Relatório de ASA

Hugo Guerreiro
83475

João Sousa
83487

May 2, 2017

1 Introdução

Foi-nos apresentado um problema que consistia em desenvolver um software para ajudar o Sr.João Caracol a construir uma rede de ligações aéreas e rodoviárias que interligue todas as cidades do Bananadistão.

Sabendo que a construção de aeroportos e de estradas tem um custo, o software tem de apresentar quantos aeroportos e quantas estradas têm de ser construídas para minimizar o valor de construção da rede ou indicar se não é possível ligar todas as cidades na mesma rede devolvendo "Insuficiente".

2 Solução

2.1 Descrição da solução

O problema proposto pode ser resolvido encontrando uma arvore abrangente de custo mínimo utilizando o algoritmo de Kruskal com conjuntos disjuntos. Podemos representar então o problema da seguinte forma: As cidade podem ser representadas como vértices de um grafo e cada aresta do grafo é uma ligação entre duas cidades. Arestas que representam ligações terrestres entre duas cidades têm como peso o custo de construir a estrada entre estas.

Cidades onde é possível construir um aeroporto ligamos esse vértice a um vértice auxiliar que representa o céu, onde a aresta que o liga tem um peso igual ao custo de construir o aeroporto na respetiva cidade.

Para achar a arvore MST primeiramente, construímos todas as estradas e corremos o algoritmo de Kruskal na lista de arestas construída O custo total desta rede , numero de cidades e aeroportos a construir são guardados para futura comparação. De seguida ligamos todas as cidades onde e possível construir um aeroporto ao céu, correndo Kruskal novamente.

Para finalizar, compara-se o custo total das duas árvores calculadas, escolhendo a configuração com menor custo. Caso seja igual, escolhe-se a que usa menos aeroportos.

2.1.1 Insuficiência do input

Uma árvore tem obrigatoriamente v-1 arestas, onde v é o numero de vértices. Logo, caso alguma das árvores calculadas não contenha v-1 arestas então o resultado obtido nessa execução do algoritmo deverá ser Insuficiente pois não é uma árvore[1].Construir o grafo a partir do input recebido

2.2 Passos da solução

1ºpasso: Receber o input inicial e criar todas as arestas do tipo estrada.
2º Passo: Correr kruskal sobre o grafo.
3º Passo: Receber resto do input, criar arestas entre as cidades com aeroportos e ligar ao vértice auxiliar.
4ºpasso: Correr novamente Kruskal sobre o grafo.
5ºpasso: Comparar o custo das árvores calculadas e imprimir a mais favorável. Caso ambas tenham um número de arestas diferente do número de vértices menos um, então imprime "Insuficiente".

2.2.1 Pseudocódigo

```
G.E ← Empty list that will contain the edges
G.V ← Empty list that will contain the vertexes
Result1 ← Instance of a structure that contains the result
Result2 ← Instance of a structure that contains the result
Get-input()
Build-Roads(G.E)
Make-set(G.V)
Result1 ← KRUSKAL(G)
Get-input()
Build-aeroports-and-append-to(G.E)
Make-set(G.V)
Result2 ← KRUSKAL(G)
Compare-results(Result1, Result2)
Output-result()
function KRUSKAL(G)    Sort G.E in increasing order
    Result ← Instance of a structure that contains the result
    n ← 0
    for each (u,v) in G.E do
        if FIND-SET(u) ≠ FIND-SET(v) then
        UNION(u,v)
        UPDATE-RESULT(result)
        n ← n + 1
```

```
            end if
        end for
        if n ≠ E.V -1 then
            Result ← "Insuficiente"
        end if
    return Result
end function
```

3 Análise

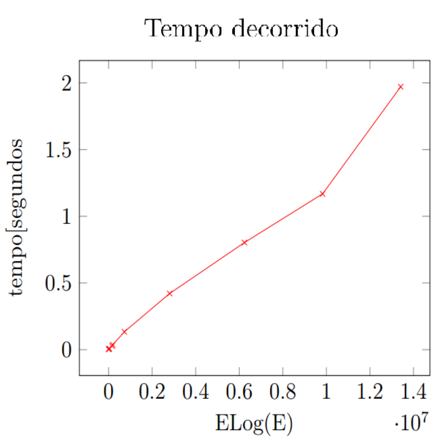
3.1 Complexidade

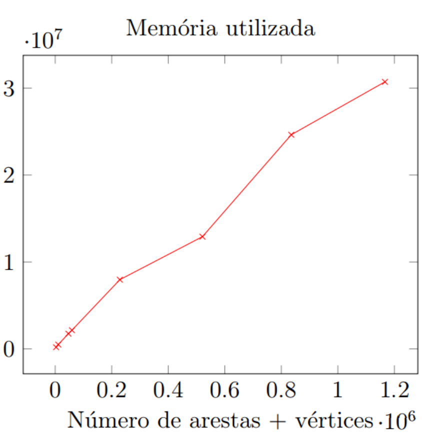
continuar a escrever Considerando as partes da solução mais relevantes podemos concluir a complexidade da seguinte forma:

- Inicializar a lista de vértices tem uma complexidade temporal e espacial de $\mathcal{O}(E)$ onde E é o numero de vértices.
- Construir todas as arestas(estradas e aeroportos) tem uma complexidade temporal e espacial de $\mathcal{O}(V)$ onde V é o número de arestas.
- Executar duas vezes o algoritmo de Kruskal é O(ELog(V)) pois utilizámos o std::sort como algoritmo de ordenação que tem uma complexidade temporal de $\mathcal{O}(ELog(E))$ e conjuntos disjuntos com as operações "find" e "union by rank" para verificar que vértices estão em cada componente.[2][3]

Podemos então concluir que o algoritmo executa-se com complexidade temporal $\mathcal{O}(E + V + ELog(E)) = \mathcal{O}(ELog(E))$ e complexidade espacial de $\mathcal{O}(E + V)$.

3.2 Demonstração de resultados





Ao observar os gráficos podemos concluir que seguem uma distribuição aproximadamente linear, pelo que podemos concluir as complexidades do ponto anterior.

4 Referências

References

[1] https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

[2] https://en.wikipedia.org/wiki/Tree_(graph_theory)

[3] Cormen, Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009) [1990]. Introduction to Algorithms (3rd ed.), MIT Press and McGrawHill.ISBN 0-262-03384-4.