ASA – Relatório do Projeto 2

Grupo 12

# - **Introdução:**

O nosso problema tem o objectivo de otimizar o custo de construir aeroportos e estradas que ligam um conjunto de cidades, o problema foi resolvida usando o conceito das árvores abrangentes com custo mínimo, o problema pode também ter mais um objectivo que é especificar as cidades e as estradas que formam a rede otimizada.

# - **Descrição da solução:**

**Modelação da Problema, Algoritmo e as estratras de Dados**

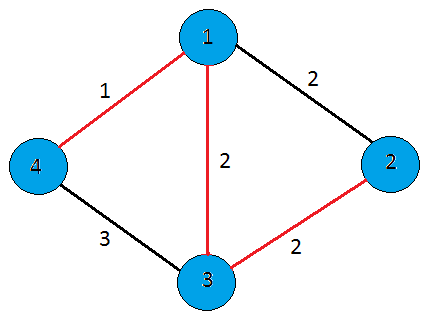
Para este problema, os aeroportos e as estradas são bem modelados por um **grafo não dirigido**, uma vez que as ligações entre estradas e aeroportos podem ser feitas em ambos os sentidos e **pesado**, com arcos do tipo:

1. aéreo: tem 0 como inicio e qualquer aeroporto como fim, tem peso positivo dado pelo custo do aeroporto, que vai ser adicionado na fase de inserir os aeroportos.
2. rodoviário: representa uma ligação entre dois cidades, tem como peso o custo da estrada, que vai ser adicionado na segunda fase de inserção.

Para se determinar o custo mínimo da rede, que tem o menor número de aeroportos, vamos aplicar o algoritmo de *Kruskal*, sendo o objetivo determinar a árvore abrangente mínima que passa por todos os vértices com o custo mínimo. Para isso, são utilizados dois grafos, em que o primeiro tem com vértices dados pelos arcos rodoviários de todos as cidades; o segundo com tem os arcos aéreos e rodoviários incluindo os arcos fictícios (espaço aéreo) da cidade adicional representada pelo 0.

As estraturas de dados usadas no nosso algoritmo são duas:

1. A primeira estratura é o grafo que é representado pelo número de vértices *V* e por um vetor de arcos que simplifica a aplicação do algoritmo de *Kruskal* (usa de espaço).
2. A segunda estrutura de dados é os conjuntos disjuntos (*Union find Sets*) que usam árvores com compressão de caminhos e união por categorias para reduzir a complexidade de procura e de ligação. Têm 3 operacoes basicas: *make\_set, find\_set, union*, e cada conjunto disjunto usa de espaço, ou seja, tem 2 vetores, *parent* e *category.*

**- O algoritmo**

No inicio, consideramos o grafo constituído somente pelas estradas, como pode ser observado na figura 1.

É possível formar uma árvore abrangente das estradas, com um custo total . Assim, temos de verificar se existir outra rede com aeroportos com um custo menor.

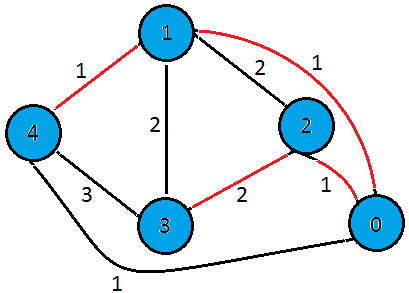
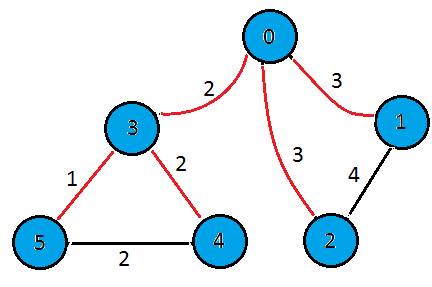


Figure 1: Modelação

Os aeroportos vão ser considerados como arcos para um vértice adicional (vértice fictício 0) como pode ser observado na figura 2. Como neste caso só há um aeroporto e temos de ter mais que 2 aeroportos, então definimos um contador, que aumenta quando adicionar o primeiro aeroporto, e depois de segundo aeroporto comeca aumentar o contador dos aeroportos incluídos.

No caso anterior, notamos que o custo atual é igual ao anterior, então escolhemos a primeira rede que reduz o número de aeroportos.

Figure 2: Adicionar arcos de 0 para os aeroportos



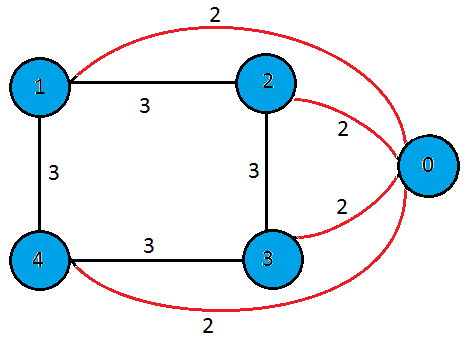
Se os arcos não formarem uma rede conexa, então simulamos o algoritmo *Kruskal* sobre o grafo dos arcos aéreos e rodoviários.

Notamos que o arco aparece antes do arcosegundo ao definição de operador(<) que compara os arcos.

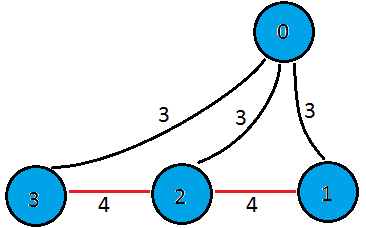
Este caso apareceu no exemplo anterior, em que o arco foi antes de arco no vetor de arcos após oredena-lo quando correr Kruskal sobre o segundo grafo.

Figure 3: Grafo nao conexo pelas estradas

Figure 4: O custo dos aeroportos é menor que o das estradas



Aqui temos aeroportos, cada um tem um custo 2, e 4 estradas com custo 3, a rede das estradas tem custo 9, que é maior que o custo de construir 4 aerportos.



Aqui temos 3 aeroportos que podem ser ligados por 2 estradas cada um de custo 4, A rede rodoviária tem custo 8 que é menor que 9. O output de *Kruskal* sobre a rede dos aeroportos e as estradas.

Figure 5: O custo dos aeroportos e maior que as estradas apesar que os arcos tem menor custo

# - **Análise teórica: A Complexidade**

A função da solução (*MST\_Solution*) pode ser analisada da seguinte maneira:

1. a ordenar os arcos, usando o operador < (definido na linha 23), para dar a prioridade aos arcos aéreos.
2. para as operações de *Find\_Set* e *Union*.
3. As operações Find\_Set e Union, (definidas na linha 61 e 66), têm complexidade calculada pela função de Ackerman , porque usamos árvores com compressão de caminhos e união por categorias.
4. A verificação da rede ser conexa, custa e tem a operação *Find\_Set*, sendo feita no tempo .

Deste modo, a complexidade total de tempo é:

O espaço usado e devido ao vetor dos arcos dos ambos grafos, do conjunto disjunto de cada Grafo, entao no total, o espaço usado .

**- Nota**

O nosso codigo passou todos os testes no Mooshak.

**- Referências**:

1. CLRS, Introduction to algorithms and data structures, Chapter23, Minimum spanning tree.
2. S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, Algorithms, Chapter5, Greedy algorithms, Subchapter1, Minimum spanning tree.
3. Quora, A diferenca entre Kruskal e Prim:

<https://www.quora.com/What-is-the-difference-in-Kruskals-and-Prims-algorithm>

1. Union Find sets by rank and path compression:

<https://en.wikipedia.org/wiki/Disjoint-set_data_structure>

1. Disjoint sets data structure ,MIT open course ware:

<https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-design-and-analysis-of-algorithms-spring-2012/lecture-notes/MIT6_046JS12_lec16.pdf>