TÉCNICO LISBOA

Análise e Síntese de Algoritmos

2º Projeto 2013/2014 – Grupo 34

70577 João Godinho
73951 Pedro Silva

Introdução

No âmbito da cadeira de Análise e Síntese de Algoritmos foi-nos proposto que resolvêssemos um problema relacionado a segurança em vários tipos de eventos, particularmente aqueles que envolvem grandes grupos de pessoas. O problema resume-se à existência de pontos ligados entre eles, no máximo, de uma forma. Certos pontos destes são designados como pontos críticos, caso haja um caminho entre os pontos críticos, a probabilidade de ser iniciado um motim é mais elevada. Foi-nos portanto incumbida a tarefa de barrar ligações de modo a isolar pelo menos um ponto crítico dos restantes. Estes cortes tem que ser feitos de maneira eficiente, de modo a que o número de cortes seja mínimo.

Solução

Depois de analisarmos o *input* e *output* dos ficheiros de exemplo, e visto que precisávamos de isolar um ponto crítico dos restantes, decidimos que a melhor abordagem seria fazer combinações possíveis entre pares de pontos críticos e selecionar o par que tiver o menor número de caminhos. Deste modo, decidimos aplicar um grafo a este problema. Tendo em conta que seria necessário calcular o número de caminhos entre cada par de pontos críticos, decidimo-nos pela implementação de uma adaptação do algoritmo de *Edmonds-Karp* com o fluxo máximo de cada arco a um. Com esta solução em mente, a adaptação do algoritmo envolvia a aplicação repetida de uma BFS removendo sempre o caminho encontrado, até não haver mais caminhos por descobrir.

O retorno da adaptação do algoritmo será então o número de vezes que a BFS encontrou um caminho. O resultado final será o valor mínimo da adaptação do algoritmo aplicada a todos os pares de pontos críticos.

A nossa solução foi desenvolvida em C, de modo que tivemos que implementar certas estruturas que nos permitiram aplicar a adaptação do algoritmo de *Edmonds-Karp.* As estruturas necessárias para o funcionamento da solução são as seguintes:

- *Graph*
  - *int nNodes:* número de nós do grafo;
  - *int *nNeighbors:* número de vizinhos de cada nó, onde o nó é o índice do vetor.
  - *int **neighbours:* lista de vizinhos de cada nó, aceder ao segundo vizinho do primeiro nó seria neighbors[0][1];
  - *int **nodesInSearch:* indica se a ligação entre nós ainda existe, onde os índices são os nós e o valor estará a TRUE se a ligação existir, FALSE caso contrário.
- *Queue*
  - *int *array:* lista que contém os nós que estão na *queue*.
  - *int size:* número de nós na *queue;*
  - *int front:* posição da frente da *queue;*
  - *int rear:* posição do fim da *queue;*

Com as estruturas descritas anteriormente foi-nos possível implementar a nossa adaptação do algoritmo de *Edmonds-Karp*.

Análise Teórica

O projeto encontra-se dividido em 2 fases: o tratamento do ficheiro de entrada e o cálculo do número de caminhos entre todos os pares de pontos críticos. Na fase do tratamento do ficheiro de entrada, o projeto terá uma complexidade O(V + E + (P * V)), onde V representa o número de nós, E o número de ligações e P o número de problemas para serem resolvidos.

Na fase de cálculo do número de caminhos entre todos os pares de pontos críticos, o projeto terá uma complexidade de O(P * V^2 * (E + V * E^2)), visto que para cada problema a ser resolvido (O(P)), teremos que fazer todas as combinações de pares de pontos críticos (O(V^2)) e aplicar o algoritmo de *Edmonds-Karp* a esse mesmo par. Tendo em conta que o algoritmo de *Edmonds-Karp* tem complexidade O(V * E^2), e que sempre que corremos o algoritmo é necessário fazer inicializações com complexidade O(E), ficamos com a fórmula em cima.

Concluímos que a complexidade temporal deste projeto é de O(V + E + (P * E) + (P * V^2 * (E + V * E^2))).

Avaliação Experimental

Para testarmos o tempo que a nossa implementação demorava a correr *inputs* de diferentes tamanhos, criámos uma pequena aplicação que recebendo um número de arcos e de problemas, criava esse mesmo número de arcos e criava arcos + 1 nós, ou seja, dado o número 4, a aplicação devolvia: 5 4 | 1 2 | 2 3 | 3 4 | 4 5, seguido de o número dado de problemas e um número aleatório de pontos críticos, conforme os limites do grafo.

Foram criados 4 ficheiros com 100, 200, 300 e 400 arcos, todos com um problema e com 2, 77, 189 e 69 pontos críticos, respetivamente, cada teste foi corrido 5 vezes numa máquina com um *i5@2.40GHz* com *4GB* de memória num sistema operativo *Kubuntu 13.10*, os testes foram corridos em linha de comandos sem tarefas que fossem dispensáveis (*e.g.:* ambiente gráfico).

Nos seguintes gráficos é possível observar o tempo de execução do algoritmo para cada ficheiro de teste.

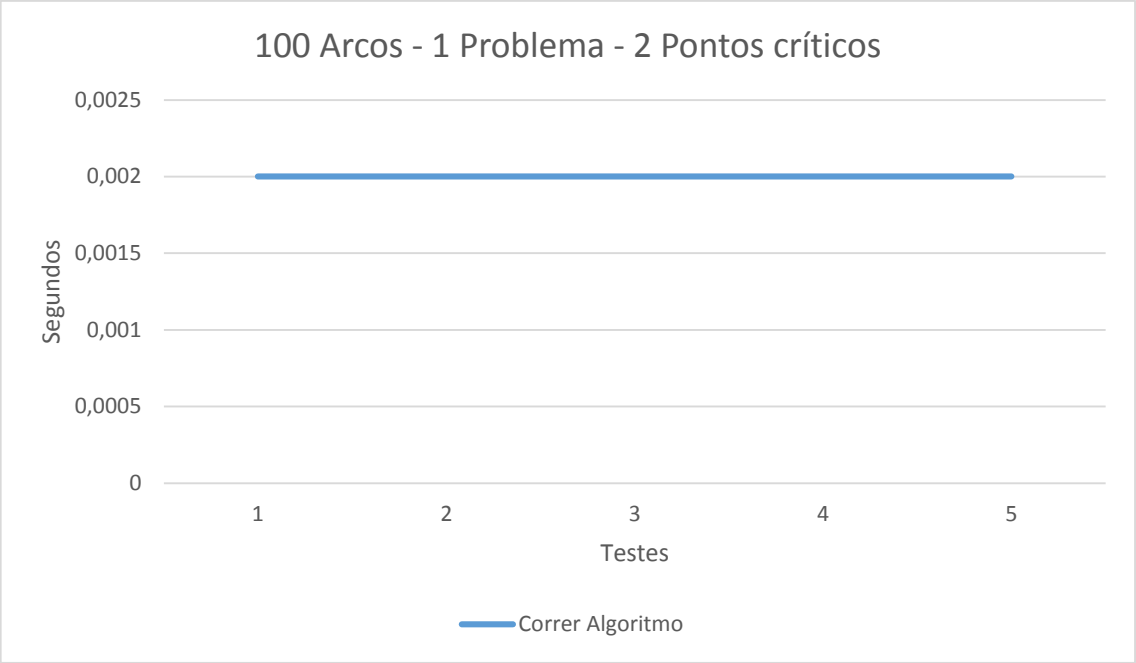

Gráfico 1

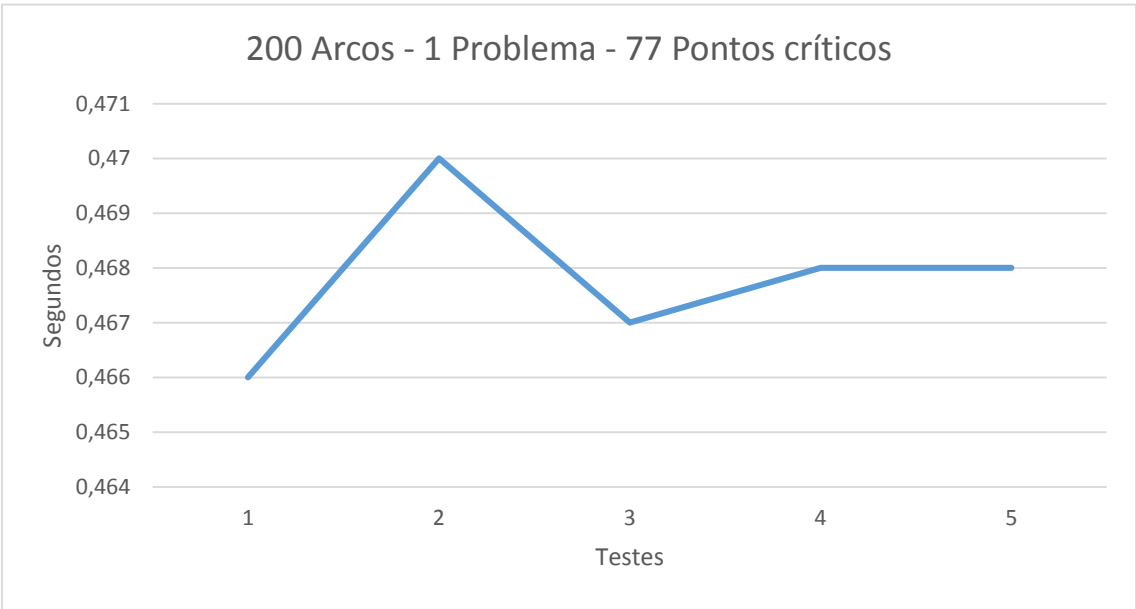

Gráfico 2

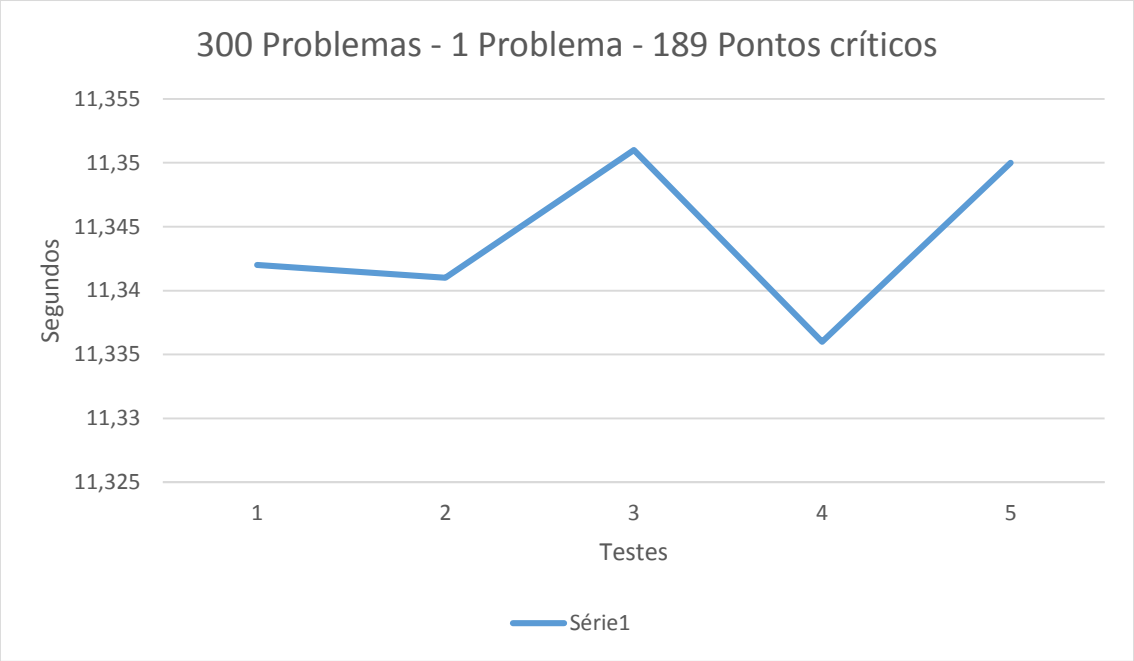

Gráfico 3

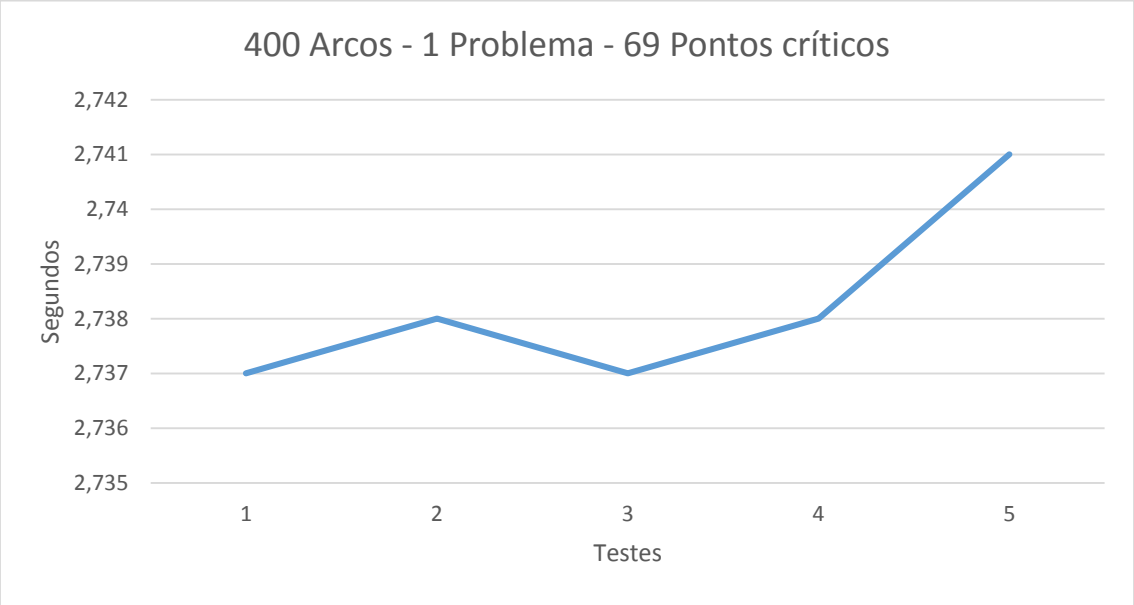

Gráfico 4