*Introduction to Algorithms: 6.006*
Massachusetts Institute of Technology
Erik Demaine, Ronald L. Rivest, and Nir Shavit

November 22, 2013
Handout # 6

# Problem Set 6, Problem 1

DeJuan Anderson
Collaborators: None
Recitation: WF10

---

This problem set is due **Thursday, December 5** at **11:59PM**.

This solution template should be turned in through our submission site. [1]

For written questions, full credit will be given only to correct solutions that are described clearly *and concisely.*

Please fill in the TA and recitation section you attend. Otherwise you may not be able to get your problem sets back in section!

---

[1] Register an account, if you haven't done so. Then go to Homework, Problem Set 6, and upload your files.

**Problem 6-1.** [35 points] **Party Bus**

You're on one of the new Party+Tour Buses, which serves as both a party bus and a tour bus. The bus makes $n$ stops, $s_1, s_2, \ldots, s_n$. You only have time to get out at $k$ of the stops to see tourist attractions. (You need $n - k$ time to party!) You may decide to stop at fewer stops, however. If you get out at stop $s_i$, you get $h_i$ happiness points. Additionally, each time you decide to party instead of getting out, if you have just partied for $j \geq 0$ consecutive stops, then you get $j$ additional happiness points. (Thus your happiness grows roughly quadratically with long stretches of partying.) Design and analyze an efficient dynamic programming algorithm to compute the optimal $\leq k$ stops to get out at, in $O(n^2k)$ time.

========== ==========

========== ==========

Since we're designing this algorithm with Dynamic Programming, let's follow the 5 steps to DP:

1)Define the subproblems.

2)Guess an aspect of the solution.

3)Relate the solutions of our various subproblems.

4)Recurse + Memoize OR build Bottom-Up.

5)Solve the original problem.

## 1) What are the subproblems?

Each subproblem is whether or not to get off at a stop or to stay on the bus and party harder. Choose based on the decision which results in maximal happiness.

## 2) What aspect of the solution shall we guess?

Let's guess whether or not to get off at the last stop. Thus, our subproblems are prefixes.

## 3)How does this guess relate to the remaining subproblems?

Knowing which we should to at the end makes it easier to determine what we will do before that point. If we guess we're going to get off at the last stop and earn h(n) happiness, then we may be more likely to party before the last stop if we'll earn more happiness than having gotten off, and we know we have k-1 more chances to get off. Likewise, if we know we're going to party at the last stop and earn some happiness from that, we're more likely to have partied at the previous stop unless H(n) is <=1. As stated before, this transforms the remaining subproblems into prefixes of our list of stops vs partying.

## 4)How would we establish a recursion to solve this?

We would first guess whether or not staying on or partying at the last stop grants the most happiness. With this, we then step up one to the second-to last stop, and see which technique would earn us the greater sum of happiness at that step, and so on and so forth. This implies that in our case, a memoization+recursion from the base case to the end would best suit our

needs, as with the guess as our basis, we can build our happiness tree from the root up to the very first stop.

## 5) How do we solve the original problem with this method?

We would continuously combine our subproblem solutions as we went up the tree, constantly trying to optimize our happiness, and so once we finish the last decision, that is, whether or not to party at the first stop, we'll have found our plan for the entire trip. Thus, the solution is itself a subproblem.

## The algorithm:

Assuming H(N) is a dictionary containing stop: happiness earned by touring, we would first guess whether or not to get off at h(n) or party at stop n. With that guess, we visit the second path above and take the optimal happiness decision, memoizing the happiness earned by previous decisions. If we get off, we decrement our K by 1, and if our K hits 0 before we are at the first stop, we know we'll have to have partied until this stop for the rest of the tree to work, so we can instantly calculate the remaining happiness we'll have earned at that time by entering a loop for the remaining n-k stops and just calculating the happiness sum.

In terms of calculating the happiness sum we would need to set up a recursion which takes in the current value of J, which is the number of times in a row we haven't gotten off the bus, and our current number of stops that we've gotten off at, call this big K. We would then at each stop do max(H(stop number), J+1) if our number K $<$ k. If K = k, we skip and just compute J+1 repeatedly for the remaining stops.

We repeat, storing the value of the happiness we've earned and compute the outcomes for the two guesses. We then store the highest happiness path for our guess. Once we're done computing the results for each of the two options on our guess, we compare them and take the one with higher happiness.

## Analyzing the algorithm:

There are a maximum of N subproblems. Each subproblem has two choices. However, to relate the subproblems, we can only make a certain decision k times, so there's a bounding factor of K there. We also fully take the entire path twice: One for guessing that we get off at the last stop, and one for assuming that we party at the last stop. With this in mind, our total runtime comes out to be:

O(N) subproblems for one guess * O(N) subproblems for the other guess

* O(1) for the two choices per subproblem, accessing H(n) and comparing J party points vs H(n) tour points

* O(K) for the relation of getting off vs partying limitations

= O($N^2 * k$) time to solve the entire problem.