Computer Science 3711
Winter 2004

Midterm Exam

Answers

1. (9 marks) Circle the letters associated with the appropriate answers in the following multiple choice questions. Note that some of these questions may have more than one answer whose letter needs to be circled.

(a) (3 marks) Which of the following is true of the function $T(n) = 2^{n-4}$?

a) $T(n)$ is $\Theta(2^{n+3})$ b) $T(n)$ is $\Omega(n^{1000})$ c) $T(n)$ is $O(2^{n-10})$

Answer:

- **Part (a) is true**: $c_1 2^{n+3} \leq 2^{n-4} \leq c_2 2^{n+3}$ when $c_1 = 2^{-7}$, $c_2 = 1$, and $n \geq n_0 = 1$.
- **Part (b) is true**: This can be restated as follows:

$$\begin{aligned} 2^{n-4} &\geq cn^{1000} \\ \log_2 2^{n-4} &\geq \log_2 cn^{1000} \\ n - 4 &\geq \log_2 c + 1000 \log_2 n \\ n - (4 + 1000 \log_2 n) &\geq \log_2 c \end{aligned}$$

  As $n - (4 + 1000 \log_2 n) \geq 0$ when $n \geq 4 + 1000 \log_2 n$ and $\log_2 1 = 0$, the above holds for $c = 1$ and $n \geq n_0 = 2^{14}$.
- **Part (c) is true**: $2^{n-4} = \frac{1}{2^4} 2^n \leq c2^{n-10} = \frac{c}{2^{10}} 2^n$ when $c = 2^6$ and $n \geq n_0 = 1$.

(b) (3 marks) Which of the following is true of the function $T(n) = 9^{\log_3 n}$?

a) $T(n)$ is $O(2^{\frac{n}{64}})$ b) $T(n)$ is $\Theta(3^{\log_9 n})$ c) $T(n)$ is $\Omega(n \log_2 n^2)$

Answer: A key simplification here is $9^{\log_3 n} = (3 \cdot 3)^{\log_3 n} = 3^{\log_3 n} \cdot 3^{\log_3 n} = n \cdot n = n^2$.

- **Part (a) is true**: This can be restated as follows:

$$\begin{aligned} n^2 &\leq c2^{\frac{n}{64}} \\ \log_2 n^2 &\leq \log_2 c2^{\frac{n}{64}} \\ 2 \log_2 n &\leq \log_2 c + \frac{n}{64} \\ 2 \log_2 n - \frac{n}{64} &\leq \log_2 c \\ 2 \log_2 n - \frac{n}{2^6} &\leq \log_2 c \end{aligned}$$

  As $2 \log_2 n - \frac{n}{2^6} \leq 0$ when $2 \log_2 n \leq \frac{n}{2^6}$ and $\log_2 1 = 0$, the above holds for $c = 1$ and $n \geq n_0 = 2^{11}$.

- **Part (b) is false**: Note that $3^{\log_9 n} = n^{\log_9 3}$ and $\log_9 3 = \frac{1}{2}$. As $n^2/n^{\frac{1}{2}}$ goes to ∞ as n goes to ∞, $9^{\log_3 n} = n^2 \not\leq cn^{\frac{1}{2}} = c3^{\log_9 n}$ for any constant c when n is sufficiently large.
- **Part (c) is true**: $cn \log_2 n^2 = 2cn \log_2 n \leq n^2$ when $c = \frac{1}{2}$ and $n \geq n_0 = 1$.

(c) (3 marks) Which of the following is true of the function $T(n) = n^2$?

a) $T(n)$ is $\Theta((n-5)^3)$ b) $T(n)$ is $O(57 \log_2 n)$ c) $T(n)$ is $\Omega((n+4)^2)$

Answer:

- **Part (a) is false**: $n^2 \leq c(n-5)^3$ when $c = 1$ and $n \geq n_0 = 10$ (as $10^2 = 100 < 125 = 5^3 = (10-5)^3$). However, $n^2 \geq c(n-5)^3 \rightarrow \frac{n^2}{(n-5)^3} \geq c$ is not true for any constant c relative to sufficiently large n as the term $\frac{n^2}{(n-5)^3}$ goes to 0 as n goes to ∞.
- **Part (b) is false**: $c(57 \log_2 n) \geq n^2 \rightarrow c \geq \frac{n^2}{57 \log_2 n}$ is not true for any constant c relative to sufficiently large n as the term $\frac{n^2}{57 \log_2 n}$ goes to ∞ as n goes to ∞.
- **Part (c) is true**: As $(n+4)^2 = n^2 + 8n + 16 \leq n^2 + 8n^2 + 16n^2 = 25n^2$, $c(n+4)^2 \leq n^2$ when $c = \frac{1}{25}$ and $n \leq n_0 = 1$.

2. (12 marks)

a) (4 marks) Give the recurrence for the time complexity of the following recursive algorithm:

```
procedure FUNKY-REC(m, n)
    if (m < 1)
        sum = 0
        for (i = 1; i <= n; i++)
            sum = sum + (FUNKY-ITR(m) / i) - j
        return(sum)
    else if (m == 1)
        sum = 10
        for (i = 1; i <= m; i++)
            sum = sum + FUNKY-ITR(n) / i * j
        return(sum)
    else if (m > 1)
        sum = 0
        for i = 1 to n * n do
            l = FUNKY-REC(m/5, n)
            sum = FUNKY-ITR([square root of n]) * l
        return(sum)
    else
        return(m)
```

Assume that procedure `FUNKY-ITR(x)` runs in $O(x^4)$ time.

Answer:

$$T(m,n) = \begin{cases} O(n) & m < 1 \\ O(n^4) & m = 1 \\ n^2 T(\frac{m}{5}, n) + O(n^4) & m > 1 \end{cases}$$

b) (8 marks) Derive an upper bound for $T(n)$ using any method discussed in class or in the textbook, where

$$T(n) = \begin{cases} 0 & n \leq 3 \\ T(n-4) + cn & n > 3 \end{cases}$$

Answer:

$$\begin{aligned}
T(n) &= T(n-4) + cn \\
&= cn + T(n-4) \\
&= cn + c(n-4) + T(n-8) \\
&= cn + c(n-4) + c(n-8) + T(n-12) \\
&= cn + c(n-4) + c(n-8) + c(n-12) + T(n-16) \\
&\cdots \\
&= (\sum_{i=0}^{\frac{n}{4}} c(n-4i)) + 0 \\
&= (cn \sum_{i=0}^{\frac{n}{4}} 1) - (4c \sum_{i=0}^{\frac{n}{4}} i) \\
&= cn(\frac{n}{4}) - 4c(\frac{\frac{n}{4}(\frac{n}{4}+1)}{2}) \\
&= \frac{4c}{16}n^2 - \frac{2c}{16}(n^2 + 4n) \\
&= \frac{2c}{16}n^2 - \frac{8c}{16}n \\
&\leq \frac{2c}{16}n^2 \\
&= O(n^2)
\end{aligned}$$

3. (9 marks) Consider the following algorithm:

```
sum = 0;
for i = 1 to n * n do
    j = T1(n)
    if (T2(i, j, n)
        for k = 1 to n do
            if ((k % 2) == 0)
                l = T3(n, k) - 2
            sum = (sum * j / l)
    sum = sum * T3(n)
if ((sum % 2) == 0)
    sum = sum / 2
else
    sum = sum * 2
```

a) (3 marks) Give the parameterized asymptotic worst-case time complexity for this algorithm. Please include a term for how many times T2 evaluates to `true` in your expression.

Answer:

$$T(n) = n^2(T(T1) + T(T2) + T(T3)) + f(T2() == true)nT(T3)$$

b) (3 marks) Give the asymptotic worst-case time complexity of this algorithm when the T1, T2, and T3 operations require $O(1)$, $O(n^2)$, and $O(1)$ time, respectively, and we don't know how many times T2 evaluates to `true`.

Answer: In the worst case, we assume that T2 always evaluates to `true`. Hence, $f(T2() == true) = n^2$, giving us

$$\begin{aligned} T(n) &= n^2(T(T1) + T(T2) + T(T3)) + f(T2() == true)nT(T3) \\ &= n^2(O(1) + O(n^2) + O(1)) + n^2 n O(1) \\ &= O(n^2) + O(n^4) + O(n^2) + O(n^3) \\ &= O(n^4) \end{aligned}$$

c) (3 marks) Give the asymptotic worst-case time complexity of this algorithm when the T1, T2, and T3 operations require $O(n^3)$, $O(\log_2 n)$, and $O(n^2)$ time, respectively, and we know that T2 evaluates to `true` $O(n \log_2 n)$ times.

Answer:

$$\begin{aligned} T(n) &= n^2(T(T1) + T(T2) + T(T3)) + f(T2() == true)nT(T3) \\ &= n^2(O(n^3) + O(\log_2 n) + O(n^2)) + (n \log_2 n)nO(n^2) \\ &= O(n^5) + O(n^2 \log_2 n) + O(n^4) + O(n^4 \log_2 n) \\ &= O(n^5) \end{aligned}$$

4. (18 marks)

a) (4 marks) How are problems that have only combinatorial solution-space tree (CST) algorithms different from problems that have divide-and-conquer algorithms? Phrase your answer in terms of the problem properties given in class.

Answer: Problems that have divide-and-conquer algorithms satisfy the optimal substructure property, *i.e.*, there is recursive decomposition of the problem such that the solution to an instance of a particular size can be stated in terms of the solutions of problems of the same type on smaller-size inputs, while problems that have only CST algorithms do not satisfy this property.

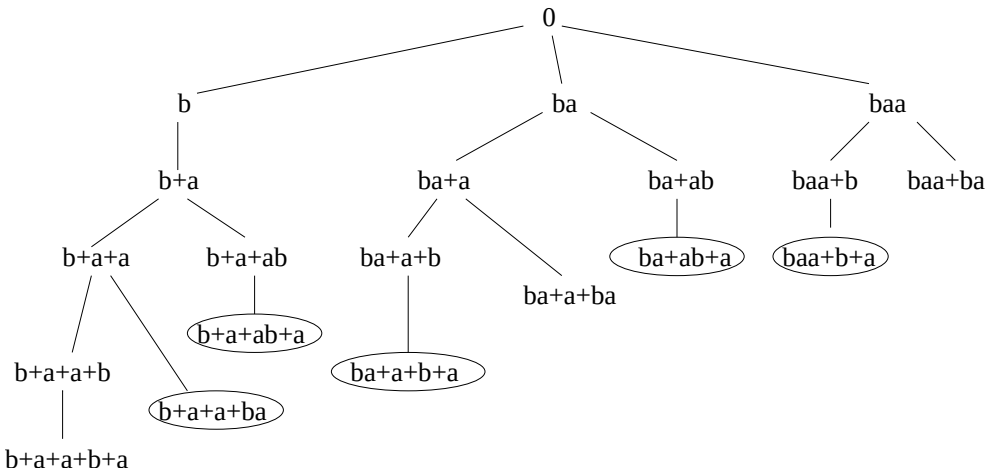
b) (14 marks) Consider the following algorithm:

```
DFS-V(sol, C, num)
    if (SIZE(sol) > num)
        print("pruned: decryption uses too many codewords")
    else if (ISCOMPLETE(sol))
        if (SIZE(sol) == num)
            print(sol)
    else
        for (i = 1; i <= NUMCODEWORDS(C); i++)
            if (CANEXTEND(sol, C, i)
                solc = EXTENDCOPY(sol, C, i)
                DFS-V(solc, C, num)
```

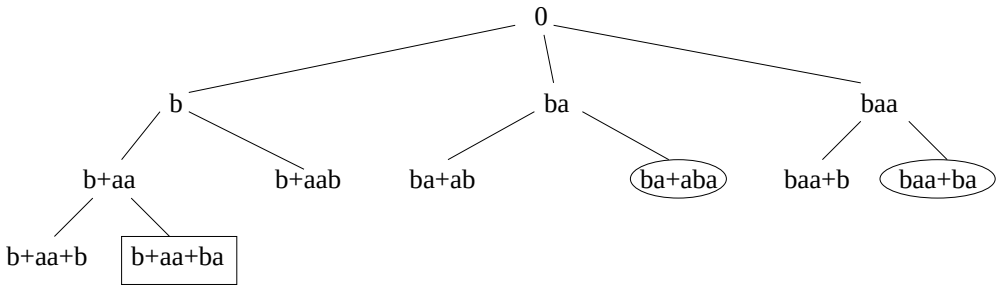
The problem solved here prints all decryption-solutions for a ciphertex T relative to a codeword-dictionary C that have *num* **distinct** codewords in the decryption-sequence (note that this is different from the problem solved in Assignments #2 and 3, in which we were interested in the **total** number of codewords in the decryption-sequence). In this algorithm, *sol* is a (possibly partial) decryption of T relative to C, SIZE(*sol*) returns the number of distinct codewords in the decryption-sequence in *sol*, CANEXTEND(sol, C, i) determines whether or not the decryption in *sol* can be extended by the codeword indexed i in C, and EXTENDCOPY(sol, C, i) returns a copy of *sol* in which the codeword indexed i in C has been added to the decryption-sequence. To answer this question, sketch

the implicit tree of solution-nodes generated by the algorithm above (marking leaf-nodes which are printed by a circle and nodes that are pruned with a square) with the call DFS-V(sol, C, num) when sol is the empty decryption, $T = baaba$ and C and num have the following values:

i) (7 marks) $C = \{a, b, ab, ba, baa\}$ and $num = 3$:



ii) (7 marks) $C = \{b, aa, ab, ba, aba, aab, baa\}$ and $num = 2$:



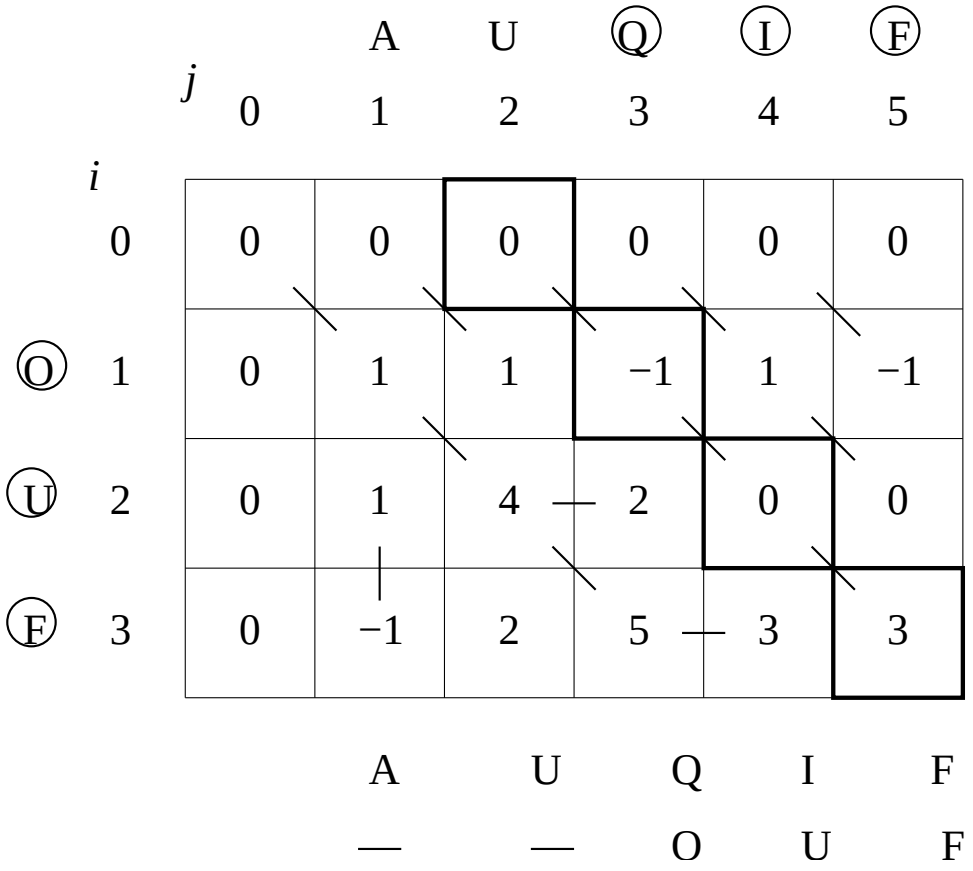
5. (12 marks) The length of the longest weighted common subsequence (LWCS) of two strings s and s' is defined by the following recurrence:

$$D(i,j) = \begin{cases} 0 & i = 0 \text{ or } j = 0 \\ \max(D(i-1,j-1) + COST(s(i), s'(j)), & \\ \quad D(i,j-1) - 2, & \\ \quad D(i-1,j) - 2) & \text{otherwise} \end{cases}$$

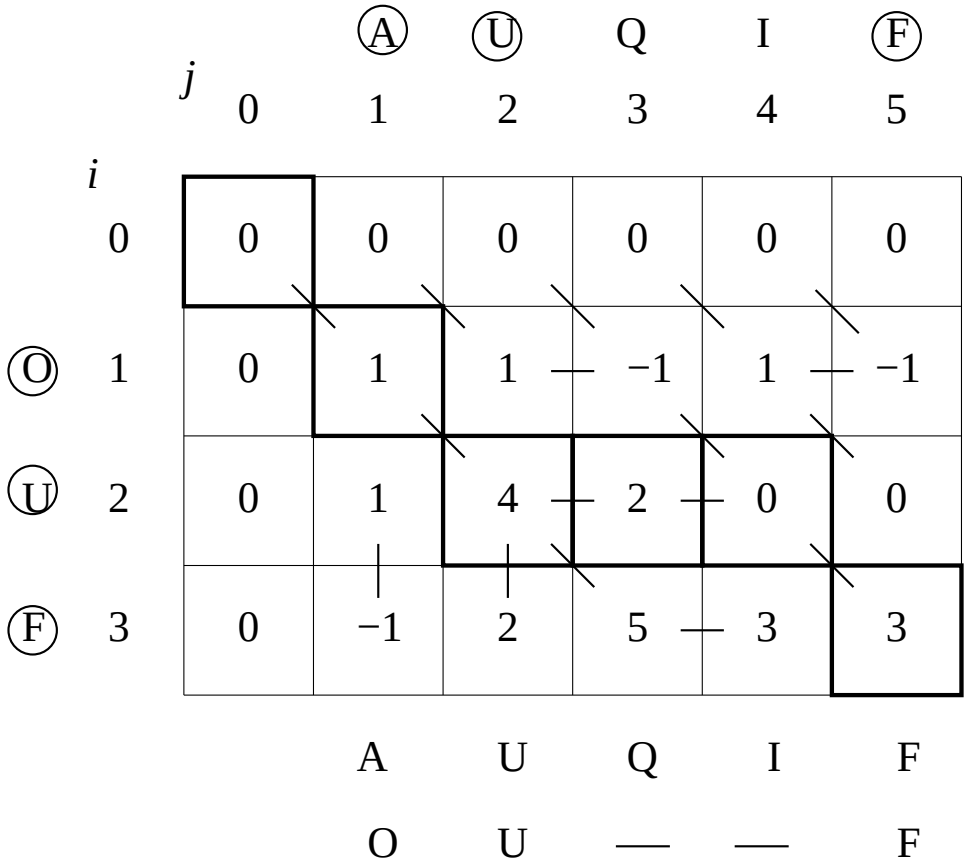
For any two symbols x and y, function $COST(x, y)$ returns 3 if $x = y$, 1 if either (i) x and y are both vowels or (ii) x and y are both consonants, and -1 otherwise (recall that a vowel is one of the letters $\{'A', 'E', 'I', 'O', 'U'\}$ and a consonant is any letter of the alphabet that is not a vowel). Given the above, determine the longest

weighted common subsequence of the strings **AUQIF** and **OUF** – that is, fill in the dynamic programming matrix given below (including one backpointer per matrix-cell), show one of the backpointer paths that gives an optimal LWCS (given that traceback starts at the lower right-hand corner matrix-cell, *i.e.*, $D(|s|, |s'|)$, and goes back to a matrix cell in row 0 or column 0), and show the LWCS corresponding to this path, *i.e.*, the sequence of matching symbol-pairs induced by the recursive subclause $\max(D(i-1, j-1) + COST(s(i), s'(j))$ and its associated diagonal backpointers.

Answer: If one resolves ties for optimal matrix-cell value in the given order of the `max`-cases in the recursive part of the recurrence, *i.e.*, diagonal / left / up backpointer, the solution is as follows:



If, however, we store all optimal-value backpointers, one of the possible solutions is as follows:



The recurrence given here is very closely related to that describing a maximum similarity local alignment of two strings, in which one attempts to find the most similar substrings of two given strings; the interested reader is referred to [1, Section 11.7] for details.

References

[1] D. Gusfield. 1997. **Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology**. Cambridge University Press.