Sorting Algorithms

Santiago Toll Leyva
Universidad de Artes Digitales

Guadalajara, Jalisco

Email: idv16a.stoll@uartesdigitales.edu.mx

Profesor: Efraín Padilla

Agosto 05, 2019

Brute Force vs Dynamic Programming

Teoria

El metodo de fuerza bruta realiza todas las operaciones necesarias para completar el algoritmo; cumple su funcion sin almacenar datos en memoria. Por otro lado la programacion diamica consiste en reutilizar resultados previos para evitar computo innecesario; este metodo utiliza mas memoria pero en muchos casos reduce el tiempo de ejecucion del algoritmo drasticamente.

Planteamiento del problema

Utilizar ambas tecnicas utilizando el algoritmo ”Longest Common Subsequence” que se encarga de encontrar la subsecuencia mas larga que existe en dos cadenas.

Solucion del problema

Se utilizan dos cadenas de 20 caracteres para probar el algoritmo; el tiempo de ejecucion de cada funcion es medido para comparar la diferencia entre el tiempo de ejecucion de cada una.

Tambien se agrego una tercer funcion que se encarga de imprimir la subsecuencia mas larga(esta funcion solo se implemento en Dynamic Programming ya que obtenerla con Brute Force tomaba demasiado tiempo).

Codigo

Longest Common Subsequence(Brute Force).

Listing 1. LCS(Brute Force) C++

```
/*****************************************************************/
/* Longest Common Subsequence(Brute Force)                       */
/*****************************************************************/
int BruteLCS(string first, string second,int firstSize,int secondSize) {
  if (firstSize == 0 || secondSize == 0)
  {
    return 0;
  }

  else if (first[firstSize - 1] == second[secondSize - 1])
  {
    return 1 + BruteLCS(first, second, firstSize - 1, secondSize - 1);
  }
  else
  {
    return Max(BruteLCS(first, second, firstSize, secondSize - 1), BruteLCS(first, second, firs
  }

}
```

Longest Common Subsequence(Dynamic Programming).

Listing 2. LCS(Dynamic Programming) C++

```
/*****************************************************************/
/* Longest Common Subsequence(Dynamic Programming)               */
/*****************************************************************/
int DynamicLCS(char * first, char * second)
{
  int lcs[firstLenght+1][secondLenght+1];
  int i, j;

  for (i = 0; i <= firstLenght; i++)
  {
    for (j = 0; j <= secondLenght; j++)
    {
      if (i == 0 || j == 0)
        lcs[i][j] = 0;

      else if (first[i - 1] == second[j - 1])
        lcs[i][j] = lcs[i - 1][j - 1] + 1;

      else
        lcs[i][j] = Max(lcs[i - 1][j], lcs[i][j - 1]);
    }
  }
  return lcs[firstLenght][secondLenght]-1;
}
```

Print Longest Common Subsequence(Dynamic Programming).

Listing 3. Print LCS(Dynamic Programming) C++

```
/*****************************************************************/
/* Print Longest Common Subsequence (Dynamic Programming)        */
/*****************************************************************/
int PrintLCS(char * first, char * second)
{
  int lcs[firstLenght + 1][secondLenght + 1];
  int i, j;

  for (i = 0; i <= firstLenght; i++)
  {
    for (j = 0; j <= secondLenght; j++)
    {
      if (i == 0 || j == 0)
        lcs[i][j] = 0;

      else if (first[i - 1] == second[j - 1])
        lcs[i][j] = lcs[i - 1][j - 1] + 1;

      else
        lcs[i][j] = Max(lcs[i - 1][j], lcs[i][j - 1]);
    }
    const int index = lcs[firstLenght + 1][secondLenght + 1];

    int i, j;
    for (i = firstLenght, j = secondLenght; i > 0 && j > 0;) {
      if (first[i - 1] == second[j - 1])
      {
        //Insert char at the beginning of string.
        currentLongest.insert(currentLongest.begin(), first[i - 1]);
        i--;
        j--;
      }
      else if (lcs[i - 1][j] > lcs[i][j - 1])
      {
        i--;
      }
      else
      {
        j--;
      }
    }
  }
  currentLongest.resize(lcs[firstLenght][secondLenght] - 1);
  return lcs[firstLenght][secondLenght] - 1;
}
```

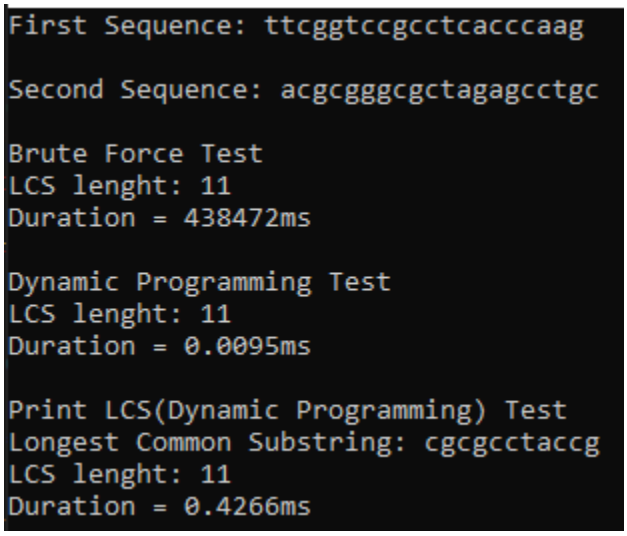
Driver Function.

Listing 4. LCS Test C++

```
/*****************************************************************/
/* Driver Function                                               */
/*****************************************************************/
int main()
{
  //SetSequences();
  std::cout << "First Sequence: " << firstSequence << std::endl << std::endl;
  std::cout << "Second Sequence: " << secondSequence << std::endl << std::endl;

  //Brute Force Test
  std::cout << "Brute Force Test" << std::endl;
  auto startTime = high_resolution_clock::now();
  int longest = BruteLCS(firstSequence,secondSequence,strlen(firstSequence),strlen(secondSeque
  auto endTime = high_resolution_clock::now();
  duration<float, std::milli> duration = (endTime - startTime);
  cout << "LCS lenght: " << longest << std::endl;
  cout << "Duration = " << duration.count() << "ms" << std::endl;

  //Dynamic Programming Test
  std::cout << std::endl << "Dynamic Programming Test" << std::endl;

  startTime = high_resolution_clock::now();
  longest = DynamicLCS(firstSequence, secondSequence);
  endTime = high_resolution_clock::now();
  duration = (endTime - startTime);
  cout << "LCS lenght: " << longest << std::endl;
  cout << "Duration = " << duration.count() << "ms" << std::endl;

  //Print LCS(Dynamic Programming)
  std::cout << std::endl << "Print LCS(Dynamic Programming) Test" << std::endl;
  startTime = high_resolution_clock::now();
  longest = PrintLCS(firstSequence, secondSequence);
  endTime = high_resolution_clock::now();
  duration = (endTime - startTime);
  cout << "Longest Common Substring: " + currentLongest << std::endl;
  cout << "LCS lenght: " << longest << std::endl;
  cout << "Duration = " << duration.count() << "ms" << std::endl;
}
```

Resultados

En la siguiente imagen se puede observar el tiempo que tarda en terminar cada una de las funciones. Como se puede ver Dynamic Programming es mucho mas rapido que Brute force. Dependiendo del problema y la cantidad de memoria con la que se cuenta puede ser una mejor manera de abordar el problema.



Como se puede observar en la imagen los resultados de la prueba fueron los siguientes:
-Brute Force : 438.472 segundos
-Dynamic : 0.0000095 segundos
-Dynamic Print : 0.0004266 segundos

La complejidad de Brute Force es O($2^{(m+n)}$)$dondemynsonlalongituddelaprimerysegundacadenarespectivamente.$
Por otro lado la complejidad de el enfoque dinamico es O(mn); donde m y n representan lo mismo que en el caso de brute force.