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

$$
\begin{aligned}
f_1 &= 1 \\
f_2 &= 1 \\
f_n &= f_{n-1} + f_{n=2}
\end{aligned}
$$

- 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

```
long long int fib(long long int i) {
    if (i<=2)
      return 1;
    else
      return fib(i-1) + fib(i-2);
}

int main() {
    printf("f_50 = %d\n",fib(50));
}
```

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)

```
long long memo[100];
long long int fib(long long int i) {
  if (memo[i] > -1) return memo[i];
  if (i<=2) return 1;
    else return memo[i] = fib(i-1) + fib(i-2);
}

int main() {
  memset(memo,-1,sizeof memo);  // in cstring
  printf("f_50 = %lld\n",fib(50));
}
```

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

The DP Solution (Bottom Up)

```
void init() {
  memo[1] = 1;   memo[2] = 1;
  for(int i=3; i<100; i++)
     memo[i] = memo[i-1] + memo[i-2];
}

long long int fib(long long int i) {
  return memo[i];
}

int main() {
    init();
    printf("f_50 = %lld\n",fib(50));
}
```

Saving Space

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

```
long long int fib(int n, long long int i, long long int j) {
  if (n==1)
     return i;
  else
     return fib(n-1,j,i+j);
}

int main() {
  printf("f_50 = %lld\n",fib(50,1,1));
}
```

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

```
int costs[21][21];
int N,M,C;

int shop(int money, int garment) {
   int ans;

   if (money<0) return -10000000;
   if (garment == C) return M - money;

   ans = -1;
   for(int model=1; model <= costs[garment][0]; ++model)
      ans = max(ans, shop(money - costs[garment][model], garment+1));
   return ans;
}
```

The Memoized Version

```
int memo[21][201];

int shop(int money, int garment) {

   if (money<0) return -10000000;
   if (garment == C) return M - money;

   int & ans = memo[garment][money];

   if (ans > -1) return ans;

   for(int model=1; model <= costs[garment][0]; ++model)
      ans = max(ans, shop(money - costs[garment][model], garment+1));

   return ans;
```

Returning the Decisions

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

Bottom Up Style

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