Instituto Superior Técnico

Análise e Síntese de Algoritmos

Projeto 2

Grupo 64:

Jorge Rebelo Albergaria Pacheco, nº 86457

Introdução

No âmbito da cadeira de Análise e Síntese de Algoritmos pertencente à Licenciatura de Engenharia Informática e de Computadores (LEIC) da Universidade do Instituto Superior Técnico, foi proposto como segundo projeto implementar um algoritmo com o fim de separar um plano primário de um secundário, tendo em conta os seus custos respetivos, assim como o custo de vizinhança, de modo a que o custo total (ou seja, a soma de todos os custos) seja mínimo.

O algoritmo foi desenvolvido na linguagem de programação C, o que implica a implementação manual de estruturas como listas ligadas e filas, ambas necessárias ao desenrolar da execução do algoritmo.

Descrição da solução e análise teórica

A um nível básico, o algoritmo dispõe de quatro estruturas: um *node* (ou seja, vértice), um *path* (ou seja, um arco), uma componente de lista que guarda um *node*, assim como uma componente de lista que guarda um *path*.
Um *node* guarda informações básicas e necessárias para a execução do programa (coordenadas, peso primário e de cenário, a sua cor, o seu *parent*, as suas arestas e os valores de *flow* de entrada e saída), enquanto que um *path* guarda as coordenadas dos *nodes* a que está associado, assim como a sua capacidade total, o seu *flow* e uma variável binária que decide se está saturado (por outras palavras, se o seu *flow* iguala a sua capacidade total).
Ora, algumas destas variáveis associadas são necessárias ao algoritmo que será utilizado para a resolução do problema: o algoritmo de *Edmonds-Karp*, utilizando uma procura em largura.

O programa consiste em quatro fases: criação/inicialização das variáveis, aplicação do algoritmo de *Edmonds-Karp*, separação dos planos tendo em conta o corte fornecido e, finalmente, o cálculo do custo total e apresentação dos resultados.

O algoritmo começa por interpretar os *inputs* fornecidos, evocando uma função denominada por *initializeEverything*, que consiste em quatro *loops* destinados a receber as "matrizes" que contém os pesos principais, de cenário e de vizinhança. Com a exceção da matriz ter dimensões de 1x1, que faz com que a função termine imediatamente após receber o custo principal e de cenário, esta função tem uma complexidade de O(4 * (M*N)), ou seja, O(M*N), sendo M e N as dimensões da matriz principal. É de notar que os valores de *flow* de entrada e saída de cada *node* são reduzidos ao valor mínimo entre o custo primário e de cenário, de modo a reduzir o número de ocorrências em que a BFS (*Breadth-First Search*) necessita de ser invocada. De facto, estes dois valores representam dois *nodes* S e T, que representam a entrada e saída do grafo, que se encontram "ligados" a todos os *nodes* do grafo com capacidades iguais ao peso primário e de cenário, respetivamente. Em outras palavras, a complexidade desta fase será relativamente simples:

$$O(M*N).$$

Depois, o algoritmo de *Edmonds-Karp* é invocado, onde, enquanto houver um caminho de S para T, o algoritmo de procura em largura é chamado, o que possui um custo O(V + E), sendo V o número total de *nodes* (M*N) e E o número total de arcos (((M – 1) * N) + ((N – 1) * M)). O "caminho" calculado pela busca é interpretado numa função denominada *MakePath*, que percorre a lista fornecida(O(V)) e traduz o caminho para uma nova lista, passando pelos *nodes* no sentido inverso, verificando se poderia ter chegado a este *node* pelo ponto S, o que leva ao fim da função, caso contrário, passa para o *node* marcado como "pai" deste *node*(O(V^2), dado que percorre a lista de dimensão V, V vezes). Dito isto, calcula-se que a complexidade desta fase seja:

$$O((V^2) + V + E + (V*(E^2))).$$

A terceira parte do algoritmo consiste em correr o algoritmo de busca em largura uma única vez, dado que este guarda uma lista ligada contendo todos os *nodes* que o algoritmo percorreu, sendo que todos os *nodes* que são atingíveis pelo S considerados como parte do plano de cenário e os outros pertencentes ao plano principal. Dado que o algoritmo BFS é executado (O(V + E)) e o seu resultado percorrido (O(V)), espera-se que a complexidade desta fase seja:

$$O(V + E).$$

A última fase do algoritmo consiste apenas em demonstrar os resultados, o que implica percorrer a matriz do grafo uma última vez, sendo a sua complexidade:

$$O(V).$$

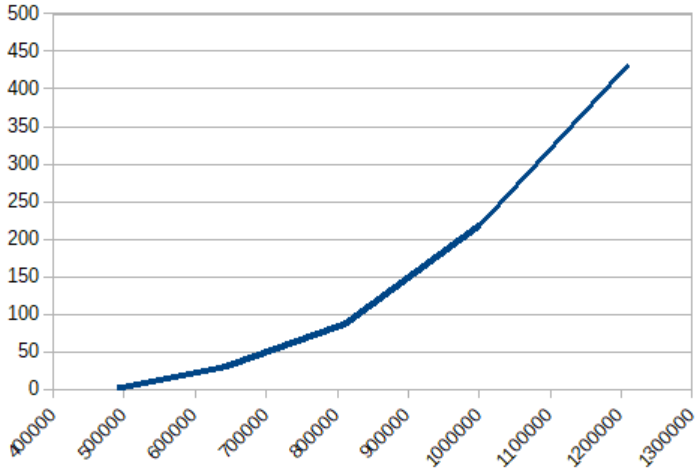
Em conclusão, estaremos a lidar com uma complexidade igual a:

$$O((V^2) + (E^2)).$$

Avaliação experimental

O modelo do algoritmo consome mais memória, de modo a melhorar o seu tempo de execução, dito isto, alguns testes públicos não puderam ser corridos (complexidade igual ao teste "t700.in" e superior), dado que o *root* do sistema terminava abruptamente a execução do algoritmo, devido ao elevado consumo de memória. Algumas mudanças poderiam ser feitas, como modificar a estrutura do *node* para guardar ponteiros das suas arestas, em vez de guardar cópias, o que poderia reduzir o espaço ocupado pelas arestas em metade, dado que cada aresta é partilhada por dois *nodes*. Ademais, poder-se-ia guardar simplesmente os *parents* de cada *node* em vez de criar uma lista ligada dos mesmos, o que elimina a necessidade de percorrer a mesma múltiplas vezes, aumentando ainda mais a eficiência do programa, reduzindo O(V^2) da sua complexidade.

Segue-se um gráfico que representa o tempo de execução, em segundos, tendo em conta o número de *nodes* inseridos como *input:*



No entanto, a progressão de memória utilizada pelo programa (número de bytes em função do número de *nodes*) apresenta uma característica mais linear: