We have continued to expand our Algorithmic Pizza Parlor stores, and now are going to try to gain new customers by advertising. We have a budget of $B$ dollars to spend on advertising, and $n$ possible ways to spend it (billboards, television spots, radio ads, etc). For each possible spending method $i$, we can allocate anywhere between 0 and $B$ dollars, but must do so in even increments of one dollar (we cannot allocate partial dollars). Our marketing department has come up with a series of functions $\{f_i\}$ to determine how effective (in terms of customers gained) each advertising method will be. If we spend $d$ dollars on advertising method $i$, we will gain $f_i(d)$ customers. These functions are all non-decreasing: if $d < d'$, then $f_i(d) \leq f_i(d')$. You may assume that $f_i(0) = 0$ for all $i$, if you would like.

Given these functions, use dynamic programming to design an algorithm that will determine the maximum number of new customers we can gain within our budget. Provide the recursive solution (including the base case), explain briefly why it is correct, and state what the running time of the iterative algorithm would be.

You do not need to give the iterative algorithm, nor do you need to produce a listing of where to spend the money – simply have your algorithm find the best possible number of customers gained within the budget.

Hint: The correct solution is not to figure out where to put the last marginal dollar.
Traveling Salesperson Problem

Consider the Traveling Salesperson problem. We are given a simple (not necessarily complete) directed graph. Our goal is to find the Hamiltonian Cycle of lowest total weight.

Example:

<table>
<thead>
<tr>
<th>Tour</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1, v_2, v_3, v_4, v_1$</td>
<td>22</td>
</tr>
<tr>
<td>$v_1, v_3, v_2, v_4, v_1$</td>
<td>26</td>
</tr>
<tr>
<td>$v_1, v_3, v_4, v_2, v_1$</td>
<td>21</td>
</tr>
</tbody>
</table>

The last tour listed is the optimal one for this input.

A dynamic programming algorithm

In general, we could enumerate every possible tour in time $O(n!)$, although this is probably a poor idea. Use dynamic programming to produce a better algorithm for this problem. You should not expect to get a polynomial running time for this problem.

Instead, let’s use dynamic programming, allowing for super-polynomial running time.

Hint 1: Without loss of generality, every tour “begins” and “ends” at $v_1$. Every such cycle can be thought of as going from $v_1$ to some $v_j$, going through zero or more vertices along the way, and then returning to $v_1$ going through the remaining vertices.

Hint 2: Any subset of vertices can be represented with a bit vector of $n$ bits. The $i$th bit corresponds to whether or not $v_i$ is included. However, you should focus on the concept, and instead treat this as if your algorithm can take a parameter of a subset of vertices. The detail of how to represent the set is important when you implement the algorithm.