Projecto I
Análise e Síntese de Algoritmos
2011/2012

Relatório de Projecto

62462 Miguel Fonseca
<miguelcsfonseca@ist.utl.pt>

I. Introdução

Este relatório descreve uma solução para o problema do carteiro (*Chinese postman problem*), baseada no algoritmo de Hierholzer, adaptado para responder a um critério lexicográfico de selecção da solução. É feita uma breve análise da complexidade da solução, referindo nuances trazidas pela implementação concreta da solução, escrita em C e usando estruturas de dados simples.

II. Descrição do problema e sua solução

Por trás do problema do carteiro jaz a teoria dos circuitos eulerianos. Estes definem-se como circuitos que, num grafo não-dirigido, percorrem cada arco uma e uma só vez e, sendo ciclos, acabam no mesmo vértice em que começam. Modelando as ruas como arcos e as suas intersecções como vértices dum grafo, torna-se óbvio que resolver o problema do carteiro passa por encontrar um circuito euleriano para a área do carteiro.

Um grafo G admite um circuito euleriano se for dito euleriano. Dois critérios devem ser verificados para tal. O primeiro é imediato: G deve ser conexo (*connected graph*); no nosso problema, é expectável que todas as intersecções (vértices) possam ser atingidas a partir de qualquer uma delas seguindo uma dada sequência de ruas (arcos). O segundo é o de que cada vértice tem valência par. Intuitivamente, pode-se pensar nisto da seguinte forma: cada vez que se visita um vértice a partir dum arco, quer-se sair por outro arco; assim, uma visita em que não se fique “preso” num vértice implica consumir dois arcos do vértice. Enquanto restarem arcos por

consumir, quer-se continuar a visitar vértices de forma a ser possível, no final, regressar, por arcos não consumidos, ao vértice de partida.

Estas condições verificadas, é possível construir um circuito euleriano. Para tal, um algoritmo ao mesmo tempo simples e razoavelmente eficiente é o de Hierholzer. Informalmente:

- toma-se um vértice de partida e segue-se uma rota qualquer de vértices, consumindo os arcos percorridos pelo caminho, até se voltar ao vértice de partida e este já não ter arcos de saída não consumidos;
- enquanto restarem arcos por consumir no grafo, toma-se um novo vértice inicial, um que tenha arcos de saída não consumidos, e segue-se uma nova rota, parando sob a mesma condição que na primeira iteração;
- cada nova rota desenhada pode então ser fundida na anterior, juntando as duas num vértice que tenham em comum.

No final, acaba-se com uma única rota – um circuito euleriano – resultante das fusões progressivas de sub-rotas.

No entanto, visivelmente, este algoritmo permite, consoante a sua implementação, obter diferentes circuitos para um mesmo grafo. Ora, o problema do carteiro admite como solução um só circuito específico para cada grafo válido – trata-se do caminho cujo *output* é o menor por ordem lexicográfica. O *output* do caminho é a sequência de vértices visitados, cada um dos quais identificado por um inteiro. Para garantir que se obtém o circuito certo, quer-se então, dentro das possibilidades do grafo, consumir em primeiro lugar os arcos entre os vértices de menor número. Assim, para cada sub-rota, a partir dum vértice inicial, face a uma escolha entre dois arcos, escolhe-se aquele que leva ao vértice de menor número. Tomando o exemplo *Input 1* do enunciado, a primeira sub-rota obtida, partindo do vértice 1, é *1 2 4 3 5 1*.

Resta todavia a questão da fusão de sub-rotas. No exemplo *Input 1*, depois de se percorrer a primeira sub-rota, restam os arcos (lembre-se que são não-dirigidos) *2-5*, *5-4*, *4-6* e *6-2*. Poder-se-ia pensar que a segunda sub-rota fosse começar no menor vértice, *2*, obtendo-se *2 5 4 6 2*. No entanto, isto implicaria que, a seguir, se fizesse a fusão das duas sub-rotas num vértice *2*

da primeira sub-rota, *1 2 4 3 5 1*. Ora, como se vê, a tendência é para que, em cada sub-rota, os vértices de menor número estejam distribuídos mais para o início da sequência. Assim, sendo a fusão traria vértices de maior número (aqui, *4*, *5* e *6*) para o início do *output*. Isto resolve-se fazendo a escolha certa do vértice inicial para cada sub-rota: quer-se o vértice *v* visitado por último (excluindo o da ponta, que fecha o caminho) na sub-rota anterior tal que ainda haja arcos por consumir com *v* como extremidade. No nosso exemplo, o último vértice visitado pela primeira sub-rota é *5*, e ainda restam arcos de saída para ele, por isso é seleccionado como vértice de partida para a segunda sub-rota, e obtém-se para esta *5 2 6 4 5*. A fusão faz-se então precisamente no último vértice *5* da primeira sub-rota, e, não restando arcos por consumir, obtém-se o circuito *1 2 4 3 5 2 6 4 5 1*.

III. Análise da solução e sua implementação

O programa age em três tempos: uma fase de preparação, uma de acumulação de sub-rotas, e uma de fusões sucessivas.

Na fase de preparação, cria-se, à medida que se lê de *standard input*, para cada vértice *v* do problema, uma lista dos vértices para os quais *v* tem arcos. Usam-se para isto listas simplesmente ligadas. Cada inserção de um vértice (representado por um inteiro) numa lista é feita ordenadamente. Assim, para conter todas as listas de vértices, inicializa-se um vector de tamanho linear com V e inicializam-se V listas. De seguida, para popular as listas, fazem-se $2 \times E$ inserções – o grafo sendo não-dirigido, faz-se o dobro das inserções. A complexidade das inserções depende de V e E: um grafo denso vai ter listas mais longas com mais trabalho de ordenação a fazer. Se supusermos que uma lista tem em média $\mathrm{N} = E \div V$ nós, e considerando que uma inserção tem um custo no pior caso de $O(N)$, a fase de preparação tem um desempenho médio:

$$V \times 2 \times E \times O(N) = O(E^2).$$

No fim desta fase, verifica-se que o problema tem solução, o que corresponde a verificar que todos os vértices têm valência não-nula e par. Para isso, inquire-se V listas quanto ao seu tamanho. À hora da redacção, consultar o tamanho de uma lista nesta implementação é linear com o seu tamanho, mas é trivial manter um contador e tornar a consulta de tempo

constante. Assim, esta verificação tem custo:

$V \times O(1) = O(V)$.

Para construir uma sub-rota, determina-se primeiro o vértice inicial. Para isso, examina-se a sub-rota anterior de trás para a frente até se encontrar um vértice candidato. Isto, na prática, é trivial, o candidato encontrando-se na cauda da sub-rota; uma análise mais rica procuraria determinar a complexidade desta procura. Notaremos o seu tempo $P(V, E)$. De seguida, tendo-se o vértice inicial v_1, a construção da sub-rota faz-se fazendo `pop` de um vértice v_2 da lista de vértices, `remove` de v_1 da lista de v_2, e `push` de v_2 na lista da sub-rota. As operações `pop` e `push` são $O(1)$. `remove` implica encontrar a primeira ocorrência de v_1 na lista de v_2; esta procura é linear com o tamanho da lista. Assim, a construção de uma sub-rota tem custo:

$O(P(V, E) + N) = O(P(V, E) + E \div V)$.

Estimar o número de sub-rotas necessárias à resolução de um problema concreto não fará parte desta análise; conjectura-se que dependa da densidade do grafo em questão. Notando este número $iter(V, E)$, a acumulação de sub-rotas tem custo:

$iter(V, E) \times O(P(V, E) + E \div V)$.

Finalmente, para reunir todas as sub-rotas numa, itera-se sobre o conjunto das sub-rotas (portanto, $iter(V, E) - 1$ vezes) e, de cada vez, liga-se duas listas L_1 e L_2 no elemento e com a chamada `subst(L1, e, L2)`. O valor e está na cabeça de L_1 e é obtido em $O(1)$. `subst` percorre uma lista à procura de e e altera apontadores para juntar as duas listas; é assim linear com o tamanho de L_1. O conjunto das fusões tem então custo:

$(iter(V, E) - 1) \times O(E \div V)$.

Se, na implementação, tivéssemos agrupado a construção de rotas com a fusão, como é sugerido na descrição da solução, o processo inteiro teria custo:

$iter(V, E) \times O(P(V, E) + E \div V)$.

Contas feitas, tem-se os custos:

- inicialização + verificação: $O(E^2 + V)$
- ciclo principal: $iter(V, E) \times O(P(V, E) + E \div V)$

De notar que estes resultados poderiam ser melhorados indexando as listas ligadas. As operações de procura passariam assim de $O(E \div V)$ para $O(log(E \div V))$.

IV. Resultados da avaliação experimental

O programa gera, para os exemplos de *input* dados no enunciado, o *output* esperado em cada caso. Mais, passa na bateria de testes disponibilizada aos alunos. Da bateria, o teste `t08`, com *401* vértices e *80200* arcos, é passado em *1,07 s* de processamento numa máquina de 32 bits com um *core* de *1,6 GHz*. O exame da execução pela ferramenta Valgrind revela que são feitas *321 212* alocações de memória, num total de *2 569 672 bytes*. Note-se, no entanto, que o programa não passa nos últimos dois testes da bateria utilizada pelo sistema de submissão Mooshak.

V. Referências

en.wikipedia.org: artigos “Route inspection problem”, “Eulerian path”