Modelagem, Estruturas de Dados e Algoritmos do Simulador de Coleta de Lixo

Elias Pio Mendes Neto[1], Jeffry Smith Alves Da Silva[2]

[1]Instituto de Ensino Superior - ICEV – Teresina – PI – Brasil

[2] Instituto de Ensino Superior - ICEV – Teresina – PI – Brasil

jeffry.smith@somosicev.com, elias.neto@somosicev.com

Abstract. *This paper presents the technical modeling, data structures, and algorithms implemented in a waste collection simulation system. The simulator, developed in Java with a graphical interface, models the main elements of urban solid waste collection, including zones, small and large trucks, transfer stations, and landfill logistics. The system uses efficient data structures such as maps and lists to track operations and performance metrics. The implemented algorithms simulate the collection, transfer, and disposal processes, enabling the analysis of operational efficiency and resource requirements. The simulator also provides detailed reports and supports CSV export for further analysis.*

Resumo. *Este artigo apresenta a modelagem técnica, as estruturas de dados e os algoritmos implementados em um sistema de simulação de coleta de lixo. O simulador, desenvolvido em Java com interface gráfica, modela os principais elementos da coleta de resíduos sólidos urbanos, incluindo zonas, caminhões pequenos e grandes, estações de transferência e logística de envio ao aterro. O sistema utiliza estruturas de dados eficientes, como mapas e listas, para rastrear operações e métricas de desempenho. Os algoritmos implementados simulam os processos de coleta, transferência e destinação final, permitindo a análise da eficiência operacional e da necessidade de recursos. O simulador também gera relatórios detalhados e permite exportação em CSV para análise adicional.*

1. Introdução

Este artigo apresenta a modelagem, as estruturas de dados e os algoritmos implementados no Simulador de Coleta de Lixo, uma aplicação Java baseada em interface gráfica (Swing) para simular a operação de coleta e transporte de resíduos sólidos urbanos. O sistema foi desenvolvido para permitir a análise detalhada do fluxo de caminhões, acúmulo de lixo nas zonas da cidade, uso de estações de transferência e logística de envio ao aterro sanitário. O simulador oferece relatórios completos, exportação de dados e reinicialização dinâmica, tornando-se uma ferramenta útil para estudos de logística urbana e gestão de resíduos.

2. Modelagem do Sistema

A modelagem do sistema foi orientada a objetos, representando os principais elementos do processo de coleta de lixo urbano:

- **Zonas Urbanas**: Representadas pela classe Zona, cada zona possui um nome e um contador de lixo acumulado (lixoTotal).
- **Caminhões Pequenos**: Modelados pela classe CaminhaoPequeno, responsáveis pela coleta nas zonas e transporte até as estações de transferência. Cada caminhão possui nome, capacidade (em toneladas) e contador de viagens.
- **Estações de Transferência**: Implementadas na classe EstacaoTransferencia, acumulam o lixo recebido dos caminhões pequenos até atingir o volume necessário para envio ao aterro.
- **Caminhões Grandes**: Representados pela classe ViagemCaminhaoGrande, transportam grandes volumes de lixo das estações até o aterro.
- **Aterro Sanitário**: O destino final dos resíduos, monitorado indiretamente pelo acúmulo de lixo enviado pelas estações.

A modelagem considera restrições de capacidade dos caminhões, horários de operação (incluindo horários de pico, definidos em SimulacaoUtils.ehHorarioPico), regras de sorteio de zonas e estações, além de métricas de desempenho como tempo de espera e quantidade de lixo transportado.

Exemplo de definição de zona:

```
public class Zona {
   public String nome;
   public int lixoTotal = 0;
   public Zona(String nome) {
      this.nome = nome;
   }
}
```

3. Estruturas de Dados Utilizadas

O sistema utiliza diversas estruturas de dados para garantir eficiência e rastreabilidade:

- **Map<String, Zona> zonas**

  Armazena as zonas urbanas, indexadas pelo nome, permitindo acesso rápido e atualização do lixo acumulado.

- **Map<String, CaminhaoPequeno> caminhões**

  Mantém os caminhões pequenos disponíveis, indexados pelo nome, facilitando o controle individual de viagens e capacidades.

- **List<ViagemCaminhaoPequeno> viagensPequenos**

  Lista todas as viagens realizadas por caminhões pequenos, com detalhes como zona, estação, horário, carga e tempo de viagem.

- **List<ViagemCaminhaoGrande> viagensGrandes**

Registra as viagens dos caminhões grandes para o aterro, incluindo estação de origem, horário, se foi em horário de pico, tempo de viagem e carga transportada.

- **Map<String, Integer> lixoPorZona**

  Quantidade total de lixo coletado por zona, permitindo análise detalhada da geração de resíduos por região.

- **Map<String, Integer> caminhaoPorZona**

  Quantidade de caminhões que passaram por cada zona, útil para avaliar a distribuição da frota.

- **Map<String, List<String>> zonasPorCaminhao**

  Histórico das zonas percorridas por cada caminhão pequeno, possibilitando rastreamento individual das rotas.

- **EstacaoTransferencia estacaoA, estacaoB**

  Objetos que representam as estações de transferência, contendo atributos como carga acumulada, total de lixo enviado e número de caminhões grandes usados.

Além dessas, são utilizados objetos auxiliares para controle de métricas globais, como totalLixoColetado, totalViagensRealizadas, tempoTotalEsperadoEstacaoA e temp oTotalEsperadoEstacaoB.

Exemplo de inicialização das estruturas:

```
Map<String, Zona> zonas = new HashMap<>();
Map<String, CaminhaoPequeno> caminhoes = new HashMap<>();
EstacaoTransferencia estacaoA;
EstacaoTransferencia estacaoB;
private List<ViagemCaminhaoPequeno> viagensPequenos = new ArrayList<>();
private List<ViagemCaminhaoGrande> viagensGrandes = new ArrayList<>();
private Map<String, List<String>> zonasPorCaminhao = new HashMap<>();
```

4. Algoritmos Implementados

4.1. Simulação das Viagens

O algoritmo principal da simulação está no método realizarTodasViagens() da classe ColetaLixoFrame. Para cada caminhão pequeno, o método executa:

1. **Sorteio de Zona e Estação**

   Para cada viagem, sorteia-se aleatoriamente uma zona (de acordo com a estação associada) e um horário de operação:

```
String nomeEstacao;
String nomeZona;
if (rand.nextBoolean()) {
   nomeEstacao = "Estação A";
```

```
    String[] zonasA = {"Zona Centro", "Zona Norte", "Zona Leste"};
    nomeZona = zonasA[rand.nextInt(zonasA.length)];
} else {
    nomeEstacao = "Estação B";
    String[] zonasB = {"Zona Sul", "Zona Sudeste"};
    nomeZona = zonasB[rand.nextInt(zonasB.length)];
}
int horario = horarioInicio + rand.nextInt(horarioFim - horarioInicio + 1);
```

2. **Cálculo de Carga**

   A carga coletada é sorteada dentro de um intervalo proporcional à capacidade do caminhão.

   ```
   int carga = rand.nextInt(caminhao.capacidade * 1000 - 499) + 500;
   ```

3. **Registro da Viagem**

   Os dados da viagem são registrados em listas e mapas auxiliares:

   ```
   viagensPequenos.add(new ViagemCaminhaoPequeno(
       caminhao.nome, nomeZona, nomeEstacao,
   SimulacaoUtils.formatarHora(horario), pico, carga,
   SimulacaoUtils.calcularTempoViagem(caminhao, pico)
   ));
   zonasVisitadas.add(nomeZona);
   ```

4. **Acúmulo nas Estações**

   O lixo coletado é acumulado na estação correspondente. Ao atingir 20 toneladas, um caminhão grande é despachado para o aterro:

   ```
   estacao.cargaAcumulada += carga;
   if (estacao.cargaAcumulada >= 20000) {
       viagensGrandes.add(new ViagemCaminhaoGrande(
           estacao.nome, SimulacaoUtils.formatarHora(horarioEnvio), envioPico,
   tempoEnvioGrande, estacao.cargaAcumulada
       ));
       estacao.totalLixoEnviado += estacao.cargaAcumulada;
       estacao.cargaAcumulada = 0;
       estacao.caminhoesGrandesUsados++;
   }
   ```

5. **Atualização de Métricas**

   São atualizados os totais de lixo, viagens, tempos de espera e histórico de zonas percorridas:

   ```
   lixoPorZona.put(nomeZona, lixoPorZona.getOrDefault(nomeZona, 0) + carga);
   caminhaoPorZona.put(nomeZona, caminhaoPorZona.getOrDefault(nomeZona, 0) + 1);
   totalLixoColetado += carga;
   totalViagensRealizadas++;
   zonasPorCaminhao.put(caminhao.nome, new ArrayList<>(zonasVisitadas));
   ```

4.2. Cálculo de Tempo de Viagem

O tempo de viagem é calculado considerando se o horário é de pico ou não, utilizando funções auxiliares da classe SimulacaoUtils:

```
public static boolean ehHorarioPico(int minutos) {
   if (minutos >= 6*60 && minutos < 8*60) return true;
   if (minutos >= 12*60 && minutos < 14*60) return true;
   if (minutos >= 17*60 && minutos < 19*60) return true;
   return false;
}
public static int calcularTempoViagem(CaminhaoPequeno caminhao, boolean pico) {
   return pico
      ? (20 + caminhao.capacidade * 2 + new java.util.Random().nextInt(15))
      : (10 + caminhao.capacidade * 2 + new java.util.Random().nextInt(10));
}
```

4.3. Relatórios e Exportação

O sistema gera relatórios detalhados com métricas de desempenho, histórico de viagens e estatísticas por zona. Os dados podem ser exportados em formato CSV, utilizando iteração sobre as listas e mapas para compor as linhas do arquivo:

```
// Exportação CSV
writer.append("Caminhão Pequeno;Zona;Estação;Horário;Pico;Carga (kg);Tempo Viagem (min)\n");
for (ViagemCaminhaoPequeno v : viagensPequenos) {
   writer.append(v.caminhao).append(";")
      .append(v.zona).append(";")
      .append(v.estacao).append(";")
      .append(v.horario).append(";")
```

```
        .append(v.pico ? "Pico" : "Fora Pico").append(";")
        .append(String.valueOf(v.carga)).append(";")
        .append(String.valueOf(v.tempoViagem)).append("\n");
}
```

4.4. Reinicialização

O método reiniciarSimulacao() limpa todas as estruturas de dados e reinicializa os objetos, permitindo uma nova execução da simulação sem reiniciar o programa:

```
void reiniciarSimulacao() {
   caminhoes.clear();
   zonas.clear();
   lixoPorZona.clear();
   caminhaoPorZona.clear();
   viagensPequenos.clear();
   viagensGrandes.clear();
   zonasPorCaminhao.clear();
   // ... reinicialização dos contadores e objetos ...
}
```

5. Conclusão

O simulador foi projetado com foco em flexibilidade, clareza e extensibilidade, utilizando estruturas de dados adequadas para o rastreamento eficiente das operações e métricas do sistema. Os algoritmos implementados permitem simular cenários realistas de coleta de lixo urbano, facilitando a análise de desempenho e a tomada de decisões para a gestão de resíduos. O uso de Java Swing proporciona uma interface amigável e interativa, enquanto a exportação de relatórios e a reinicialização dinâmica tornam o sistema prático para múltiplos cenários de simulação.

6. Referências

Documentação oficial Java SE: https://docs.oracle.com/javase/8/docs/api/

SBC - Sociedade Brasileira de Computação: https://www.sbc.org.br/