Dynamic Programming

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Objectives

Your Objectives:

- Explain the difference between dynamic programming and greedy algorithm
- Be able to use the top down and the bottom up approach
- Use the *memory saving trick* for the bottom up approach
- Solve the Fibonacci Sequence
- Solve UVA 11450 - Wedding Shopping

When Greediness Fails

- Greedy Algorithms have:
  - Optimal sub-structures
  - The Greedy Property
- Dynamic Programming Algorithms:
  - Optimal sub-structures, but **overlapping**
  - The Greedy Property does not hold!

A Very Simple Example

$$f_1 = 1$$
$$f_2 = 1$$
$$f_n = f_{n-1} + f_{n=2}$$

- Elegant definition, but terrible computationally!
- $f_5 = f_4 + f_3 = (f_3 + f_2) + (f_2 + f_1) = ((f_2 + f_1) + f_2) + (f_2 + f_1)$
- But... what if we could *remember* what we tried to compute before?

The Näive Solution

```
0 long long int fib(long long int i) {
1     if (i<=2)
2       return 1;
3     else
4       return fib(i-1) + fib(i-2);
5 }
6
7 int main() {
8     printf("f_50 = %d\n",fib(50));
9 }
```

When we run this....

```
% time ./a.out
f_50 = 12586269025
./a.out  52.18s user 0.01s system 99% cpu 52.279 total
```

The DP Solution (Top Down)

```
0 long long memo[100];
1 long long int fib(long long int i) {
2   if (memo[i] > -1) return memo[i];
3   if (i<=2) return 1;
4     else return memo[i] = fib(i-1) + fib(i-2);
5 }
6
7 int main() {
8   memset(memo,-1,sizeof memo);  // in cstring
9   printf("f_50 = %lld\n",fib(50));
10 }
```

```
% time ./a.out
f_50 = 12586269025
./a.out  0.00s user 0.00s system 84% cpu 0.002 total
```

The DP Solution (Bottom Up)

```
0 void init() {
1   memo[1] = 1;    memo[2] = 1;
2   for(int i=3; i<100; i++)
3       memo[i] = memo[i-1] + memo[i-2];
4 }
5
6 long long int fib(long long int i) {
7   return memo[i];
8 }
9
10 int main() {
11      init();
12      printf("f_50 = %lld\n",fib(50));
13 }
```

Saving Space

- If you do not need the previous rows of the table, you can use this trick:

```
0 long long int fib(int n, long long int i, long long int j) {
1   if (n==1)
2       return i;
3   else
4       return fib(n-1,j,i+j);
5 }
6
7 int main() {
8   printf("f_50 = %lld\n",fib(50,1,1));
9 }
```

The problem

- You have M dollars to spend, and want to maximize your spending.
- You have to select 1 article of clothing from each of $1 \leq C \leq 20$ categories.
- Each category has $1 \leq K_C \leq 20$ choices of different pricing.

Try to make a näive recursive enumeration version of the solution!

The Recursive Version

```
0 int costs[21][21];
1 int N,M,C;
2
3 int shop(int money, int garment) {
4    int ans;
5
6    if (money<0) return -10000000;
7    if (garment == C) return M - money;
8
9    ans = -1;
10   for(int model=1; model <= costs[garment][0]; ++model)
11      ans = max(ans, shop(money - costs[garment][model], garment+1));
12   return ans;
13 }
```

The Memoized Version

```
0 int memo[21][201];
1
2 int shop(int money, int garment) {
3
4    if (money<0) return -10000000;
5    if (garment == C) return M - money;
6
7    int & ans = memo[garment][money];
8
9    if (ans > -1) return ans;
10
11   for(int model=1; model <= costs[garment][0]; ++model)
12      ans = max(ans, shop(money - costs[garment][model], garment+1));
13
14   return ans;
```

Returning the Decisions

```
0 void print_shop(int money, int g) {
1     if (money < 0 || g == C) return;
2     for (int model = 1; model <= costs[g][0]; model++) // which model?
3         if (shop(money - price[g][model], g + 1) == memo[g][money]) {
4             printf("%d%c", price[g][model], g == C-1 ? '\n' : '-');
5         print_shop(money - price[g][model], g + 1);
6         break;
7     }
8 }
```

Bottom Up Style

```
0 for (g = 1; g <= price[0][0]; g++)
1    if (M - price[0][g] >= 0)
2       reachable[0][M - price[0][g]] = true;
3 for (g = 1; g < C; g++)
4    for (money = 0; money < M; money++)
5       if (reachable[g-1][money])
6          for (k = 1; k <= price[g][0]; k++)
7             if (money - price[g][k] >= 0)
8                reachable[g][money - price[g][k]] = true;
9 for (money = 0; money <= M && !reachable[C - 1][money]; money++);
10 if (money == M + 1) printf("no solution\n");
11 else
12    printf("%d\n", M - money);
```

Bottom Up Style: Saving Space

```
for (g = 1; g <= price[0][0]; g++)
   if (M - price[0][g] >= 0)
      reachable[0][M - price[0][g]] = true;
cur = 1; prev = 0;
for (g = 1; g < C; cur = 1-cur, prev=1-prev, g++)
   for (money = 0; money < M; money++)
      if (reachable[prev][money])
         for (k = 1; k <= price[cur][0]; k++)
            if (money - price[cur][k] >= 0)
               reachable[cur][money - price[cur][k]] = true;
for (money = 0; money <= M && !reachable[cur][money]; money++);
if (money == M + 1) printf("no solution\n");
   else
      printf("%d\n", M - money);
```