Traveling Salesman Problem

conceitos importantes para o entendimento da explicação no final

- CIRCUITO EULERIANO:
  - O caminho euleriano busca visitar todas as arestas somente uma vez;
  - E o circuito faz o caminho começando e terminando no mesmo vértice;
  - Existe um algoritmo em tempo linear para resolver esse problema;

- CIRCUITO (CICLO) HAMILTONIANO:
  - O caminho hamiltoniano visita todos os vértices somente uma vez;
  - E o circuito faz o caminho começando e terminando no mesmo vértice;
  - Este problema é NP-completo;

- TSP:
  - Procura visitar todos os vértices somente uma vez, começando e terminando no mesmo vértice e com o menor custo possível;
  - Pode ser visto como um Circuito Hamiltoniano de menor custo;
  - É um problema NP-completo;
  - Dado um grafo completo com n vértices e custos não-negativos, o problema é NP-completo e os algoritmos de aproximação por um fator constante também são NP-difíceis, ou seja, se o custo da solução ótima é C, encontrar um algoritmo com um custo no máximo kC também é NP-Difícil;
  - Usando uma heurística para o TSP;
    - Usando uma heurística gulosa;
    - Inicie com um vértice arbitrário. Procure o vértice mais próximo do último vértice adicionado que não esteja no caminho e adicione ao caminho a aresta que liga esses dois vértices;
    - Quando todos os vértices estiverem no caminho, adicione uma aresta conectando o vértice inicial e o último vértice adicionado;
    - Complexidade: $O(n^2)$, sendo n o número de vértices, ou $O(d)$, sendo d o conjunto de distâncias entre os vértices;
    - Aspecto negativo: embora todas as arestas escolhidas sejam localmente mínimas, a aresta final pode ser bastante longa;
  - Podemos encontrar um algoritmo 2-aproximação para o TSP, mas somente se colocarmos algumas restrições ao TSP;
  - Metric TSP:
    - As arestas são não-negativas: $d(x,y) >= 0$;
    - Ir de x para y custa o mesmo que ir de y para x: $d(x,y) = d(y,x)$;
    - Desigualdade triangular: $d(x,y) + d(y,z) >= d(x.z)$, se você quer ir de x pra z, é sempre mais barato ir direto do que passar por um vértice intermediário e depois ir para z;

        - Caso a desigualdade triangular não se aplique, adicione k a cada aresta: mantém o mesmo caminho, mas altera o valor do custo;
        - O problema em si também é NP-difícil;
        - Mas existem algoritmos de p-aproximação para o TSP com essas restrições em tempo polinomial;

- ## TSP 2-Aproximação
    - s: multiconjunto de arestas (a mesma aresta pode aparecer mais de uma vez);
    - c(S): soma dos pesos de todas as arestas de S;
    - O TSP quer encontrar o melhor Circuito Hamiltoniano;
    - H*g : Circuito Hamiltoniano de menor custo;
    - c(H*g) : custo do circuito, e consequentemente é a solução ótima do TSP;
    - Sabemos que a solução contém todos os vértices e queremos eles conectados de forma mínima: MST (Minimum Spanning Tree) → Algoritmo polinomial: Prim ou Kruskal; Menor custo para conectar todos os vértices; Mas obviamente não se preocupa com quantas vezes um nó foi visitado;
    - Seja T uma MST, vendo a partir de um nó raiz teremos uma árvore. Rode um DFS nessa árvore;
    - Encontraremos um ciclo e chamaremos de C. O problema é que vários vértices são visitados mais de uma vez;
    - Mas lembrando que o grafo é completo (existem arestas entre todos os vértices) e da desigualdade triangular (é mais barato ir de x até z diretamente do que passar por um y intermediário);
    - Removendo as duplicatas, temos o ciclo C';
    - c(C') <= c(C);
    - c(C) = 2c(T); Perceba que o caminho é o dobro → ida e volta;
    - Logo, c(C') <= 2c(T);
    - O Ciclo Hamiltoniano ótimo H*g:
        - Se removermos uma aresta qualquer, teremos T', uma spanning tree (não necessariamente a mínima);
        - c(H*g) >= c(H*g - e) >= c(T);
        - Se removermos uma aresta, então o custo deve diminuir, logo: c(H*g) >= c(H*g - e);
        - E o custo dessa T' não pode ser menor do que o custo da MST T;

    - Sabendo que c(C') <= 2c(T) e c(H*g) >= c(H*g - e) >= c(T), multiplicando a segunda desigualdade por 2:
        - 2c(H*g) >= 2c(H*g - e) >= 2c(T);
        - Logo, c(C') <= 2c(H*g);
        - Em outras palavras: o custo do algoritmo de aproximação é menor ou igual a 2 vezes o custo da solução ótima;

- Conceitos Importantes:
  - Heurística:
    - Algoritmo que pode produzir um bom resultado (ou até a solução ótima), mas pode também não obter a solução ou mesmo obter uma solução distante da ótima;
    - Pode ser determinística ou probabilística;
    - Pode haver instâncias em que uma heurística nunca vai encontrar uma solução;

  - Algoritmos aproximados para problemas NP-completos:
    - Para projetar algoritmos polinomiais para “resolver” um problema de otimização NP-completo é necessário relaxar o significado de resolver;
    - Resolvemos a exigência de que o algoritmo tenha sempre de obter a solução ótima;
    - Procuramos algoritmos eficientes que não garantem obter a solução ótima, mas sempre obtêm uma próxima da ótima;
    - Tal solução, com valor próximo da ótima, é chamada de solução aproximada;
    - Um algoritmo aproximado para um problema P é um algoritmo que gera soluções aproximadas para P;
    - Para ser útil, é importante obter um limite para a razão entre a solução ótima e a produzida pelo algoritmo aproximado;
    - No caso do TSP, seja a solução ótima C* e a solução aproximada C, a razão entre as soluções nos dá uma forma de medir a qualidade (C / C*). Essa razão será sempre maior ou igual a 1;
    - Chama-se razão de aproximação p(n);
    - Um algoritmo de aproximação pode ser denominado algoritmo p(n) - aproximação;
    - Por exemplo, um algoritmo 2-aproximação nunca supera 2 nessa razão entre a solução aproximada e a ótima.