University of Nevada, Las Vegas Computer Science 477/656 Spring 2023

Answers to Assignment 3: Due Saturday February 25, 2023

1. Compute the Levenshtein edit distance between "`kitchen`" and "`chicken`." Show the matrix.

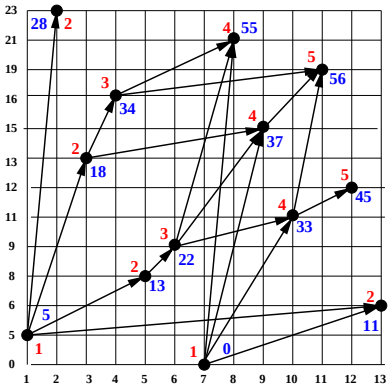
		k	i	t	c	h	e	n
	0	1	2	3	4	5	6	7
c	1	1	2	3	3	4	5	6
h	2	2	2	3	4	3	4	5
i	3	3	2	3	4	4	4	5
c	4	4	4	3	3	4	5	5
k	5	4	5	4	4	4	5	6
e	6	5	5	5	5	5	4	5
n	7	6	6	6	6	6	5	4

2. I stated this problem incorrectly, and there is no way you could have worked it. Therore, it will not be graded.

3. Consider problem of finding the maximum sum monotone increasing subsequence of

   5, 23, 13, 16, 8, 9, 0, 21, 15, 11, 19, 12, 6

   Draw a figure of the corresponding directed acyclic graph, as defined in the handout `dynamic.pdf`.



   Along the vertical axis, only the values of terms of the sequence are shown. The labels on the horizontal axis refer to position in the sequence. For example, the 6^{th} term of the sequence is 9. The definition of the graph given in the handout requires an arc from (i, k) to (j, ℓ) if $i < j$ and $k < \ell$. However, to reduce clutter, only the transitive reduction of that graph is drawn. For example, the arc from (5,8) to (8,15) is not drawn, because it is in the transitive closure of the arcs from (5,8) to (6,9) and from (6,9) to (8,15).

   Using dynamic programming, two numbers are computed at each vertex. The red numeral indicates the maximum length of any monotone increasing subsequence which ends at that vertex, and the blue numeral indicates the maximum sum of any monotone increasing subsequence ending at that vertex. We conclude that the maximum length of any monotone increasing subsequence is 5, and there are two solutions: 5,8,9,11,12 and 5,8,9,15,19. The maximum sum of any monotone increasing subsequence is 56, the sum of the subsequence 5,8,9,15,19.

4. Write an algorithm that solves the following problem. Given a sequence of coins of positive value, find the maximum value set of coins which do not contain any three coins that are consecutive in the initial sequence. You need to describe the algorithm in English, not C++ code. (See the handout `dynamic.pdf`)

   There are several ways to work this problem. Here is what I think is the easiest to understand. Let $x[i]$ be the value of the i^{th} coin. We say that a set of coins is *legal* if it has no three consecutive coins. Let

$F[i]$ be the maximum value of any legal subset of the first i coins. $F[n]$ is the maximum value of any legal subset of the entire row.

The non-recursive part is to let $F[0] = 0$, $F[1] = x[1]$, and $F[2] = x[1] + x[2]$.

We now consider the recursive case, namely $i \geq 3$. There are three possible legal subsets to consider.

Case 1: It is best not to choose the i^{th} coin. In that case, $F[i] = F[i-1]$.

Case 2: It is best to choose the i^{th} coin, but not the $(i-1)^{\text{st}}$ coin. In that case, $F[i] = x[i] + F[i-2]$.

Case 3: It is best to choose both of the last two coins. In that case, the $(i-2)^{\text{nd}}$ coin will not be chosen, and $F[i] = x[i] + x[i-1] + F[i-3]$.

Our dynamic program is to compute the $F[i]$ for $i \geq 3$ in order of increasing i, using the formula:

$$F[i] = \max(F[i-1], x[i] + F[i-2], x[i] + x[i-1] + F[i-3])$$

5. For this problem, see the handout `recursive.pdf`. Solve the following recurrences, giving the answers using Θ notation.

   Use the anti-derivative method:

   (a) $F(n) = F(n-1) + n^3$

   $F(n) = \Theta(n^4)$, the the anti-derivative of x^3 is $\frac{x^4}{4} = \Theta(x^4)$.

   (b) $F(n) = F(n - \sqrt{n}) + \sqrt{n}$

   Move $F(n - \sqrt{n})$ to the left side, then divide both sides by $\sqrt{n}$. We obtain

   $$\frac{F(n) - F(n - \sqrt{n})}{\sqrt{n}} = 1$$

   The left side is approximately the derivative of $F(n)$, since $\sqrt{n}$ is close enough to zero. The anti-derivative of 1 is n. Thus, $F(n) = \Theta(n)$.

   Use the master theorem:

   (c) $F(n) = F(n/4) + 1$

   $F(n) = \Theta(\log n)$

   (d) $F(n) = 2F(n/4) + \sqrt{n}$

   $F(n) = \Theta(\sqrt{n} \log n)$

   (e) $F(n) = 4F(n/2) + n$

   $F(n) = \Theta(n^2)$

   (f) $F(n) = 4F(n/2) + n^3$

   $F(n) = \Theta(n^3)$

6. Consider the following recursive function.

```
int f(int n)
 // input condition: n > 0
 {
  if(n <= 2) return n;
  else return f(n/2)+f(n/2+1)+n;
 }
```

(a) Using the master theorem, compute the asymptotic complexity of the function $f(n)$, expressing the answer using Θ notation.

Approximating n/2+1 to n/2, we obtain the recurrence:

$$f(n) = 2f(n/2) + n$$

Thus $f(n) = \Theta(n \log n)$ by the master theorem.

(b) Using the master theorem, compute the time complexity of the the program. Let $T(n)$ be the time to compute $f(n)$. Our recurrence is

$$T(n) = 2T(n/2) + 1$$

since it only takes one step to compute n, not n steps. Thus $T(n) = \Theta(n)$ by the master theorem.

(c) What follows is a portion of a C++ program which computes $f(n)$ using dynamic programming. Fill in the missing code in the `main()`.

```
int const n;
int f[n+1];

int main()
 {
  f[1] = 1;
  f[2] = 2;
  // FILL IN THE MISSING CODE HERE
  for(int i = 3; i <= n; i++)
   f[i] = f[i/2]+f[i/2+1]+i;
  cout << "f(" << n << ") = " << f[n] << endl;
  return 1
 }
```

(d) What is the asymptotic time complexity of your code?

$\Theta(n)$