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

Relatório atividade 6 - Steps

BCC402 - ALGORITMOS E PROGRAMACAO AVANCADA

Kayo Xavier Nascimento Cavalcante Leite - 21.2.4095

Professor: Rafael Alves Bonfim

Ouro Preto
31 de março de 2025

Sumário

Lista de Códigos Fonte

1 Código e enunciado.

Na Atividade 6 - o problema selecionado foi Steps. O Problema tem como objetivo determinar o número mínimo de passos para ir de x a y em uma linha reta, onde cada passo pode variar no máximo 1 unidade em relação ao anterior, com primeiro e último passos sendo 1. 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/limon2009/ACM-UVA-Online-Judge-solution/blob/master/846.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++ Steps.cpp -o executavel

e para rodar basta utilizar .\executavel no cmd.

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

2 Problema 846: Steps

Objetivo: Determinar o número mínimo de passos para ir de x a y em uma linha reta, onde cada passo pode variar no máximo 1 unidade em relação ao anterior, com primeiro e último passos sendo 1.

2.1 Regras dos Passos

- Primeiro e último passo devem ser de tamanho 1.
- Cada passo subsequente pode ser $+1$, 0, ou -1 em relação ao anterior.
- Passos não podem ser negativos.

2.2 Entrada e Saída

- **Entrada**:
  - Número de casos de teste n.
  - Para cada caso: dois inteiros x e y ($0 \leq x \leq y < 2^{31}$).
- **Saída**: Número mínimo de passos para cada caso.

2.3 Estratégia de Solução

Passos:

1. **Pré-Cálculo**:
   - Gere o array A onde $A[i] = i \times (i + 1)$. Este valor representa a soma máxima alcançável com uma sequência de passos otimizada.
2. **Análise da Diferença**:
   - Calcule $target = y - x$.
   - Se $target = 0$, retorne 0 passos.

3. **Busca Binária**:
   - Encontre o maior p onde $A[p] < target$ usando busca binária modificada.
4. **Determinação do Resultado**:
   - Se $target \leq A[p] + (p+1)$, o número de passos é $2p+1$.
   - Caso contrário, o número de passos é $2p+2$.

2.4 Detalhes Matemáticos

- **Soma Máxima por Passos**:

$$A[p] = \sum_{k=1}^{p} k + \sum_{k=p}^{1} (k-1) = p(p+1)$$

- **Intervalo de Decisão**:
  - Se $target$ está na primeira metade do intervalo $[A[p], A[p+1]]$, usa-se $2p+1$ passos.
  - Se está na segunda metade, usa-se $2p+2$ passos.

```
Algoritmo CalcularPassos

Constantes:
  LIMITE = 46341 // Limite para pre-calculo

Variaveis Globais:
  A[0..LIMITE]: Array para guardar valores pre-calculados (A[i] = i * (i+1))

Funcao Precalcular():
  A[0] = 0
  PARA i DE 1 ATE LIMITE:
    A[i] = i * (i + 1)
Fim Funcao

// Funcao de busca: encontra o maior indice p tal que A[p] < chave
Funcao BuscarIndiceP(chave):
  // Implementa busca (binaria ou linear) no array A
  // Retorna p tal que A[p] < chave <= A[p+1]
  // (Detalhes da busca binaria especifica omitidos por clareza)
  p = ... // Resultado da busca
  RETORNAR p
Fim Funcao

Funcao CalcularResultado(n, m):
  alvo = m - n
  p = BuscarIndiceP(alvo)
  diferenca = alvo - A[p] // Quanto alvo excede A[p]

  // Verifica se alvo esta na primeira ou segunda metade do intervalo (A[p], A
      [p+1]]
  // O tamanho do intervalo e 2*p + 2. O ponto medio e A[p] + p + 1.
  SE diferenca <= p + 1:
    resultado = 2*p + 1
  SENAO:
    resultado = 2*p + 2

  IMPRIMIR resultado
Fim Funcao

```

```
Funcao Principal():
  Precalcular() // Chama a pre-computacao uma vez

  LER num_casos_teste
  ENQUANTO num_casos_teste > 0:
    LER n, m
    SE n == m:
      IMPRIMIR 0
    SENAO:
      CalcularResultado(n, m)
    num_casos_teste = num_casos_teste - 1
  FIM ENQUANTO
Fim Funcao
```

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 e testinput.txt, sendo o primeiro o próprio caso de teste disponibilizado pelo exercício e o segundo caso de teste encontrado na plataforma https://www.udebug.com/. 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
45 48
45 49
45 50
```

output esperado

```
3
3
4
```

Os demais testes se encontram no diretório da atividade