

CSE 202 - Algorithms

Greedy Algorithms

5/19/03

CSE 202 - Greedy Algorithms



Greedy Algorithms

Optimization problem: find the best way to do something.

- E.g. match up two strings (LCS problem).

Search techniques look at many possible solutions.

- E.g. dynamic programming or backtrack search.

A greedy algorithm

- Makes choices along the way that seem the best.
- Sticks with those choices.

For some problems, greedy approach always gets optimum.

For others, greedy finds good, but not always best.

- If so, it's called a greedy heuristic or approximation.

For still others, greedy approach can do very poorly.

The problem of giving change

Vending machine has huge supply of quarters, dimes and nickels.

Customer needs N cents change (N is multiple of 5).

Machine wants to give out few coins as possible.

Greedy approach:

```
while (N > 0) {  
    give largest denomination coin  $\leq N$ ;  
    reduce N by value of that coin;  
}
```

Aside: Using division, it could make decisions faster.

Does this return the fewest number of coins?

More on giving change

Thm: Greedy algorithm always gives minimal # of coins.

Proof:

- Optimum has ≤ 2 dimes.
 - Quarter and nickel better than 3 dimes.
- Optimum has ≤ 1 nickel
 - Dime better than 2 nickels.
- Optimum doesn't have 2 dimes + 1 nickel
 - It would use quarter instead.
- So optimum & greedy have at most \$0.20 in non-quarters.
 - That is, they give the same number of quarters.
- Optimum & greedy give same on remaining $\leq \$0.20$ too.
 - Obviously.

More on giving change

Suppose we run out of nickels, put pennies in instead.

- Does greedy approach still give minimum number of coins?

Formally, the Coin Change problem is:

Given k denominations d_1, d_2, \dots, d_k and given N ,
find a way of writing $N = i_1 d_1 + i_2 d_2 + \dots + i_k d_k$ such
that $i_1 + i_2 + \dots + i_k$ is minimized.
"Size" of problem is k .

Is the greedy algorithm always a good heuristic?

That is, does there exist a constant c s.t. for all instances
of Coin Change, the greedy algorithm gives at most c times
the optimum number of coins?

How do we solve Coin Change exactly?

5

CSE 202 - Greedy Algorithms

Coin Change by Dynamic Programming

Let $C(N)$ = min # of coins needed to give N cents.

Detail: $C(0) = 0$ and, for $N < 0$, $C(N) = \infty$.

Optimal substructure: After an optimal solution gives
customer one coin, the remaining change must be
given optimally.

So $C(N) = 1 + \min \{ C(N-d_1), C(N-d_2), \dots, C(N-d_k) \}$


6

CSE 202 - Greedy Algorithms

Pennies, Dimes and Quarters

Recall: $C(N) = 1 + \min \{ C(N-d_1), C(N-d_2), \dots, C(N-d_k) \}$

0	5	10	15	20	25	30	35	40	45	50	55	60	65	70
0	5													

 $\min \{ 5+5, 0+1 \} = 1$

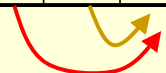
7

CSE 202 - Greedy Algorithms

Pennies, Dimes and Quarters

Recall: $C(N) = 1 + \min \{ C(N-d_1), C(N-d_2), \dots, C(N-d_k) \}$

0	5	10	15	20	25	30	35	40	45	50	55	60	65	70
0	5	1	6											

 $\min \{ 1+5, 5+1 \} = 6$

8

CSE 202 - Greedy Algorithms

Pennies, Dimes and Quarters

Recall: $C(N) = 1 + \min \{ C(N-d_1), C(N-d_2), \dots, C(N-d_k) \}$

0	5	10	15	20	25	30	35	40	45	50	55	60	65	70
0	5	1	6	2	1									

give 5 pennies
give dime
give quarter

Pennies, Dimes and Quarters

Recall: $C(N) = 1 + \min \{ C(N-d_1), C(N-d_2), \dots, C(N-d_k) \}$

0	5	10	15	20	25	30	35	40	45	50	55	60	65	70
0	5	1	6	2	1	?								

give 5 pennies
give dime
give quarter

Complexity of Coin Change

Greedy algorithm (non-optimal) takes $O(k)$ time.

Dynamic Programming takes $O(kN)$ time.

- This is NOT necessarily polynomial in k .
 - Better way to define "size" is the number of bits needed to specify an instance.
 - With this definition, N can be almost 2^{size} .
 - So Dynamic Programming is exponential in size.
- In fact, Coin Change problem is NP-hard.
 - So no one knows a polynomial-time algorithm for it.

Linear Partition Problem

Given a list of positive integers, s_1, s_2, \dots, s_N , and a bound B , find smallest number of *contiguous* sublists s.t. each sum of each sublist $\leq B$.

I.e.: find partition points $0 = p_0, p_1, p_2, \dots, p_k = N$ such that for $j = 0, 1, \dots, k-1$,

$$\sum_{i=p_j+1}^{p_{j+1}} s_i \leq B$$

Greedy algorithm:

Choose p_1 as large as possible.

Then choose p_2 as large as possible. Etc.

Greedy is optimal for linear partition

Thm: Given any valid partition $0 = q_0, q_1, \dots, q_k = N$,
then for all j , $q_j \leq p_j$. (The p_i 's are greedy solution.)

Proof: (by induction on k).

Base Case: $p_0 = q_0 = 0$ (by definition).

Inductive Step: Assume $q_j \leq p_j$.

We know $\sum_{i=q_{j+1}}^{q_{j+1}} s_i \leq B$ (since q 's are valid).

So $\sum_{i=p_{j+1}} s_i \leq B$ (since $q_j \leq p_j$).

So $q_{j+1} \leq p_{j+1}$ (since Greedy chooses p_{j+1} to be as large as possible subject to constraint on sum).

Variant on Linear Partitioning

New goal: partition list of N integers into exactly k contiguous sublists so that the maximum sum of a sublist is as small as possible.

Example: Partition $\langle 16, 7, 19, 3, 4, 11, 6 \rangle$ into 4 sublists.

- We might try $16+7, 19, 3+4, 11+6$. Max sum is $16+7=23$.

Try out (at board):

- Greedy algorithm: add elements until you exceed average.
- Divide-and-conquer: break into two nearly equal sublists.
- Reduce to previous problem: binary search on B .
- Dynamic programming.

Scheduling Unit time Tasks

Given N tasks (N is problem size):

- Task i must be done by time d_i .
- Task i is worth w_i .

You can perform one task per unit time. If you do it before its deadline d_i , you get paid w_i .

Problem: Decide what to do at each unit of time.

- *Aside: This is an off-line scheduling problem: You know entire problem before making any decisions.*
- *In an on-line problem, you get tasks one-at-a-time, and must decide when to schedule it before seeing next task.*
- *Typically, it's impossible to solve an on-line problem optimally, and the goal is to achieve at least a certain % of optimal.*

Glossary (in case symbols are weird)

$\subseteq \in \forall \exists \Theta \Omega \Sigma \geq \approx * \neq \mathbb{N} \mathbb{R} \mathbb{Q} \mathbb{Z} \lfloor \lceil \equiv \infty$

subset \in element of ∞ infinity

\forall for all \exists there exists

Θ big theta Ω big omega Σ summation

\geq $\succ=$ \leq $\prec=$ \approx about equal

\neq not equal \mathbb{N} natural numbers(\mathbb{N})

\mathbb{R} reals(\mathbb{R}) \mathbb{Q} rationals(\mathbb{Q}) \mathbb{Z} integers(\mathbb{Z})