SuperComputação

Insper - Engenharia da Computação - 2021.2 - Prof. André Filipe M. Batista

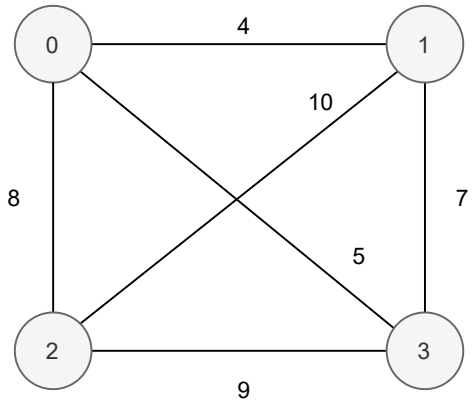
Avaliação Intermediária - 01/10/2021

Instruções

- A prova tem duração de 03 horas (180 minutos);
- A interpretação do enunciado faz parte da avaliação. Eventuais dúvidas serão resolvidas via *chat* do *Teams*;
- É permitida a consulta ao material da disciplina (tudo o que estiver no repositório do Github da disciplina e no site http://insper.github.io/supercomp. Isso também inclui suas próprias soluções aos exercícios de sala de aula, mas não inclui materiais não digitais, tampouco outros materiais além dos citados;
- É permitido consultar a documentação de C++ nos sites http://cplusplus.com e https://cppreference.com;
- A prova é individual. Qualquer consulta a outras pessoas durante a prova constitui violação do código de ética do Insper.
- Boa prova!

Questão nro. 1 (4 pontos): Considere o problema de n cidades onde precisamos selecionar k centros de distribuição de modo a minimizar a distância entre as cidades e esses centros. Esse cenário configura um grafo de n vértices conectado, e precisamos determinar quais k vértices podem ser estabelecidos como centros de distribuição, de modo a minimizar a distância entre as demais cidades e seus respectivos centros (assuma que o centro de uma cidade é aquele em que a distância da cidade para o centro é mínima). Esse problema consiste, portanto, na escolha de k cidades que irão formar k clusters, de modo que a distância intra-cluster seja mínima e a distância inter-cluster seja máxima.

Este é um problema NP-HARD e, para tanto, vamos precisar resolvê-lo via abordagem de aproximação. Faremos uso de uma abordagem *heurística gulosa*. Considere o seguinte grafo, contendo $n = 4$ cidades, onde se deseja determinar $k = 2$ centros.



A abordagem gulosa consiste em escolher uma das cidades como o primeiro centro. Por exemplo, podemos iniciar pela cidade 0. Uma vez que a cidade 0 foi escolhida como um centro, precisamos escolher mais um centro, que agora seja o mais distante possível de 0, o que nos leva a escolher o centro 2, já que a distância

entre 0 e 1 é 4, 0 e 3 é 5 e 0 e 2 é 8 (maior valor). Uma vez que conseguimos k centros, nosso algoritmo finaliza.

Sua tarefa: Implemente a abordagem gulosa descrita acima. Seu programa deve receber como entrada um arquivo com a seguinte estrutura:

```
4 2
0 4 8 5
4 0 10 7
8 10 0 9
5 7 9 0
```

Observe que a primeira linha é formada por n e k, e as demais linhas são formadas pela matriz de distâncias de cada nó do grafo. Neste caso, esse é o input que representa o grafo da imagem. A saída do seu programa deve ser uma listagem das cidades que foram estabelecidas como centros, e uma listagem dos respectivos centros de cada cidade do grafo. No exemplo em questão, a saída deve ser:

```
0 2
0 0 2 0
```

O que indica que as cidades 0 e 2 foram estabelecidas como centros (observe que a saída deve ser na ordem crescente do número da cidade) e, na segunda linha, temos os centros para quais cada cidade i pertence.

Seu algoritmo guloso deve iniciar estabelecendo a cidade 0 como primeiro centro e, em seguida, seguir na estratégia gulosa estabelecendo os demais centros.

O seguinte código-fonte pode lhe ajudar a processar o arquivo de input e criar uma matriz de pesos, que irá armazenar as distâncias entre as cidades do grafo. Fique à vontade para usá-lo ou não na sua resolução.

```
int main(){
    int n;
    cin >> n;
    int k;
    cin >> k;
    int** pesos = new int*[n];
    for(int i = 0;i<n; i++){
        pesos[i] = new int[n];
    }
    for(int i =0; i < n; i ++){
        for(int j = 0; j < n; j++){
            cin >> pesos[i][j];
        }
    }
}
```

No seu pacote de prova, você estará recebendo quatro arquivos de input para serem utilizados na resolução. Ao submeter a sua prova, você deve encaminhar, além do código-fonte, os arquivos de output, de modo que se o arquivo de input se chama `questao1-input-10.txt`, o arquivo de output deve ser `questao1-output-10.txt`. Você deve, portanto, encaminhar o output gerado pelo programa para cada input fornecido.

Questão nro. 2 (1 ponto): Com relação ao problema anterior, a estratégia gulosa nos garante uma solução ótima? Justifique.

Questão nro. 3 (1 ponto): Proponha uma estratégia de busca local para resolução do problema da questão 1. Apresente um pseudo-código.

Questão nro. 4 (1 ponto): Em sala de aula, nós implementamos diversas estratégias para a mochila binária. Explique a importância de buscar um balanço entre *exploration* e *exploitation*.

Questão nro. 5 (2 pontos): No projeto da disciplina, estamos implementando soluções para o problema MIN SET-COVER. Uma possível variação para o problema é a existência de custos associados aos subconjuntos. Por exemplo, considere a seguinte situação, onde U é o universo de cidades e S_i ($i = 1, 2, 3$) são subconjuntos de U:

```
U = {1, 2,  3, 4, 5}
S = {S_1, S_2, S_3}
S_1 = {4, 1, 3}, Custo(S_1) = 5
S_2 = {2, 5}, Custo(S_2) = 10
S_3 = {1, 4, 3, 2}, Custo(S_3) = 3
```

Para esta instância, o min set-cover é formado pelos subconjuntos S_2 e S_3, com custo 13.

Observe que nesse caso, além do número de subconjuntos ser mínimo, nós estamos também buscando minimizar o custo. Seja I o conjunto de todos os subconjuntos da solução para o problema, um possível algoritmo guloso para resolver seria adicionar o subconjunto S_i à I que minimize a seguinte relação:

$$Custo(S_i)/|S_i - I|$$

Ou seja, o custo de S_i em função do número de elementos que S_i contribui efetivamente para I.

Sua tarefa: Escreva um pseudo-código que resolva o problema MIN SET-COVER com essa abordagem gulosa, considerando agora que o problema possui custo (1,0 ponto). Implemente em C++ a solução específica para o exemplo apresentado nessa questão. Não há necessidade de haver arquivo de input. Você pode criar as variáveis que representam os subconjuntos e seus elementos de maneira *hard-coded.* (1,0 ponto).

Questão nro. 6 (1 ponto): No material complementar da prova, você encontra o arquivo `badprimes.cpp`. Avalie o arquivo e, com uso da ferramenta *valgrind*, execute o profiling do código, indicando quais trechos do código podem ser alvo de otimização.