Competitive Programming

From Problem 2 Solution in O(1)

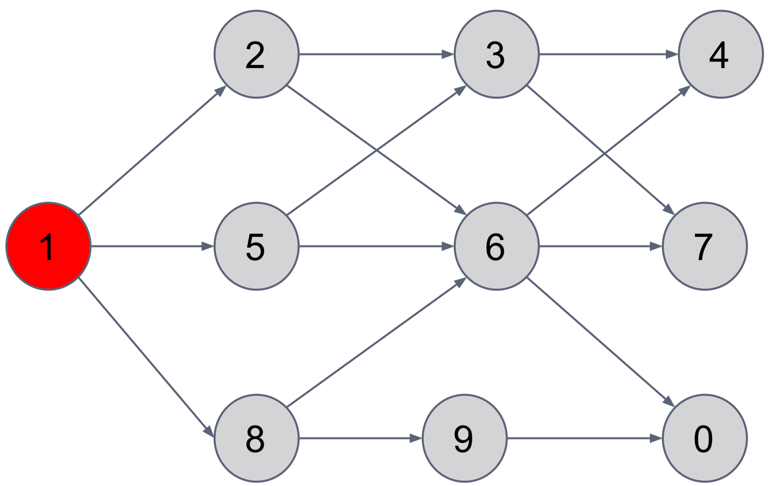
Greedy Algorithms

Greedy Properties

Mostafa Saad Ibrahim
PhD Student @ Simon Fraser University

Recall: Search for a solution

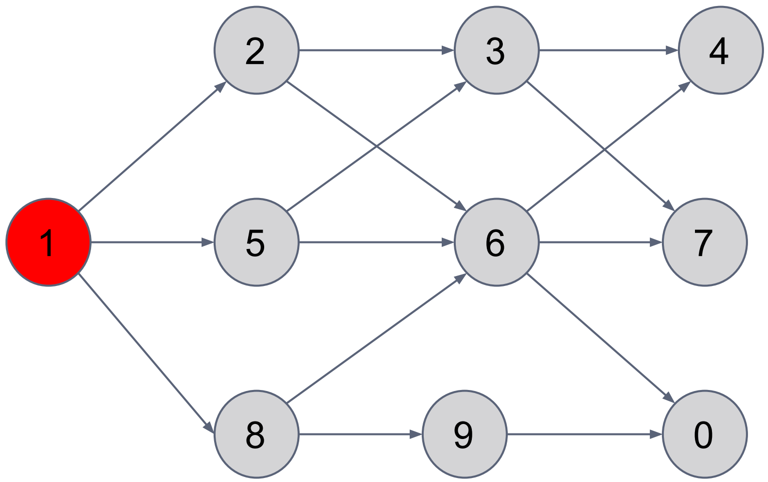
- Assume we have several states to find some solution path starting from 1



Assume **optimal solutions** are:
{1, 2, 3, 7}
{1, 5, 6, 7}
{1, 5, 3, 7}
{1, 2, 6, 0}

Recall: Dynamic Programming

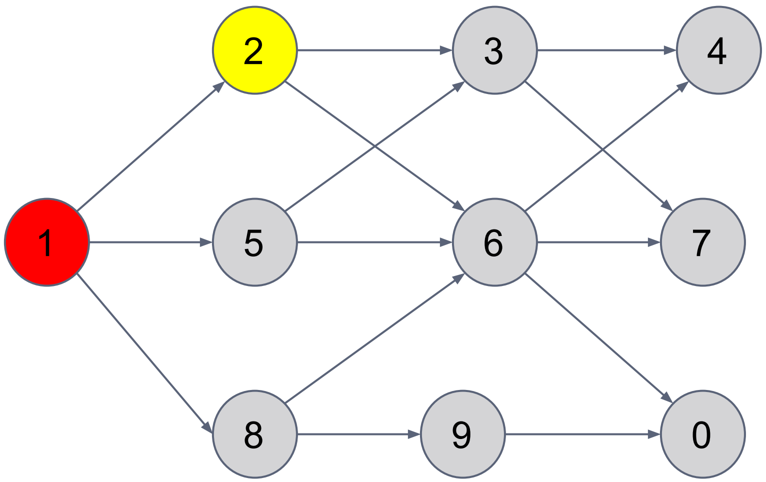
- DP will go in every possible path, check all
  - Don't **repeat** calculations (otherwise, it is backtrack)



- **Try 2**
  - Try 3
    - Try 4
    - Try 7
  - Try 6
    - ...
- **Try 5**
  - try 3
    - **COMPUTED**
  - try 6
- **Try 8**
  - ….

Recall: Greedy

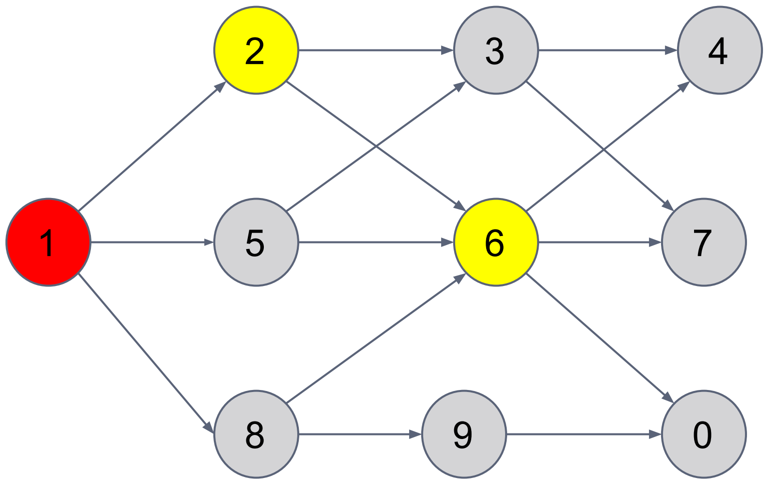
- Greedy make 1 choice only..local choice
  - This step is in the optimal choice



- **I am sure it is 2**

Recall: Greedy

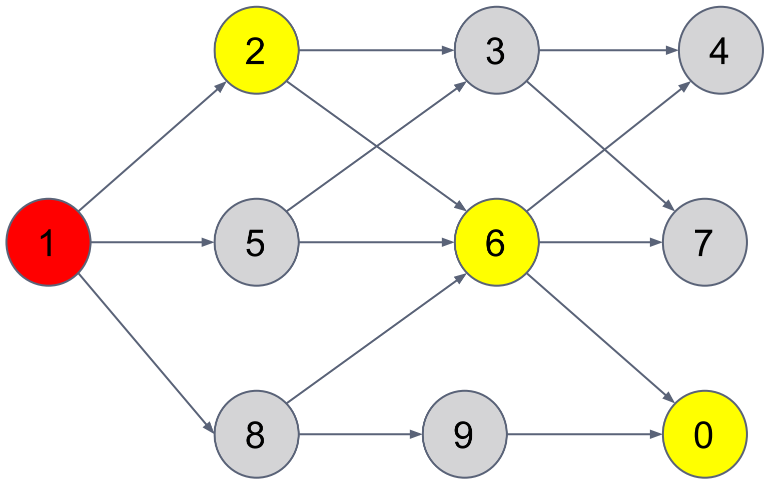
- Greedy make 1 choice only..local choice
  - This step is in the optimal choice



- **I am sure it is 2**
- **I am sure it is 6**

Recall: Greedy

- Greedy makes 1 local choice only per step
  - This step is part of the optimal choice



- **I am sure it is 2**
- **I am sure it is 6**
- **I am sure it is 0**

Recall: {1, 2, 6, 0} is **ONE** of optimal solutions

Greedy is very efficient...but few problems can be greedy

Each step is a new **sub-problem**

Proving Greedy

- Generally, by induction and by contradiction
  - Regardless of your approach, you may need them
- Directly, prove greedy properties
  - Next of this session
- Use greedy proof methods
  - Staying ahead
  - Exchange arguments
- Advanced [Out of scope]
  - Matroids
  - Greedoids
  - Matroid embeddings

Greedy properties

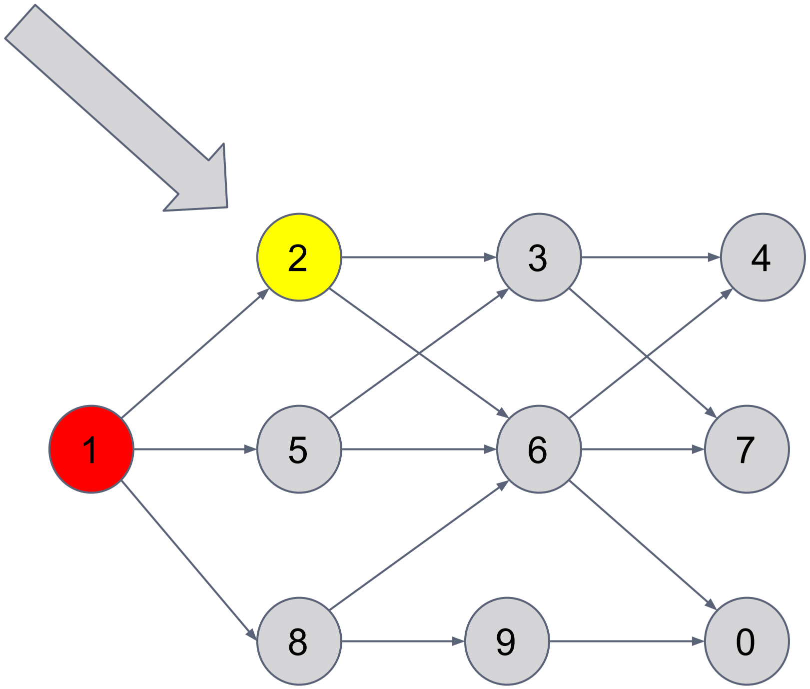
- Greedy Algorithm is 2 steps
  - Make a local **choice**
  - Solve **subproblem**
- For correctness, we must prove 2 properties
  - Greedy **choice** property
    - Multiple solutions: Proof by contradiction
    - Only one solution: Adhock proof
  - Optimal **substructure** property
    - Proof by contradiction: sub-problem can’t be better

Greedy choice property

General technique (multiple solutions):

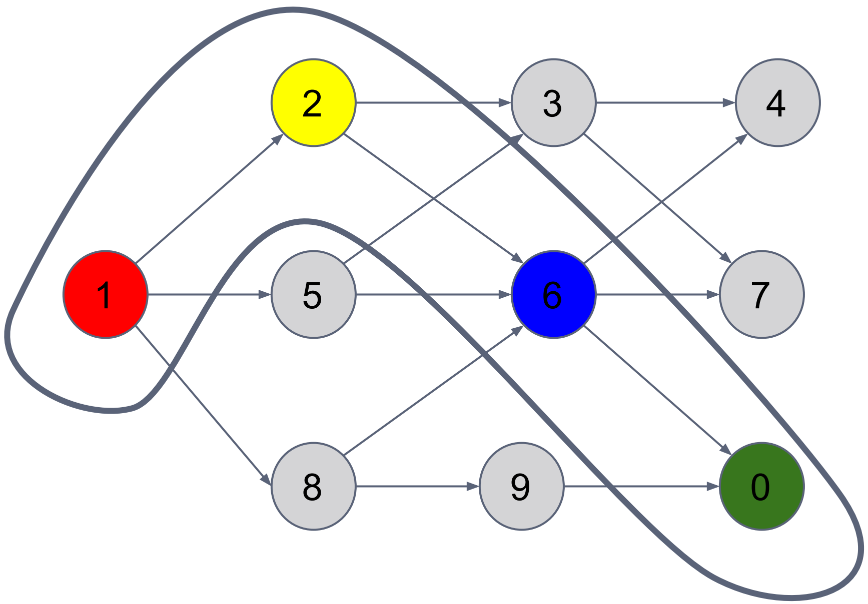
- Let S be optimal that doesn't include choice
- Let S' = Modified S to contains the choice
- Show S' as good as S

Is 2 part of some optimal solution?



Optimal substructure property

- We used this in DP too. It relates optimality of sub-problem to the problem
- An optimal solution to the **problem contains** an optimal solution to **subproblems**
- One problem optimal solution: F(1) = [1, 2, 6, 0]
  - Sub-optimal solutions: F(2) = [2, 6, 0], F(6) = [6, 0], F(0) = [0]
  - E.g. remember DP: in every sub-state, try all and pick best sub-solution

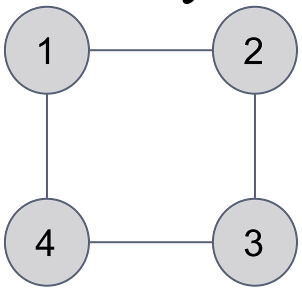


SP and Optimal substructure

- Is subpath of a shortest path is a shortest path?
  - We can prove that.
- Simple Shortest path SP(A, B) in graphs
  - Let SP(1, 6) = [1, 2, 5, 3, 8, 4, 6]
  - SP(1, 6) tells us other **7*6/2 sub-shortest paths**
  - E.g. SP(2, 4) = [2, 5, 3, 8, 4] and SP(8, 6) = [8, 4, 6]
- Note: if SP(1, 6) path passes with node 3
  - Then SP(1, 6) = SP(1, 3) + SP(3, 6)
  - E.g. SP(1, 3) = [1, 2, 5, **3**]
  - E.g. SP(3, 6) = [**3**, 8, 4, 6]
- That is why floyd-warshal and Dijsktra work

LP and Optimal substructure

- Is subpath of a longest path is a longest path?
  - No, we can find counterexample.
- Simple Longest path LP(A, B) in graphs
  - From graph: LP(1, 3) = **[1, 2, 3]** or [1, 4, 3]
  - Then LP(1, 2) = [1, 2], based on optimal substructure
  - However: LP(1, 2) = [1, 4, 3, 2] from graph
  - Then problem sub-solution is not solution sub-problem
  - Longest path doesn’t satisfy the optimal substructure



Change-making problem

- Given money X and Coin types: what is the least number of coins to sum X?
- E.g. X = 47, Coins = {1, 5, 10, 25}
  - Use 25+10+10+1+1 = 5 coins
- Greedy: Pick largest possible coin value first
  - Works for: {1, 5, 10, 25}
  - Fails for {1, 5, 10, **12**, 25}
- General solution: Dynamic programming, (leave it, pick it/try it again)

Change-making problem

Greedy Coin-Changing Algorithm

```
Sort coins denominations by increasing value:
c_1 < c_2 < ... < c_n.

S ← φ                    S = coins selected.
while (x ≠ 0)
   let p be largest integer such that c_p ≤ x
   if (p = 0)
      return "no solution found"
   x ← x - c_p
   S ← S ∪ {p}
return S
```

Fails for {1, 5, 10, 12, 25}
E.g. X = 15
Algorithm gives: 12, 1, 1, 1
Correct is: 10+5

Src: http://www.cs.princeton.edu/~wayne/cs423/lectures/greed-4up.pdf

Change-making problem

- Greedy Choice Property: [1, 5, 10, 25]
  - Target: if we picked largest possible coin Ck
  - Prove, this coin is part of some optimal solution
  - Only 1 solution...Ad Hoc proof...study 2 points
  - **The maximum # of coins** of each type in optimal solution
  - E.g. we can maximum have 4 ones. If 5 ones, replace them with 1 coin of value 5. Similarly maximum 1 5.
  - **The maximum value** using the first K coins
  - E.g. using first coin max value is 4. Using first 2 coins, max value is 9. Notice, we can't make 10 using first 2 coins. We can't make 25 using first 3 values, in optimal solutions

Change-making problem

- Consider optimal way to change amount $c_k \leq x < c_{k+1}$.
- Greedy takes coin k.
- Suppose optimal solution does not take coin k.
  - it must take enough coins of type $c_1, c_2, \ldots, c_{k-1}$ to add up to x.

k	c_k	Max # taken by optimal solution	Max value of coins 1, 2, . . . , k in any OPT
1	1	4	4
2	5	1	4 + 5 = 9
3	10	2	20 + 4 = 24
4	25	3	75 + 24 = 99
5	100	no limit	no limit

2 dimes $\Rightarrow$ no nickels

Src: http://www.cs.princeton.edu/~wayne/cs423/lectures/greed-4up.pdf

Change-making problem

- Optimal substructure property
  - If best way to change 34¢ is {25, 5, 1, 1, 1, 1}
  - Then best way to change 29¢ is {25, 1, 1, 1, 1}
  - Then best way to change 7¢ is {5, 1, 1}...2^n subsets
- Formally:
  - Assume optimal solution is: F(X) = {C1, C2….Cm}
  - To build X, Assume greedy picked C1
  - Then F(X-v(C1)) = {C2, ...Cm} by optimal substructure
  - Let optimal F(X-v(C1)) = {A1, A2...Ak} where k < m
  - Then, {C1, A1, A2...Ak} has fewer coins than F(X), which is contradiction.

Sorting and Greedy

- Recall...Greedy makes locally optimal choice at each stage
- Many greedy algorithms use sorting … where is the local **choice per step**?
- Think like, we have N numbers
  - In first step, we pick smallest one..remains N-1 numbers
  - In second step...we pick smallest among N-1
  - and so on...these are local choices
- So overall of local choices = sorted array

Sorting and Greedy

- Many greedy solutions are as following:
- 1) Sort items based on some criteria
  - shortest value, earliest start time, processing time...etc
- 2) For every item
  - 2.1) CanBeMyLocalOptimalChoice(item)
  - 2.1.1) Add it to the global optimal solution
- Example: think in Kruskal Algorithm
  - Sort edges decreasing. Add it if no cycles
- Sometime, the reverse: E.g. remove item from overall items (e.g. Reverse-delete algorithm)

General guidelines

- Competitions: Optimal greedy only
- Can do it with DP / Bruteforce? Do it
- Otherwise
  - Brainstorm on all possible heuristics...put in ur ideas' pool
  - Don't think/verify….just brainstorm
- Pick most promising idea
  - Find some example that breaks it...think hard
  - Wrong? think what to fix generally...push idea to the pool
- Good heuristic so far? Can you prove? or informal strong intuition? Yes => do it

General guidelines

- Can’t prove in short time?
  - Code it..greedy usually is little code
  - Either submit…or write small brute force solution
  - write random case generator and test greedy vs brute force
- Training offline? Put much time to prove it
- Sometimes a strategy doesn’t have much intuition, but correct and provable
- Greedy in hard problems is tricky, proving is a must, as it reveals the different scenarios

Readings

- Overview of properties
- Properties with activity selection problem
- Properties with Fractional Knapsack
- Read Greedy Chapter from algorithms book
  - Introduction to Algorithms

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً