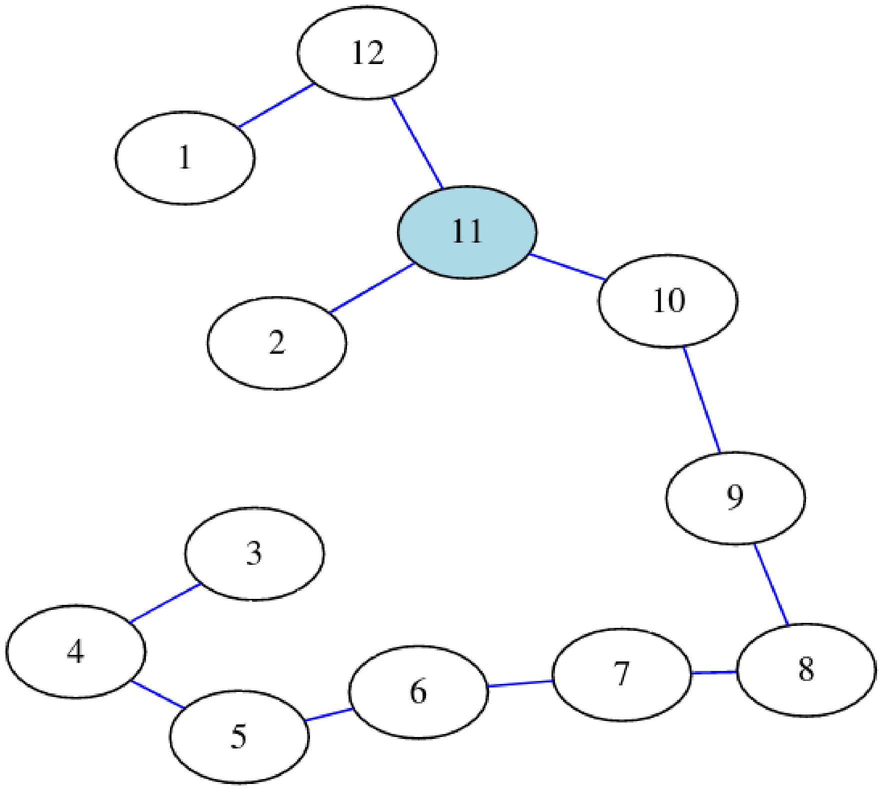
BUSCA LOCAL - PASSO A PASSO

1. Começamos com uma solução inicial para o problema, obtida através de um algoritmo construtivo aplicado a um grafo de exemplo.

    a. No grafo em questão, estamos buscando a melhor solução de uma árvore geradora mínima.



2. Usamos essa árvore como entrada para a função 'FirstBest_Neighbor()', e começamos um loop. Enquanto a função 'FirstBest_Neighbor' continua retornando soluções alternativas, nosso programa permanece na busca da melhor solução.

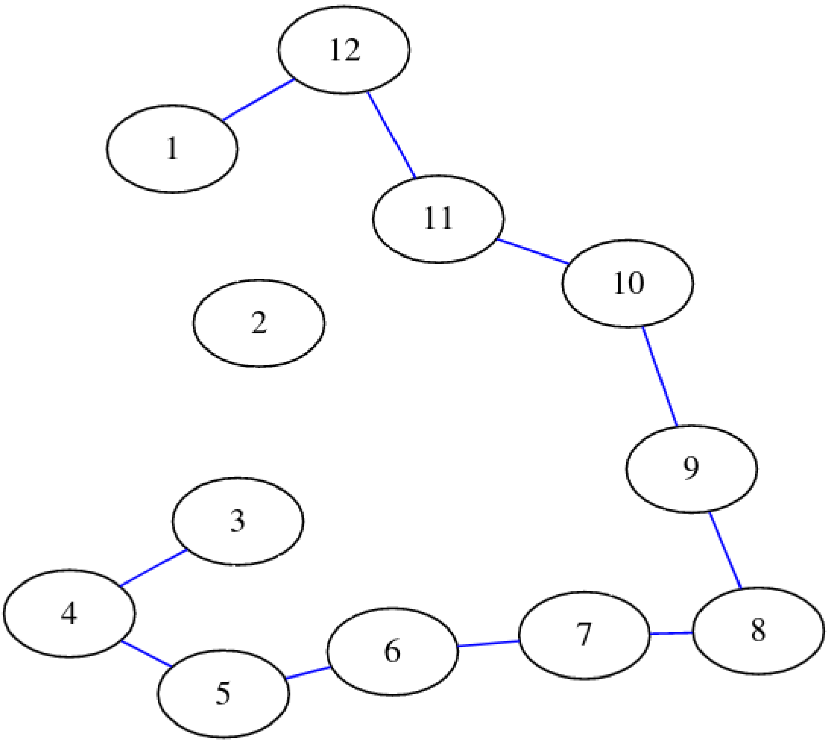
    a. Organizamos uma lista com os vértices da árvore geradora mínima, ordenando todos os vértices d-Branch em ordem crescente com base em um valor calculado a partir da soma do grau das arestas do vértice mais o grau das arestas obrigatórias do vértice.

        Vértice 11 (grau = 3);

    b. Selecionamos o primeiro vértice d-Branch da lista como 'v' e escolhemos um dos vértices adjacentes a 'v' como 'u' (Conexões na árvore T).

        11 - 2;
        11 - 10;
        11 - 12;

    c. Removemos 'v' e ‘u' da árvore 'T', obtendo a partir disso uma floresta.



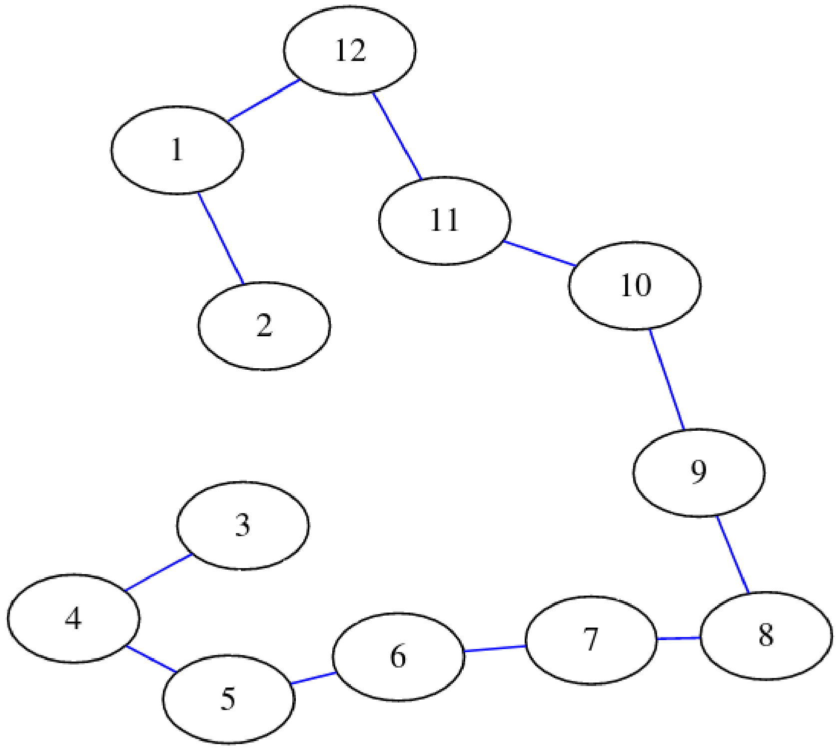
d. Para identificar possíveis arestas que conectam a floresta em um único grafo, liste os graus dos vértices não pertencentes à árvore (T) no grafo original G. Isso ajuda a determinar as conexões adicionais que podem transformar a floresta em uma árvore novamente.

Arestas	**Wsod em G**
1 -- 2	5
7 -- 9	6
2 -- 3	7
3 -- 5	7
3 -- 6	7

e. Selecionar aresta que atende ao requisito e retorno para a função de ‘FirstBest_Neighbor()’.

f. Se encontrarmos uma nova solução que torne a árvore 'T' conexa, a adicionamos à árvore e utilizamos uma condição 'if' para verificar se essa nova solução tem menos vértices d-Branch do que a solução inicial.

g. Se a nova solução tiver menos vértices d-Branch, a consideramos como a nova melhor solução. Caso contrário, continuaremos a busca por vértices, primeiro entre os adjacentes a 'v' e depois entre os próximos vértices d-Branch.


12
1
11
10
2
9
3
4
6
7
8
5