem mais: se z e w são curtas para os respectivos autômatos, então zw é curta para $\mathcal{A}$.