Universidade Federal de Ouro Preto - UFOP
Departamento de Ciência da Computação - DECOM

Relatório atividade 5 - Pairsumonious Numbers

BCC402 - ALGORITMOS E PROGRAMACAO AVANCADA

Kayo Xavier Nascimento Cavalcante Leite - 21.2.4095

Professor: Rafael Alves Bonfim

Ouro Preto
31 de março de 2025

Sumário

Lista de Códigos Fonte

1 Código e enunciado.

Na Atividade 4 - o problema selecionado foi PairsumoniousNumbers). O Problema tem como objetivo dadas todas as somas de pares de N números, reconstruir os números originais em ordem não decrescente ou indicar impossibilidade. O código comentado e documentado, casos de teste e executável pré compilado se encontram no .zip da atividade. O código foi feito com base na referência encontrada no site:

`https://github.com/evandrix/UVa/blob/master/10202.cpp`

Caso queira, para rodar e compilar o código, é necessário ter o compiler g++ e utilizar o seguinte comando no terminal dentro do diretório da pasta da atividade específica:

Compilando e rodando o exercício

```
para compilar:
g++ PairsumoniousNumbers.cpp -o executavel

e para rodar basta utilizar .\executavel no cmd.

para utilizar os cenários de teste:
.\executavel < sampleinput.txt
.\executavel < testinput.txt
```

2 Pairsumonious Numbers: pseudocódigo e descrição do problema

Objetivo: Dadas todas as somas de pares de N números, reconstruir os números originais em ordem não decrescente ou indicar impossibilidade.

2.1 Entrada e Saída

- **Entrada**:
  - Número N $(2 < N < 10)$.
  - $N \times (N-1)/2$ inteiros representando as somas de todos os pares.
- **Saída**:
  - Os N números originais em ordem não decrescente ou `Impossible`.

2.2 Estratégia de Solução

Passos Principais:

1. **Ordenação**: Ordene as somas em ordem crescente.
2. **Hipótese Inicial**:
   - Assume $a[0] = N_1 + N_2$ e $a[1] = N_1 + N_3$.
   - Testa cada $a[t]$ $(t \geq 2)$ como $N_2 + N_3$.
3. **Validação**:
   - Calcula $N_1 = \frac{a[0]+a[1]-a[t]}{2}$.
   - Deriva $N_2 = a[0] - N_1$ e $N_3 = a[1] - N_1$.
   - Verifica se $N_2 + N_3 = a[t]$.

4. **Reconstrução Iterativa**:
   - Para $N_4, N_5, \ldots$, utiliza a menor soma restante como $N_1 + N_i$.
   - Remove as somas correspondentes $N_i + N_j$ da lista.
5. **Caso de Falha**: Se alguma soma necessária não for encontrada, a hipótese é inválida.

2.3 Detalhes de Implementação

- **Lista Encadeada**: Estrutura para remoção eficiente de somas usadas.
  - `prev_` e `nxt`: Índices para simular lista duplamente encadeada.
  - `remove(i)`: Elimina o índice i da lista.
- **Otimizações**:
  - Ordenação inicial para garantir $a[0] \leq a[1] \leq \ldots$.
  - Uso de sentinela para busca simplificada.
- **Casos Especiais**:
  - Números negativos: Tratados naturalmente pela aritmética.
  - Somas duplicadas: Validadas pela remoção sequencial.

```
// Objetivo: Descobrir N numeros secretos (N1, N2... Nn)
//           tendo apenas a lista de todas as somas possiveis entre pares deles (N1+N2, N1+N3, ..., Nn-1 + Nn).

// 1. Preparacao:
//    a. Ler N (quantos numeros secretos).
//    b. Calcular n2 = N*(N-1)/2 (quantas somas teremos).
//    c. Ler as n2 somas fornecidas e guardar em um array `Somas[]`.
//    d. Ordenar o array `Somas[]` do menor para o maior.
//       (Agora, Somas[0] = N1+N2, Somas[1] = N1+N3, assumindo N1 < N2 < N3 ...)

// 2. A Grande Ideia (Deducao Inicial):
//    a. Usamos uma formula magica: 2*N1 = (N1+N2) + (N1+N3) - (N2+N3).
//    b. Sabemos N1+N2 (e Somas[0]) e N1+N3 (e Somas[1]).
//    c. Nao sabemos qual soma em `Somas[]` corresponde a N2+N3.
//    d. Vamos *testar* cada `Somas[t]` (comecando de t=2) como um *candidato* para ser N2+N3.

// 3. Testando um Candidato (Somas[t] == N2+N3?):
//    a. Calcular `valor_teste = Somas[0] + Somas[1] - Somas[t]`.
//    b. Se `valor_teste` nao for par ou for negativo, este `Somas[t]` nao pode ser N2+N3. Pular para o proximo candidato (proximo 't').
//    c. Se for valido, calculamos os candidatos para os 3 primeiros numeros secretos:
//       - `N1_cand = valor_teste / 2`
//       - `N2_cand = Somas[0] - N1_cand`
//       - `N3_cand = Somas[1] - N1_cand`
//    d. *Verificacao Rapida*: `N2_cand + N3_cand` deve ser igual ao nosso candidato `Somas[t]`. Se nao for, pular para o proximo 't'.

// 4. Reconstrucao Iterativa (Se a verificacao rapida passou):
//    a. Guardar N1_cand, N2_cand, N3_cand como os primeiros numeros da nossa solucao provisoria `Resultado[]`.
//    b. Criar uma "lista de somas disponiveis" contendo todas as somas em `Somas[]` EXCETO Somas[0], Somas[1] e Somas[t] (que ja usamos/explicamos).
//       (O codigo usa uma lista ligada para fazer isso eficientemente).
```

```
30 //    c. Loop para encontrar os numeros restantes (N4, N5, ..., Nn):
31 //       i. Pegar a *menor* soma ‘S_min‘ que ainda esta na "lista de somas
       disponiveis".
32 //          (A ideia e que S_min = N1 + Ni, onde Ni e o proximo numero secreto
        a descobrir).
33 //       ii. Calcular o candidato ‘Ni_cand = S_min - N1_cand‘.
34 //       iii. Remover ‘S_min‘ da "lista de somas disponiveis".
35 //       iv. *Validacao Crucial*: Para cada numero ‘Nj‘ *ja encontrado* (N2,
       N3,..., N(i-1)):
36 //           - Calcular a soma esperada ‘S_esperada = Ni_cand + Nj‘.
37 //           - Procurar ‘S_esperada‘ na "lista de somas disponiveis".
38 //           - Se ‘S_esperada‘ NAO for encontrada, entao a nossa hipotese
       inicial (Somas[t] == N2+N3) estava errada. Parar esta tentativa e voltar
       ao passo 2d para testar o proximo ’t’.
39 //           - Se ‘S_esperada‘ for encontrada, remove-la da "lista de somas
       disponiveis".
40 //       v. Se todas as validacoes passaram, adicionar ‘Ni_cand‘ a solucao
       provisoria ‘Resultado[]‘.
41 //    d. Se o loop terminar e encontrarmos todos os N numeros, a hipotese
       inicial estava CORRETA!
42
43 // 5. Resultado:
44 //    a. Se o passo 4d foi bem sucedido para algum ’t’:
45 //       - Imprimir os numeros na solucao ‘Resultado[]‘. Terminar.
46 //    b. Se testamos todos os ’t’ possiveis e nenhum funcionou:
47 //       - Imprimir "Impossivel".
48
49 // Funcao principal ‘main‘: Le a entrada, chama o processo acima (passos 1 a
       5) para cada caso de teste.
```

Código 1: Pseudocódigo do problema.

3 Casos teste - Input e output esperado.

Para os casos de teste do problema, foi disponibilizado junto a pasta do mesmo os seguintes arquivos :sampleinput.txt e testinput.txt, sendo o primeiro o próprio caso de teste disponibilizado pelo exercício e o segundo caso de teste encontrado na plataforma https://www.udebug.com/. Além disso, encontra-se também o arquivo com os outputs esperados para cada input. Ambos os resultados foram validados e tiveram o output esperado.

sampleinput.txt

```
3 1269 1160 1663
3 1 1 1
5 226 223 225 224 227 229 228 226 225 227
5 216 210 204 212 220 214 222 208 216 210
5 -1 0 -1 -2 1 0 -1 1 0 -1
5 79950 79936 79942 79962 79954 79972 79960 79968 79924 79932
```

output esperado

```
383 777 886
Impossible
111 112 113 114 115
101 103 107 109 113
-1 -1 0 0 1
39953 39971 39979 39983 39989
```

Os demais testes se encontram no diretório da atividade