Universidade de Lisboa - Instituto Superior Técnico
Licenciatura em Engenharia Informática e de Computadores
Análise e Síntese de Algoritmos

1º Projeto

Gonçalo Marques, 84719 Manuel Sousa, 84740

Introdução

Com este projeto pretendemos apresentar uma solução eficiente para o problema enunciado, explicar a sua implementação e fazer uma análise teórica e experimental da complexidade temporal e espacial do mesmo.

Descrição do problema

O problema consiste em relacionar um conjunto de fotos. As fotos estão dentro de uma caixa, sendo possível retirar sucessivos pares de fotos, e indicar em cada tirada qual das duas vem primeiro cronologicamente. No fim destas relações, devemos imprimir as fotografias por ordem cronológica. No entanto, algumas das ordens atribuídas aos pares podem tornar a relação das fotos incoerente (caso incoerente), sendo impossível chegar a uma ordem válida. Para além deste problema, é possível haver falta de informação nas relações de modo a chegar a uma ordem cronológica única (caso insuficiente). Todos estes casos têm de ser resolvidos eficientemente.

Ora este problema pode ser descrito como um problema num grafo dirigido, em que:

- cada fotografia é um nó
- uma ligação cronológica entre fotografias corresponde a uma aresta

Assim, o problema descrito é equivalente ao de encontrar num grafo dirigido $G(V, E)$ (com $N = \#V$ e $L = \#E$) uma ordenação topológica única.

É necessário ter em conta a deteção de ciclos no grafo, para verificar ligações incoerentes, e caso estas não existam, comprovar a existência de um caminho Hamiltoniano na ordem topológica obtida de forma a provar uma única ordem.

Algoritmo utilizado

Tendo a solução do problema sido estudada nas aulas, foi utilizado o algoritmo estudado - o algoritmo de ordenação topológica baseada em *Depth-first search* (DFS). Uma ordenação topológica de um grafo acíclico é uma ordem linear em que cada nó vem antes

de todos os nós para os quais este tenha arestas de saída [1]. A descrição do algoritmo será baseada na noção de tempo de fim, embora estes tempos não sejam calculados. São utilizadas também, estruturas para indicar o estado de visita de cada nó.

Estruturas utilizadas:

- **G**[u][**i**] - O grafo G foi representado como uma lista de adjacências (foi utilizado um std::vector (array dinâmica) em vez de std::list (lista duplamente ligada), pelo facto de a implementação do std::vector ser bastante mais eficiente que a de std::list [2]).
- **visited**[u] - O nó u foi visitado?
- **closed**[u] - O nó u foi fechado?
- **ts**[u] - Estrutura que contem nós ordenados topologicamente

Todas estas estruturas foram implementadas com o std::vector de C++ que tem tempo de inserção no fim $O(1+)$ e de acesso $O(1)$. Com tempo de inserção $O(1+)$ (constante amortizado), queremos dizer que para inserir N elementos a complexidade de pior caso é $O(N)$, apesar da complexidade de pior caso de inserção de 1 elemento não ser $O(1)$ (de facto é também $O(N)$). [2]

Explicação do algoritmo

Para o caso de um grafo dirigido com pelo menos 2 nós, é executada uma pesquisa em profundidade ao longo dos nós, a partir do nó 1. Para cada nó visitado na DFS é guardada informação relativa ao seu estado de visita (vetor visited e closed). Depois para cada nó, chama o ciclo DFS a todos os nós adjacentes não visitados. Se um nó já tiver sido marcado como visitado, então verifica-se se este já é um nó fechado (vetor closed). Caso não tenha sido fechado, então é porque forma um ciclo no grafo, tornado a solução incoerente. Um grafo dirigido é acíclico se a árvore DFS não constituir arestas para trás. Como arestas para trás são arestas (u, v) sendo que u liga ao antecessor v, numa árvore DFS, se não existirem arcos para trás então existem apenas arcos de árvore, ou seja, não há ciclos no grafo. [3]

Quando um nó termina o seu ciclo DFS, é marcado como fechado (vetor closed) e é adicionado à ordenação (vetor ts).

É preciso garantir ainda que os nós da ordenação obtida formam um caminho Hamiltoniano (i.e. caminho que passa por todos os vértices de um grafo, visitando-os apenas uma só vez [4]). Se existir um caminho Hamiltoniano então é garantido que existe apenas uma ordem topológica. Para verificar este caso, basta ter em conta todos os pares de nós consecutivos $(u, v) \in ts$: onde u é antecessor de v na ordem topológica, tenha uma aresta ligada a v. [5]

Análise assintótica temporal teórica do algoritmo

O algoritmo visita cada nó exatamente uma vez, sendo a função de DFS chamada exatamente N vezes. Como percorremos todos os nós adjacentes a um certo nó, dentro de uma chamada DFS, podemos concluir que a complexidade de todas as chamadas pode ser descrita como $\sum_{i=1}^{N} \#G[i] = L$. Assim, a complexidade do algoritmo neste caso é $O(max(N, L)) = O(N + L)$. Não é possível arranjar uma única ordenação topológica sem de facto percorrer todos os nós e arestas pelo menos uma vez, o que implica a complexidade de pior caso de $\Omega(N + L)$. Para comprovar a existência de um caminho Hamiltoniano, no máximo, teremos de percorrer todo o grafo, pelo que não altera a complexidade de pior caso.

Para as soluções incoerentes, podemos verificar a existência de ciclos em $O(N)$, pois se existir uma aresta para trás, vai ser encontrada impreterivelmente antes de visitar N nós distintos. Isto acontece porque num grafo acíclico, $|L| \leq |N| - 1$. No entanto como a complexidade temporal da inserção de N nós e E arestas num grafo representado por lista de adjacências fica em O(N + L), que é superior a O(N). Então, a complexidade de melhor caso é $O(N + L)$ e pior caso $\Omega(N + L)$.

Concluímos então que teremos sempre complexidade de melhor caso $O(N + L)$ e ao mesmo tempo complexidade de pior caso $\Omega(N + L)$, portanto estamos perante uma solução ótima.

Análise assintótica espacial

Como se verifica pelas estruturas utilizadas, o algoritmo utiliza 2 estruturas com N booleanos (visited e closed), 1 estrutura com N inteiros (ts), e uma estrutura com N vetores, mas com o total de L inteiros no seu interior. Durante a recursividade, nunca é atingido um nível de profundidade superior a N, pelo que a memória de stack tem complexidade de pior caso $O(N)$ (num grafo linear). Assim a complexidade espacial será $O(N + L)$. Se não for contabilizada a memória utilizada para representar o grafo, o algoritmo tem complexidade espacial $O(N)$.

Análise assintótica temporal experimental do algoritmo

Para se realizar a análise experimental temporal do algoritmo, foi utilizada a aproximação que cada instrução de C corresponde em média a um número fixo de instruções de CPU. Assim, em primeiro lugar procedeu-se à geração de K diferentes casos de teste aleatórios para uma das soluções possíveis, também escolhidas aleatoriamente (para cada caso foi escolhido aleatoriamente um N e um $L \geq N - 1$).

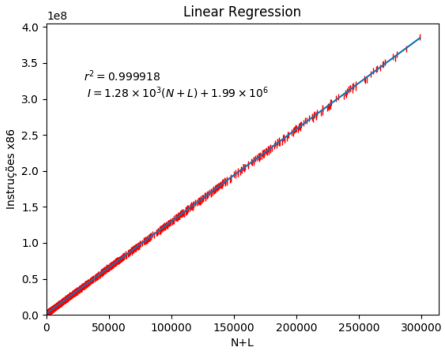

Figura 1: Análise experimental com parâmetros $2 \leq N \leq 10000$ (1000 riscos)

De seguida, para cada caso de teste foi utilizada a ferramenta perf para contar o número de instruções de CPU utilizadas em modo utilizador durante a execução do programa - I. Por fim, fez-se o plot do gráfico de I em função de $N + L$. Obteve-se assim os resultados da Figura 1.

Sendo o coeficiente de correlação linear muito próximo de 1, a análise experimental comprova o resultado teórico esperado.

Referências

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, e C. Stein. Introduction to Algorithms (3rd Edition). *Topological Sort - 22.4*, pages 612-614. MIT Press, 2009.

[2] Programming languages - c++. *ISO/IEC 14882:2003*, page 879, 2003.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, e C. Stein. Introduction to Algorithms (3rd Edition). *Depth-first search - 22.3*, pages 603-612. MIT Press, 2009.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, e C. Stein. Introduction to Algorithms (3rd Edition). *NP-Completeness - 34.2-6*, page 1066. MIT Press, 2009.

[5] Robert S. e Kevin W. Algorithms 4th Edtion. *Graphs - 4.2.25*, page 598. Addison-Wesley Professional, 2011.