Relatório 1º projecto ASA 2021/2022

Grupo: tp015

Aluno(s): José Cutileiro (99097) e João Costa (99088)

1º Problema - LIS

Descrição do Problema e da Solução

A resolução deste problema tem como base o uso de uma estrutura de elementos. Cada elemento guarda o seu valor, um ponteiro para o elemento seguinte e dois valores adicionais*. Estes permitem obter resultados mais rapidamente (elementos de programação dinâmica). A solução é construída durante o processo de leitura dos valores, poupando assim pelo menos um ciclo de execução.

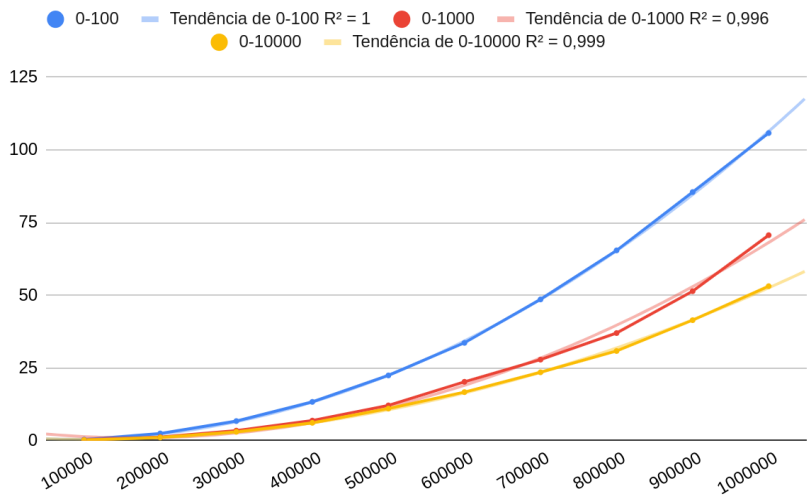
*lis: tamanho da maior subsequência & occur: número de ocorrências da lis

Análise Teórica

- Inicializar o primeiro elemento, O(1)
- Leitura dos valores,num ciclo while, e obter vetores que resolvem o problema O(n)
    - Inicializar novo elemento, O(1)
    - Atualizar a LIS*, O(n), ((melhor caso: $\Omega(1)$*)
    - Inserir o novo elemento no local adequado, O(n), (melhor caso: $\Omega(1)$*)
- Obter respostas finais de acordo com os vetores auxiliares, O(n)

Complexidade global da solução: O(n^2) e no melhor caso $\Omega(n)$

*o melhor caso é quando a sequência dada é crescente



Avaliação Experimental dos Resultados

Para cada testagem, escolhemos uma amostra de 3 intervalos de valores*. Podemos ver que o nosso algoritmo favorece testes com menos valores repetidos. É também de notar que o nosso algoritmo obtém o resultado mais rapidamente se tivermos uma sequência ordenada (ou quase ordenada).

*valores entre: 1 e 100 // 1 e 1000 // 1 e 10000.

Conclusão: O gráfico está de acordo com a teoria lecionada, gerando o início de uma parábola, representando um algoritmo de complexidade O(n^2). (aproximação a n² de 1).

2º Problema - LCIS

Descrição do Problema e da Solução

A resolução deste problema tem como base elementos básicos de programação dinâmica, utilizando por isso um vetor com memória adicional*. A solução é construída em simultâneo com a leitura da segunda sequência permitindo a otimização de pelo menos um ciclo.

*LCIS - Longest common increasing subsequence

Recursão utilizada

LCIS - No código em c++, de modo a evitar guardar valores obsoletos que só gastariam tempo e memória cingimos a LCIS a um vetor unidimensional, mas caso LCIS fosse uma matriz (2x2) com o grupo de valores completo teríamos:

$$\mathrm{LCIS}(i,j) = \begin{cases} max(Temp(i,j)+1, max[0,1,..,j-1](LCIS(i))) & \text{if } seq1(i) = seq2(j) \\ 0 & \text{otherwise.} \end{cases}$$

Temp - No código escrito em c++, temp é um vetor temporário tratando apenas do problema de cada iteração, mas se caso tivéssemos acesso a todos os valores existentes em Temp teríamos:

$$\mathrm{Temp}(i+1,j) = \begin{cases} max(Temp(i,j), max[0,1,..,j-1](LCIS(i))) & \text{if } seq1(i) < seq2(j) \\ Temp(i,j) & \text{otherwise.} \end{cases}$$

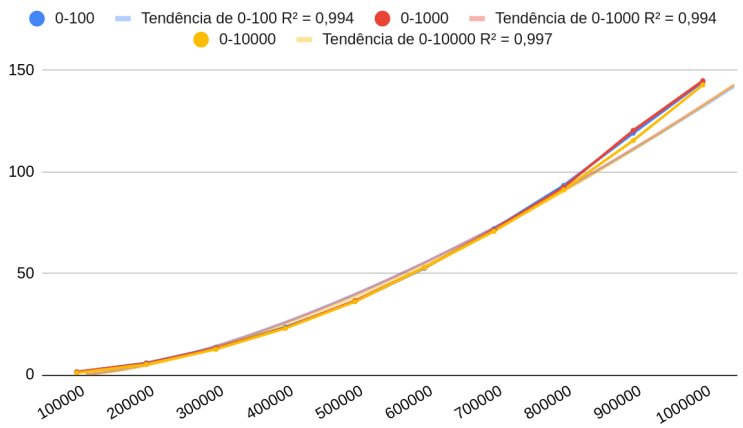
Nota: max{k=0;j-1}(LCIS(i) representa o valor máximo de do vetor LCIS(i) desde k=0 até k = j-1

Análise Teórica

- Leitura e análise dos dados de entrada, num ciclo *while*, e obtenção da solução , O($n + m$)
  - Construção da sequência 1 e hash auxiliar*, O(1)
  - Processamento das entradas da sequência 2, ignorando as que não constam na sequência 1, resultando em O(m) (no melhor caso $\Omega(1)$)
- Apresentação do maior valor encontrado, percorrendo uma lista de soluções, O(n)

Complexidade global da solução: O($(m) * (n + m)$) e no melhor caso $\Omega(n + m)$

*permitirá uma procura com complexidade O(1) no processamento da próxima sequência



Avaliação Experimental dos Resultados

Para cada instância de teste, optamos por correr cada uma 3 vezes, com intervalos de valores diferentes, para comparar entre ocasiões com valores repetidos e valores distintos*. Apesar disso, não existem alterações significativas nos resultados.

*valores entre: 0 e 99 // 0 e 999 // 0 e 9999

Conclusão: O gráfico está de acordo com a teoria lecionada, criando a parte inicial do que virá a ser uma parábola, concordando com a complexidade do algoritmo, O(n * m) para valores aleatórios. (testes feitos com n = m).