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

Relatório atividade 10 -The Problem with the Problem Setter

BCC402 - ALGORITMOS E PROGRAMACAO AVANCADA

Kayo Xavier Nascimento Cavalcante Leite - 21.2.4095

Professor: Rafael Alves Bonfim

Ouro Preto
1 de abril de 2025

Sumário

Lista de Códigos Fonte

1 Código e enunciado.

Na Atividade 10 o problema selecionado foi The Problem with the Problem Setter. O objetivo é alocar problemas de um conjunto a categorias de um teste, respeitando que cada problema seja atribuído a apenas uma categoria e que cada categoria receba a quantidade exata de problemas especificada. A entrada define o número de categorias, o número de problemas e as categorias possíveis para cada problema. A saída indica se a alocação é possível e, caso seja, lista os problemas por categoria. 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:

`hthttps://github.com/KHvic/uva-online-judge/blob/master/10092-the%20Problem%20with%20the%20Proble`

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++ Problem.cpp -o executavel

e para rodar basta utilizar .\executavel no cmd.

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

2 Problema 10092: The Problem with the Problem Setter

2.1 Descrição do Problema

O objetivo é alocar problemas de um conjunto a categorias de um teste, respeitando que cada problema seja atribuído a apenas uma categoria e que cada categoria receba a quantidade exata de problemas especificada. A entrada define o número de categorias, o número de problemas e as categorias possíveis para cada problema. A saída indica se a alocação é possível e, caso seja, lista os problemas por categoria.

2.2 Entrada e Saída

- **Entrada**:
  - Número de categorias nk e problemas np.
  - Para cada categoria, o número de problemas necessários.
  - Para cada problema, as categorias às quais pode pertencer.
- **Saída**:
  - "1"se possível, seguido da lista de problemas por categoria.
  - "0"se impossível.

2.3 Estratégia de Solução

Utiliza-se um grafo de fluxo máximo:

1. **Modelagem do Grafo**:
   - Fonte conecta às categorias com capacidade igual à demanda.
   - Categorias conectam aos problemas compatíveis (capacidade 1).
   - Problemas conectam ao sumidouro (capacidade 1).

2. **Algoritmo de Edmonds-Karp**: Calcula o fluxo máximo. Se igual à demanda total, a alocação é possível.

3. **Recuperação da Solução**: Verifica arestas residuais para determinar quais problemas foram alocados a cada categoria.

2.4 Análise Matemática

Seja F o fluxo máximo e D a demanda total. Se $F = D$, existe uma alocação válida. O grafo tem $O(nk + np)$ nós e $O(nk \cdot np)$ arestas, tornando o algoritmo viável para os limites dados.

```
// Programa: Alocacao de Problemas por Categoria usando Fluxo Maximo

// Funcao AumentarFluxo(no_atual, capacidade_minima_caminho):
//   // Percorre o caminho encontrado (de tras para frente)
//   // e atualiza o fluxo nas arestas e arestas reversas.
//   Se no_atual e a fonte (inicio):
//      Define 'f' (fluxo a ser adicionado) = capacidade_minima_caminho
//      Retorna
//   Se existe pai para no_atual:
//      Chama AumentarFluxo para o pai recursivamente
//      Reduz capacidade da aresta (pai -> no_atual) em 'f'
//      Aumenta capacidade da aresta reversa (no_atual -> pai) em 'f'

// Principal
Enquanto ler K (num categorias) e P (num problemas) e (K+P nao for 0):
    Limpar dados do grafo anterior.
    TotalProblemasNecessarios = 0.

    // Construir Grafo de Fluxo:
    //   Nos: Fonte (0), Sumidouro (1), Categorias (2 a K+1), Problemas (K+2 a K+P+1)
    // Arestas e Capacidades:
    //   Fonte(0) -> Categoria(i): capacidade = demanda da categoria i
    //   Categoria(i) -> Problema(j): capacidade = 1 (se problema j pertence a categoria i)
    //   Problema(j) -> Sumidouro(1): capacidade = 1
    Ler demandas das K categorias e criar arestas Fonte -> Categoria.
    Calcular TotalProblemasNecessarios.
    Ler quais categorias cada um dos P problemas pertence.
    Criar arestas Categoria -> Problema e Problema -> Sumidouro.

    // Calcular Fluxo Maximo (Edmonds-Karp):
    FluxoMaximoTotal = 0
    Repetir:
        // Tentar achar um caminho da Fonte ao Sumidouro com capacidade > 0 usando BFS
        //   - BFS marca nos visitados e guarda o caminho (usando 'pais').
        AcharCaminhoComBFS()
        Se nenhum caminho encontrado:
            Interromper repeticao // Nao ha mais como aumentar o fluxo

        // Se achou caminho, calcular 'f' (capacidade gargalo do caminho)
        // e atualizar o fluxo no grafo usando AumentarFluxo(Sumidouro, infinito)
        AumentarFluxoNoCaminho()
        FluxoMaximoTotal = FluxoMaximoTotal + f // Adiciona fluxo do caminho ao total

    // Verificar e Imprimir Resultado:
    Se FluxoMaximoTotal == TotalProblemasNecessarios:
        Imprimir "1" // E possivel
```

```
47          // Imprimir a alocacao para cada categoria:
48          //   Verificar arestas reversas Problema -> Categoria com capacidade >
                0
49          //   Isso indica que o fluxo passou por ali (problema alocado para
                categoria)
50          ImprimirAlocacao()
51      Senao:
52          Imprimir "0" // Nao e possivel
53
54  // Fim
```

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 sendo o primeiro o próprio caso de teste disponibilizado pelo exercício. 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 15
3 3 4
2 1 2
1 3
1 3
1 3
1 3
3 1 2 3
2 2 3
2 1 3
1 2
1 2
2 1 2
2 1 3
2 1 2
1 1
3 1 2 3
3 15
7 3 4
2 1 2
1 1
1 2
1 2
1 3
3 1 2 3 2 2 3
2 2 3
1 2
1 2
2 2 3
2 2 3
2 1 2
1 1
3 1 2 3
0 0
```

output esperado

```
1
14 6 13
1 7 9
15 12 8 3
0
```