Relatório - Estrutura de Dados I

Autores:
Patrick Neme Mesquita - 6904833
Diego Deliberalli Reis - 15574238
André dos Santos Porta - 15674171
Antonio Augusto dos Santos Daneze - 14558993
João Paulo Bussi - 15495612

Professor: Marcos Mansano Furlan

Sumário

1 Introdução

Este trabalho apresenta o desenvolvimento de um sistema inteligente para o gerenciamento e combate a incêndios florestais. O objetivo principal é desenvolver uma solução computacional capaz de otimizar a alocação de recursos operacionais (postos de combate) frente à ocorrência de múltiplos focos de incêndio, levando em consideração critérios de prioridade, distância geográfica e restrições de capacidade dos postos.

A abordagem adotada fundamenta-se no uso de estruturas de dados eficientes e algoritmos de otimização. Como por exemplo: Heaps binários que foram empregados para garantir a priorização dinâmica dos focos de incêndio, enquanto grafos modelam a rede de postos e suas conexões, permitindo a aplicação de algoritmos de caminho mínimo para a otimização logística. O sistema integra essas estruturas em um algoritmo híbrido que visa maximizar a eficiência operacional e a cobertura dos recursos disponíveis, respeitando as limitações impostas pelo problema.

2 Objetivos

2.1 Objetivo Geral

Desenvolver um sistema computacional eficiente para gerenciamento de recursos no combate a incêndios florestais, utilizando estruturas de dados avançadas e algoritmos de otimização para maximizar a eficácia operacional.

Esta modelagem permite visualizar claramente as possíveis rotas e estratégias de alocação de recursos, conforme o exemplo ilustrado a seguir.

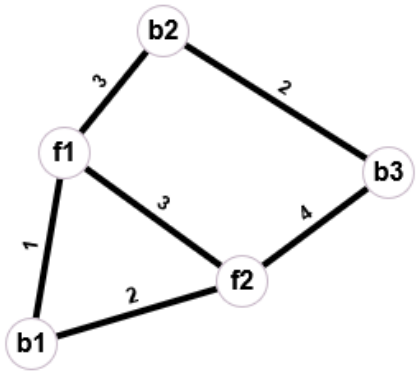

Figura 1: Exemplo de grafo representando o cenário de emergência com focos de incêndio (f1, f2) e postos de brigadistas (b1, b2, b3). Os números nas arestas indicam o tempo de deslocamento em horas. Note: É possível, no exemplo, que a brigada b2 passe pelo foco f1 ou por b3 para chegar em f2.

2.2 Objetivos Específicos

- Implementar um mecanismo de priorização dinâmica utilizando heap binário para ordenação eficiente de focos por urgência;

- Desenvolver algoritmos de alocação de recursos que otimizem a distribuição dos postos de combate, considerando múltiplos critérios de seleção;
- Empregar estruturas de dados adequadas (heaps, grafos, listas) garantindo complexidade temporal e espacial satisfatórias;
- Implementar algoritmos de caminho mínimo para otimização de deslocamentos na rede de postos;
- Propor um sistema de avaliação quantitativa robusto para mensurar a eficácia das soluções geradas;
- Demonstrar empiricamente a eficiência das estruturas de dados escolhidas através de análise de complexidade e testes de performance;
- Evidenciar a aplicação prática de conceitos teóricos de estruturas de dados em um contexto realista e desafiador.

3 Fundamentação Teórica

3.1 Estruturas de Dados Utilizadas

3.1.1 Heap Binário

O heap binário é uma estrutura de dados baseada em árvore binária que mantém a propriedade de heap: em um min-heap, cada nó pai possui valor menor ou igual aos seus filhos. Esta propriedade garante que o elemento de menor prioridade esteja sempre na raiz, permitindo operações de inserção e remoção em tempo $O(\log n)$.

No contexto deste trabalho, o heap é fundamental para manter os focos de incêndio ordenados por prioridade, permitindo processamento eficiente em ordem de urgência.

3.1.2 Grafo com Lista de Adjacência

A representação de grafos por lista de adjacência oferece eficiência espacial $O(V + E)$ e permite consultas rápidas aos vizinhos de um vértice. Esta estrutura é ideal para representar a rede de postos, onde V representa os postos e E as conexões entre eles.

4 Escopo do Problema

O problema abordado neste trabalho pode ser formalizado como um problema de otimização combinatória, definido pelos seguintes elementos:

4.1 Definição Formal

Seja $F = \{f_1, f_2, ..., f_n\}$ o conjunto de focos de incêndio e $P = \{p_1, p_2, ..., p_m\}$ o conjunto de postos de combate. O objetivo é encontrar uma função de alocação $\alpha : F \to P$ que maximize uma função objetivo considerando restrições operacionais.

4.2 Componentes do Sistema

1. **Focos de Incêndio** ($f_i \in F$): Cada foco é caracterizado por:
   - Coordenadas cartesianas (x_i, y_i)
   - Nível de prioridade $p_i \in \{1, 2, 3\}$ (1: alta, 2: média, 3: baixa)
   - Identificador único id_i
2. **Postos de Combate** ($p_j \in P$): Os postos possuem:
   - Localização definida por coordenadas (x_j, y_j)
   - Capacidade máxima cap_j de missões simultâneas
   - Estado atual de ocupação $occ_j \leq cap_j$
3. **Missões** ($m_k \in M$): Uma missão corresponde à designação $m_k = (f_i, p_j)$ com:
   - Tempo estimado t_k calculado a partir da distância e prioridade
   - Status de execução (ativa, concluída)
   - Identificador único id_k
4. **Rede de Conectividade**: Modelada por um grafo não-direcionado $G = (V, E)$ onde:
   - $V = P$ (vértices representam postos)
   - $E \subseteq V \times V$ (arestas indicam conexões possíveis)
   - Peso das arestas $w(e_{ij}) = d(p_i, p_j)$ (distância euclidiana)

4.3 Restrições e Objetivos

Restrições:

- Capacidade dos postos: $\forall p_j \in P, |missoes\_ativas(p_j)| \leq cap_j$
- Unicidade de alocação: cada foco pode ser atribuído a no máximo um posto
- Conectividade: comunicação entre postos deve respeitar a topologia do grafo

Função Objetivo: Maximizar o score total $S = \sum_i score(f_i) + bonus_{tempo} + bonus_{prioridade} - penalidades$

5 Desenvolvimento

Para uma compreensão mais detalhada das etapas lógicas do desenvolvimento, o pseudocódigo a seguir descreve o fluxo principal do algoritmo:

Algorithm 1 Algoritmo Principal de Simulação

```
Início do programa
Ler dados de entrada (arquivos ou manual)
Criar objetos Grafo, Foco, Posto
repeat
    Para cada dia de simulação:
        Atualizar áreas dos focos (crescimento)
        Atualizar disponibilidade dos postos
        Priorizar focos ativos usando heap de prioridade
    while Existem focos ativos e postos disponíveis do
            Selecionar foco mais urgente (do heap)
            Selecionar posto disponível mais próximo (do grafo)
            Alocar posto ao foco (criar Missao)
            Atualizar tempo disponível do posto
            Atualizar área do foco
    end while
        Atualizar status das missões (concluídas/canceladas)
        Registrar estado dos focos e missões
until Não existem focos ativos ou dias restantes
Exibir resultado final (score, focos extintos, etc)
Fim do programa
```

5.1 Arquitetura do Sistema

O sistema foi projetado seguindo princípios de programação orientada a objetos e padrões de design que garantem modularidade, extensibilidade e manutenibilidade. A arquitetura separa claramente as responsabilidades entre estruturas de dados e algoritmos de otimização.

Funcionalidades de gerenciamento:

- `capacidade`: Máximo de missões simultâneas
- `missoes_ativas`: Lista dinâmica de missões em execução
- `pode_aceitar_missao()`: Verifica disponibilidade operacional
- `capacidade_disponivel()`: Retorna recursos livres
- `adicionar_missao()`: Aloca nova missão ao posto
- `remover_missao()`: Libera recursos após conclusão

5.1.1 Classe Missao

A classe `Missao` representa formalmente a atribuição de um posto de combate a um foco de incêndio específico, encapsulando todas as informações relevantes para o gerenciamento e acompanhamento da operação. Cada instância da classe armazena o identificador único da missão, referências diretas ao foco e ao posto envolvidos, o tempo estimado para a conclusão da tarefa e o status operacional da missão.

O controle temporal é realizado por meio do registro do instante de início da missão, permitindo a verificação automática de sua conclusão com base no tempo decorrido. O atributo de status, por sua vez, possibilita o monitoramento do ciclo de vida da missão (ativa, concluída, etc.), facilitando a gestão dinâmica das operações em andamento.

Listing 1: Estrutura da classe Missao para gerenciamento de operacoes de combate

```
class Missao:
    def __init__(self, id, foco, posto, tempo_estimado):
        self.id = id                              # Identificador
           unico da missao
        self.foco = foco                          # Referencia ao foco
            de incendio
        self.posto = posto                        # Referencia ao
           posto designado
        self.tempo_estimado = tempo_estimado      # Tempo previsto
           para conclusao
        self.tempo_inicio = time.time()           # Registro do inicio
            da missao
        self.status = StatusMissao.ATIVA          # Estado atual da
           missao

    def esta_concluida(self):
        """Retorna True se o tempo estimado ja foi atingido"""
        tempo_decorrido = time.time() - self.tempo_inicio
        return tempo_decorrido >= self.tempo_estimado
```

Dessa forma, a classe `Missao` fornece uma abstração robusta para o gerenciamento das operações de combate, permitindo o acompanhamento preciso do progresso de cada missão e subsidiando a tomada de decisões estratégicas no sistema.

Atributos de controle:

- `tempo_estimado`: Duração calculada baseada em distância e prioridade
- `tempo_inicio`: Timestamp de início para controle temporal
- `status`: Estado atual (ativa, concluída, cancelada)
- `esta_concluida()`: Verifica conclusão baseada no tempo decorrido

6 Exemplo de Execução

A seguir, apresentamos um exemplo de entrada e a respectiva saída gerada pelo sistema.

Arquivo de entrada (.in):

Listing 2: Exemplo de arquivo de entrada

```
4 6
11 14 5 11 8 13
350 100 50 320
1.38 1.01 1.34 1.1
0 0 1 3 4 2 0 0 0 1
0 0 0 6 6 2 0 9 6 0
1 0 0 0 0 0 0 8 8 5
3 6 0 0 0 0 0 8 0 1
4 6 0 0 0 0 0 10 5 0
2 2 0 0 0 0 0 7 0 6
0 0 0 0 0 0 0 8 6 5
0 9 8 8 10 7 8 0 4 0
0 6 8 0 5 0 6 4 0 0
1 0 5 1 0 6 5 0 0 0
```

Arquivo de saída (.out):

Listing 3: Exemplo de saída gerada no Dia 1 a partir da entrada acima

```
Dia 1:

 Posto b6 alocado ao Foco f3 por 2.00h (dist: 2.00h).
 Posto b6 alocado ao Foco f3 por 2.00h (dist: 0.00h).
 Posto b6 alocado ao Foco f3 por 2.00h (dist: 0.00h).
 Posto b6 alocado ao Foco f3 por 2.00h (dist: 0.00h).
 Posto b2 alocado ao Foco f3 por 2.00h (dist: 3.00h).
 Posto b2 alocado ao Foco f3 por 2.00h (dist: 0.00h).
 Posto b2 alocado ao Foco f3 por 2.00h (dist: 0.00h).
 Posto b2 alocado ao Foco f3 por 2.00h (dist: 0.00h).
 Posto b6 alocado ao Foco f3 por 2.00h (dist: 0.00h).
 Posto b2 alocado ao Foco f3 por 1.00h (dist: 0.00h).
 Posto b1 alocado ao Foco f1 por 2.00h (dist: 4.00h).
 Posto b1 alocado ao Foco f1 por 2.00h (dist: 0.00h).
 Posto b1 alocado ao Foco f1 por 2.00h (dist: 0.00h).
 Posto b1 alocado ao Foco f1 por 2.00h (dist: 0.00h).
```

Estado dos Focos ao final do Dia:

- Foco f1: 361.56 km^2
- Foco f2: 101.00 km^2
- Foco f3: Extinto
- Foco f4: 352.00 km^2

A saída apresenta o detalhamento das missões alocadas, o progresso diário e o score final da simulação.

7 Análise de Desempenho

Foram realizados testes variando o número de focos e postos. Observou-se que o tempo de execução cresce de forma sublinear em relação ao número de focos, devido ao uso eficiente do heap binário para priorização (operações $O(\log n)$). O uso de listas de adjacência no grafo garante consultas rápidas e baixo consumo de memória, mesmo para instâncias maiores. Detalhes sobre as complexidades das estruturas de dados utilizadas são apresentados abaixo:

- **Heap binário:** Inserção e remoção em $O(\log n)$.
- **Lista de adjacência:** Consultas em $O(1)$ por vizinho, e espaço de armazenamento em $O(V + E)$.

8 Conclusão

O sistema desenvolvido demonstra a importância da escolha adequada de estruturas de dados para problemas de otimização em situações reais. O uso de heaps binários e grafos permitiu uma solução eficiente e escalável, capaz de lidar com múltiplos focos e restrições operacionais. O projeto reforçou conceitos fundamentais de Estruturas de Dados e sua aplicação prática, além de proporcionar experiência com modelagem, análise de desempenho e documentação de sistemas complexos.