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

# Relatório atividade 8 - Bigger Square Please...

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 8 o problema selecionado foi Bigger Square Please...!. O objetivo é preencher um quadrado de tamanho $N \times N$ com o menor número possível de quadrados menores, cada um com tamanho entre 1 e $N-1$. A solução deve garantir que não haja sobreposição ou espaços vazios. A saída inclui o número mínimo de quadrados e suas coordenadas e tamanhos.posicionadas para um dado $N$. 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/evandrix/UVa/blob/master/10270.cpp`

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++ Bigger.cpp -o executavel

e para rodar basta utilizar .\executavel no cmd.

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

# 2 Problema 10270: Bigger Square Please...

## 2.1 Descrição do Problema

O objetivo é preencher um quadrado de tamanho $N \times N$ com o menor número possível de quadrados menores, cada um com tamanho entre 1 e $N-1$. A solução deve garantir que não haja sobreposição ou espaços vazios. A saída inclui o número mínimo de quadrados e suas coordenadas e tamanhos.

## 2.2 Entrada e Saída

- **Entrada**: Um inteiro $T$ (número de casos de teste) seguido por $T$ valores de $N$ ($2 \leq N \leq 50$).
- **Saída**: Para cada $N$, o número mínimo $K$ de quadrados seguido por $K$ linhas com coordenadas $(x, y)$ e tamanho $l$ de cada quadrado.

## 2.3 Estratégia de Solução

A solução combina pré-computação e backtracking:

1. **Pré-computação**: Resultados para $N$ até 50 são armazenados em um array estático, otimizando a resposta para casos conhecidos.
2. **Números Compostos**: Se $N$ é composto, a solução é derivada escalando a solução de seus fatores primos.
3. **Números Primos**: Para $N$ primo, um algoritmo de backtracking tenta combinações de quadrados, começando pelo maior tamanho possível.

## 2.4 Análise Matemática

Para $N$ composto com fator $d$, a solução é escalada por $N/d$. Para primos, a solução é encontrada verificando recursivamente todas as combinações válidas de quadrados, minimizando $K$. A área total dos quadrados deve ser $N^2$, e a posição de cada quadrado é verificada para evitar sobreposições.

```
// Programa: Ladrilhar Quadrado NxN com Minimo de Quadrados Menores

// --- Funcoes Principais de Logica ---

// Funcao AcharCombinacao(N, num_quadrados_alvo):
//   - Tenta encontrar (usando recursao/backtracking) um CONJUNTO
//     de 'num_quadrados_alvo' quadrados que somem area N*N.
//   - Se encontrar um conjunto:
//     - Chama TentarEncaixar para ver se esse conjunto cabe na grade.
//     - Retorna verdadeiro SE encaixou, falso caso contrario.
//   - Se nao encontrar conjunto:
//     - Retorna falso.

// Funcao TentarEncaixar(N, num_quadrados_alvo, posicao_atual):
//   - Tenta ENCAIXAR fisicamente (usando recursao/backtracking)
//     o CONJUNTO de quadrados (ja definido) na grade.
//   - Comeca a tentar na posicao_atual.
//   - Se conseguir encaixar todos:
//     - Guarda a sequencia de encaixes.
//     - Retorna verdadeiro.
//   - Se nao conseguir encaixar:
//     - Retorna falso.

// --- Funcoes Auxiliares (Manipulacao da Grade) ---
//   - CriarBorda(N)
//   - MarcarQuadrado(posicao, tamanho, ocupado_ou_livre)
//   - VerificarAreaLivre(posicao, tamanho)
//   - AcharProximaPosicaoLivre(posicao_atual)

// --- Programa Principal ---

// Definir Modo de Operacao: GERAR_SOLUCOES ou USAR_SOLUCOES_PRONTAS

Se Modo == GERAR_SOLUCOES:
  // Calcula ou deriva solucoes para N de 2 ate MAX_N
  Para N de 2 ate MAX_N:
    Se N for composto (ex: 6):
      // Cria solucao para N escalando a solucao de um fator (ex: 3)
      DerivarSolucao(N)
    Senao (N for primo ou base):
      // Encontra o menor numero K de quadrados que funciona
      K = 2 // Numero minimo inicial a testar
      Repetir:
        // Verifica se existe combinacao de K quadrados que encaixa
        Resultado = AcharCombinacao(N, K)
        Se Resultado == verdadeiro:
          Interromper // Achou o menor K para N
        Senao:
          K = K + 1 // Tenta com K+1 quadrados
      // Salva K e a sequencia de encaixe como solucao para N
    // Imprime a solucao encontrada (para copiar/colar no codigo)

Senao (Modo == USAR_SOLUCOES_PRONTAS):
  // Le casos de teste e imprime resultados pre-calculados
  Ler quantidade_testes
  Enquanto houver testes:
    Ler N
    Imprimir solucao_pre_calculada_para[N]

// 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
4
3
7
```

output esperado

```
4
1 1 2
3 1 2
1 3 2
3 3 2
6
1 1 2
3 1 1
1 3 1
2 3 1
3 2 1
3 3 1
9
1 1 4
5 1 3
1 5 3
4 5 2
4 7 1
5 4 1
5 7 1
6 4 2
6 6 2
```