INSTITUTO SUPERIOR TÉCNICO

Relatório do Projecto 1

“Cadeia de Supermercados”

Miguel Coelho – 87687

LEIC-T

Introdução ao problema:

O projecto apresenta uma cadeia de supermercados que necessita de dividir a sua rede de distribuição em várias sub-redes regionais de forma a que seja possível enviar produtos a partir de quais quer dois pontos da sub-rede regional.

Considerando a distribuição actual das redes da cadeia, o objectivo é identificar todas as sub-redes regionais que já estão formadas.

Cada ponto de distribuição está identificado com um número inteiro, começando no 1, uma ligação entre dois pontos de distribuição é identificada como <id1 id2> e uma sub-rede regional é identificada com o menor identificador de todos os pontos que pertencem á sub-rede.

Descrição da Solução:

Para a representação do problema, vamos tratar a rede de distribuição como um grafo, onde cada ponto de distribuição representa um vértice do grafo e uma ligação entre dois pontos de distribuição representa um arco entre dois vértices do grafo.

Com esta representação, o problema anterior é equivalente a identificar todos os componentes fortemente ligados do grafo e todas as ligações entre eles.

Para resolver o problema foi usado como base o algoritmo Tarjan:

Foi criada uma classe para representar cada vértice que contem o seu id, o id do seu SCC, tempo de descoberta na DFS, o seu valor de low e se se encontra no stack do Tarjan.

Foi também criada uma classe Grafo que contem informação sobre as adjacências, implementada por um vector de listas duplamente ligadas, os vértices, implementada por um vector de Vértices, as ligações entre SCC's, implementada por um set de tuplos, um stack de id's e os métodos para adicionar arcos ao grafo e executar o algoritmo Tarjan.

O programa começa por ler o input dado e instanciar a classe Grafo com a lista de adjacências, um vector com V elementos e com valor uma lista ligada vazia e o vector de vértices com V elementos e com valor uma instancia de Vértice para cada elemento.

De seguida são lidas as ligações e adicionamos cada arco á respectiva lista ligada da lista de adjacências, para facilitar toda a implementação dos vértices e das suas representações em memória, foi subtraído 1 aos id's dos vértices (e consequentemente dos SCC's) para o índice do elemento das estruturas de dados corresponder ao id do Vértice, esta alteração é desfeita quando o output está a ser apresentado.

Depois de lido o input, é aplicado o algoritmo Tarjan a todos os vértices do grafo começando no Vértice 0.

Este actualiza os valores de descoberta e low para o número da iteração actual do algoritmo (começando a 0), coloca o id do Vértice no stack e actualiza o booleano do Vértice relativo á presença no stack.

E de seguida percorre os Vértices adjacentes que ainda não foram visitados ou que estão no stack. Se ainda não foram visitados, é chamada a função do algoritmo formando recursão, se já foram visitados é actualizado o valor de low para o mínimo entre o low de si próprio e o low do vértice adjacente.

Quando são esgotados os vértices adjacentes, é feita a verificação de se o valor de low é igual ao valor de descoberta, se não, o vértice não é a raiz de um SCC e retornamos para o vértice predecessor, se sim, o vértice é a raiz de um SCC e é incrementado o contador de SCC's e são removidos todos os id's dos vértices no stack até chegar ao id da raiz do SCC e colocar os Vértices que são removidos numa lista ligada de Vértices temporária.

Enquanto é feita a remoção, guardamos menor valor de id de todos os Vértices removidos, e quando a raiz for removida iteramos sobre a lista temporária e actualizamos o valor do id de SCC de cada um dos Vértices.

Acaba assim o algoritmo Tarjan, com o vector de todos os Vértices com as informações de descoberta e o id do SCC.

Para terminar a solução, percorremos novamente a lista de adjacências e para cada arco, comparamos os valores do id do SCC a que pertencem, se forem iguais, é uma ligação entre dois Vértices do mesmo SCC, se forem diferentes representam uma ligação entre dois SCC's e guardamos num set (para evitar repetições) um tuplo de dois inteiros com o valor dos id's dos SCC's que estão ligados.

De seguida é apresentado o output com os dados calculados, primeiro é apresentado o número de SCC's calculados, depois o número de ligações entre SCC's e para finalizar é percorrido o set de ligações que apresenta os tuplos que representam cada ligação de forma ordenada.

De notar que como o set é implementado por uma arvore binária equilibrada, a travessia dos elementos de forma ordenada é trivial e também que a alteração aos ids dos Vértices originais é desfeita aqui, somando 1 a cada id antes de os apresentar.

Análise Teórica:

Na inicialização do Grafo é criada a lista de Vértices feito em O(V) de seguida são lidos e adicionados os arcos à lista ligada de adjacências que tem inserção O(1), logo esta operação custa O(E).

No algoritmo Tarjan, vai ser chamada a função tarjan_visit para cada Vértice do Grafo na função tarjan O(V). Na função tarjan_visit existe o primeiro ciclo vai chamar a função recursivamente uma vez por cada arco do grafo, ou seja, custa O(V+E).

```
void Graph::tarjan() {
    curr_d = 0;
    for (int i = 0; i < V; i++) {
        if (vertex_vector[i]->d == -1) {
            tarjan_visit(i);
        }
    }
}
```

Chamada inicial de visit

```
for (Vertex *v : adj_list[u]) {
    if (v->d == -1 || v->is_on_stack == true) {
        if (v->d == -1) {
            tarjan_visit(v->id);
        }
        u_ptr->low = min(u_ptr->low, v->low);
    }
}
```

Chamada recursiva nas arestas

Quando são explorados todos arcos adjacentes de um determinado Vértice verifica-se se este é a raiz de um SCC, se sim, é executado o ciclo que remove os ids do stack em O(1), coloca o Vértice numa lista ligada em O(1) e faz a comparação entre dois inteiros em O(1), este ciclo é efectuado tantas vezes quantos Vértices existirem no SCC que está a ser calculado, em geral, visto que todos os Vértices vão pertencer a um SCC este ciclo vai ser efectuado uma vez por vértice, ou seja, O(V) o mesmo se pode dizer para o ciclo seguinte que actualiza o valor do id do SCC para todos os elementos do SCC que acabou de ser calculado, mais uma vez, vai ser executado V vezes ao longo do programa, esta condição vai ser então O(2V), ou seja, O(V)

```
do {
    v = L.top();
    L.pop();
    vertex_vector[v]->is_on_stack = false;
    SCC_list_temp.push_back(vertex_vector[v]);
    SCC_min_id = min(SCC_min_id, vertex_vector[v]->id);
} while (u != v);

for(Vertex* v: SCC_list_temp){
    v->SCC_id = SCC_min_id;
}
```

Ciclo que identifica os SCC's

Então, o algoritmo tarjan utilizado para a identificação de SCC's vai ser O(V + E).

Depois de estar calculado o id do SCC a que cada Vértice do grafo pertence, vamos iterar a lista de adjacências O(V+E), e no caso de ser um arco de ligação entre dois SCC's

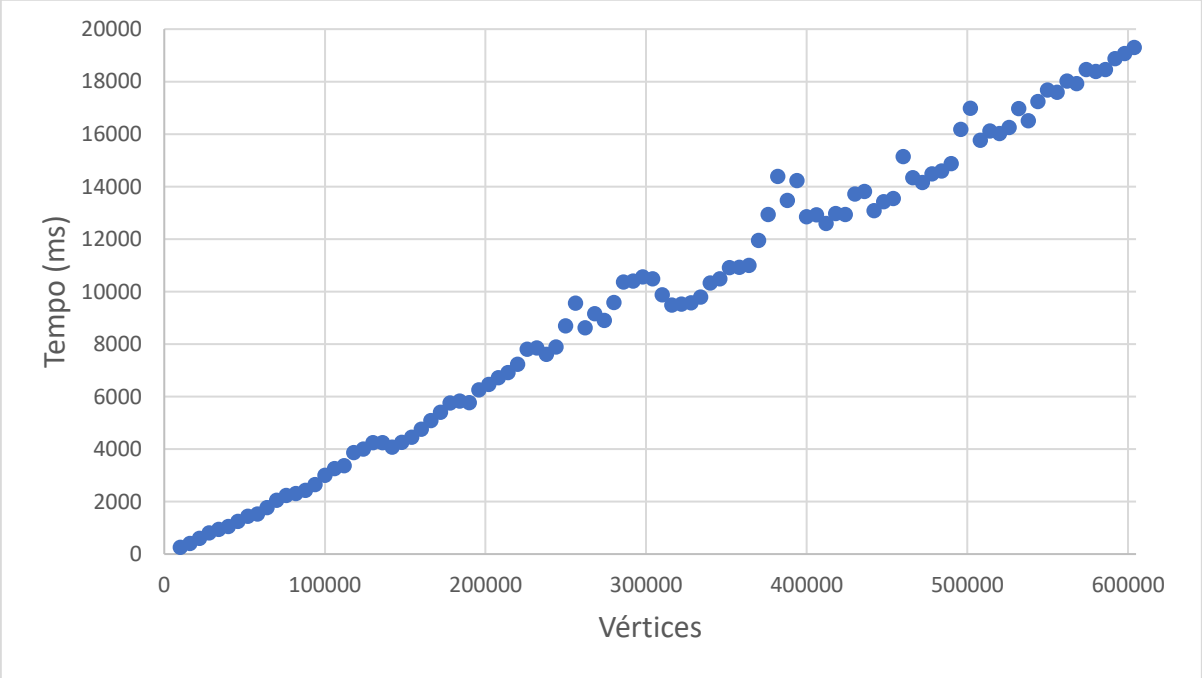
então inserimos um tuplo de dois inteiros com os ids dos SCC's a que pertencem os Vértices que formam o arco original.

Visto que o set está implementado como uma árvore binaria, o tempo de inserção é O(log(n)) logo no pior caso em que todos os arcos são ligações de SCC's este ciclo terá complexidade O((V+E) log(E)).

Para concluir, são apresentados os resultados, onde é percorrido o set de ligações, devido à implementação com uma arvore binária, é possível fazer uma travessia in-order para apresentar os resultados ordenados, ou seja, a complexidade de apresentar as ligações de forma ordenada é O(X) onde X é o número de ligações que existem entre diferentes SCC's no grafo.

Avaliação experimental:

Foram feitas 100 experiencias, onde o número de vértices varia entre 10000 e 604000 com saltos de 6000, o número de arestas é sempre 3 vezes o número de vértices, e em cada um foram criados 1000 SCC's.



Referências a material auxiliar:

Como referência, foram consultados os slides da UC de Análise e Síntese de Algoritmos do ano presente sobre o algoritmo Tarjan e a documentação da linguagem usada (C++) em: www.cplusplus.com.