SuperComputação

Insper - Engenharia da Computação - 2023.1 - Profs. André Filipe e Luciano Silva

Avaliação Intermediária - 31/03/2023

Instruções

- A prova está disponível no blackboard, e você pode fazê-la entre 08h e 21h no dia 31/03/2023. Ou seja, você pode iniciar a qualquer momento ao longo deste horário, **mas a entrega deve ser feita até às 21h**;
- A interpretação do enunciado faz parte da avaliação;
- 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 planejamento de roteiros para uma viagem de férias com amigos, em que cada dia é dedicado a uma atividade diferente e deve-se otimizar a ordem das atividades para minimizar o tempo total gasto em deslocamentos.

O problema consiste em dado um conjunto de atividades com localização, encontrar o melhor roteiro de viagem que minimize o tempo gasto em deslocamentos entre as atividades, considerando as limitações de tempo e orçamento dos viajantes.

Para resolver esse problema, podemos usar uma heurística gulosa que funciona da seguinte maneira:

1. Escolha uma atividade inicial aleatória e adicione à lista de atividades a serem visitadas.
2. Para cada atividade não visitada, calcule a distância (ou tempo) para a atividade atualmente selecionada.
3. Selecione a atividade mais próxima da atual e adicione à lista de atividades a serem visitadas.
4. Repita o passo 2 e 3 até que todas as atividades sejam visitadas.

Sua tarefa: Você deve implementar essa heurística gulosa em C++ para encontrar o melhor roteiro de viagem para um conjunto de atividades. A entrada do programa deve ser um arquivo com o seguinte formato:

```
<número de atividades>
<latitude da atividade 1> <longitude da atividade 1>
<latitude da atividade 2> <longitude da atividade 2>
...
<latitude da atividade n> <longitude da atividade n>
```

O programa deve calcular a distância em metros entre duas localizações usando a seguinte função:

```cpp
#include <cmath>

double distance(double lat1, double lon1, double lat2, double lon2) {
    const double R = 6371000;
    double phi1 = lat1 * M_PI / 180;
    double phi2 = lat2 * M_PI / 180;
    double dphi = (lat2 - lat1) * M_PI / 180;
    double dlambda = (lon2 - lon1) * M_PI / 180;
    double a = pow(sin(dphi / 2), 2) + cos(phi1) * cos(phi2) * pow(sin(dlambda / 2), 2);
    double c = 2 * atan2(sqrt(a), sqrt(1 - a));
    double distance = R * c;

    return distance;
}
```

Essa função recebe como parâmetros as latitudes e longitudes das duas localizações em graus e retorna a distância em metros. A constante `M_PI` representa o valor de pi em C++.

A saída do programa deve ser o roteiro de viagem com as atividades em ordem e o distância total gasta em deslocamentos, em metros.

Exemplo: Suponha que temos um arquivo "atividades.txt" com o seguinte conteúdo:

```
5
-23.5505 -46.6333
-22.9068 -43.1729
-23.9658 -46.3331
-25.4439 -49.2709
-22.8758 -43.3143
```

A primeira linha indica que há 5 atividades. As próximas linhas representam as coordenadas geográficas de cada atividade. Seu programa deve produzir a seguinte saída:

```
Distância total percorrida: 1420941 metros
Ordem das atividades:
1 -> 3 -> 5 -> 2 -> 4
```

onde a ordem das atividades indica o roteiro de viagem otimizado.

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 (0,5 ponto): Com relação ao problema anterior, a estratégia gulosa nos garante uma solução ótima? Justifique.

Questão nro. 3 (0,5 ponto): Proponha uma estratégia de busca exaustiva 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*. Dê um exemplo de como buscamos atingir *exploration* e outro de como buscamos atingir *exploitation* no problema da mochila binária na estratégia de **algoritmos genéticos**.

Questão nro. 5 (1 ponto): No material complementar da prova, você encontra o arquivo `montecarlo.cpp`, que calcula uma aproximação para o número π. Avalie o arquivo e, com o uso da ferramenta valgrind, execute o *profiling* do código, **indicando quais trechos do código podem ser alvo de otimização**. Para cada um destes pontos, descreva (não implemente) como você otimizaria estes pontos.

Questão nro. 6 (3 pontos): Em nosso projeto, buscamos desenvolver estratégias computacionais para a alocação de filmes que podemos assistir, restrito a uma jornada de 24 horas, e respeitando a capacidade máxima de filmes por categoria. Há muitas formas de buscar resolver este problema e uma delas é por meio da abordagem de **programação dinâmica**. A programação dinâmica é uma abordagem computacional que consiste em dividir um problema em subproblemas menores e resolver cada subproblema apenas uma vez, armazenando sua solução para uso futuro e evitando o retrabalho.

Sua tarefa: Implemente um algoritmo em C++ que resolva esse problema de nosso projeto utilizando programação dinâmica.

Vamos recordar como é a sua entrada, para que você possa compreender o pseudocódigo que estamos propondo para resolução: A entrada é composta por um arquivo texto chamado "input.txt". A primeira linha contém dois números inteiros separados por um espaço: N e M, respectivamente, representando o número de filmes disponíveis e o número de categorias diferentes. A segunda linha contém M números inteiros separados por um espaço, representando o número máximo de filmes permitidos em cada categoria. As próximas N linhas contêm três números inteiros separados por um espaço: H, F e C, respectivamente, representando o horário de início, horário de término e categoria do filme.

A seguir apresentamos o pseudocódigo proposto para resolução desse problema. **Você deve basear sua implementação nesse pseudocódigo, fazer uso dos inputs gerados em seu projeto e discutir se está obtendo resultados melhores que os obtidos até o momento em suas implementações. Não deixe de explicar como está funcionando a abordagem de programação dinâmica.**

```
Ler N, M
Ler max_filmes[M]

Criar uma lista de filmes vazia

Para i de 1 até N
    Ler H[i], F[i], C[i]
    Adicionar o filme (H[i], F[i], C[i]) na lista de filmes

Ordenar a lista de filmes por ordem crescente de hora de término F[i]

Cria uma matriz dp[N+1][M+1] preenchida com zeros

Para j de 1 até M
    dp[0][j] = max_filmes[j]

Para i de 1 até N
    Para j de 1 até M
        Se C[i] != j
```

```
            dp[i][j] = dp[i-1][j]
        Senão
            last = i - 1
            Enquanto last >= 0 e F[last] > H[i]
                last = last - 1
            Fim Enquanto
            max_count = 0
            Para k de 0 até min(dp[last][j-1], F[i]-H[i]-1)
                max_count = max(max_count, dp[last][j-1-k] + k + 1)
            Fim Para
            dp[i][j] = max(dp[i-1][j], max_count)
        Fim Se
    Fim Para
Fim Para

Imprimir dp[N][M]
```