CS170–Fall 2014 — Solutions to Homework 8

Steffan Voges, SID 23434518, `cs170-cz`

October 30, 2014

Collaborators: Ryan Flynn

1. Subsequence

Main idea. The main idea is to iterate through B, while keeping track of whether or not each letter in A has been hit yet.

Pseudocode.

```
def algorithm(A[1...n], B[1...m]):
    for i = 1, 2, ..., m
        if A[0] == B[i]:
            A = A[1:]
    if A is empty: return True
    else: return False
```

Proof of correctness.

Loop Invariant: At the beginning of every loop, the only letters in $A[1...n]$ that remain are those that have not been hit yet in order.

Base Case: Before the first iteration of the loop, no characters in $A[1...n]$ have been hit yet. Therefore, all the characters remain, and our loop invariant holds true.

Inductive Hypothesis: Before the i^{th} iteration of the loop, only the letters that haven't appeared in order from $A[1..i-1]$ remain in A.

Inductive Step: To prove our algorithm true, we need to examine our algorithm at iteration $i+1$. We know that at the beginning of the i^{th} loop, only the letters that haven't appeared in order from $A[1..i-1]$ remain in A appear due to the inductive hypothesis. Let the remaining string of A be $A[k...n]$ such that $1 \leq k \leq n$.

Case 1: $A[k]$ is equal to the next character in B

In this case, $A[k]$ is added to the set of characters we've seen, and is thus deleted from A. Then, only the characters which have been seen in order in A remain, and our loop invariant holds true.

Case 2: $A[k]$ is not equal to the next character in B

In this case, $A[k]$ is not added to the set of characters we've seen, and remains in A. A is still composed only of the characters we've seen in order so far, and our loop invariant holds true.

Thus, by proof by cases, we see that our loop invariant holds true.

By the loop invariant, only the characters from A that haven't been yet in B in order remain in A. Therefore, at the end of the i^{th} iteration, if all the characters in A have been seen, A will be empty and we will return True. If not all the characters in A have been seen, then we know that there does not exist a subsequence of A in B, and we return False since A will not be empty.

Running time. $O(m)$

Justification of running time. You complete m iterations through B, so our running time is $O(m)$.

2. Another scheduling problem

Main idea.
The main idea of the code is to start from the very last hour, and keep track of the maximum times for picking a movement, A's schedule, or B's schedule at every hour. Thus, each problem only needs to worry about the maximum values of the two subproblems to its right.
Pseudocode.

```
def iterative(A, B):
    dp = []
    dp[n][A] = get(A, n)
    dp[n - 1][A] = get(A, n -1) + dp[n][A]
    dp[n][B] = get(B, n)
    dp[n - 1][B] = get(B, n - 1) + dp[n][B]
    for i = n - 2...1:
        dp[i][A] = get(A, i) + max(get(A, i + 1), get (B, i + 2))
        di[i][B] = get(B, i) + max(get(B, i + 1), get(A, i + 2))
    return max(get(A, 1), get(B, 1))

def get(state, hour):
    return maximum power at that state if in range, else return 0
```

Proof of correctness.
Loop Invariant: At the beginning of every loop i, the elements in dp[i + 1...n] hold the maximum values of picking either A, B, or moving at that hour.
Base Case: Before the first iteration of the loop, the only values entered in the array dp are from the last two hours. Since no case with only two hours can be optimal, the maximum values hold only straight choices in either A or B, which are already filled in inside the array. Therefore, our base case holds.
Inductive Hypothesis: Before the i^{th} iteration of the loop, dp[n - i + 1...n] holds the maximum values of A and B at the hour i.
Inductive Step: To prove our algorithm true, we need to examine our algorithm at iteration $i+1$. We know that at the beginning of the i^{th} loop, dp holds the maximum values of A and B at hours n - i + 1...n due to the inductive hypothesis. Let $k =$ hour $n-i$. In our loop, we find the maximum value of picking the next consecutive hour in the same schedule, or skipping the next value and adding the number of seconds in the $k+2^{nd}$ in the other schedule for picking both A and B. Thus, we know that when analyzing hour $k-1$, dp will hold the maximum values of A and B at hours n - 1...n, and our loop invariant holds true.

If we follow our loop invariant all the way through to hour 1, we will have accumulated the maximum values for picking A or B at each hour in the schedule. Thus, we just need one more iteration in the loop to find the maximum value at hour 1 for picking both A and B.
Running time. $O(n)$
Justification of running time. We access the elements in A and B 3 times each per iteration, and we have n iterations, leading to $O(6n) = O(n)$

3. Park Tours

Main idea. YOUR ANSWER GOES HERE
Pseudocode. YOUR ANSWER GOES HERE
Proof of correctness. YOUR ANSWER GOES HERE
Running time. YOUR ANSWER GOES HERE
Justification of running time. YOUR ANSWER GOES HERE

4. Optimal binary search trees

Main idea. YOUR ANSWER GOES HERE
Pseudocode. YOUR ANSWER GOES HERE
Proof of correctness. YOUR ANSWER GOES HERE
Running time. YOUR ANSWER GOES HERE
Justification of running time. YOUR ANSWER GOES HERE

5. Beat inference

Main idea. YOUR ANSWER GOES HERE
Pseudocode. YOUR ANSWER GOES HERE
Proof of correctness. YOUR ANSWER GOES HERE
Running time. YOUR ANSWER GOES HERE
Justification of running time. YOUR ANSWER GOES HERE

6. Optional Bonus Problem: Image re-sizing

Main idea. YOUR ANSWER GOES HERE
Pseudocode. YOUR ANSWER GOES HERE
Proof of correctness. YOUR ANSWER GOES HERE
Running time. YOUR ANSWER GOES HERE
Justification of running time. YOUR ANSWER GOES HERE