NYU

CSCI-UA.0480-004 Algorithmic Problem Solving

Lecture 10

Greedy

Bowen Yu
Spring 2018

HW

- HW5 released
  - Bitmask and Trie
  - Due on Sunday

- HW6 released
  - Union find
  - Due on next Wednesday

Greedy Algorithm

- Make a greedy choice for a local configuration
- Provable that making local greedy choices leads to a final global optimum.
- A greedy choice is *never* reverted.
    - We do not go back and consider other choices earlier. Otherwise it is not greedy but complete search.

- Greedy algorithm is often ad hoc
- Requires practice + creativity!

Scarecrow

There is one row of land with N cells (N <= 100). A cell may contain a crop or is empty. To prevent crops from being eaten by crows, we can install a scarecrow in some cells. A scarecrow installed at cell i protects cell i, cell i - 1 and cell i + 1 (three consecutive cells). What is the minimum number of scarecrows we need to install to protect all crops?

Sample: # is crop, `.` is empty

```
..##..##. => ..##..##.  ans = 2
.#.#.#..# => .#.#.#..#  ans = 3
```

Scarecrow

Solution: For the leftmost crop, place a scarecrow in the cell right to it.

```
..##..##.
  [1]
..##..##.
  [1] [2]
```

```
.#.#.#..#
 [1]
.#.#.#..#
     [2]
.#.#.#..#
       [3]
```

For any optimal solution, there must be one scarecrow covering the leftmost crop. If it is not right to the crop, we can move it while still being able to get the optimum.

Time: O(N)

NYU

Flips

There is one roll of N cards (N <= 100). The front of a card is white, and the back of a card is black. Initially some cards are facing up while the others are facing down. You can select *exactly* K consecutive cards and flip them together (1 <= K <= N). Determine if it is possible to flip the cards so that all of them become facing up. If so, also determine the minimum number of flips.

Sample: # is black, `.` is white

```
N = 8, K = 3
..##.##.
```
Two moves are needed
```
..##.##. => ...###.. => ........
```

```
N = 8, K = 3
..##.#.#
```
answer is impossible

Flips

Solution: Find the leftmost black card, starting from this card flip K consecutive cards.

```
N = 8, K = 3
..##.##.

1 ..##.##.
  ....###.
2 ....###.
  ........
```

```
N = 8, K = 3
..##.#.#

1 ..##.#.#
  ....##.#
2 ....##.#
  ......##
3 ......##X
```

Time: O(NK)

This sounds intuitive, but how do we show that:

- If there exists a solution, then this strategy will find it?
- This strategy can give the minimum number of flips?

Develop a Greedy Strategy

- Find an intuition for the optimization and come up with a hypothetical greedy strategy
- Examine the strategy on some small cases and look for breakers
- **Prove the strategy**
  - This is the most important step. An unproven greedy strategy is very risky.
  - This is also the most difficult part. Many greedy algorithms are easy to describe, intuitive, but difficult to prove.

Proof: Flips

Solution: Find the leftmost black card, starting from this card flip K consecutive cards.

How do we show that:

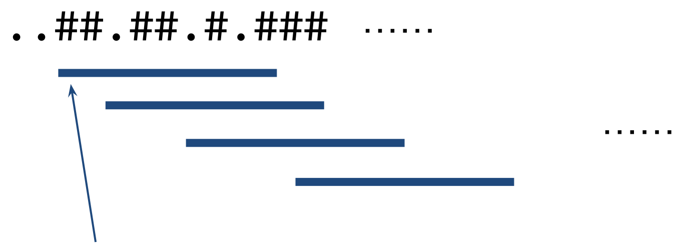
- If there exists a solution, then this strategy will find it?
- This strategy can give the minimum number of flips?

Observation

- There are never more than one flips at a same position in a solution
  - flipping twice at a same position does nothing
- The order of the flips does not matter

Proof: Flips (cont'd)

Since all flips are at distinct positions, we can order them from left to right and apply them in this order.



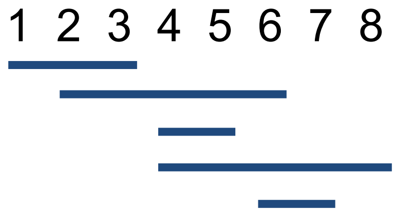
The left endpoint of the leftmost flip must be at the first black card. Otherwise we cannot have all cards being white in the end.

Therefore for every optimal solution, the leftmost flip must be the same with the first flip in our greedy strategy. Then we induct for all the flips. ∎

Event Selection

There is a room that N people want to book ($N<=10^5$). Each person wants to use the room for some time interval [s, t] ($0<=s<=t<=10^9$). Find the largest number of people who get can the room without a conflict.

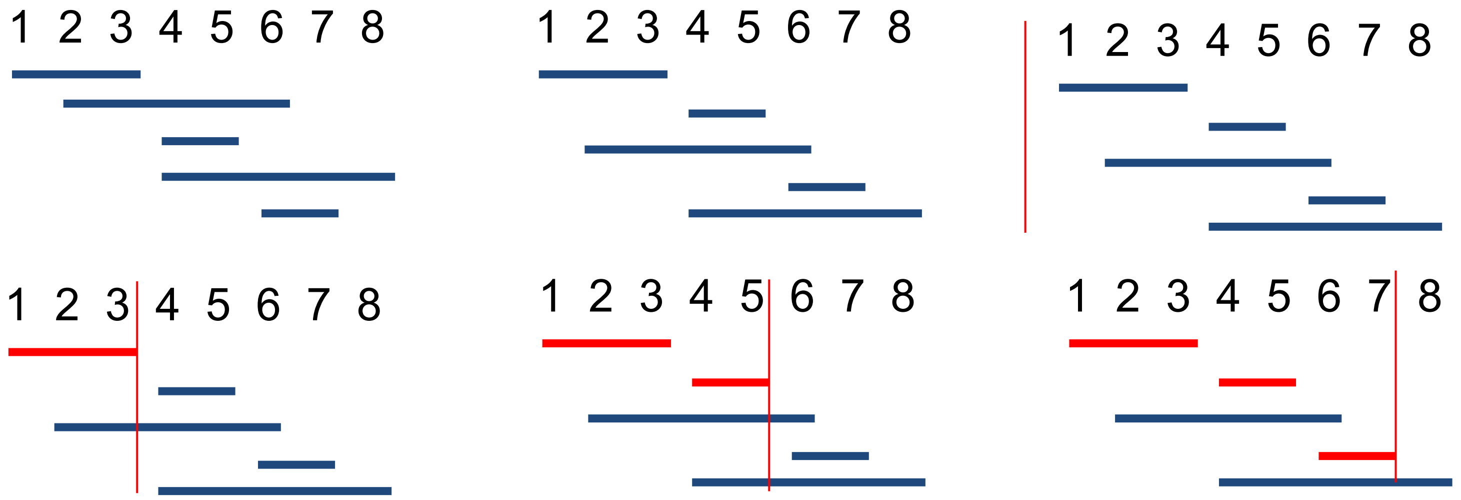
Sample:
[1, 3], [2, 6], [4, 5], [4, 8], [6, 7]



Solution:

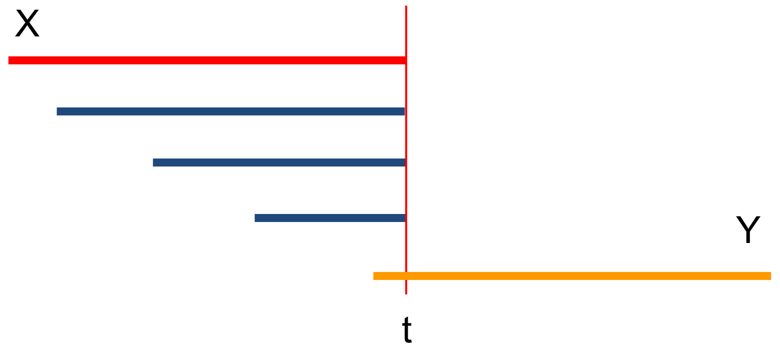
- Sort the intervals by increasing endTime (tie can be handled arbitrarily).
- Initially latestEndingTime is -1.
- For each interval X, select it if it does not contain latestEndingTime.
- Update latestEndingTime by X.endTime.

Time: O(NlogN)



Proof: Event Selection

Consider those intervals with a smallest end time t. We must have selected one of them, say X.



- If an optimal solution has one interval ending at t but it is not X, then we can replace it by X.
- If an optimal solution does not have any of those intervals ending at t, then it must contains an interval Y that contains t (otherwise we can add X without conflict). We can thus replace Y by X.

After the replacement, the remaining set of intervals will be the same. Thus our greedy algorithm preserves the opportunity to reach global optimum.

Proof of a Greedy Strategy

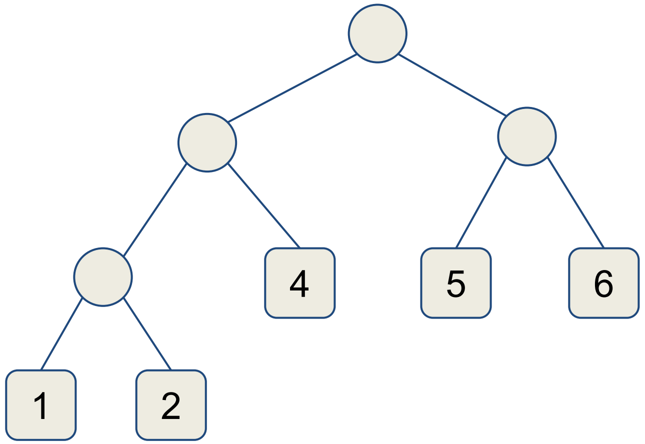
- Prove that any global optimum must contain our local choice
  - e.g. Flips
- Prove that if a global optimum does not contain our local choice, we can make some replacement to let it contain our local choice, while still being able to reach a global optimum
  - e.g. Event Selection

Huffman Tree

Arrange N non-negative integers onto a binary tree's leaves to minimize

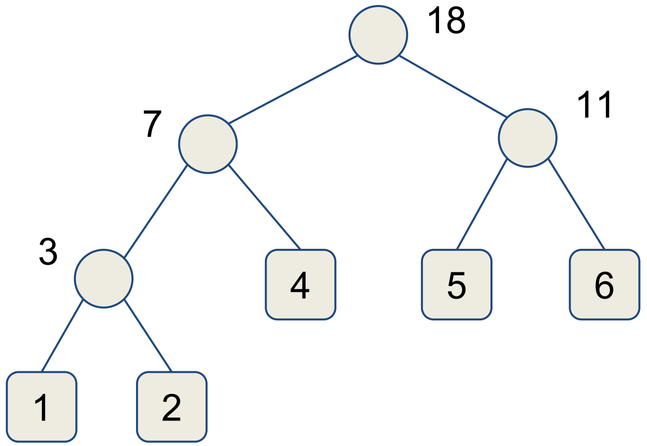
$$\sum_x x \cdot depth(x)$$

Sample:
[1, 2, 4, 5, 6]
Cost = (1+2)*3 + (4+5+6)*2
= 39



Huffman Tree Algorithm

- Let S contain all integers. Each integer is a node.
- Select the two smallest integers x, y from S. Create a new node with x, y as its children. Let this new node have value x + y. Remove x, y from S. Add x + y to S.
- Repeat until S contains one single integer.



- Proof of correctness
  - Show that any optimal tree must have the two smallest integers as a pair of deepest sibling nodes; otherwise we can swap to reduce cost.
  - Show that after processing the two smallest integers, the problem becomes an identical sub-problem where the same strategy applies.
- Complexity
  - Using a heap to store S, the total time complexity is O(NlogN)

Coins

There are N types of coins of distinct values (integers in $[1, 10^9]$). Each type of coin has unlimited supply. Given a total value S ($1<=S<=10^9$), determine the minimum number of coins needed to compose S. There is always a coin of value 1.

Sample:
coins = [1, 2, 5, 10, 50]
S = 88
answer is 7 coins: 50, 10, 10, 10, 5, 2, 1

Where Greedy Doesn’t Work

Incorrect Greedy Strategy: Use as many largest coins as possible

coins = [1, 4, 5, 6]
S = 9

Greedy gets 4 coins: 6, 1, 1, 1
But only 2 are needed: 4, 5

NYU

Coins

- If we look at real-world coin (bill) values: [1, 2, 5, 10, 50], then the greedy strategy works.

- Someone claims that if we order the coins $C_1 < C_2 < \cdots < C_N$ and we have

$$\sum_{1 \le i < j} C_i \le C_j$$

then the greedy strategy will work (?)

- e.g. the binary coin system: [1, 2, 4, 8, 16, ...]

Coins

Someone claims that if we order the coins $C_1 < C_2 < \cdots < C_N$ and we have

$$\sum_{1 \le i < j} C_i \le C_j$$

then the greedy strategy will work (?)

- Breaker: coins = [1, 8, 9], S = 24
- Before implementing or proving a "greedy strategy", try to find small counterexamples that break it.

* Given a set of coins, check if greedy applies is a more complicated topic.
https://naq17.kattis.com/problems/canonical

ZigZag Subsequence

Given a list of N integers ($N <= 10^5$), determine the length of its longest zigzag subsequence. A sequence is zigzag if $x_1 > x_2 < x_3 > x_4 \ldots$

Sample:

[2, 4, 1, 3, 5] => [$\underline{2}$, 4, $\underline{1}$, $\underline{3}$, 5], ans = 3

[4, 4, 2, 3, 3, 5, 1] => [4, $\underline{4}$, $\underline{2}$, 3, $\underline{3}$, 5, $\underline{1}$], ans = 4

Solution: Pick peak and valley alternatively.

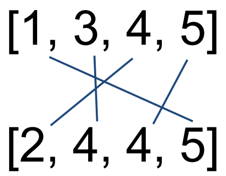
peak/valley: last element in an increasing/decreasing streak

Proof: For every increasing streak, we can choose at most two elements (its first and last element). The greedy strategy indeed chooses two elements.

Pairing Sticks

There are N black and N white sticks (N<=100) of different integer lengths. You are to pair every black stick X with one white stick Y. You are also given two constant integers C (cap), F (fine). If X.length + Y.length > C, you have to pay F*(X.length + Y.length - C) dollars. Determine the minimum amount of dollars you have to pay.

Sample:
Black: [1, 3, 4, 5]
White: [2, 4, 4, 5]
C = 7, F = 10
answer is 20, we can have



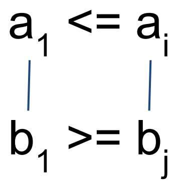
Pairing Sticks

Solution: Pair the shortest white stick with the longest black stick.

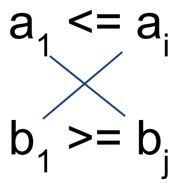
Proof:
Let white sticks' lengths be $a_1 <= a_2 <= \ldots <= a_n$
Let black sticks' lengths be $b_1 >= b_2 >= \ldots >= b_n$

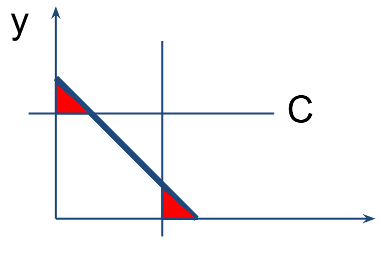
If in an optimal solution, we don't have (a_1, b_1), then we must have (a_1, b_j) and (a_i, b_1). In this case we can swap them to obtain a potentially smaller cost. This is because:

$S = a_1 + b_1 + a_i + b_j$ is a constant, consider the line $x + y = S$



can be no worse than





We have to pay in the red area. So we'd better move (x, y) to the middle. Correspondingly, we'd pair a_1 with b_1

NYU

Job Scheduling

There are N jobs ($N<=1000$). Job i takes T_i days to complete and has a delay fine F_i. Starting from day 0, we complete the jobs one by one. If a job is started on day d, we have to pay a delay fine of $d * F_i$. Determine the minimum total delay fine we have to pay if we order the jobs optimally.

Sample:
N = 4
T_i = [3, 1, 2, 5]
F_i = [4, 1000, 2, 5]

One optimal order: [2, 1, 3, 4]
Job 2 started on day 0, fine = 0
Job 1 started on day 1, fine = 1*4 = 4
Job 3 started on day 4, fine = 4*2 = 8
Job 4 started on day 6, fine = 6*5 = 30
Total fine = 42

Job Scheduling

Solution: For a pair of jobs x and y, if TxFy <= TyFx, then do job x before job y.

Proof:

- Define: better(x, y) ⇔ TxFy <= TyFx
- Trivially for two jobs x and y, the optimal order is [x, y] ⇔ better(x, y)
- Consider three jobs x, y, z. We want to show better(x, y), better(y, z), **better(x, z)** => [x, y, z] is optimal

Proof: Job Scheduling

- better(x, y), better(y, z), better(x, z) => [x, y, z] is optimal
  - There are 6 orders: xyz, xzy, yxz, yzx, zxy, zyx
  - Since we already know better(x, y) and better(y, z), xzy, yxz, zyx cannot be optimal. We only need to discuss xyz, yzx, zxy
  - As an example, we show cost(xyz) <= cost(yzx)
    - cost(xyz) <= cost(zxy) is similar.

cost(xyz) - cost(yzx)
= (TxFy + TxFz + TyFz) - (TyFz + TyFx + TzFx)

$$= \underbrace{(TxFy - TyFx)}_{\text{better}(x,\ y)} + \underbrace{(TxFz - TzFx)}_{\text{better}(x,\ z)} \le 0$$

Proof: Job Scheduling (cont'd)

- For three jobs x, y, z, we have proven that better(x, y), better(y, z), better(x, z) => [x, y, z] is optimal. We can extend this to more jobs.
- For three jobs a, b, c with optimal order [a, b, c], and a fourth job d, we can use induction to show better(a/b/c, d) => [a, b, c, d] is optimal
  - Obviously, **d**abc, a**d**bc, ab**d**c are not optimal
  - Let "+" denote job combination, x+y is a virtual job (Tx+Ty, Fx+Fy).
  - better(a, d) and better(b, d) => better(a+b, d) => [a+b, c, d] is optimal => [a, b, c, d] is optimal

Max OR

You are given a 32-bit unsigned integer X and a range [L, R]. Find an integer Y (L<=Y<=R) so that X | Y (bitwise OR) is maximum. If there is a tie, choose the smallest Y.

Sample:
X = 100, L = 50, R = 60 => Y = 59

```
X   1100100      R   0111100
Y   0111011      Y   0111011
X|Y 1111111      L   0110010
```

Max OR

Solution:

- Iterate each bit from highest to the lowest
- If the bit is set in X
  - Then we do not need this bit to have a larger OR value
  - But if not setting this bit results in Y < L, then we must set this bit
    - We haven’t set the lower bits yet. How do we know that Y can be < L? Just try to set all lower bits to 1 and see if Y < L.
- If the bit is not set in X
  - Then we need this bit to have a larger OR value
  - But if setting this bit results in Y > R, then we must skip this bit

Correctness is implied by the greedy construction and proof is omitted.

Gas Stations

You drive a car along a road. Your car initially has a full tank of capacity C. You start at x=0 and want to reach x=L. There are $N<=10^5$ gas stations on the road. The i-th station is at Xi (0<Xi<L) and sells gas for Pi dollars per gallon. At each gas station, you can refill any amount of gas till a full tank. Determine the minimum total cost to reach x=L, assuming it costs 1 gallon of fuel to drive 1 unit of distance.

Sample:

C = 4, L = 6
X1 = 1, P1 = 3
X2 = 2, P2 = 2
X3 = 5, P3 = 1

Drive to X2, fuel 4->2
At X2, buy 1 gallon, fuel 2->3
Drive to X3, fuel 3->0
At X3, buy 1 gallon, fuel 0->1
Drive to L=6, fuel 1->0