UNIVERSIDADE ESTADUAL DE MARINGÁ

DEPARTAMENTO DE INFORMÁTICA

CIÊNCIA DA COMPUTAÇÃO

ARTHUR MANUEL BANDEIRA

67226

RESGATE EM QUEDA LIVRE - URI 1552

ALGORITMOS EM GRAFOS

DANIEL KIKUTI

MARINGÁ

SETEMBRO/2015

Sumário

1. Descrição do Problema

O problema escolhido foi o Resgate em Queda Livre - URI 1552 que consiste em conseguir uma forma de encontrar conexões entre pessoas em queda livre, as pessoas, ou melhor, vértices, podem ser representados dispostos em um plano cartesiano e estes vértices são interligados por arestas que representam a conexão entre as pessoas que fará com que estas sejam salvas todas juntas, o peso das arestas representa a distância entre as pessoas, o comprimento mínimo, em metros, de teia necessário para interligar duas pessoas.

1.1. Representação

O gráfico abaixo (Figura 1) representa a entrada abaixo na qual a primeira linha é o número de casos de teste, a segunda o número de pessoas no grupo, ou número de vértices do grafo e as linhas seguintes, as coordenadas x e y das pessoas na malha.

```
1
4
1 5
1 4
2 3
3 2
```

Os pesos das arestas são os menores custos para interligar dois vértices, o método utilizado para encontrar o menor custo entre as arestas será apresentado nas sessões seguintes.

Os vértices estão representados em sua posição no plano cartesiano *(x, y)* de acordo com os valores passados na entrada.

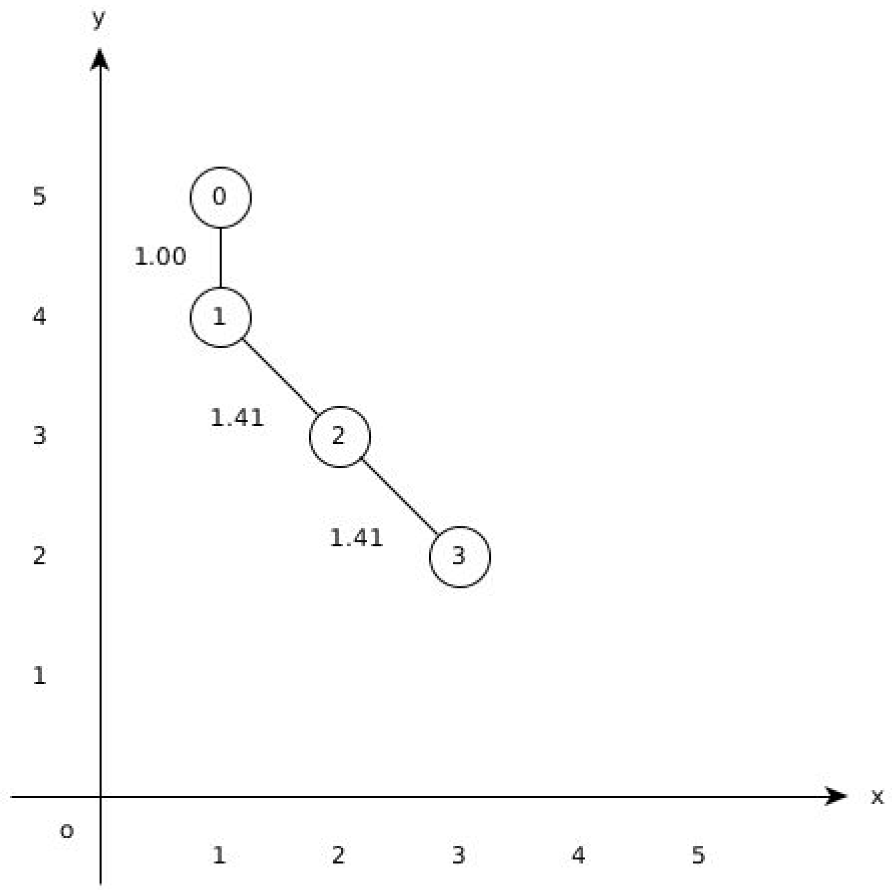

Figura 1: Representação do problema em um grafo no plano cartesiano

2. Técnica Utilizada

Como nosso objetivo é conectar as pessoas uma a uma até que todas estejam interligadas com o menor custo possível, podemos conseguir esse resultado encontrando uma Árvore Geradora Mínima, para isso aplicaremos o algoritmo de Kruskal. A aplicação do algoritmo é feita através de chamadas a partir do método *kruskal*().

Lendo a entrada e encontrando a posição das pessoas no plano cartesiano temos nossos vértices e o grafo, ainda sem arestas. Com a chamada do algoritmo de Kruskal, o método *makeSet*() inicializa os vértices definindo o próprio vértice como pai e seu ranking como 0, as informações pai e ranking serão utilizadas quando o vértice for avaliado para verificar se o mesmo pertence a árvore geradora mínima.

Em seguida o método *criaArestas*() recebe os vértices e todas as arestas são avaliadas, se a aresta encontrada não gera um ciclo com as já presentes no conjunto, esta aresta é adicionada ao conjunto. Caso a aresta já esteja presente no conjunto, com orientação invertida, a aresta é descartada. Este método também calcula o peso entre as arestas, a distância entre uma e outra.

Com o resultado do método *criaArestas*(), foi criada uma lista que é ordenada em ordem crescente de custo, a partir dessa ordem, avalia os vértices que formam a aresta no método *encontrar*(), este método retorna o pai de cada vértice e caso os pais destes vértices sejam diferentes o método *unir*() define a hierarquia, o ranking dos vértices na árvore e cria a aresta entre os vértices, garantindo assim a não existência de ciclos.

3. Estruturas de Dados

Para a leitura das entradas foi importada a biblioteca *stdin* e para uso de funções matemáticas como exponenciação e raiz quadrada a biblioteca *math*.

Para definição das variáveis que precisam ser frequentemente acessadas como ranking, pai e o grafo, foi utilizada a estrutura dicionário nativa do Python, que implementa mapeamentos, uma coleção de associações entre valores, o primeiro elemento do dicionário é a chave e o segundo o conteúdo.

No método *ler*(), os valores de entrada são recebidos e o grafo é montado, lendo as linhas com a função *stdin.readline*() e dividindo as informações da linha com a função *split*().

No método *criaArestas*(), para encontrarmos a distância entre os pontos x e y no plano cartesiano, foi utilizada a definição de Geometria que diz que “por dois pontos passa apenas uma reta”. Conhecendo as coordenadas dos pontos conseguimos definir um triângulo ABC e aplicar o Teorema de Pitágoras, assim:

- Cateto BC: $y_b - y_a$
- Cateto AC: $x_b - x_a$
- Hipotenusa AB: distância D

$$D = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$$

Com isso, utilizando as funções de raiz quadrada *sqrt*() e *pow*() da biblioteca *math*, conseguimos definir a distância entre vértices e peso das arestas.

A chamada da função *sort*() no método *kruskal*() foi utilizada para ordenar a lista de arestas por menor distância, dentro da função *sort*() foi utilizada a função *lambda* para selecionar o elemento da tupla que contém os pesos das arestas.

Para retornar o resultado do algoritmo no formato esperado, a variável total foi incrementada com as distâncias mínimas entre os vértices e para esse valor ser exibida em metros a variável ao final de sua computação foi dividida por 100.

4. Análise de Complexidade

A complexidade do algoritmo dependerá de algumas das estruturas utilizadas.

A função *sort*() utilizada para ordenar os elementos da lista de arestas tem complexidade *O(n lg n)*. As estruturas de dicionário utilizadas tem complexidade *O(n)* no pior caso e *O(1)* nos casos normais.

O método *criaArestas*(), devido às duas estruturas *for* terá tempo de execução definido pelo número de chaves no dicionário grafo é passado para a função ao quadrado. Para as entradas do algoritmo serem lidas, temos uma estrutura *for* que percorre o número de casos de teste *c* e dentro desta outro *for* percorrendo o número de vértices *n*, portanto para leitura e atribuição no grafo necessitaremos de tempo *O(c*n)*.

O algoritmo de Kruskall tem complexidade *O(E lg E)*.

5. Casos de Teste

Os casos de teste utilizados foram os dois casos já definidos na descrição do problema e mais alguns casos criados com um gerador de números aleatórios.

Todos os casos de teste foram avaliados na funcionalidade *Toolkit* do URI e coincidiram com os resultados esperados, apesar do algoritmo ter apresentado o erro *Time Limit Exceeded* quando submetido ao URI. O tempo limite para este problema no URI é de 3 segundos e segundo o avaliador o algoritmo está sendo executado com 4 segundos.

SUBMISSÃO :: 2812218

Problema:	1552 - Resgate em Queda Livre
Resposta:	Time limit exceeded
Linguagem:	Python 3
Tempo:	4.000
Submissão:	11/09/2015 - 14:43:49

Figura 2: Resultado com tempo de 4 segundos quando submetido ao URI

6. Referências

KIKUTI, Daniel. **AULA 13 – Arvores Geradoras Mínimas (MST – Minimum Spanning Trees) [Parte II].** 2015. Disponível em: <http://moodle.din.uem.br/pluginfile.php/9166/mod_resource/content/1/aula14.pdf>. Acesso em: 10 set. 2015.

ESPERANÇA, Claudio. **Python: Dicionários.** Disponível em: <http://www.dcc.ufrj.br/~sadoc/python/dicionarios.pdf>. Acesso em: 10 set. 2015.

OLIVEIRA, Gabriel Alessandro De. **Distância entre dois pontos**; *Brasil Escola*. Disponível em <http://www.brasilescola.com/matematica/distancia-entre-dois-pontos.htm>. Acesso em: 10 set. 2015.

<http://stackoverflow.com/users/224671/kennytm>. **How to sort with lambda in Python.** 2015. Disponível em: <http://stackoverflow.com/questions/3766633/how-to-sort-with-lambda-in-python>. Acesso em: 10 set. 2015.

<http://stackoverflow.com/users/367273/npe>. **What is the complexity of this python sort method?** 2013. Disponível em: <http://stackoverflow.com/questions/14434490/what-is-the-complexity-of-this-python-sort-method>. Acesso em: 10 set. 2015.

TimeComplexity. Disponível em: <https://wiki.python.org/moin/TimeComplexity>. Acesso em: 10 set. 2015.

Data Structures. Disponível em: <https://docs.python.org/2/tutorial/datastructures.html>. Acesso em: 10 set. 2015.

GROUP, Intemodino. **Gerador de Números Aleatórios.** Disponível em: <http://randomnumbergenerator.intemodino.com/pt/gerador-de-numeros-aleatorios.html>. Acesso em: 11 set. 2015.

JUDGE, Uri Online. **TOOLKIT - 1552.** Disponível em: <https://www.urionlinejudge.com.br/judge/pt/problems/toolkit/1552>. Acesso em: 11 set. 2015.

JUDGE, Uri Online. **URI Online Judge.** Disponível em: <https://www.urionlinejudge.com.br/judge/>. Acesso em: 11 set. 2015.