TCSS 343 - Week 4

Jake McKenzie

September 24, 2018

Dynamic Programming

"An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."
. . .
Richard Bellman's **Principle of Optimality**

"What we choose means more than what was handed to us by chance."
. . .
Ada Palmer

" If 'dynamic programming' didn't have such a cool name, it would be known as 'populating a table'".
. . .
Mark Dominus

1. Today we're going to explore dynamic programming. Below are three implementations of the fibonacci algorithm that I wrote in python. I want you to draw the **"tree"** for each then reflect on how the "bottom up" apprach is different from the other two? (Hint: They are all trees but also different types of trees. This is a key insight in my opinion in idea in understanding dynamic programming)

```
1    # recusive fibonacci
2    def F(n):
3        if n == 0: return 0
4        elif n == 1: return 1
5        else: return F(n-1) + F(n-2)
```

```
7    # "top down" memoized recursive fibonacci
8    memo = {}
9    def Fib(n):
10       if n in memo: return memo[n]
11       if n == 0: f = 0
12       elif n == 1: f = 1
13       else: f = Fib(n-1) + Fib(n-2)
14       memo[n] = f
15       return f
```

```
17   # "bottom up" iterative fibonacci
18   def fib(n):
19       fn = [0,1]
20       for i in range(n >> 1):
21           fn[0] += fn[1]
22           fn[1] += fn[0]
23       return fn[n % 2]
```

2. I found these really cool recursive Fibonacci formulas: $F_{2n+1} = F_{n+1}^2 + F_n^2$ and $F_{2n} = 2F_{n+1}F_n - F_n^2$ now can you use them to find F_{11} and F_{10}(worth noting that $F_2 = 1$, $F_1 = 1$, and $F_0 = 0$)?

3. Can you now write an iterative method that computes F_{2n} and F_{2n+1} using dynamic programming?

4. How is this different from the basic Fibonacci formula? What is the time complexity of this function? Space complexity?

```
1    # Rules: X,Y > 0
2    def game_of_Stanley_Gill(X,Y):
3        x = X
4        u = X
5        y = Y
6        v = Y
7        while x != y:
8            if x > y:
9                x = x - y
10               u = v + u
11           elif y > x:
12               y = y - x
13               v = u + v
14   print((x + y) >> 1) #GCD(X,Y)
15   print((u + v) >> 1) #LCM(X,Y)
```

5. This is a game shown to Edsgar Dijkstra(pictured right) when he was still an undergraduate, attributed to Stanley Gill(pictured left), an early computer scientist. Now it is true that $2XY = xv + yu$, which can be seen when the variables are initialized. But is it always true given that $X, Y > 0$? Show why this is true. (Hint: You can use the comments to help you along. Remember that the $>> 1$ operation is the same as $/2$.)

6. Hopefully in problem 1 I was able to impart on you the reality that most dynamic programming problems can be thought of as graph problems, specifically directed acyclic graphs, DAG for short. Even if you don't solve them with DAGs I think it's a useful headspace for a large class of problems. We can typically solve DP problems without constructing a graph. If we aren't going to use a graph there are other useful tools that you may not have run across. Let's try to understand what a bitmask is. I've been peppering them into the packets all quarter but let's finally dive deep into them. Mask in bitmask means hiding something. Bitmask is nothing but a binary number that represents something.

Please connect the corresponding "bitmask" with their set operations or arithmetic operations. I encourage you to play around with the operation and really try understand what they're doing. All operations are on two intergers X and Y. I know this may seem disjointed from what you've covered in class but I have used all of these in solving DP problems. Often times we aren't just worried about our time complexity, but we're also worried about space complexity. Using integers instead of arrays to represent data is sometimes vital in solving hard problems. For example: we may represent the set $\{5, 4, 3, 2, 1\}$ as 11111 in binary which is 31 in decimal and the set $\{4, 2, 1\}$ as 01011 in binary which is 11 in decimal.

a) $X \& 1$ I)Union of two sets

b) $X \wedge Y$ II)Arithmetic negation

c) $(X >> 31)\&1$ III)Test for set membership

d) $X|Y$ IV)2 to the power of X(also Singleton Set)

e) $X \& Y$ V)Barrel shift left

f) $X \& = (X - 1)$ VI)Intersection of two sets

g) $(X << Y)|(X >> (32 - Y))$ VII)Signed bit

h) $!X + 1$ IX)Clears lowest "ON" bit in X

i) $1 << X$ X)Symmetric Difference of two sets

j) $(X \& (1 << Y))\,! = 0$ XI)Even/Odd check

7. The N-Queens is one of the classic brain teasers. The problem can be stated simply: On a NxN square checkerboard, place N queens in a way so that no queen threatens any of the other ones, ie. shares column, row or diagonal.**Below I will ask you a series of questions on how you might make a plan of attack to solving this problem.** Below are all possible "unique" solutions for an $N = 8$. There are actually 92 correct solutions in all. It's worth noting why we need dynamic programming for such a problem. There are $\frac{64!}{(64-8)!8!}$ ways of arranging 8 queens on a chessboard.
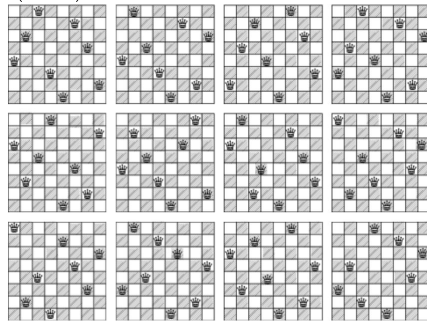
7.0 How many (non-unique)solutions are there for $N = 1, 2, 3, 4, 5, 6$

7.1 What are the subproblems to the global problem of computing all possible solutions to this problem?

8. Analysis of N-Queen

a. Given a method isSafe that checks to see if the board is safe. This method will take $O(n)$ time as it iterates through our board every time.

b. To places a queen, on each iteration we must go through the board everytime which takes $O(n)$ time.

c. In each iteration of this loop, there is an isSafe invocation which is $O(n)$ and a recursive call.

Given this information can you come up with a recurrence relation for the N-Queen problem?

9. Time for some graph review. Shortest path algorithms on graphs all fixated on the concept of relaxation. To put simply, relaxing an edge from one vertex to another means temporarily violating the shortest path criteria locally to check to see if you can find a new global shortest path. The relaxation algorithm is as follows(some python code for it):

```python
1    def relax(graph):
2        for _ in range(1, len(graph.vertices())):
3            for u,v,data in graph.edges(data=True):
4                if distance[v] > distance[u] + data["weight"]:
5                    distance[v] = distance[u] + data["weight"]
```

Why is the relaxation algorithm safe? Can you use the triangle inequality to prove it? Use this lemma: The relaxation algorithm maintains the invariant that V.d $\geqslant \delta$(S,V)$\forall$v$\in$V
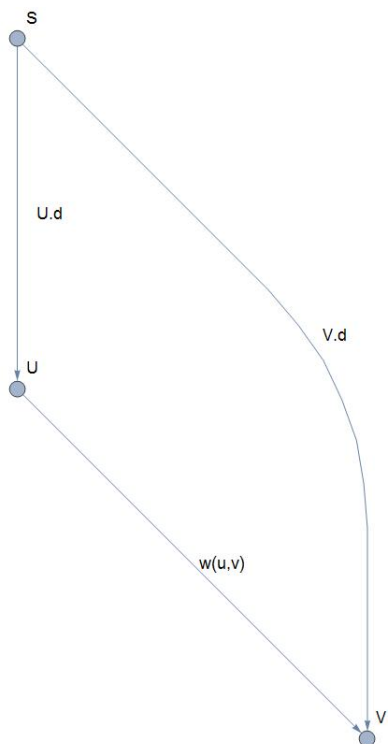
Triangle inequality: δ(S,V) $\leqslant \delta$(S,U) $+\delta$(U,V)

A. Say you have access to a function **dict** that returns true if its input is a valid English word, and false otherwise. We are given as input a sentence from which the punctuation has been stripped (for example: "dynamicprogrammingisfabulous"). Assuming calls to dict take unit time, give an $O(n^2)$ time algorithm to figure out whether an input string of length n can be split into a sequence of valid words or not. HINT: Try to remove valid words from the end of the input string.

r	sp_r	sc_r
1	5	1
2	9	6
3	26	18
4	31	22
5	36	28

B. So imagine you've graduated from University of Washington in Tacoma and you got a nice job at a startup in San Francisco. It's the weekend and you want to drive down to Monterrey to enjoy some wine with friends. Now depending on what scenery you're driving through you find yourself speeding through that locale for each time unit you're in it. It's the weekend you don't really care how much time it takes you to get to Monterrey, but you defintely don't want a ticket. What's the maximum amount of scenic value you can get out of your weekend drive to Monterrey?

r\speedlimit	0	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75
{1}	0															
{1,2}	0															
{1,2,3}	0															
{1,2,3,4}	0															
{1,2,3,4,5}	0															

C. Let $s[1..m]$ and $t[1..n]$ be the two strings to be matched. Let $M(i, j)$ be the number of mismatches in the best alignment of $s[1..i]$ and $t[1..j]$. Which of the following is a correct recursive formulation of $M(i, j)$ for $i, j > 0$? Note: By convention $s[1..0]$ and $t[1..0]$ are taken to be the empty string, so $M(i, 0) = i$ for $i \in \{1, 2, ..., m\}$ and $M(0, j) = j$ for $j \in \{1, 2, ..., n\}$.

a) $M(i, j) = 2 + M(i - 1, j - 1)$, if $s[i] \neq t[j]$
   $min(M(i - 1, j), M(i, j - 1))$, otherwise

b) $M(i, j) = M(i - 1, j - 1)$, if $s[i] = t[j]$
   $min(M(i - 1, j), M(i, j - 1))$, otherwise

c) $M(i, j) = 1 + M(i - 1, j - 1)$, if $s[i] = t[j]$
   $1 + min(M(i - 1, j), M(i, j - 1))$, otherwise

d) $M(i, j) = M(i - 1, j - 1)$, if $s[i] = t[j]$
   $1 + min(M(i - 1, j), M(i, j - 1))$, otherwise

D. You have a bag that you want to fill with toys. There are N toys, and the ith toy has weight $w[i]$. The bag can hold a total weight of at most W. Maximize the number of toys you can take with you in the bag.

Formally, choose a set of as many toys as possible, such that the sum of their weights is $\leqslant W$.

Constraints: $N \leqslant 1000$; $w[i] \leqslant 10,000 \; \forall \; i$.

Choose the best option out of the following

a) The simplest solution to this problem is using Dynamic Programming.

b) This problem cannot be solved using Dynamic Programming.

c) This problem can be solved with Dynamic Programming, but has a simpler solution without Dynamic Programming.

E. You may not have seen the operation $\sim$. It means "asymptotic to". To say $f(n) \sim g(n)$ that means that $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} \to 1$(not just a constant, we can have that $f(n) \in \Theta(g(n))$ but also $f(n) \nsim g(n)$ ex: $f(n) = n$ and $g(n) = 2n$). Now for SSSP (Single Source Shortest Path) problem if we solve it via brute force we find the runtime is $O((n-2)!)$(there are $n-1$ vertices and $(V-1)!$ possible permutations of paths through the graph).

I want you to show why brute force on graphs to solve the SSSP problem is worse than the usual way we solve similar problems by showing $(n-2)! \sim 2^{n \log n}$. Reminder that there are 2^n possible partitions to nongraph problems like the subset sum problem, which can be thought of as a SSSP problem.

F. Robert Sapolsky is a famous neuroendocrinologist (big word that means he studies hormones & stress in humans and primates) and he's one of my favourite thinkers. He likes to use this sequence of numbers, $4, 14, 23, 34, \ldots$ in his neuroscience class to illustrate an important point in categorical thinking. For purposes of our algorithms course I will call these numbers "Sapolsky Numbers". Please compute the 10th Sapolsky Number. Can you come up with a recurrence formula for the Sapolsky numbers? (If you get stuck ask me questions)

Why was this hard? If you think you came up with a solution please tell me.