Greedy Algorithms

2016-03-05

Tony Wong

Coin Problem

- Suppose you are given infinitely many **coins** of the following denominations:
- $10, $5, $2, $1, $0.5, $0.2, $0.1

- What is the **minimum** number of coins required to produce **$M**?
  - M = 1.6
  - M = 4.5
  - M = 12.8

Coin Problem

- Suppose you are given infinitely many **stamps** of the following denominations:
- $2.3, $2.2, $2, $1.7, $1, $0.5, $0.2, $0.1

- What is the **minimum** number of stamps required to produce **$M**?
  - M = 2.5
  - M = 3.7
  - M = 4.9

Coin Problem

- For **some** coin systems, we can repeatedly pick the highest denomination that is larger than the current remaining value and the result must be **optimum**
- For **some other** coin systems (e.g. stamps), this technique / algorithm will not yield an optimum result
- A coin system is **canonical** ***if and only if*** GREEDY(*M*) == OPT(*M*) for all *M*
- Most, if not all, currencies in the world are canonical

Greedy Algorithm

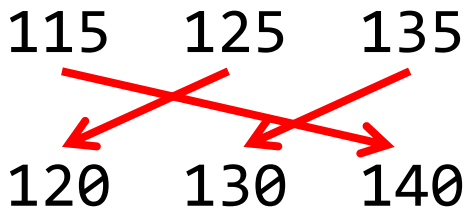
- A **greedy algorithm** tackles a problem by choosing the **locally optimal step** and then solve the **subproblem**
- It is not a specific algorithm – it is just an idea
- It is important that for a problem to be solvable using a greedy algorithm previous choices should not be relevant and need not be reconsidered

Solution to Coin Problem

- Answer to Main Problem =
  Optimal step's effect to answer + Answer to Subproblem
- For the Coin Problem example
  - Problem: the remaining amount K
  - Local optimal step: take the largest denomination
    $$d = \max\{D_i | \, D_i \leq K\}$$
  - Subproblem: $K \leftarrow K - d$
  - Answer for remaining amount K: $\text{Ans}(K) = 1 + \text{Ans}(K - d)$
  - Answer to main problem: $\text{Ans}(N)$
  - Base case: $\text{Ans}(0) = 0$

J054 Competition

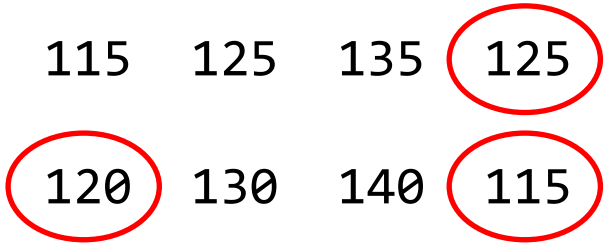
- You and your opponent both have **N** cows
- Cows have a strength attribute
  - The cow with strictly strength wins
- Strength of your cows: Ai and your opponents: Bi
- You know your opponent's sequence.
  Try to win as many rounds as possible



J054 Competition

- In 50% of the inputs: $1 \leq N \leq 10$
- Brute force solution:
  - Try every permutation of 1, 2, 3, ..., N
    - 1, 2, 3, 4, 5
      1, 2, 3, 5, 4
      ...
      5, 4, 3, 2, 1
  - Check the number of wins and determine the best permutation
- Time Complexity: O(N!)

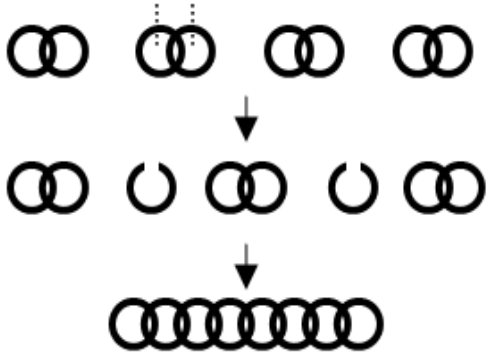
J054 Competition



- Observe that we should try to win by as little as possible, and lose by as much as possible
- Main problem: You and your opponents have **N** cows
- Subproblem: You and your opponent have **K** cows remaining
- Local optimum step for Cow K
  - Find the strongest remaining cow that has strength < A_K
    - Number of wins increases by 1
  - If there is none, (all cows have strength >= A_K)
    - Find the strongest remaining cow, and lose to it (sacrifice)
- Complexity: $O(N^2)$
- With sorting, the complexity can be reduced to O(N lg N)

J082 Joining Metal Chains

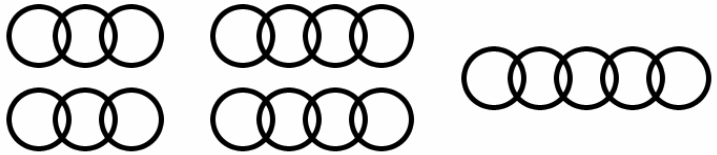
- Given some chains having of different lengths
- Open some rings to join the metal chains together
- What is the minimum number of open / close operation pairs?



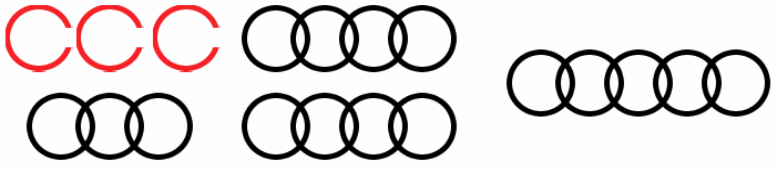
- Four chains of 2 can be joined by opening 2 rings only
- Note: Input is given in ascending (non-decreasing) order

J082 Joining Metal Chains

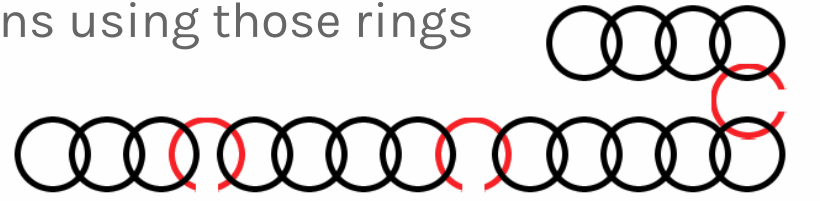
- 3 3 4 4 5



- Step 1) Open the rings from the chain of length 3



- Step 2) Join the chains using those rings



- Optimal step: Join the longest two chains using a ring from the shortest chain

Fractional Knapsack

- You went to 7-ELEVEn and bought a SLURPEE cup from the cashier
- There are four flavors to choose from
- Suppose that the cup's volume is **V** = 600 mL
- What is the maximum amount of sugar you can consume?

Flavor	Volume Available (mL)	Total Sugar (g)
Coke	320	35
Coke Zero	220	0
Fanta (Orange)	240	40
Watermelon	200	30

Fractional Knapsack

- Main problem: Remaining volume **V**
- Optimal step: Choose the one with highest sugar per unit volume among the remaining flavors, and fill as much as possible (up to the remaining flavor volume available, or remaining cup volume, whichever is less)

Flavor	Volume Available	Total Sugar	Sugar per unit volume
Coke	320	35	0.109
Coke Zero	220	0	0.000
Fanta (Orange)	240	40	0.167
Watermelon	200	30	0.150

- Answer = 40 + 30 + 35 * (600 – 440) / 320
  = 70 + 17.5 = 87.5

Fractional Knapsack

- Sort the flavors by sugar per unit volume
- Greedily fill the cup starting from the top of the list (highest sugar per unit volume) until the cup is full

```
3  bool cmp(pair<int, int> a, pair<int, int> b) {
4    // first: sugar, second: volume
5    // compare a.sugar / a.volume > b.sugar / b.volume
6    return a.first * b.second > b.first * a.second;
7  }
8  pair<int, int> flavors[100000];
9  int main() {
10   ...
11   sort(flavors, flavors + n, cmp);
12   for (int i = 0; i < n; i++) {
13     ...
```

- A lot of greedy solutions involve **sorting**
  - Order the choices from best to worst

0-1 Knapsack

- You are a thief and have a bag of volume V
- N objects: each object has volume A and value B
- Which objects should you take so that the total value is maximized?
- Proposed greedy solution: sorting by value per unit volume
  - Does it work?
  - If not, can you think of a counter example?

J042 Currency Exchange

- You know the exchange rates up to **N** days in the future
- You have US$ **M** at the beginning
- Using optimal trading strategy, how much would you have at the end of day **N**?

SAMPLE TESTS

	Input	Output
1	3 100 1.05 1.08 0.90	120.00

J042 Currency Exchange

- You cannot short-sell
- Where are the profit opportunities?

For the purple arrow,
the rate went down from
1 USD = 120.2 JPY to
1 USD = 118.5 JPY

Let's we convert 10 USD to JPY
at 1 USD = 120.2 JPY
We'll have 1202 JPY

Later, convert the JPY to USD
at 1 USD = 118.5 JPY
We'll have 10.143 USD

J042 Currency Exchange

- A long slope can be broken down into several days

$$\frac{a_0}{a_k} = \frac{a_0}{a_1} \cdot \frac{a_1}{a_2} \cdot \dots \cdot \frac{a_{k-1}}{a_k}$$

1 USD = 118.9355 JPY +0.03400 (0.029%)
Feb 25, 1:23PM GMT

J042 Currency Exchange

- Main problem: Amount on day N
- Subproblem: Consider day i – 1 and day i
- Optimal step: Compare today's rate and yesterday's rate
  - If lower, sell yesterday and buy today
  - Otherwise, do nothing
- Base Case: Day 1 = $M

S011 Activities

- There are **N** activities. Activity **i** starts at **Si** and ends at **Ei**
- You can only attend on activity at a time
- You must attend an activity in whole, not attend at all
- What is the maximum number of activities that you can attend?

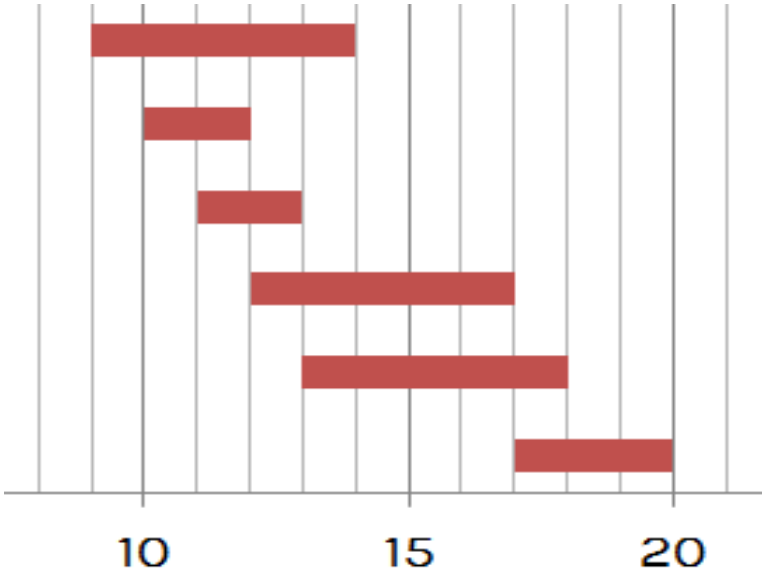
SAMPLE TESTS

	Input	Output
1	5 2000080809 2000080810 2000080811 2000080815 2000080812 2000080814 2000080813 2000080815 2000080814 2000080818	3 1 3 5

S011 Activities

- 6 activities

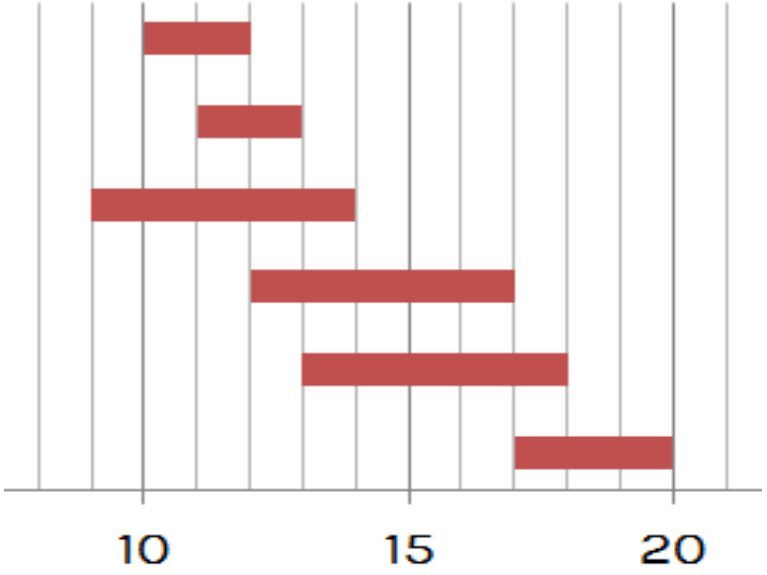
Si	Ei
9	14
10	12
11	13
12	17
13	18
17	20



S011 Activities

- Reordering the activities by **Ei**

Si	Ei
10	12
11	13
9	14
12	17
13	18
17	20



S011 Activities

- Sort the activities according to **Ei**
- Initialize **T** = 0
- Optimal step:
  - Pick first activity that has **Si ≥ T**
    the one with the earliest end time
  - **T ← Ei**

M0633 Children's Game

- Given **N** numbers, what is the largest number that can be formed by concatenating them?

SAMPLE TESTS

	Input	Output
1	4 123 124 56 90	9056124123
2	5 123 124 56 90 9	99056124123

- Sorting the numbers would not pass samples 1 and 2
- Sorting the numbers as strings would not pass sample 2

M0633 Children's Game

- Consider: A = 50 B = 5
  - A is lexicographically greater than B
  - Optimal step = 550 (B followed by A)
- Consider: A = 59 B = 5
  - This time, A is also lexicographically greater than B
  - However, optimal step = 595 (A followed by B)
- Instead of comparing A and B, compare the results of the two methods, i.e., concat(A, B) and concat(B, A)
  - Compare 505 and 550; 595 and 559
  - Since the lengths are equal, they can be compared directly

M0633 Children's Game

- Implementation:

```cpp
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
string s[50];
bool cmp(string& a, string& b) {
  return a + b > b + a;
}
int main() {
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    cin >> s[i];
  }
  sort(s, s + n, cmp);
  for (int i = 0; i < n; i++) {
    cout << s[i];
  }
  cout << endl;
  return 0;
}
```

(Alternative for C++11)

```cpp
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
string s[50];
int main() {
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    cin >> s[i];
  }
  sort(s, s + n, [](string& a, string& b) {
    return a + b > b + a;
  });
  for (int i = 0; i < n; i++) {
    cout << s[i];
  }
  cout << endl;
  return 0;
}
```

M0633 Children's Game

- How to prove that the algorithm is correct?
  - Let's say there are 3 strings A, B and C
  - If we have $A + B > B + A$ and $B + C > C + B$
    - "A before B and B before C"
  - Does that imply "A should be placed before C", i.e. is $A + C > C + A$ **true**?
- Let's say the lengths of A, B and C are a, b, c respectively
  - $A \times 10^b + B > B \times 10^a + A$ $\quad B \times 10^c + C > C \times 10^b + B$
  - $A \times (10^b - 1) > B \times (10^a - 1)$ $\quad B \times (10^c - 1) > C \times (10^b - 1)$
  - $A / (10^a - 1) > B / (10^b - 1)$ $\quad B / (10^b - 1) > C / (10^c - 1)$
  - $A / (10^a - 1) > C / (10^c - 1)$
  - $A \times (10^c - 1) > C \times (10^a - 1)$
  - $A \times 10^c + C > C \times 10^c + A$ Q.E.D.

If the relation is **transitive**, then we can sort the items by just comparing two at a time

Conclusion

- When to use?
  - When you intuition tells you that greedy solution might work
  - A counter example is not found
  - Full feedback (e.g. HKOI, Mini Competitions)
  - Points given per case + you have no idea
    - Greedy solution may pass some simple tests
- It is not realistic to prove the correctness during the competition

Suggested Tasks

- 01014 Stamps
- D109 Giving changes
- M0632 Machine Scheduling
- J044 Amazing Robot
- J064 Cave Adventures
- J154 Father's Will
- Google Code Jam 2008 1A Minimum Scalar Product