SuperComputação

Insper - Engenharia da Computação - 2022.1 - Profs. André Filipe e Luciano Silva

Avaliação Intermediária - 05/04/2022

Instruções

- A prova tem duração de 03 horas (180 minutos);
- 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): Suponha que temos m máquinas idênticas ($m \geq 2$) e n jobs. Para cada job i, temos t_i, que corresponde ao seu respectivo tempo de execução em qualquer uma das máquinas. Nosso objetivo é alocar esses jobs nas m máquinas de uma maneira balanceada, minimizando o tempo de execução da máquina mais carregada. Ou seja, após a distribuição das tarefas às máquinas, a soma dos tempos das tarefas pertencentes à máquina com a maior carga entre todas deve ser a mínima possível.

Podemos reconhecer esse problema em diversos cenários reais. Por exemplo, pode-se considerar uma fábrica com duas ou mais máquinas em uma linha de produção. Qual a ordem de tarefas a serem executadas em cada máquina, considerando reduzir o seu tempo máximo de utilização?

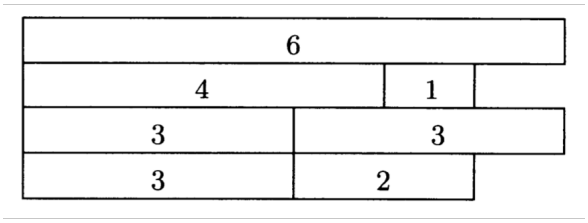
Seja A_j o conjunto de jobs que foram alocados na máquina j. Definimos $T_j = \sum_{i \in A_j} t_i$ como a carga da máquina j. Um possível algoritmo de **heurística gulosa** para este problema é dado por:

1. Para cada máquina i, faça:
   1.1. $T_i = 0$ *(as máquinas começam sem carga)*
   1.2. $A_j = \emptyset$ *(as máquinas começam sem jobs alocados)*
2. Para cada job j em ordem decrescente de seu tempo de execução:
   2.1. Determinar i a máquina com a menor carga;
   2.2. $A_i = A_i \cup \{j\}$ (*alocamos j na máquina i*)
   2.3. $T_i = T_i + T_j$ (*atualizamos a carga da máquina*)
3. Retornamos o maior valor de carga (T_i) para nosso conjunto de máquinas.

Por exemplo, considere um cenário de 7 jobs e 4 máquinas. Um possível arquivo de input para nosso problema pode ser:

```
7 4
3
2
4
1
3
3
6
```

Observe que a primeira linha é formada por n jobs e m máquinas, e as demais linhas correspondem ao tempo de execução de cada job. Após a execução do algoritmo acima, a alocação de cada máquina (considere cada linha da imagem abaixo como uma máquina) seria da seguinte forma.



O nosso programa deveria, portanto, retornar o valor 6, pois é o maior tempo de execução entre as máquinas existentes.

Sua tarefa: implemente em C++ a versão desse algoritmo guloso. 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 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.

Questão nro. 5 (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.

Questão nro. 6 (2 pontos): Em 01 de abril de 2022 uma equipe de pesquisadores anunciou que conseguiu sequenciar 100% o genoma humano. Os genes de um ser humano estão espalhados por 46 cromossomos,

estruturas muito complexas no núcleo das células. Há regiões neles difíceis de decifrar. Esforços concentrados nas últimas décadas haviam fracassado em decifrar os pares de bases ("letras" do minimalista alfabeto químico-genético, A, T, C e G) nessas regiões. Os buracos tomavam 8% do genoma, apesar dos anúncios de "finalização". O trabalho dos pesquisadores tapou tais lacunas com 238 milhões de letras acrescentadas. Não é pouca coisa, num texto — o genoma — composto por 3.054.832.041 caracteres; se fosse um livro, teria mais de 1,5 milhão de páginas.

Em nosso projeto de disciplina estamos buscando desenvolver técnicas para fazer o alinhamento de sequências de DNA, oriundo de uma área denominada bioinformática. O interesse nesta área deve-se, principalmente, ao fato dos biólogos necessitarem do poder de processamento dos computadores para a análise de dados obtidos através das pesquisas experimentais, procurando alinhar e encontrar padrões entre diferentes sequências de nucleotídeos ou aminoácidos. Alinhamento de sequências, simplificadamente, é o posicionamento e a comparação entre cadeias de caracteres, elaborando inferências que podem ser feitas a partir das partes similares que elas apresentam. Conseguir um bom alinhamento entre as sequências de várias espécies é geralmente uma tarefa difícil e envolve vários problemas. Por exemplo, a diferença de tamanho entre os diversos trechos que apresentam similaridade, os tamanhos variados das sequências, entre outros problemas que podem ser encontrados.

Chegou o momento de você executar a sua implementação do algoritmo de alinhamento Local de Smith-Waterman desenvolvido ao longo do projeto em sequências reais. Nos arquivos de prova você encontra o arquivo `input_seq` que contém duas sequências do gene *Opsin1*, uma do ser humano e outra de um rato. Esse gene está ligado a incapacidade de enxergar cores.

Sua tarefa: Você deve executar o algoritmo de Smith-Waterman para o arquivo `input_seq` e obter o score de alinhamento dessas sequências. Quanto maior o score, mais esse gene é similar entre as espécies de animais. Mas atenção, agora estamos trabalhando com dados reais! Será que seu programa vai executar? **Tenha cuidado! Salve seus arquivos e toda a prova antes de executar.** E o que você vai fazer se não rodar? Caso isso aconteça, você deve avaliar quais trechos de sua implementação podem estar causando a lentidão ou incapacidade de execução. Você deve apresentar uma proposta (não precisa implementar) de onde mudar seu código-fonte caso isso aconteça. Não esqueça de justificar, ok? **Na submissão desta questão, numere ou realce os trechos do seu código que poderiam ser objeto de modificação, fazendo menção a eles na sua proposta.**