CS3233
Competitive Programming

Dr. Steven Halim

Week 04 – Problem Solving Paradigms
(Dynamic Programming 1)

Outline

- Mini Contest #3 + Break + Discussion + Admins
- Dynamic Programming – Introduction
  - Treat this as **revision** for ex CS2010/CS2020 students
  - **Listen carefully** for the other group of students!
  - I open consultation slots (Mon/Fri) for NUS students who need help with this topic, especially those who did not go through CS2010/CS2020 before
- Dynamic Programming
  - Some Classical Examples
- PS: I will use the term **DP** in this lecture
  - OOT: DP is NOT Down Payment!

Wedding Shopping

EXAMPLE 1

Motivation

- How to solve UVa 11450 (Wedding Shopping)?
  - Given 1 ≤ **C** ≤ 20 classes of garments
    - e.g. shirt, belt, shoe
  - Given 1 ≤ **K** ≤ 20 different models for each class of garment
    - e.g. three shirts, two belts, four shoes, …, each with its own price
  - Task: Buy **just one** model of **each class** of garment
  - Our budget 1 ≤ **M** ≤ 200 is limited
    - We cannot spend more money than it
    - But we want to spend the maximum possible
  - What is our maximum possible spending?
    - Output "no solution" if this is impossible

- Budget M = 100
  - Answer: 75

Model / Garment	0	1	2	3
0	8	6	4	
1	5	10		
2	1	3	3	7
C = 3	50	14	23	8

- Budget M = 20
  - Answer: 19
    - Alternative answers are possible

Model / Garment	0	1	2	3
0	4	6	8	
1	5	10		
C = 2	1	3	5	5

- Budget M = 5
  - Answer: no solution

Model / Garment	0	1	2	3
0	6	4	8	
1	10	6		
C = 2	7	3	1	7

Greedy Solution?

- What if we buy the most expensive model for each garment which still fits our budget?
- Counter example:
  - M = 12
  - Greedy will produce:
    - no solution
  - Wrong answer!
    - The correct answer is 12
    - (see the green dotted highlights)
      - Q: Can you spot one more potential optimal solution?

Model / Garment	0	1	2	3
0	6	4	8	
1 ?	5	10		
C = 2	1	5	3	5

Divide and Conquer?

- Any idea?

?

Complete Search? (1)

- What is the potential **state** of the problem?
  - g (which garment?)
  - id (which model?)
  - money (money left?)
- Answer:
  - (money, g) or (g, money)
- Recurrence (recursive backtracking function):

```
shop(money, g)
  if (money < 0) return -INF
  if (g == C) return M – money
  return max(shop(money – price[g][model], g + 1), ∀model ∈ [1..K]
```

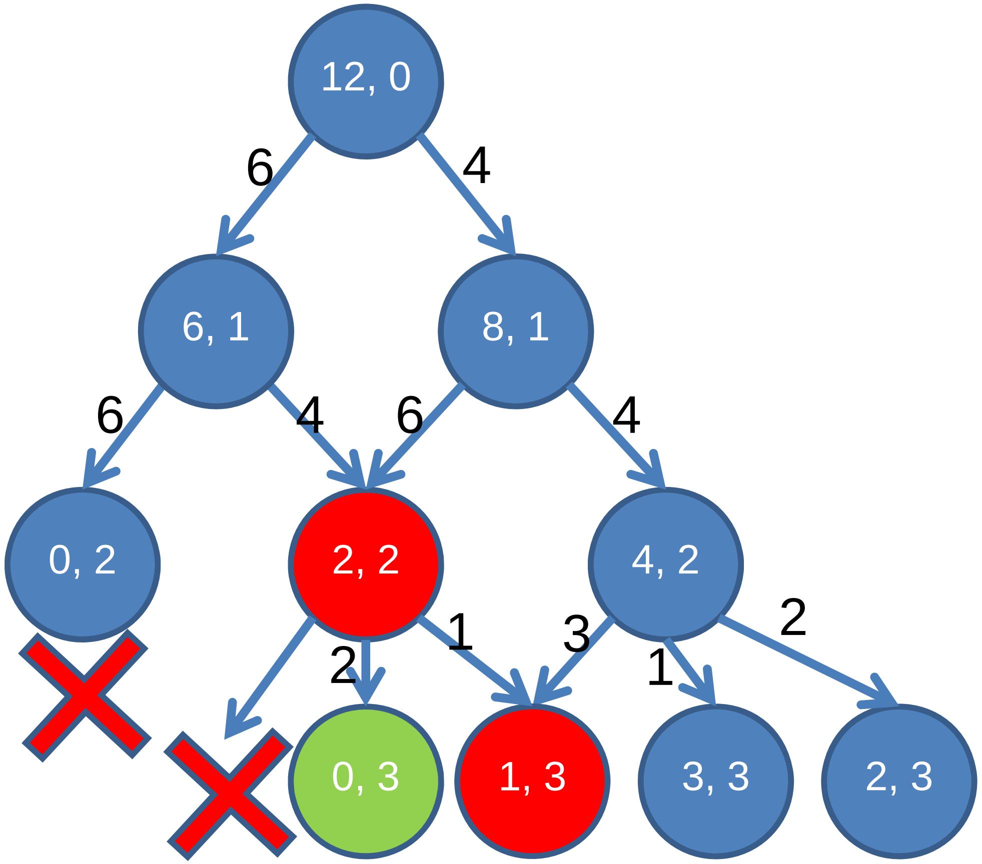
Complete Search? (2)

- But, how to solve this?
  - M = 200 (maximum)
- Time Complexity: 20^{20}
  - Too many for 3s time limit ☹

Model / Garment	0	1	…	19
0	32	12		55
1	2	53		4
2	1	3		7
3	50	14		8
4	3	1		5
5	4	3		1
6	5	2		5
…				
19	22	11		99

Overlapping Sub Problem Issue

- In the simple 20^{20} Complete Search solution, we observe **many** **overlapping sub problems**!
  - Many ways to reach state (money, g), e.g. see below, M = 12



Model / Garment	0	1	2
0	6	4	
1	4	6	
C = 2	1	2	3

DP to the Rescue (1)

- DP = Dynamic Programming
  - Programming here is not writing computer code, but a **“tabular method”**!
    - a.k.a. **table** method
  - A programming paradigm that you must know!
    - And hopefully, master…

DP to the Rescue (2)

- Use DP when the problem exhibits:
  - Optimal sub structure
    - Optimal solution to the original problem contains optimal solution to sub problems
      - This is **similar** as the requirement of **Greedy algorithm**
      - If you can formulate complete search recurrences, you have this
  - Overlapping sub problems
    - Number of **distinct sub problems** are actually “small”
    - But they are **repeatedly computed**
      - This is **different** from **Divide and Conquer**

DP Solution – Implementation (1)

- There are two ways to implement DP:
  - Top-Down
  - Bottom-Up
- Top-Down (Demo):
  - Recursion as per normal + **memoization table**
    - It is just a simple change from backtracking (complete search) solution!

Turn Recursion into Memoization

initialize memo table in main function (use ‘memset’)

return_value **recursion(params/state)** {

  if this state is already calculated,

    simply return the result from the memo table

  calculate the result using **recursion(other_params/states)**

  save the result of this state in the memo table

  return the result

}

Dynamic Programming (Top-Down)

- For our example:

```
shop(money, g)
  if (money < 0) return -INF
  if (g == C) return M – money
  if (memo[money][g] != -1) return memo[money][g];
  return memo[money][g] = max(shop(money – price[g][model], g + 1),
    ∀model ∈ [1..K]
```

- As simple as that ☺

If Optimal Solution(s) are Needed

- Clever solution for Top-Down DP
  - (See solution for Bottom-Up DP in Example 2)
- For our example:

```
print_shop(money, g)
  if (money < 0 || g == C) return
  for each model ∈ [1..K]
    if shop(money – price[g][model], g + 1) == memo[money][g]
      print "take model = " + model + " for garment g = " + g
      print_shop(money – price[g][model], g + 1)
      break
```

- As simple as that ☺

DP Solution – Implementation (2)

- Another way: Bottom-Up:
  - Prepare a table that has size equals to the number of distinct states of the problem
  - Start to fill in the table with base case values
  - Get **<u>the topological order</u>** in which the table is filled
    - Some topological orders are natural and can be written with just (nested) loops!
  - Different way of thinking compared to Top-Down DP
- Notice that both DP variants use “table”!

Dynamic Programming (Bottom-Up)

- For our example:
  - Start with with table **can_reach** of size 20 (g) * 201 (money)
    - The state (money, g) is reversed to (g, money) so that we can process bottom-up DP loops in row major fashion
    - Initialize all entries to 0 (false)
    - Fill in the first row with money left (column) reachable after buying models from the first garment (g = 0
  - Use the information of current row g to update the values at the next row g + 1

- Budget M = 20
  - Answer: 19
    - Alternative answers are possible

Garment \ Model	0	1	2	3
0	4	6	8	
1	5	10		
C = 2	1	3	5	5

money =>

g =>

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0
1	0	0	1	0	1	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0
1	0	0	1	0	1	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0
2	0	1	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0

Top-Down or Bottom-Up?

- Top-Down
  - Pro:
    - Natural transformation from normal recursion
    - Only compute sub problems when necessary
  - Cons:
    - Slower if there are many sub problems due to recursive call overhead
    - Use exactly O(states) table size (MLE?)
- Bottom Up
  - Pro:
    - Faster if many sub problems are visited: no recursive calls!
    - Can save memory space*
  - Cons:
    - Maybe not intuitive for those inclined to recursions?
    - If there are X states, bottom up visits/fills the value of all these X states

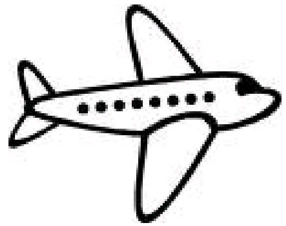
Flight Planner

(study this on your own)

EXAMPLE 2

Click me to jump to the next section

Motivation

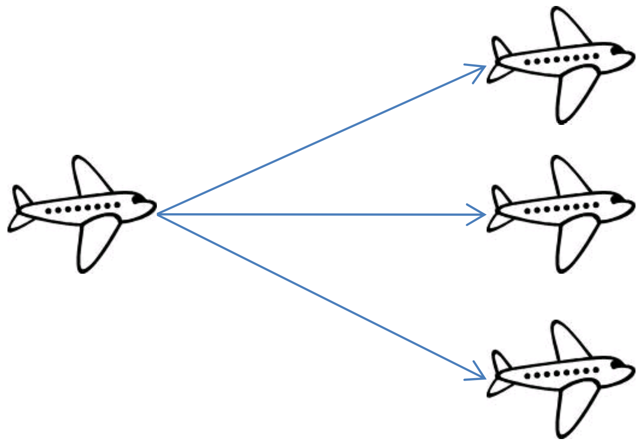


- How to solve this: 10337 (Flight Planner)?
  - Unit: 1 mile altitude and 1 (x100) miles distance
  - Given wind speed map
  - Fuel cost: {**climb** (+60), **hold** (+30), **sink** (+20)} - wind speed wsp[alt][dis]
  - Compute min fuel cost from (0, 0) to (0, X = 4)!

```
1   1   1   1    | 9
1   1   1   1    | 8
1   1   1   1    | 7
1   1   1   1    | 6
1   1   1   1    | 5
1   1   1   1    | 4
1   1   1   1    | 3
1   1   1   1    | 2
1   9   9   1    | 1
1  -9  -9   1    | 0
========================
0   1   2   3   4 (x100)
```

Complete Search? (1)

- First guess:
  - Do complete search/brute force/**backtracking**
  - Find *all possible* flight paths and pick the one that yield the minimum fuel cost



Complete Search? (2)

- Recurrence of the Complete Search
  - **fuel**(alt, dis) =

```
min3(60 - wsp[alt][dis] + fuel(alt + 1, dis + 1),
     30 - wsp[alt][dis] + fuel(alt    , dis + 1),
     20 - wsp[alt][dis] + fuel(alt - 1, dis + 1))
```

  - Stop when we reach final state (base case):
    - alt = 0 and dis = X, i.e. fuel(0, X) = 0
  - Prune infeasible states (also base cases):
    - alt < 0 or alt > 9 or dis > X!, i.e. return INF*
  - Answer of the problem is **fuel(0, 0)**

Complete Search Solutions (1)

- Solution 1

```
1    1    1    1     | 9
1    1    1    1     | 8
1    1    1    1     | 7
1    1    1    1     | 6
1    1    1    1     | 5
1    1    1    1     | 4
1    1    1    1     | 3
1    1    1    1     | 2
1    9    9    1     | 1
1   -9   -9    1     | 0
========================
0    1    2    3    4 (x100)
```

29+ 39+ 39+ 29=**136**

- Solution 2

```
1    1    1    1     | 9
1    1    1    1     | 8
1    1    1    1     | 7
1    1    1    1     | 6
1    1    1    1     | 5
1    1    1    1     | 4
1    1    1    1     | 3
1    1    1    1     | 2
1    9    9    1     | 1
1   -9   -9    1     | 0
========================
0    1    2    3    4 (x100)
```

29+ 39+ 69+ 19=**156**

Complete Search Solutions (2)

- Solution 3

```
1    1    1    1     | 9
1    1    1    1     | 8
1    1    1    1     | 7
1    1    1    1     | 6
1    1    1    1     | 5
1    1    1    1     | 4
1    1    1    1     | 3
1    1    1    1     | 2
1    9    9    1     | 1
1   -9   -9    1     | 0
========================
0    1    2    3    4 (x100)
```

29+ 69+ 11+ 29=**138**

- Solution 4

```
1    1    1    1     | 9
1    1    1    1     | 8
1    1    1    1     | 7
1    1    1    1     | 6
1    1    1    1     | 5
1    1    1    1     | 4
1    1    1    1     | 3
1    1    1    1     | 2
1    9    9    1     | 1
1   -9   -9    1     | 0
========================
0    1    2    3    4 (x100)
```

59+ 11+ 39+ 29=**138**

Complete Search Solutions (3)

- Solution 5

```
1   1   1   1     | 9
1   1   1   1     | 8
1   1   1   1     | 7
1   1   1   1     | 6
1   1   1   1     | 5
1   1   1   1     | 4
1   1   1   1     | 3
1   1   1   1     | 2
1   9   9   1     | 1
1  -9  -9   1     | 0
========================
0   1   2   3   4 (x100)
```

29+ 69+ 21+ 19=**138**

- Solution 6

```
1   1   1   1     | 9
1   1   1   1     | 8
1   1   1   1     | 7
1   1   1   1     | 6
1   1   1   1     | 5
1   1   1   1     | 4
1   1   1   1     | 3
1   1   1   1     | 2
1   9   9   1     | 1
1  -9  -9   1     | 0
========================
0   1   2   3   4 (x100)
```

59+ 21+ 11+ 29=**120(OPT)**

Complete Search Solutions (4)

- Solution 7

```
1   1   1   1    | 9
1   1   1   1    | 8
1   1   1   1    | 7
1   1   1   1    | 6
1   1   1   1    | 5
1   1   1   1    | 4
1   1   1   1    | 3
1   1   1   1    | 2
1   9   9   1    | 1
1  -9  -9   1    | 0
========================
0   1   2   3   4 (x100)
```

59+ 21+ 21+ 19=**120(OPT)**

- Solution 8

```
1   1   1   1    | 9
1   1   1   1    | 8
1   1   1   1    | 7
1   1   1   1    | 6
1   1   1   1    | 5
1   1   1   1    | 4
1   1   1   1    | 3
1   1   1   1    | 2
1   9   9   1    | 1
1  -9  -9   1    | 0
========================
0   1   2   3   4 (x100)
```

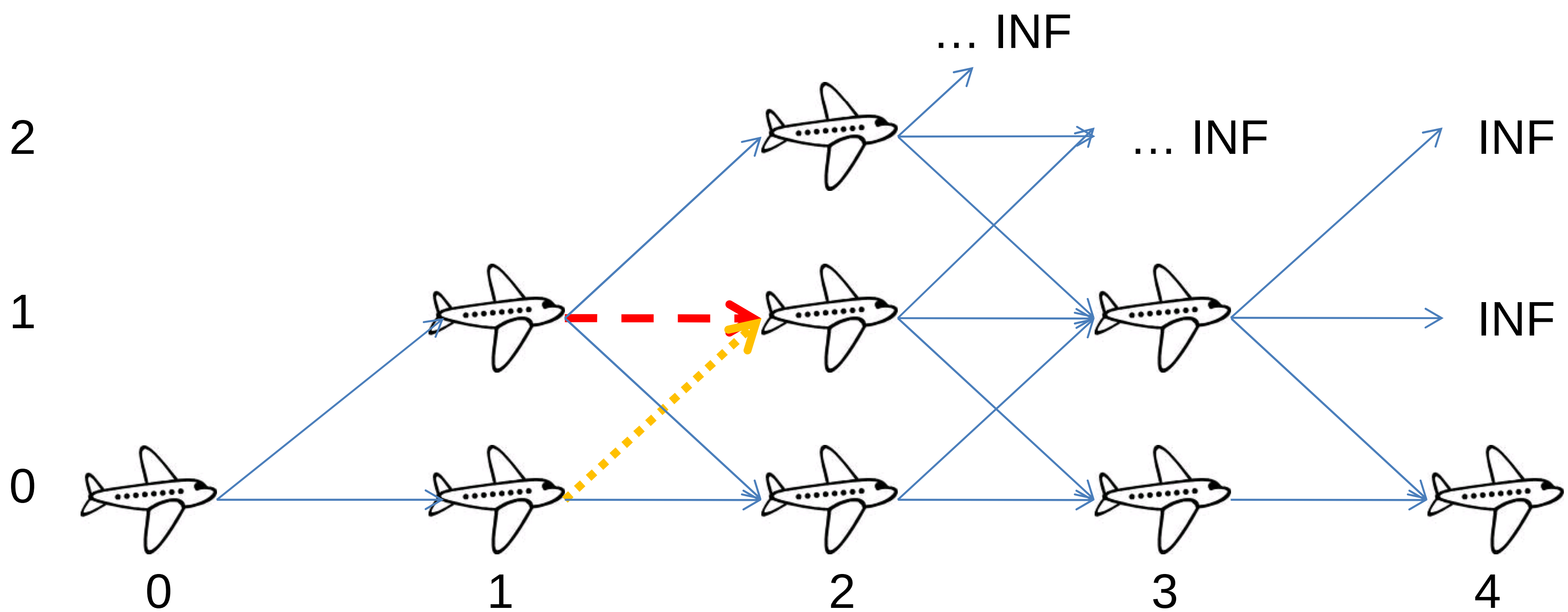
59+ 51+ 19+ 19=**148**

Complete Search? (3)

- How large is the search space?
  - Max distance is 100,000 miles
    Each distance step is 100 miles
    That means we have **1,000** distance columns!
    - Note: this is an example of "coordinate compression"
  - Branching factor per step is 3… (climb, hold, sink)
  - That means complete search can end up performing $3^{1,000}$ operations…
  - Too many for 3s time limit ☹

Overlapping Sub Problem Issue

- In simple $3^{1,000}$ Complete Search solution, we observe **many** **overlapping sub problems**!
  - Many ways to reach coordinate (alt, dis)



DP Solution

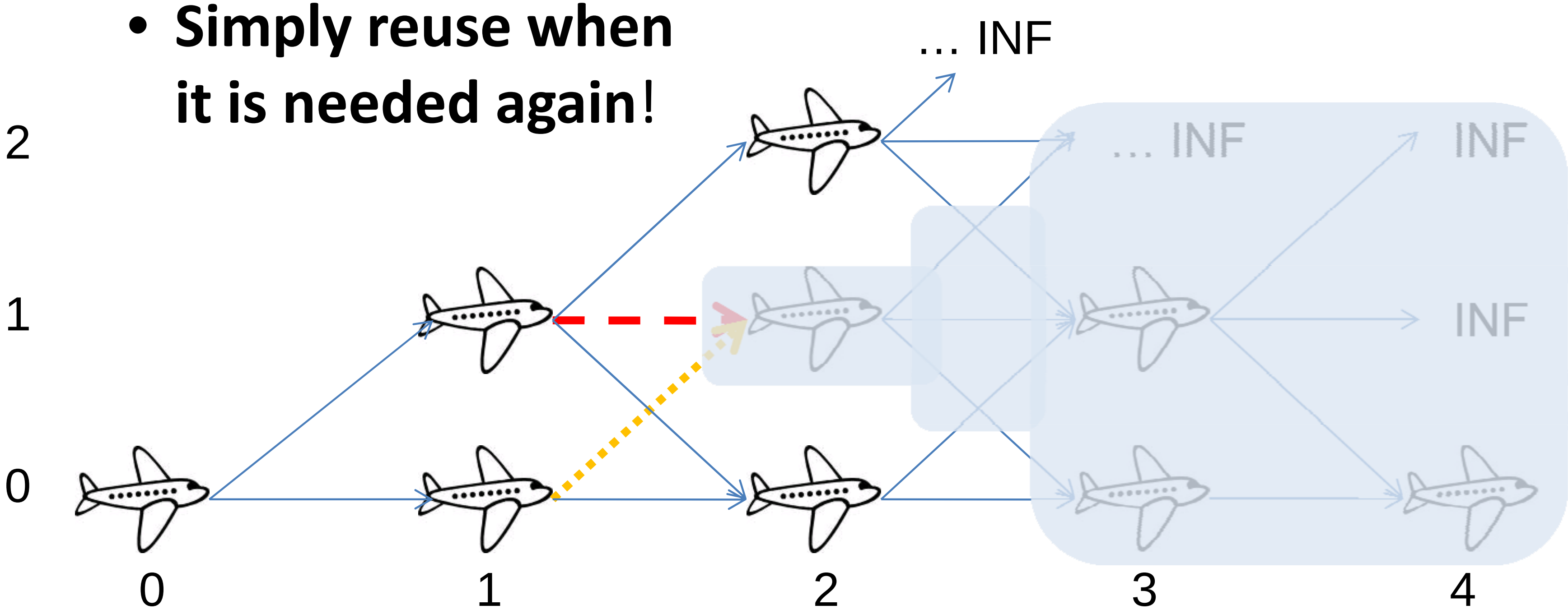
- Recurrence* of the Complete Search
  - ```
    fuel(alt, dis) =
      min3(60 - wsp[alt][dis] + fuel(alt + 1, dis + 1),
           30 - wsp[alt][dis] + fuel(alt    , dis + 1),
           20 - wsp[alt][dis] + fuel(alt - 1, dis + 1))
    ```
- Sub-problem fuel(alt, dis) can be **overlapping**!
  - There are only 10 alt and 1,000 dis = **10,000** states
  - A lot of time saved if these are not re-computed!
    - Exponential $3^{1,000}$ to polynomial 10*1,000!
    ```

alt > 2 not shown

2	-1	-1	-1	∞	∞
1	-1	-1	40	19	∞
0	-1	-1	-1	29	0
	0	1	2	3	4

DP Solution (Top Down)

- Create a 2-D table of size 10 * (X/100) ← Save Space!
  - Set "-1" for unexplored sub problems (memset)
  - Store the computation value of sub problem
    - **Simply reuse when it is needed again!**



DP Solution (Bottom Up)

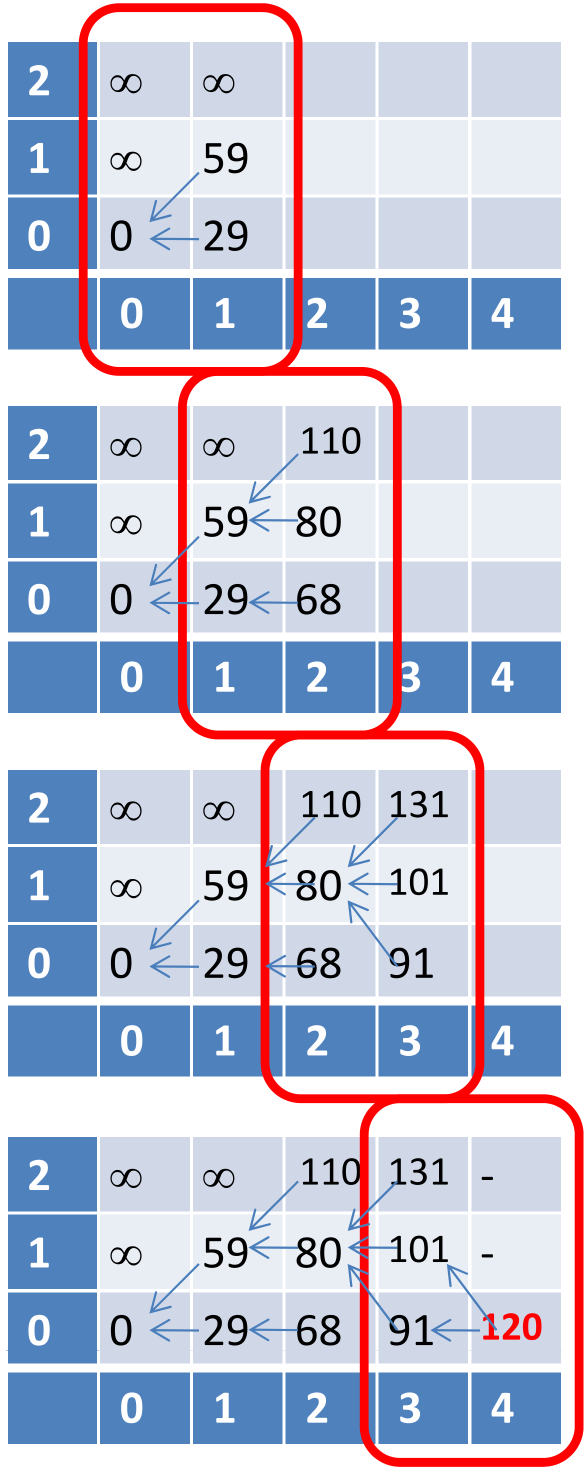
```
fuel(alt, dis) =
  min3(20 - wsp[alt + 1][dis - 1] + fuel(alt + 1, dis - 1),
       30 - wsp[alt    ][dis - 1] + fuel(alt    , dis - 1),
       60 - wsp[alt - 1][dis - 1] + fuel(alt - 1, dis - 1))
```

```
1    1    1    1      | 9
1    1    1    1      | 8
1    1    1    1      | 7
1    1    1    1      | 6
1    1    1    1      | 5
1    1    1    1      | 4
1    1    1    1      | 3
1    1    1    1      | 2
1    9    9    1      | 1
1   -9   -9    1      | 0
========================
0    1    2    3    4 (x100)
```

Tips:
(space-saving trick)

We can reduce one storage dimension by only keeping 2 recent columns at a time…

But the time complexity is unchanged: O(10 * X / 100)



If Optimal Solution(s) are Needed

- Although not often, sometimes this is asked!
- As we build the DP table,
  record which option is taken in each cell!
  - Usually, this information is stored in different table
  - Then, do recursive scan(s) to output solution
    - Sometimes, there are more than one solutions!

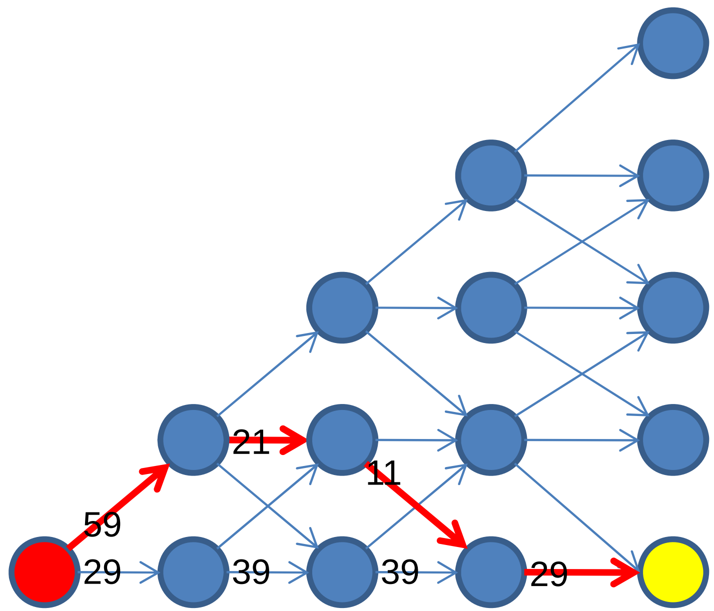
2	∞	∞	110	131	-
1	∞	59	80	101	-
0	0	29	68	91	**120**
	0	**1**	**2**	**3**	**4**

Shortest Path Problem? (1)

- Hey, I have alternative solution:
  - Model the problem as a **DAG**
  - Vertex is each position in the unit map
  - Edges connect vertices reachable from vertex (alt, dis), i.e. (alt+1, dis+1), (alt, dis+1), (alt-1, dis)
    - Weighted according to flight action and wind speed!
    - Do not connect infeasible vertices
      - alt < 0 or alt > 9 or dis > X

Visualization of the DAG

```
1   1   1   1     | 9
1   1   1   1     | 8
1   1   1   1     | 7
1   1   1   1     | 6
1   1   1   1     | 5
1   1   1   1     | 4
1   1   1   1     | 3
1   1   1   1     | 2
1   9   9   1     | 1
1  -9  -9   1     | 0
========================
0   1   2   3   4 (x100)
```



Source

What is the **shortest path** from source to destination?

Shortest Path Problem? (2)

- The problem: find the **shortest path** from vertex (0, 0) to vertex (0, X) on this DAG…
- O(V + E) solution exists!
  - V is just 10 * (X / 100)
  - E is just 3V
  - Thus this solution is as good as the DP solution

Break

- Coming up next, discussion of some **Classical DPs**:
  - Max Sum (1-D for now) → Kadane's Algorithm
  - Longest Increasing Subsequence (LIS) → O(n log k) solution
  - 0-1 Knapsack / Subset Sum → Knapsack-style parameter!
  - Coin Change (the General Case) → skipped, see textbook
  - Traveling Salesman Problem (TSP) → bitmask again :O

- I will try to cover as many as possible, but will stop at 9 pm ☺; the details are in Chapter 3 of CP2.9

Let’s discuss several problems that are solvable using DP

First, let’s see some classical ones…

LEARNING VIA EXAMPLES

Max Sum (1D)

- Find a **contiguous sub-array** in 1D array A with the **max sum**

i	0	1	2	3	4	5	6	7	8
A[i]	1	-2	**6**	**3**	**2**	-12	-6	7	1

- The answer is **{6, 3, 2}** with max sum 6 + 3 + 2 = 11

Can we do this in $O(n^3)$?

Can we do this in $O(n^2)$?

Can we do this in $O(n)$?

Longest Increasing Subsequence

- Find the Longest Increasing **Subsequence** (LIS) in array A
  - Subsequence is not necessarily contiguous

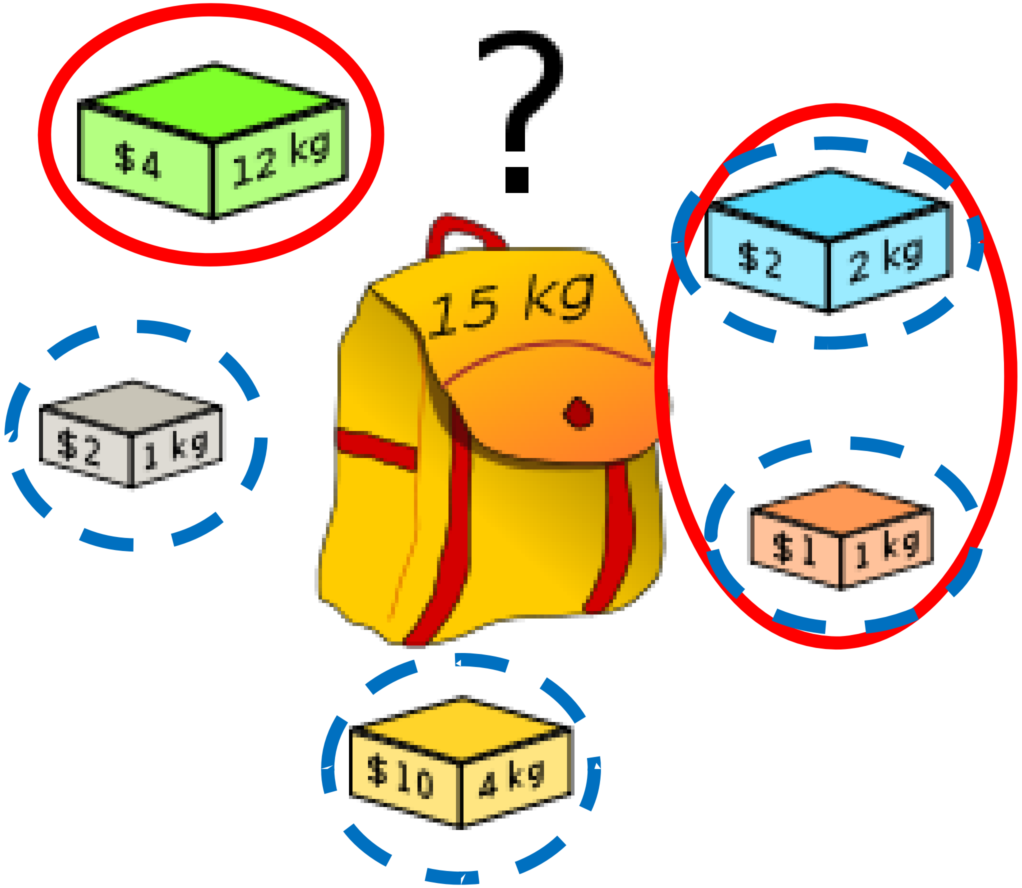
i	0	1	2	3	4	5	6	7
A[i]	**-7**	10	9	**2**	**3**	**8**	8	1

- The answer is {-7, 2, 3, 8} with length 4

Can we do this in $O(n^2)$?

Can we do this in $O(n \log k)$?

0-1 Knapsack / Subset Sum



Red = 15 kg, $ 7

Blue = 8 kg, $ 15

Can we do this in O(nS)?

n = # items
S = knapsack size

Traveling Salesman Problem (TSP)

dist	0	1	…	n-1
0				
1				
…				
n-1				

Traveling Salesman Problem (TSP)

- State: tsp(pos, **bitmask**)
- Transition:
  - If every cities have been visited
    - $tsp(pos, 2^N-1) = dist[pos][0]$
  - Else, try visiting unvisited cities one by one
    - $tsp(pos, bitmask) =$
      $\min(dist[pos][nxt] + tsp(nxt, bitmask \mid (1 << nxt)))$
      $\forall nxt \in [0..N-1], nxt \mathrel{!=} pos, bitmask \ \& \ (1 << nxt) == 0$

Summary

- We have seen:
  - Basic DP concepts
  - DP on some **classical** problems
- We will see more DP next week:
  - DP on **non classical** problems
  - DP and its relationship with DAG
  - DP on Math & String Problems
  - Some other “cool” DP (optimization) techniques

Good References about DP

- CP2.9, obviously ☺
  - Section 3.5 first
    - Then Section 4.7.1 (DAG), 5.4 (Combinatorics), 6.5 (String + DP), 8.3 (more advanced DP), parts of Ch 9
- http://people.csail.mit.edu/bdean/6.046/dp/
  - Current USACO Director
- TopCoder Algorithm Tutorial
  - http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=dynProg