DYNAMIC PROGRAMMING

Thursday, November 1, 2018 5:10 PM

- ✓ brute
- ✓ transform
- ✓ decrease
- ✓ divide
- ✓ greedy
- dynamic programming (DP)

<u>Idea</u>: - used to construct solutions to optimization problems step by step
- unlike greedy, which knows which move to make next, DP tries ALL possibilities
- uses caching to speed up the computation:
  it stores the answers to subproblems to avoid recomputations

<u>Ex</u>: Coin change problem (revisited)

Greedy doesn't work when denominations are 1, 5, 7, 10, 25

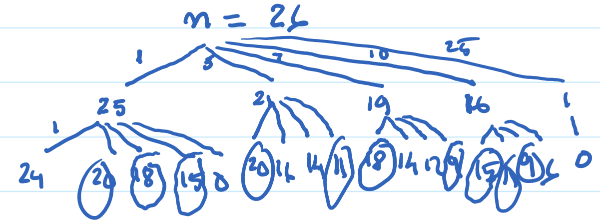
<u>Example</u>: $n = 14$

greedy: 10, 1, 1, 1, 1 optimal: 7, 7

Dynamic Programming

```
DP-coinchange (n)    // denominations = 1,5,7,10,25
{
              0 1 2 3  4
     d[] = { 1,5,7,10,25 };
     if (n==0)
          return { };
     ans.size = n+1;
     for (i=0; i<5; ++i)
     {
         if (n-d[i] >= 0)
              temp = {d[i]} U DP-coinchange(n-d[i]);
         if (temp.size() < ans.size())
                  ans = temp;
     }
     return ans;
}
```

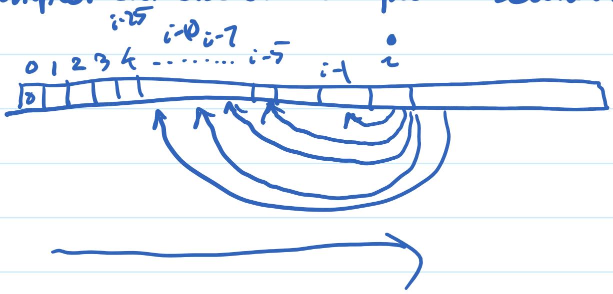
This implementation runs very slowly



because multiple recursive calls are made for the same value (20, 18, 11, 9, ⋯)

OBVIOUS IMPROVEMENT. store the subsolutions to avoid duplication.

Bottom-up implementation: compute solutions for 1, 2, 3, ----, n




```
DP-coinchange (n)
{ d[] = {1,5,7,10,25};
  S[0] = 0;
  for (i=1; i≤n; ++i)
  {    S[i] = ∞;
       for (j = 0; j < 5; ++j)
       {   if (i ≥ d[j])
              S[i] = min (S[i], 1 + S[i-d[j]]);
       }
  }
  return S[n];
}
```

$n \times 5 = 5 \times n$

1+S[8]=3
1+S[4]=5
1+S[2]=3

Ex: n = 14

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S	0	~~∞~~	∞	∞	∞	∞	∞	~~∞~~	2	3	1	2	2	3	2

1+S[13] = 4
1+S[9] = 4
1+S[7] = 2
1+S[4] = 5

1+S[0]
1+0=1

1+S[1]
1+1
2

1+S[2]
3

1+S[3]
4

~~1+4~~
1+0

1+S[5]
2

1+S[1]=2

1+S[6]
1+S[2]
~~3~~
1+S[0]=0

1+S[7]=2
1+S[3]=4
1+S[1]=2

1+S[9] = 4
1+S[5] = 2
1+S[3] = 4
1+S[0] = 1

1+S[10] = 2
1+S[6] = 3
1+S[4] = 5
1+S[1] = 2

1+S[11] = 3
1+S[7] = 2
1+S[5] = 2
1+d[2] = 3

1+S[12] = 3
1+S[8] = 3
1+S[6] = 3
1+S[3] = 4

```
DP-CC (n, d[1..k])
{
    S[0] = 0;
    for (i=1; i≤n; ++i)
    {
        S[i] = 0;
        for (j = 1; j ≤ k; ++j)
            if (i ≥ d[j])
                S[i] = min( S[i], 1 + S[i-d[j]] );
    }
    return S[n];
}
```

Running time: $\Theta(kn)$

Subsolutions: $S[i]$ = best solution for for amount i

MOVIE PROBLEM (revisited)

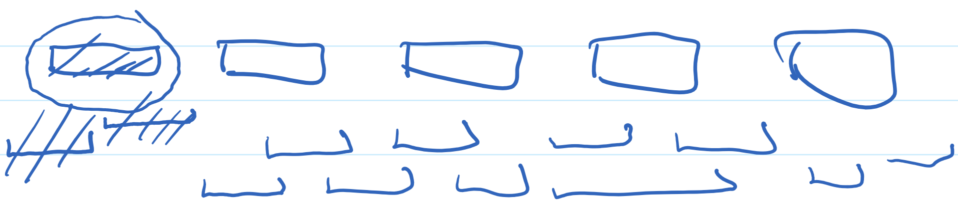
Input: a set of movies, each movie is an interval 

Output: a set of nonoverlapping movies whose total rating is maximized

Ex: 

No known greedy solutions, but there is a DP solution

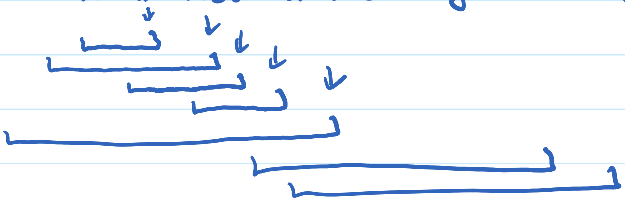
Idea: solution can be constructed one movie at a time.
What should be the first movie? Try them all
Regardless of what we pick for our first movie, the remaining must be optimal for the set of movies that do not overlap with our choice.



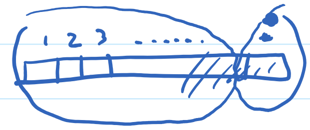
Efficient Implementation:
Sort the movies in increasing FINISH TIME

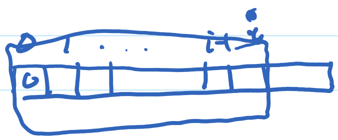
Efficient Implementation :

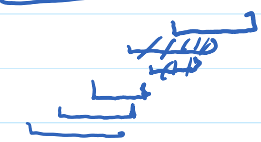
Sort the movies in increasing FINISH TIME



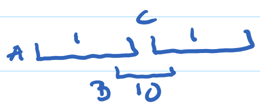
```
MOVIE ( M[1..n] ) // each movie M[i] has 3 fields : start, finish, rating
{   S[0] = 0;  M[0] = {-∞,-∞,0};
Θ(nlgn)  merge-sort ( M[1..n] ) ; // in increasing finish time
    for (i = 1; i ≤ n; ++i)
    {
        S[i] = S[i-1]
O(n^2)  for (j = i-1;  M[j].finish > M[i].start; --j )
            ;
        S[i] = max ( S[i], M[i].rating + S[j] ;
    }
    return S[n];
}
```

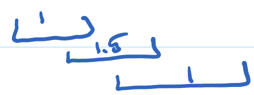


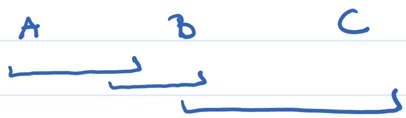




Ex :







	0	1	2	3
S	0	~~0~~ 1	~~1~~ 10	10 ~~1+1=2~~

	0	1	2	3
S	0	1	~~1~~ 1.5+0	~~1.5~~ 1+1=2

Subsolutions : $S[i]$ = best solution for $M[1..i]$ (SORTED in increasing finish time)

MATRIX CHAIN MULTIPLICATION

COIN CHANGE (re-revisited)

INPUT : amount n, denominations $d[1..k]$, available$[1..k]$

OUTPUT: size of the smallest set of coins in the given denominations whose sum is n and the number of coins in denom $d[i] \le$ available$[i]$

Ex: $n = 14$ $\quad d = [1, 5, 7, 10, 25]$

$a = [4, 2, 1, 1, 2]$ $\quad$ best $= 1 + 1 + 5 + 7$

$a = [1, 2, 1, 1, 2]$ $\quad$ best $=$ no solutions

Subproblems $\quad S[j][i]$ = best ans for amount i using the first j denominations

Base case: $j = 0$ (allowed no denominations)

$n = 1$

$d[] = \{1, 5\}$

$a[] = \{4, 2\}$

0 pennies

~~1~~

~~2~~

~~3~~

~~4~~

$n = 14$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
no denomination	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
one denomination	0	1	[illegible]	3	4	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$d[1] = 1$, $a[1] = 4$	0	1	2	3	4	1	2	3	4	5	2	3	4	5	6
	0														
	0														
	0														
	0														

Limited-CC $(n, d[1..k], a[1..k])$ $\quad$ d: denominations; a: availability

```
{
    for (i = 1; i <= n; ++i)
        S[0][i] = ∞;        // no solutions if no denominations are allowed
    for (j = 0; j <= k; ++j)
        S[j][0] = 0;
    for (j = 1; j <= k; ++j)    // adding more denomination
        for (i = 1; i <= n; ++i)
        {
            S[j][i] = ∞;
            for (m = 0; m <= a[j]; ++m)          <-- if (i >= m*d[j])
                S[j][i] = min(S[j][i], m + S[j-1][i - m*d[j]])
        }
    return S[k][n];
}
```