Lista 4 - MAC0328

Matheus de Mello Santos Oliveira - 8642821

1 Implementação

Queremos listar as pontes de um grafo não-dirigido, para isso as seguintes observações seguem de forma natural para ajudar a resolver o problema. Um arco visto durante uma execução de uma DFS, pode ser um arco da floresta de DFS, um arco de avanço ou um arco de retorno. Como estamos trabalhando com grafos não dirigidos não existem arcos cruzados. Para cada arco sempre há o arco antiparalelo e portanto se existisse um arco cruzado implicaria que o vértice final deste arco não visitou o vértice atual durante sua execução da DFS, o que é um absurdo, pois ela foi iniciada e finalizada antes da iteração atual. Arcos de retorno não triviais (retornam a um vértice diferente do pai do vértice em questão) ou de avanço, neste contexto, de grafos não-dirigidos, implicam que este arco pertence a um circuito e portanto não podem ser pontes. Isto restringe nossa busca somente a arcos da árvore de DFS e de retorno triviais como candidatos a pontes. Utilizando o vetor low, lowest preorder number, que representa o menor número de pre-ordem atingível por v ou algum descendente de v na árvore de DFS, podemos detectar o nosso objetivo, pois se um vértice v e seus descendentes atingem como menor pre-ordem possível v em si, isto significa que todos os seus descendentes formam um conjunto de vértices que atingem somente a si mesmos e dependem de v para atingir o resto da componente de G que estamos verificando no momento e portanto $v - p[v]$ é uma ponte. Seguindo esta ideia, implementamos o algoritmo como uma simples DFS, que calcula o vetor pre e low, assim como no algoritmo de Tarjan, mas sem precisar verificar arcos cruzados como no algoritmo de componentes fortemente conexas. Sempre que $pre[v] = low[v]$ basta verificar se o vértice em questão é uma raiz da atual árvore de DFS, ou seja, se $p[v] \neq v$, se este não for então a aresta entre v e $p[v]$ em G é uma ponte. Marcamos um vetor b da seguinte maneira, se o arco $u - v$ é uma ponte então $b[v] = u$, $b[v] = -1$ caso contrario.

2 Uso do programa

O programa recebe um arquivo que define um grafo e imprime a quantidade e os detalhes de cada uma de suas pontes. O arquivo de entrada in deve ser composto da seguinte forma: Primeira linha contém dois inteiros, o número de vértices n e o número de arestas m. Nas próximas m linhas dois inteiros por linha contendo a informação de cada uma das arestas, início e fim. Este projeto contém 3 arquivos de teste nomeados t1.in, t2.in e t3.in. Para compilar:

$ make

Para executar o código:

$ ./ep4 [in] Onde in é um arquivo com um grafo definido de acordo com as regras acima descritas. O programa exibe na primeira linha da saída o número p de pontes e p linhas cada uma delas imprimindo 2 inteiros que representam os vértices de início e fim de cada uma das arestas que são pontes. A função UGRAPHbridges(Graph G) está localizada em seu próprio arquivo bridge.c e referente bridge.h (única dependência sendo a biblioteca do curso, neste caso list.c e list.h), para ser testada de forma independente, mas enviamos junto com o projeto um teste de unidade para demonstração local do algoritmo.

3 Testes

t1.in consiste em um grafo com 4 pontes, sendo elas as arestas $0-1$, $1-2$, $1-3$ e $3-4$.

t2.in consiste em um grafo com 2 pontes, sendo elas as arestas $2-3$ e $3-4$.

t3.in consiste em um grafo sem pontes.