UFMG

Trabalho Prático 3 (TP3)

Algoritmos 1

Ítalo Leal Lana Santos | Matrícula: 2024013893 | italolealanasantos@ufmg.br
Universidade Federal de Minas Gerais (UFMG) | Belo Horizonte/MG | dezembro de 2025

---

1. Introdução

O problema computacional consiste em selecionar o maior grupo possível de duendes para a equipe do Papai Noel, garantindo que nenhum par de duendes conflitantes esteja na mesma equipe. Quando existem múltiplas soluções com o mesmo tamanho, deve-se escolher aquela com menor ordem lexicográfica.

Na Teoria dos Grafos, este é o problema do **Conjunto Independente Máximo** (Maximum Independent Set – MIS). Dado um grafo G = (V, E), onde os vértices representam duendes e as arestas representam conflitos, o objetivo é encontrar o maior subconjunto S ⊆ V tal que não existam arestas conectando quaisquer dois vértices em S.

O problema MIS é NP-Difícil, porém a restrição N ≤ 40 permite o uso de técnicas exponenciais otimizadas, especificamente **Meet-in-the-Middle** combinado com **Programação Dinâmica SOS (Sum Over Subsets)**, reduzindo a complexidade de O(2^N) para O(N × 2^(N/2)).

2. Modelagem

2.1 Representação do Grafo

Utilizou-se uma representação baseada em **máscaras de bits** (bitmasks) para eficiência:

- **Vértices**: Os N duendes são indexados de 0 a N-1.
- **Arestas**: A lista self.adj[i] armazena um inteiro onde cada bit ligado indica um conflito. Se o bit j está ativo em self.adj[i], então os duendes i e j brigam entre si.

Exemplo: Se o duende 2 briga com os duendes 0, 3 e 5, então self.adj[2] terá os bits 0, 3 e 5 ligados (valor binário: 101001).

2.2 Divisão do Problema (Meet-in-the-Middle)

Para evitar a explosão combinatória de 2^40 ≈ 10^12 estados, divide-se o conjunto de vértices em duas metades:
Grupo A: primeiros ⌊N/2⌋ duendes (índices 0 a mid-1)
Grupo B: últimos ⌈N/2⌉ duendes (índices mid a N-1)

Esta divisão permite processar cada metade separadamente e depois combinar os resultados, reduzindo drasticamente o espaço de busca.

3. Solução

3.1 Visão Geral do Algoritmo

A solução implementa três etapas principais:

1. **Geração de conjuntos independentes em B** via backtracking
2. **Construção da SOS DP** para consultas eficientes
3. **Combinação com conjuntos de A** e seleção da melhor solução

3.2 Pseudocódigo

```
Função Resolver(N, ListaDeConflitos):
    // Divisão do Problema
    Metade <- N / 2
    GrupoA <- Vértices[0 ... Metade-1]
    GrupoB <- Vértices[Metade ... N-1]

    // 1. Processamento do Grupo B (Backtracking)
    // Retorna lista de tuplas (mascara, tamanho)
    ConjuntosValidosB <- GerarIndependentes(GrupoB)

    // Inicialização da SOS DP
    Para cada (mask, tam) em ConjuntosValidosB:
        DP_Size[mask] <- tam
        DP_Mask[mask] <- mask  // Guarda a máscara para reconstrução

    // 2. Execução da SOS DP (Propagação de Bits)
    Para i de 0 até (Tamanho(GrupoB) - 1):
        Bit <- 1 << i
        Para mask de 0 até (2^Tamanho(GrupoB) - 1):
            Se (mask & Bit):
                Prev <- mask ^ Bit
                // Atualiza se o subconjunto anterior for maior ou
                // se tiver mesmo tamanho mas for lexicograficamente menor
                Se DP_Size[Prev] >= DP_Size[mask]:
                    Atualizar DP[mask] com dados de DP[Prev] (com ordem léxical)

    // 3. Processamento do Grupo A e Combinação
    // Retorna tuplas (maskA, tamA, proibidosEmB)
    ConjuntosValidosA <- GerarIndependentes(GrupoA, calcula_impacto_remoto=True)

    MelhorSolucao <- Vazio
    MascaraCheiaB <- (1 << Tamanho(GrupoB)) - 1

    Para cada (maskA, tamA, proibidosEmB) em ConjuntosValidosA:
        // Define quais duendes de B são permitidos (não conflitam com A)
        PermitidosB <- MascaraCheiaB XOR proibidosEmB

        // Consulta O(1) na DP
        MelhorB_Tam <- DP_Size[PermitidosB]
        MelhorB_Mask <- DP_Mask[PermitidosB]

        Total <- tamA + MelhorB_Tam
        Se Total > MelhorSolucao.Tamanho:
            MelhorSolucao <- (Total, maskA, MelhorB_Mask)
        Senão Se Total == MelhorSolucao.Tamanho:
            AplicarCritérioDeDesempateLexicográfico()

    Retornar ReconstruirSolução(MelhorSolucao)
```

3.3 Etapa 1: Geração dos Conjuntos Independentes

A função get_independent_sets utiliza **backtracking** para enumerar todos os subconjuntos independentes válidos de um grupo:
Para cada vértice i no grupo:
- Opção 1: não incluir i no conjunto
- Opção 2: incluir i (se não conflitar com vértices já escolhidos)
  - Atualizar máscara de proibições locais e máscara de proibições no outro grupo

Retorno: lista de tuplas (máscara_local, tamanho, proibições_remotas)
Para o grupo B, gerar todos os conjuntos independentes sem considerar impacto externo.
Para o grupo A, calcula-se adicionalmente quais vértices de B cada conjunto proíbe.

3.4 Etapa 2: SOS DP (Sum Over Subsets Dynamic Programming)

A técnica SOS DP permite, para qualquer máscara M, consultar rapidamente o melhor subconjunto que utiliza apenas bits presentes em M.
Construção:

1. Inicializa dp_size[mask] e dp_mask[mask] com valores dos conj. indep. válidos
2. Para cada bit i de 0 a (N/2)-1:
    - Para cada máscara que contém o bit i:
        - Propaga informação da máscara sem o bit i
        - Atualiza se encontrar solução maior ou lexicograficamente melhor

Resultado: dp_size[M] contém o tamanho do maior conjunto independente que é subconjunto de M, e dp_mask[M] armazena a máscara correspondente.

3.5 Etapa 3: Combinação e Seleção da Melhor Solução

Para cada conjunto independente válido em A:

1. Calcula-se forbidden_b: quais duendes de B conflitam com os escolhidos em A
2. Define-se allowed_b = mask_b_full ^ forbidden_b (complemento)
3. Consulta-se em O(1): size_b = dp_size[allowed_b]
4. Calcula-se tamanho total: total = size_a + size_b
5. Atualiza-se a melhor solução considerando:
    - Primeiro critério: maior tamanho
    - Desempate: menor ordem lexicográfica

3.6 Critério Lexicográfico

A função is_lex_smaller(mask1, mask2) compara duas máscaras lexicograficamente:

- Calcula a diferença XOR entre as máscaras
- Identifica o bit menos significativo da diferença
- A máscara que possui este bit ativo representa o menor índice, logo é lexicograficamente menor

Aplicação: Utilizado tanto na SOS DP quanto na seleção final para garantir a ordem crescente mínima.

4. Análise de Complexidade

4.1 Complexidade de Tempo

Componentes principais:

- **Geração de B**: O(2^(N/2)) subconjuntos via backtracking
- **SOS DP**: O((N/2) × 2^(N/2)) iterações
- **Geração de A**: O(2^(N/2)) subconjuntos
- **Combinação**: O(2^(N/2)) consultas em O(1) cada

Complexidade total: **O(N × 2^(N/2))**
Para N = 40: aproximadamente 40×2^20 ≈ 42 milhões de operações, perfeitamente viável.

4.2 Complexidade de Espaço

Armazenamento principal:

- dp_size: array de 2^(N/2) bytes → ~1 MB para N=40
- dp_mask: array de 2^(N/2) inteiros → ~8 MB para N=40
- valid_a e valid_b: listas de tuplas → O(2^(N/2))
- Adjacências: N inteiros → desprezível

Complexidade total: **O(2^(N/2))**
Para N = 40: ~10 MB de memória, extremamente eficiente.

5. Considerações Finais

5.1 Desafios Encontrados

A implementação da **SOS DP** foi a parte mais desafiadora, exigindo compreensão profunda de como propagar informações através de superconjuntos. Garantir que a propagação respeitasse tanto o critério de tamanho quanto o lexicográfico simultaneamente demandou atenção especial.

O **critério de desempate lexicográfico** também apresentou complexidade, pois era necessário comparar máscaras bit a bit e garantir que esta lógica funcionasse corretamente tanto na DP quanto na combinação final.

5.2 Facilidades

A **divisão Meet-in-the-Middle** foi intuitiva após compreender a ideia central. A representação via **bitmasks** tornou as operações de conflito extremamente eficientes, permitindo verificações em O(1) através de operações bitwise.

5.3 Otimizações Implementadas

- Uso de bytearray para dp_size, economizando memória
- Cálculo incremental de forbidden_b durante a geração de A
- Comparações lexicográficas via operações bitwise (XOR e detecção de LSB)
- Conversão final de máscaras para lista apenas uma vez

5.4 Aprendizados

Este trabalho consolidou conhecimentos sobre técnicas avançadas para problemas exponenciais, demonstrando como restrições do problema ($N \leq 40$) permitem soluções criativas. A combinação de Meet-in-the-Middle com SOS DP é uma técnica poderosa aplicável a diversos problemas de otimização combinatória.

6. Referências

CORMEN, Thomas H. et al. **Introduction to Algorithms**. 3rd ed. MIT Press, 2009.

LAAKSONEN, Antti. **Competitive Programmer's Handbook**. 2018. Disponível em: https://cses.fi/book/book.pdf

Material didático da disciplina **Algoritmos I - UFMG/DCC**, 2025. Slides sobre Programação Dinâmica e Divisão e Conquista.

GEEKSFORGEEKS. **Sum Over Subsets (SOS) DP**. Disponível em: https://www.geeksforgeeks.org/sum-over-subsets-sos-dp/

CODEFORCES. **Meet in the Middle Technique**. Disponível em: https://codeforces.com/blog/entry/95571

Python Software Foundation. **Python 3.9+ Documentation**. Disponível em: https://docs.python.org/3/