Trabalho Prático 3: Alimentação Saudável

Entrega: 06/12/2015

November 24, 2015

1 Introdução

O Departamento de Nutrição pretende promover um evento sobre alimentação saudável e precisa da ajuda do DCC. Sabe-se que para uma pessoa ter uma vida saudável, ela precisa ingerir uma quantidade calorias diárias que variam de acordo com sua idade, peso, altura e restrições pessoais (diabetes, alergias ou ser vegetariano). Centenas de pessoas irão participar deste evento e eles precisam criar planos alimentares para cada uma delas. A fim de automatizar esse processo, eles pediram ajuda aos alunos de AEDSIII.

1.1 Exemplo

João precisa consumir diariamente exatamente 313 calorias em seu café da manhã e os valores calóricos dos alimentos disponíveis para ele são: 30, 34, 40, 12, 50, 20, 45, 70, 63, 41, 110, 17, 130, 23, 50. Seu algoritmo deve ser capaz de definir se é possível criar uma dieta balanceada com essas opções. Nesse caso a resposta seria sim, pois escolhendo o conjunto 50, 110, 130, 23 ele consumiria 313 calorias. Não é possível escolher o mesmo alimento mais de uma vez.

2 Instruções sobre a Implementação

Dada uma entrada com valores positivos inteiros e um valor S de soma, encontre se existe um conjunto cuja soma seja igual a S. Lembre-se que:

- A saída do algoritmo deve ser ”sim” ou ”nao”;
- Podem existir valores repetidos na entrada;
- A computação paralela deve ser empregada e o aluno deve definir a quantidade de processos que serão utilizados.

3 Execução

O trabalho deve ser executado da seguinte forma, onde o único argumento 't' indica o número de threads que serão utilizadas:

```
./tp -t <threads>
```

4 Formato de Entrada e Saída

Seu programa deverá ler a entrada da entrada padrão ($stdin$) e gravar a saída na saída padrão ($stdout$). Na primeira linha da entrada estará um número inteiro T, $0 < T \leq 100$, que representa o número de instâncias do problema a serem simuladas. A partir daí, as linhas pares definem os valores da somas, $0 < S \leq 1000$, e as linhas ímpares os conjuntos para encontrar cada soma. Cada conjunto terá no mínimo um elemento e no máximo 300. Todos os valores serão inteiros.

Entrada:

```
1
313
30 34 40 12 50 20 45 70 63 41 110 17 130 23 50
```

Seu algoritmo deve imprimir 'sim' ou 'nao' para cada instância, uma resposta por linha.

Saída:

```
sim
```

5 O que entregar

Você deve submeter uma documentação de até 12 páginas contendo uma descrição de sua solução para o problema, além de uma análise de complexidade de tempo dos algoritmos envolvidos e uma análise da memória gasta pelo seu programa. Siga as diretrizes sobre como fazer uma documentação que foram disponibilizadas no portal minha.ufmg.

Dessa vez, além disso, você deve cobrir os seguintes itens em sua documentação:

- A formalização do problema como um problema de decisão.
- A prova de que o problema de decisão é NP-completo.

Estes itens são essenciais para seu trabalho e serão bastante considerados na correção do mesmo. Dessa forma, é altamente recomendável responder essas questões em sua documentação.

Além da documentação, você deve submeter um arquivo compactado no formato .zip contendo todos os arquivos de código (.c e .h) que foram implementados em uma pasta com seu nome. Além dos arquivos de código, esse arquivo compactado deve incluir um makefile.

Finalmente, **o algoritmo deve ser paralelizado de alguma forma**, caso você entregue um trabalho que não utilize a biblioteca **pthread** ele não será corrigido.

6 Avaliação

Seu trabalho será avaliado quanto a documentação escrita e à implementação. Eis uma lista **não exaustiva** de critérios de avaliação utilizados.

Documentação

Introdução Inclua uma breve explicação do problema que está sendo resolvido no seu trabalho e um resumo da sua solução.

Solução do Problema Você deve descrever a solução do problema de maneira clara e precisa. Para tal, artifícios como pseudocódigos, exemplos ou figuras podem ser úteis. Note que documentar uma solução não é o mesmo que documentar seu código. **Não** é preciso incluir trechos de código em sua documentação nem mostrar detalhes de sua implementação, exceto quando os mesmos influenciem o seu algoritmo principal, o que se torna interessante.

Análise de Complexidade Inclua uma análise de complexidade de tempo dos principais algoritmos implementados e uma análise de complexidade de espaço das principais estruturas de dados de seu programa.

Avaliação Experimental Sua documentação deve incluir os resultados de experimentos que avaliem o tempo de execução de seu código em função de características da entrada. Cabe a você gerar entradas para esses experimentos. Por exemplo: se esse trabalho fosse sobre ordenação, seria interessante mostrar como o tempo de execução de cada algoritmo varia quando o número de items a serem ordenados aumenta. Para tal, um gráfico mostrando o tempo de execução em função do tamanho da entrada pode ser interessante. Você também deve interpretar os resultados obtidos. Comente sobre cada gráfico ou tabela que você apresentar mostrando o que é possível concluir a partir dele.

Para esse trabalho é necessário fazer uma análise sobre o desempenho do paralelismo empregado no código e o número de threads.

Limite de Tamanho Sua documentação deve ter no máximo 12 páginas. Todo o texto a partir da página 13, se existir, será desconsiderado.

Guia Há um guia e exemplos de documentação disponíveis no moodle.

Implementação

Linguagem e Ambiente

- Implemente tudo na linguagem **C**. Você pode utilizar qualquer função da biblioteca padrão da linguagem em sua implementação, mas **não** deve utilizar outras bibliotecas. **Trabalhos em outras linguagens de programação serão zerados. Trabalhos que utilizem outras bibliotecas também.**
- Os testes serão executados em **Linux**. Portanto, garanta que seu código compila e roda corretamente nesse sistema operacional. A melhor forma de garantir que seu

trabalho rode em Linux é escrever e testar o código nele. Há dezenas de máquinas com Linux nos laboratórios do DCC que podem ser utilizadas. Você também pode fazer o download de uma variante de Linux como o **Ubuntu** (`http://www.ubuntu.com`) e instalá-lo em seu computador ou diretamente ou por meio de uma máquina virtual como o **VirtualBox** (`https://www.virtualbox.org`). Há vários tutoriais sobre como instalar Linux disponíveis na web.

Casos de teste Seu trabalho será executado em um conjunto de entradas de teste.

- Essas entradas *não* serão disponibilizadas para os alunos até que a correção tenha terminado. Faz parte do processo de implementação testar seu próprio código.
- Você perderá pontos se o seu trabalho produzir saídas incorretas para algumas das entradas ou não terminar de executar dentro de um tempo limite preestabelecido. Esse tempo limite é escolhido com alguma folga. Garanta que seu código roda a entrada de pior caso em no máximo alguns minutos e você não terá problemas.
- A correção será **automatizada**. Esteja atento ao formato de saída descrito nessa especificação e o siga precisamente. Por exemplo: se a saída esperada para uma certa entrada o número 10 seguido de uma quebra de linha, você deve imprimir apenas isso. Imprimir algo como "A resposta e: 10" contará como uma saída **errada**.
- Os exemplos mostrados nessa especificação são parte dos casos de teste.
- Os testes disponíveis no MOODLE são apenas preliminares. Os monitores executarão os trabalhos nos demais casos de teste.
- **Você deve entregar algum código e esse código deve compilar e executar corretamente para, pelo menos, um dos testes disponibilizados no MOODLE. Se isso não ocorrer, a nota do trabalho prático será zerada.**

Alocação Dinâmica Você deverá fazer uso das funções `malloc()` ou `calloc()` da biblioteca padrão C, bem como liberar *tudo* o que for alocado utilizando `free()`, para gerenciar o uso da memória.

Makefile Inclua um makefile na submissão que permita compilar o trabalho.

Qualidade do Código Seu código deve ser bem escrito:

- Dê nomes a variáveis, funções e estruturas que façam sentido.
- Divida a implementação em módulos que tenham um significado bem definido.
- Acrescente comentários sempre que julgar apropriado. Não é necessário parafrasear o código, mas é interessante acrescentar descrições de alto nível que ajudem outras pessoas a entender como sua implementação funciona.
- Evite utilizar variáveis globais.
- Mantenha as funções concisas. Seres humanos não são muito bons em manter uma grande quantidade de informações na memória ao mesmo tempo. Funções muito grandes, portanto, são mais difíceis de entender.
- Lembre-se de indentar o código. Escolha uma forma de indentar (tabs ou espaços) e mantenha-se fiel a ela. Misturar duas formas de indentação pode fazer com que

o código fique ilegível quando você abri-lo em um editor de texto diferente do que foi utilizado originalmente.

- Evite linhas de código muito longas. Nem todo mundo tem um monitor tão grande quanto o seu. Uma convenção comum adotada em vários projetos é não passar de 80 caracteres de largura.
- **Tenha bom senso**.

7 Considerações Finais

- Essa especificação não é isenta de erros e ambiguidades. Portanto, se tiverem problemas para entender o que está escrito aqui, pergunte aos monitores.
- Você **não** precisa de utilizar uma IDE como Netbeans, Eclipse, Code::Blocks ou QtCreator para implementar esse trabalho prático. No entanto, se o fizer, **não** inclua os arquivos que foram gerados por essa IDE em sua submissão.
- Esteja atento ao tamanho da entrada e às complexidades de seus algoritmos.
- **Seja honesto**. Você não aprende nada copiando código de terceiros nem pedindo a outra pessoa que faça o trabalho por você. Se a cópia for detectada, sua nota será zerada e os professores serão informados para que tomem as devidas providências.