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

Relatório atividade 7 - Repackaging

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 7 - o problema selecionado foi Repackaging. O Problema tem como objetivo dado um conjunto de pacotes contendo xícaras de três tamanhos diferentes, determinar se é possível desempacotar e redistribuir todas as xícaras em novos pacotes com quantidades iguais de cada tamanho. Cada pacote original é definido por três inteiros positivos (S_1, S_2, S_3), onde S_i representa a quantidade de xícaras do tamanho i. Não há pacotes com $S_1 = S_2 = S_3$. 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/Sharknevercries/Online-Judge/blob/master/UVA/Volume%20C/10089%20Repackaging.cp`

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++ Repackaging.cpp -o executavel

e para rodar basta utilizar .\executavel no cmd.

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

2 Problema 10089: Repackaging

2.1 Descrição do Problema

Dado um conjunto de pacotes contendo xícaras de três tamanhos diferentes, determinar se é possível desempacotar e redistribuir todas as xícaras em novos pacotes com quantidades iguais de cada tamanho. Cada pacote original é definido por três inteiros positivos (S_1, S_2, S_3), onde S_i representa a quantidade de xícaras do tamanho i. Não há pacotes com $S_1 = S_2 = S_3$.

2.2 Entrada e Saída

- **Entrada**: Vários casos de teste. Cada caso inicia com um inteiro N (número de pacotes), seguido por N linhas com as especificações dos pacotes.
- **Saída**: "Yes"se for possível redistribuir conforme desejado, "No"caso contrário.

2.3 Estratégia de Solução

O problema é resolvido utilizando uma abordagem geométrica baseada em vetores e ângulos:

1. **Modelagem Vetorial**: Para cada pacote (a, b, c), calcula-se o vetor $(b - a, c - a)$. Este vetor representa a diferença relativa entre as quantidades de xícaras.
2. **Cálculo de Ângulos**: Utiliza-se a função `atan2`(y, x) para obter o ângulo polar de cada vetor em radianos.
3. **Ordenação e Verificação de Gaps**: Os ângulos são ordenados, e verifica-se o maior intervalo entre ângulos consecutivos. Se houver um intervalo maior que π radianos, os vetores estão contidos em um semicírculo, tornando impossível a solução.

2.4 Análise Matemática

Seja θ_i o ângulo do vetor i. Após ordenar $\theta_1, \theta_2, \ldots, \theta_N$, o maior intervalo entre ângulos consecutivos é calculado como:

$$\text{gap}_{\max} = \max\left(\max_{i=1}^{N-1}(\theta_{i+1} - \theta_i),\ 2\pi - (\theta_N - \theta_1)\right)$$

Se $\text{gap}_{\max} > \pi$, a resposta é `No`, caso contrário, `Yes`.

2.5 Complexidade

A complexidade é dominada pela ordenação dos ângulos, resultando em $O(N \log N)$ por caso de teste, adequado para $N \leq 1000$.

```
// Objetivo: Verificar se um conjunto de pontos (representados por angulos)
//           cabe dentro de algum semicirculo (180 graus).

// 1. Preparacao Inicial:
//    a. Definir o valor de PI (aproximadamente 3.14159...).

// 2. Processar Conjuntos de Pontos:
//    a. Ler N (quantidade de pontos no conjunto atual).
//    b. Se N for 0, parar o programa.
//    c. Criar um array `Angulos[]` para guardar N angulos.
//    d. Para cada ponto i de 0 ate N-1:
//       i. Ler os tres numeros a, b, c que definem o ponto.
//       ii. Calcular o ANGULO correspondente a direcao (c-a, b-a) usando
   atan2.
//       iii. Guardar o angulo calculado em `Angulos[i]`.
//    e. Ordenar todos os angulos no array `Angulos[]` do menor para o maior.

// 3. Encontrar o Maior Espaco Vazio entre os Angulos:
//    a. Inicializar uma variavel `maior_espaco` com 0.
//    b. Calcular os espacos entre angulos vizinhos (apos ordenar):
//       i. Para i de 0 ate N-2:
//          - `espaco = Angulos[i+1] - Angulos[i]`.
//          - Se `espaco` for maior que `maior_espaco`, atualizar `
   maior_espaco = espaco`.
//    c. Calcular o espaco "dando a volta" no circulo (entre o ultimo e o
   primeiro angulo):
//       i. `espaco_circular = (2 * PI) - (Angulos[N-1] - Angulos[0])`.
//       ii. Se `espaco_circular` for maior que `maior_espaco`, atualizar `
   maior_espaco = espaco_circular`.

// 4. Tomar a Decisao:
//    a. Comparar o `maior_espaco` encontrado com PI (180 graus).
//    b. Se `maior_espaco` for MAIOR que PI:
//       i. Existe um "buraco" maior que um semicirculo entre os pontos.
//       ii. Imprimir "No" (nao cabem em um semicirculo).
//    c. Senao (se `maior_espaco` for MENOR ou IGUAL a PI):
//       i. O espaco vazio e pequeno o suficiente para que todos os pontos
//          possam ser cobertos por algum semicirculo.
//       ii. Imprimir "Yes".
//    d. Voltar ao passo 2a para ler o proximo N.
```

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

```
4
1 2 3
1 11 5
9 4 3
2 3 2
4
1 3 3
1 11 5
9 4 3
2 3 2
0
```

output esperado

```
Yes
No
```

Os demais testes se encontram no diretório da atividade