TP2 – Algoritmos I
Aluno: Robert Garcia da Silva, Matrícula: 2019436676

Documentação TP2

Pontos-chave observados para a solução do problema

Para o problema apresentado no trabalho observou-se o seguinte ponto chave para a solução:

- Conseguir obter um caminho, ciclovia que passe por todos os pontos de interesse, sem que ocorra ciclos, que não passe por um vértice (ponto de interesse) mais de uma vez, de forma a minimizar o custo de construção e maximizar a atratividade total da ciclovia construída.

Modelagem Matemática e Computacional

Como visto nas aulas da disciplina e analisando o problema apresentado, a modelagem matemática para o problema foi utilizando Grafos não direcionados e ponderados, devido a forte ligação e intuição do problema e sua representação com essa estrutura matemática.

Para a modelagem escolhida foi adotado a representação:

Todos os pontos de interesse (PI) foram representados como vértices de um mesmo grafo G, as ligações entre os PI foram representadas por arestas não direcionadas e ponderadas de forma que:

- O custo da construção entre dois PI, aresta (u, v), pertencente a G foi representado como o peso01 da aresta (u, v). A atratividade foi representada como um peso02 dessa mesma aresta (u, v), logo, cada aresta (u, v) possui dois pesos, um peso associado ao custo de construção daquela aresta (parte da ciclovia total) e outro peso associado a atratividade da mesma aresta.

A Figura1 mostra um exemplo de um grafo modelado, segundo esse critérios, representando a Figura 3, CasoTeste01.txt, do roteiro do trabalho.

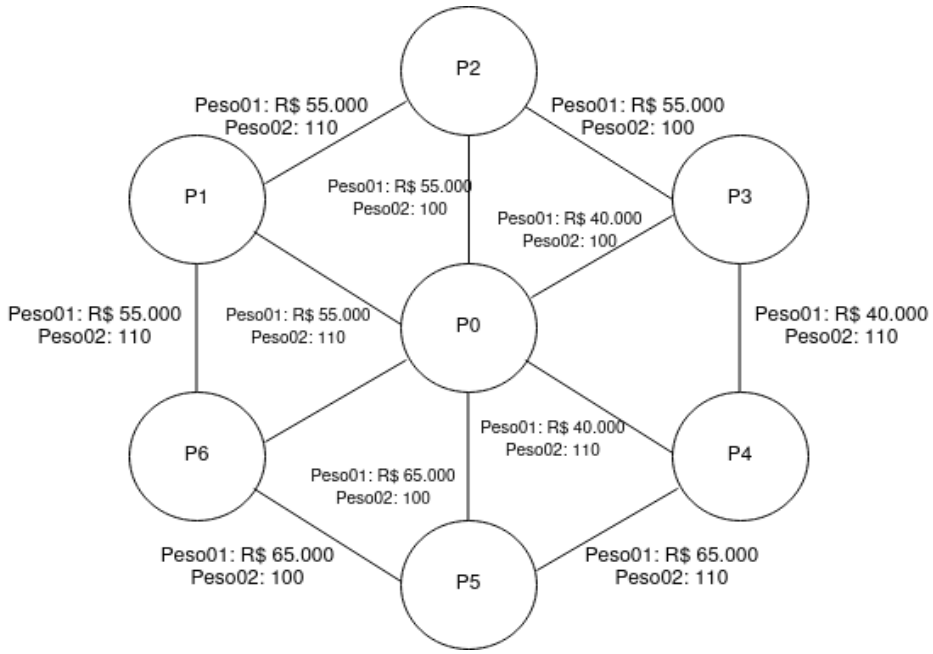

Figure 1: Exemplo modelagem, Figura 3 do roteiro do trabalho.

Para a solução do ponto-chave colocado acima, dado o que foi apresentado na disciplina e a modelagem escolhida para o problema, usando grafos, foram utilizados dois algoritmos para a tentativa de solução: Algoritmo de Prim e Algoritmo de Kruskal.

Dado os testes realizados utilizando ambos os algoritmos sobre o problema proposto, foi descartado a utilização do Algoritmo de Prim, por não apresentar a saída correta para todos os casos de teste, dado a natureza do algoritmo, e utilizado então o **Algoritmo de Kruskal** para obter uma **árvore geradora mínima** para solução do problema.

O ponto-chave para a solução do problema, utilizando o algoritmo de Kruskal foi realizar a ordenação, ordenação necessário para aplicar o Algoritmo de Kruskal, de cada aresta do grafo G tomando dois critérios (baseado nos dois pesos). O primeiro critério baseando-se no peso01 (custo) em que foi ordenado em ordem crescente, visando minimizar o custo, e então tomando-se um segundo critério baseado no peso02 (atratividade) da aresta de forma que para duas arestas que possuem o mesmo peso01, aquela aresta que possui a maior atratividade é considerada menor, na ordenação das arestas, que a aresta que possui menor atratividade, visando maximizar a atratividade total da ciclovia construída.

Para a representação computacional do grafo G foi adotado a construção de duas estruturas (classe), a primeira chamada Aresta, que representa cada aresta (u, v) do grafo G, contendo os atributos: vértice1 (representando u), vértice2 (representando v), peso01 (representando o custo da aresta (u,v)) e peso02 (representando o atratividade da aresta (u, v)) e a segundo chamada Grafo, em

que o grafo é representado usando-se um **vetor contendo cada aresta de G** instanciada anteriormente utilizando a estrutura Aresta.

O algoritmo de Kruskal é o algoritmo chave para a solução do problema como um todo , é o algoritmo responsável para a solução do problema. Seja **G(V, A)** o grafo modelado para o problema, em que V representa o número de vértices e A o número de Arestas. Na solução usada no presente trabalho o desempenho do algoritmo de Kruskal é dado principalmente por:

- Utilização do std::sort, com complexidade dada por , segundo [2], $N*\log_2(N)$, em que N representa a quantidade de elementos que serão ordenados, para o presente trabalho tem-se o desempenho limitado por por $A*\log_2(A)$.

- Na construção do algoritmo de kruskal foram utilizados duas funções, “kruskal_find()” e “kruskal_union()”, que como podem ser visto no pseudocódigo abaixo, são executadas, respectivamente 2V e V vezes, somando 3V.

Temos então uma complexidade limitada por **O(3V + $A*\log_2(A)$)**.

O Pseudocódigo de uma possível implementação, implementação usada no presente trabalho, implementação simples do algoritmo de Kruskal, bem como a análise feita, entendida são apresentados abaixo.

```
/*Pseudocódigo Algoritmo Kruskal do presente trabalho
Seja G(V, A) o grafo de entrada para o algoritmo de Kraskal, em que
V representa o número de vértices e A o número de arestas.
*/



Kruskal_find(subconjunto, vertice):
        Se subconjunto[vertice] == -1:
                retorna vertice;
        se não chame Kurskal_find(subconjunto, subconjunto[vertice])

Kruskal_union(subconjunto, A(u,v)):
        u_set <- Kruskal_find(subconjunto, u);
        v_set <- Kruskal_find(subconjunto, v);
        subconjunto[u_set] = v_set;

Algoritmo_Kraskal(G):
        Arvore_Kruskul <- Conjunto vazio que recebe as arestas dado por Kruskul
        Subconjunto <- V subconjuntos inicializados com -1; //Inicializa V Subconjuntos como conjuntos de um único elemento
        SORT(Arestas) // Ordena as arestas por ordem Crescente //Desempenho limitado por A*log2(A)

        Para cada aresta A (u, v) faça:
                vertice1 = retorno de kruskal_find(subconjunto, u); // Chama Kruskal_find A vezes
                vertice2 = retorno de Kruskal_find(subconjunto, v); // Chama Kruskal_find A vezes
                Se vertice1 != vertice2, faça:
                        Arvore_Kruskul <- A(u, v);
                        chame Kruskal_union(subconjunto, u, v); // Chama limitado por  Kruskal_union A vezes

        retorne Arvore_Kruskal
```

Figure 2: Pseudocódigo Algoritmo de Kruskual usado no trabalho.

Referências

[1] Ime.usp.br, **algoritmos para grafos.** Algoritmo Kruskal https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/kruskal.html Acessado: Entre 10/02/2021 e 22/01/2021

[2] cplusplus.com, **std::sort** https://www.cplusplus.com/reference/algorithm/sort/ Acessado: Entre 10/02/2021 e 22/01/2021

[3] **Aulas – Jussara Almeida - DCC/UFMG** https://www.youtube.com/watch?v=j2POV6Qnrhs&feature=youtu.be Acessado: Entre 10/02/2021 e 22/01/2021

[4] **Aulas – Jussara Almeida - DCC/UFMG** https://www.youtube.com/watch?v=0n_W55NtIjc&feature=youtu.be Acessado: Entre 10/02/2021 e 22/01/2021

[5] **Wikipédia**, STL https://en.wikipedia.org/wiki/C%2B%2B_Standard_Library Acessado: Entre 10/02/2021 e 22/01/2021

[6] Youtube, **Grafos - Algoritmo de Kruskal, Marcos Castro** https://www.youtube.com/watch?v=fziFDaQ1S5I&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=74 Acessado: Entre 10/02/2021 e 22/01/2021

[7] Wikipédia, **Algoritmo de Kruskal** https://pt.wikipedia.org/wiki/Algoritmo_de_Kruskal Acessado: Entre 10/02/2021 e 22/01/2021

[8] Wikipédia, **Algoritmo de Prim** https://pt.wikipedia.org/wiki/Algoritmo_de_Prim Acessado: Entre 10/02/2021 e 22/01/2021

[9] **Árvore geradora mínima - algoritmo de Prim** https://cp-algorithms-brasil.com/grafos/mst.html Acessado: Entre 10/02/2021 e 22/01/2021

[10] A Simple Makefile Tutorial

https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/ Acessado: Entre 10/02/2021 e 22/01/2021

[11] GeeksforGeeks, Multimap in C++
https://www.geeksforgeeks.org/multimap-associative-containers-the-c-standard-template-library-stl/
Acessado: Entre 10/02/2021 e 22/01/2021

[12] cplusplus.com, **std::sort**
http://www.cplusplus.com/reference/map/multimap/begin/
Acessado: Entre 10/02/2021 e 22/01/2021