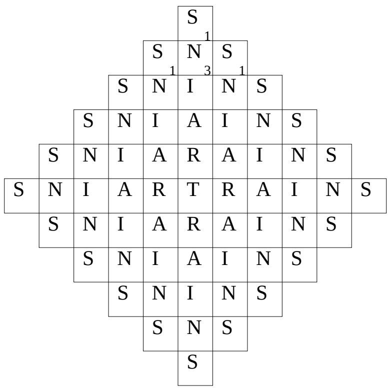
CSE 3101 Design and Analysis of Algorithms
Practice Test for Unit 5
Dynamic Programming

Jeff Edmonds

First learn the steps. Then try them on your own. If you get stuck only look at a little of the answer and then try to continue on your own.

1. In one version of the scrabble game, an input instance consists of a set of letters and a board and the goal is to find a word that returns the most points. A student described the following recursive backtracking algorithm for it. The bird provides the best word out of the list of letters. The friend provides the best place on the board to put the word. Why are these bad questions?

2. I saw this puzzle on a Toronto subway. The question is how many times the word "TRAINS" appears, winding snaking. We could count them but this might be exponential in the number of squares. Instead, for each box do a constant amount of work and write one integer. In the end, the answer should appear in the box with a "T". You should give a few sentences explaining the order you fill the boxes and how you do it and how much work it is.



3. (Answer is in the slides) A classic optimization problem is the *integer-knapsack problem.* For the problem in general, no polynomial algorithm is known. However, if the volume of the knapsack is a small integer, then dynamic programming provides a fast algorithm.

Integer-Knapsack Problem:

Instances: An instance consists of $\langle V, \langle v_1, p_1\rangle, \ldots, \langle v_n, p_n\rangle\rangle$. Here, V is the total volume of the knapsack. There are n objects in a store. The volume of the i^{th} object is v_i, and its price is p_i.

Solutions: A solution is a subset $S \subseteq [1..n]$ of the objects that fit into the knapsack, i.e., $\sum_{i\in S} v_i \leq V$.

Measure Of Success: The cost (or success) of a solution S is the total value of what is put in the knapsack, i.e., $\sum_{i\in S} p_i$.

Goal: Given a set of objects and the size of the knapsack, the goal is fill the knapsack with the greatest possible total price.

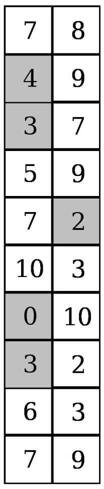
4. Stock Market Prices You are very lucky to have a time machine bring you the value each day of a set of stocks. The input instance to your problem consists of $I = \langle T, S, Price\rangle$, where T is an integer indicating your last day to be in the market, S is the set of $|S|$ stocks that you consider, and $Price$ is a table such that $Price(t, s)$ gives the price of buying one share of stock s on day t. Buying stocks costs an overhead of 3%. Hence, if you buy p dollars worth of stock s on day t, then you can sell them on day t' for $p \cdot (1 - 0.03) \cdot \frac{Price(t',s)}{Price(t,s)}$. You have one dollar on day 1, can buy the same stock many

times, and must sell all your stock on day T. You will need to determine how you should buy and sell to maximize your profits.

Because you know exactly what the stocks will do, there is no advantage in owning more than one stock at a time. To make the problem easier, assume that at each point time there is at least one stock not going down and hence at each point in time you alway own exactly one stock. A solution will be viewed a list of what you buy and when. More formally, a solution is a sequence of pairs $\langle t_i, s_i\rangle$, meaning that stock s_i is bought on day t_i and sold on day t_{i+1}. (Here $i \geq 1$, $t_1 = 1$ and $t_{last+1} = T$.) For example, the solution $\langle\langle 1,4\rangle, \langle 10,8\rangle, \langle 19,2\rangle\rangle$ means that on day 1 you put your one dollar into the 4^{th} stock, on day 10 you sell all of this stock and buy the 8^{th} stock, on day 19 you sell and buy the 2^{nd} stock, and finally on day T you sell this last stock. The value of this solution is $\Pi_i \left[(1-0.03)\cdot\frac{Price(t_{i+1},s_i)}{Price(t_i,s_i)}\right] = 1\cdot(1-0.03)\cdot\frac{Price(10,4)}{Price(1,4)}\cdot(1-0.03)\cdot\frac{Price(19,8)}{Price(10,8)}\cdot(1-0.03)\cdot\frac{Price(T,2)}{Price(19,2)}$. (Note that the symbol Π_i works the same as $\sum_i$ except for product.)

Design for a dynamic programming algorithm for this stock buying problem. Be sure to include ALL the steps given in the solution for the assignment. Hint: Ask the bird for the last "object" in the solution. Be sure to explain what this means.

5. Dynamic Programming the Narrow Art Gallery Problem:
(See ACM contest open.kattis.com/problems/narrowartgallery):
A long art gallery has $2N$ rooms. The gallery is laid out as N rows of 2 rooms side-by-side. Doors connect all adjacent rooms (north-south and east-west, but not diagonally). The curator has been told that she must close off r of the rooms because of staffing cuts. Visitors must be able to enter using at least one of the two rooms at one end of the gallery, proceed through the gallery, and exit from at least one of the two rooms at the other end. Therefore, the curator must not close off any two rooms that would block passage through the gallery. That is, the curator may not block off two rooms in the same row or two rooms in adjacent rows that touch diagonally. Furthermore, she has determined how much value each room has to the general public, and now she wants to close off the set r rooms that minimize the sum of the values of the rooms closed, without blocking passage through the gallery.
Figure 1: Shows an example of an art gallery of $N = 10$ rows and 2 columns of rooms. The number of rooms to close is $r = 5$. The number 1-10 in each room gives its value. The gray rooms indicate which should be closed in the optimal solution.



7	8
4	9
3	7
5	9
7	2
10	3
0	10
3	2
6	3
7	9

(a) Algorithmic Paradigms:
Tell me which one is **wrong** or say all are right.

A: An *iterative* algorithm takes one step at a time. During each it makes a little progress while maintaining a loop invariant.
A *recursive* algorithm asks his friends any instance that is smaller and meets the precondition. It is best not to micro managing these friends.

B: A *greedy* algorithm grabs the next best item and commits to a decision about it without concern for long term consequences.

C: A *recursive backtracking* algorithm tries various things. For each thing tried it recurses. Then it backtracks and tries something different.

D: A *dynamic programming* algorithm fills in a table with a cell for each of a set of subinstances. Each is solved in the same way as one stack frame of the recursive backtracking algorithm.

E: They are all right.

(b) Specification of the Narrow Art Gallery problem:
Tell me which one is **wrong** or say all are right.

A: An *instance* is specified by $I = \langle N, r, value(1..N, 1..2)\rangle$ where N is the number of rows. 2 is the number of columns. r is the number of rooms to close. For $n \in [1..N]$ and $side \in \{left, right\}$, $value(n, side)$ is the value of this room, i.e. the cost of closing it.

B: A *solution* is a subset of the $2N$ rooms of size r.

C: A solution is *valid* if a visitor traveling only east-west or north-south is able to enter the top of the gallery and leave though the bottom without traveling through a closed room. See the white path of rooms in the figure. For example, they closed the room of value 3 instead of the less valuable room next to it of value 2 because otherwise the public could not walk through.

D: The *cost* of such a solution is the sum of the values of the rooms closed. The goal is to minimize this cost.

E: They are all right.

(c) What is the nature of the bird?
Tell me which one is **wrong**.

A: There is no bird. There is no spoon (Quote from Matrix)

B: She helps us pose what we want to try.

C: We imagine her giving us a little answer about the solution.

D: She represents an algorithmic technique for learning part of the solution.

E: She helps us trust what we are trying so that we can go on.

(d) What is the nature of the friend?
Tell me which one is **wrong**.

A: There is no friend. There is no spoon (Quote from Matrix)

B: I know you. You are just like me.

C: You don't micro manage him because it's too confusing.

D: You can give him anything that meets the precondition and that is smaller.

E: Recursion

(e) Why does the bird tell you something about the end of the solution instead of the beginning?

A: It far too confusing the other way.

B: Esthetically the dynamic programming algorithm looks better going forward.

C: Esthetically the dynamic programming algorithm looks better going backwards.

D: It makes the algorithm faster.

E: The bird only knows about the beginning.

(f) Given the instance in the figure, I will ask bird:
"Should I close the bottom left room (i.e. labeled 7), the bottom right room (i.e. labeled 9), or neither of them."
Tell me which one is **wrong** or say all are right.

A: It is ok not to worry here about r or the higher rooms.

B: The answer she gives is $k \in \{left, right, none\}$. The number of bird answers we will have to try is 3.

C: I don't just ask just about the bottom right room, because if we just delete it, the friend's instance would be a funny shape.

D: I don't include the option of closing both of them, because that is not valid.

E: They are all right.

(g) One option is that we delete the bottom row of rooms (i.e. labeled 7&9) and we give the friend the first $N-1$ rows of rooms.
Tell me which one is **wrong** or say all are right.

A: We must also give the friend the new number r_{friend} of rooms to close which is our number r to close minus the number the bird closed.

B: We expect the friend to find the optimal solution for this.

C: The obvious solution for our instance of N rows is to close the rooms that our friend told us to close and to close the rooms that the bird told us to close.

D: The problem with this obvious solution is that if the bird tells us to close the bottom left room (i.e. labeled 7) and the friend tells us to close the room on the right in the second last row (i.e. labeled 3), then the solution will not be valid because the public would not be able to exit out of the bottom of the gallery.

E: They are all right.

Our buildings will always be rectangular with two columns. The public is allowed to enter either via the top left or the top right room. However, we are going to restrict the way that the public is allowed to leave the bottom of the gallery. We change the problem so that in addition to what has been specified above, the instance includes a parameter *door* $\in \{left, right, both\}$ which tells us whether the public can leave the gallery though the bottom left room but not the bottom right, the bottom right room but not the left, or both the bottom left and the bottom right. For our initial instance, *door* = *both*.

(h) Suppose we are given the instance in the figure and the bird tells us either to close the bottom left, bottom right or none of the rooms on the bottom row. Tell me about the instance that we give our friend.
Tell me which one is **right** or say all are wrong.

A: The set of rooms he gets is the same as the set of rooms we get.

B: We change some of the values of the rooms for our friend's instance.

C: Because the bird told us the solution for this last row of rooms, we delete this last row from our friend's instance. We give the friend our first $N_{friend} = N-1$ rows.

D: We remove just the room that the bird closes.

E: They are all wrong.

(i) Suppose we are given the instance in the figure and the bird tells us to close the bottom left room, i.e. $k = left$. Tell me about the instance that we give our friend.
Tell me which one is **right** or say all are wrong.

A: The number of rooms r that must be closes is fixed by the boss and does not change.

B: We do not need to tell the friend how many rooms to close because his bird will tell him that.

C: The bird tells us the total number of rooms to close.

D: The number of that my friend's instance needs closed is one less than the number we must close, i.e. $r_{friend} = r-1$.

E: They are all wrong.

(j) Suppose we are given the instance in the figure and the bird tells us to close the bottom left room, i.e. $k = left$. Tell me about the instance that we give our friend.
Tell me which one is **right** or say all are wrong.

A: There is no need for doors.

B: The friend's instance will have no door on the bottom left, i.e. $door_{friend} = right$. This models the fact there is a pseudo room below his bottom left room that is closed and hence the public cannot enter his bottom left room in this way.

C: The friend's instance will have no door on the bottom right, i.e. $door_{friend} = left$. This models the fact there is a pseudo room below his bottom left room that is closed and hence the public cannot enter his bottom left room in this way.

D: In this case, the friend should be given a door both on the left and on the right. This gives the public the correct level of access.

E: They are all wrong.

(k) Suppose we are given the instance in the figure except for the fact that there is no door on the bottom left, i.e. *door* = *right*. Tell me about the instance that we give our friend.
Tell me which one is **wrong** or say all are right.

A: The whole business with the doors is done to avoid the following bug. If the bird tells us to close the bottom left room (i.e. labeled 7) and the friend tells us to close the room on the right in the second last row (i.e. labeled 3), then the solution will not be valid because the public would not be able to exit out of the bottom of the gallery.

B: Closing the only room on the bottom with a door will prevent the public from leaving. Hence, we will not allow the bird to close the bottom right room, i.e. if she says that $k = right$, then we politely tell her that she is wrong.

C: We politely tell the bird she is wrong by setting $optCost_{\langle I,k\rangle} = \infty$. Being a minimization problem, this option will never be selected.

D: Sometimes when writing a recursive program, we need to change the preconditions so that the friend gives us the answer that meets our needs. When converting this into a dynamic programming algorithm, this makes a larger set of subinstances.

E: They are all right.

(l) Dynamic Programming:
Tell me which one is **wrong** or say all are right.

A: This recursive back tracking algorithm effectively tries every possible solution, i.e. brute force. The time is exponential.

B: Recursive back tracking is a common algorithmic technique that is used in artificial intelligence (certainly before machine learning).

C: A dynamic programming algorithm saves time by in a way similar to that the greedy algorithm works, i.e. if the existence of an optimal solution consistent with decision A implies the existence of an optimal solution consistent with decision B, then decision A does not need to be tried.

D: A dynamic programming algorithm saves time by not solving the same subinstance more than once.

E: They are all right.

(m) We start the dynamic programming algorithm by setting up a table indexed by all of the subinstances that some friend friend friend will have to solve.
Tell me which one is **wrong** or say all are right.

A: This table will have a dimension that is indexed by the number of rows $N' \in [0..N]$ that the friend will consider.

B: This table will have a dimension that is indexed by the number of rooms $r' \in [0..r]$ that the friend will have to close.

C: This table will have a dimension indicating the values of the each room, i.e. the numbers 0 to 10 in the figure.

D: This table will have a dimension that is indexed by the parameter $door \in \{left, right, both\}$.

E: They are all right.

(n) What needs to be true about the set S of subinstances being solved?
Tell me which one is **wrong** or say all are right.

A: Include our original instance. (Invite the bride and groom.)

B: Closed under the friend operation. For every subinstance $I' \in S$, all of I''s friends must also be in S. (If you invite your aunt, then you must invite her friends.)

C: Don't have too many things in S that are not asked by some friend's friend.

D: Sometimes it is too hard to tell if some I' is actually asked by some friend's friend. Then we just put it in S for good luck.

E: They are all right.

(o) The number of subinstance that we must solve is:

A: $N \times r \times 10 \times 2N$

B: $2N$

C: $3rN$

D: $\mathcal{O}(r+N)$

E: exponential (say in N, r, or the number of bits to write down the values of the rooms.)

(p) Each cell of the table (tables)
Tell me which one is **wrong** or say all are right.

A: Is indexed by a subinstance to be solved.

B: Stores the optimal solution for that subinstance.

C: Stores the cost of optimal solution for that subinstance.

D: Stores the bird's advice for that subinstance.

E: They are all right.

(q) What order should the subinstances in the table be completed.
Tell me which one is **wrong** or say all are right.

A: Smallest to largest.

B: In an order that nobody waits, i.e. when a subinstance is solved, all of his friends have already been solved. When its your aunt's job, her friends are already done and gotten drunk.

C: If the table is two dimentional, you have to fill it in diagonally.

D: Basecases first. Our instance last.

E: They are all right.

(r) What are the base cases?
Tell me which one is **wrong** or say all are right.

A: The smallest subinstances in your table.

B: Subinstances that don't have any friends.

C: Subinstances for which the generic code does not work.

D: Subinstances that you can solve easily on your own.

E: They are all right.

(s) The base cases are handled as follows.
Tell me which one is **right** or say all are wrong.

A: if($r \leq 0$) then return(no rooms need to be closed)

B: if($N \leq 0$) then return(no rooms to close)

C: if($N < r$) then return(more rooms to close than can be closed)

D: % The cost of the solution is the sum of that values of the rooms closed. Hence, if there are $r = 0$ rooms to close, then the cost is zero.
for all entries of the table for which $r' = 0$, $optCost[...r;..] = 0$

E: They are all wrong.

(t) Dynamic Programming
Tell me which one is **wrong** or say all are right.

A: To solve our instance, we recursively ask friends to solve smaller subinstances.

B: We loop over all subinstances from smallest to largest.

C: The word "Memoization" comes from the word "Memo," i.e. to write nodes about what has happened already.

D: To solve our instance, we look in the table to see what our friends have stored about smaller subinstances.

E: They are all right.

(u) The first thing a dynamic program does is:
Tell me which one is **right** or say all are wrong.

A: Ask the bird about an optimal solution of the inputted instance.

B: Set up the table.

C: Check if the input is a base case.

D: Check if the input has the correct format.

E: They are all wrong.

(v) We combine the cost of our friend's solution and the cost of our bird's solution k to get:
Tell me which one is **right** or say all are wrong.

A: The cost of an optimal solution $optCost_{I'}$ for the instance I'.

B: The cost of an optimal solution $optCost_{\langle I',k\rangle}$ for the instance I' from amongst those that are consistent with this bird's answer k.

C: We need to compute the optimal solution.

D: The whole idea of combining is misrepresents the technique.

E: They are all wrong.

(w) Consider a dynamic programming routine named *RoomClosures*. A key line of its is:
Tell me which one is **right** or say all are wrong.

A: $optCost_{..} = optCost[I_{friend}] + GetBirdsCost(...)$

B: $optCost_{..} = RoomClosures(I_{friend}) + optCost_{bird}$
where if($k = left$) then $opCost_{bird} = value(N', left)$.

C: $optCost_{..} = optCost[I_{friend}] + optCost_{bird}$
where if($k = none$) then $opCost_{bird} = 0$.

D: $optCost_{..} = RoomClosures(I_{friend}) + GetBirdsCost(...)$

E: They are all wrong.

(x) Dynamic Programming
Tell me which one is **wrong** or say all are right.

A: We try all bird answers.

B: Having the best, $optSol_{\langle I',k\rangle}$, for each bird's answer k, we keep the best of these best.

C: The following is reasonable pseudo code.
k_{min} = "a k that minimizes $optCost_{\langle I',k\rangle}$"

D: Options A, B, and C need to be done in dynamic programming but not in recursive backtracking.

E: They are all right.

(y) Finishing up the dynamic programming algorithm.
Tell me which one is **wrong** or say all are right.

A: The following is where the memo is being taken.
$optCost[I'] = optCost_{\langle I',k_{min}\rangle}$.

B: The following is key line of how Jeff does dynamic programming, but you likely won't hear anyone else talk about it. $birdAdvice[I'] = k_{min}$.

C: The algorithm $optSol = AlgWithAdvice\,(I, birdAdvice)$ reruns the algorithm but this time it is fast because now there really is a bird.

D: The algorithm $AlgWithAdvice$ and the greedy algorithm both follow one path down the decision tree.

E: They are all right.

(z) Running time of dynamic programming algorithms.
Tell me which one is **wrong** or say all are right.

A: The running time of a dynamic programming algorithm is the number of subinstances times the number of bird answers.

B: If an optimal solution is found in the inner loop then the time is multiplied by the number of bits to write down the solution.

C: Finding an optimal solution when the bird's advice is known is fast. The time is in linear in the number of bits to write down the solution.

D: If the input consists of n objects and each subset of these objects is a subinstance, then the number of subinstances is exponential. This occurring with the obvious dynamic programming algorithm is a very common reason for a problem to be NP-complete, i.e. no polynomial algorithm for it is known.

E: They are all right.

6. More questions past z.

   (a) Which is **true** about the running time of our RoomClosures dynamic program?

   A: The running time of this algorithm is proportional to the number of rows of rooms N. The size of the input includes the $\log N$ bits to represent N. This means that this dynamic programming algorithm runs in exponential time just like the Knapsack problem.

   B: The running time of this algorithm is proportional to the number of rooms to close r. The size of the input includes the $\log r$ bits to represent r. This means that this dynamic programming algorithm runs in exponential time just like the Knapsack problem.

   C: A and B.

   D: The running time of this algorithm is quadratic in the size of the input.

   E: They are all wrong.

   (b) Suppose instead of the number of columns being restricted to 2, the gallery could have width w. Consider extending the same basic algorithm done above.
   Tell me which one is **wrong** or say all are right.

   A: The running time would increase by a factor of w.

   B: We would still ask the bird which rooms to close on the bottom row.

   C: The number of bird answers would now be $\Theta(2^w)$.

   D: This problem is NP-complete, i.e. no polynomial algorithm for it is known.

   E: They are all right.

   (c) Write out the code for the *RoomClosure* algorithm developed here.