Dynamic Programming and Memoization

Workshop in Competitive Programming – 234901

When using recursion and
subproblems overlap,
avoid recomputation
by using DP or memoization.

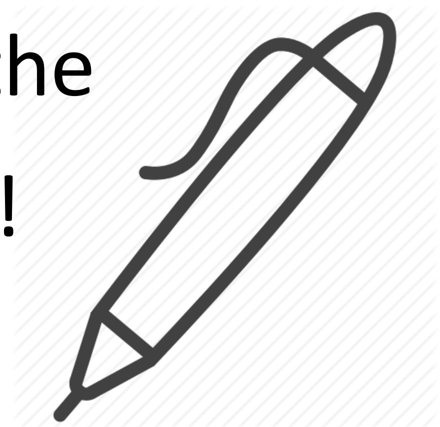
Bottom-up recursion

Storing and reusing
subproblem solutions

Be organized

- DP can get confusing.
  - It is easy to forget the big picture when working on the details.
  - Sometimes there is more than one way of defining the recursion. It is easy to mix them up.

Using a pen and paper to define everything clearly before writing the code makes a **huge** difference!



Steps for DP

1. Define the subproblem
   (the function meaning and parameters)
2. Find the recursive rule
3. Solve base cases
4. Define the target value
   (which value do you need to solve the problem)
5. Define the computation order
   (e.g. ascending order of n)
6. Start coding

definition
recursion
base
target
order
code

Let’s solve problems!

Problem: LIS

Given an array,
find a longest increasing subsequence.

Example: $\boxed{-7}$, 10, 9, $\boxed{2, 3, 8}$, 8, 1.

ans = 4

definition
recursion
target
base
order

Problem: LIS

Given an array,
find a longest increasing subsequence.

$s[i]$ = the length of the longest increasing subsequence ending in i

-----

$$s[i] = \max(1, \max_{0 \le j \le i-1, a[j] < a[i]} (\mathrm{s[j]} + 1))$$

$$ans = \max_{0 \le i < n} s[i], 0$$

$$s[0] = 1$$

Time $O(n^2)$

definition
recursion
target
base
order

Problem: LIS

$$s[i] = \max(1, \max_{0 \le j < i, a[j] < a[i]} (\mathrm{s[j]} + 1))$$
$$ans = \max_{0 \le i < n} s[i], 0$$
$$s[0] = 1$$

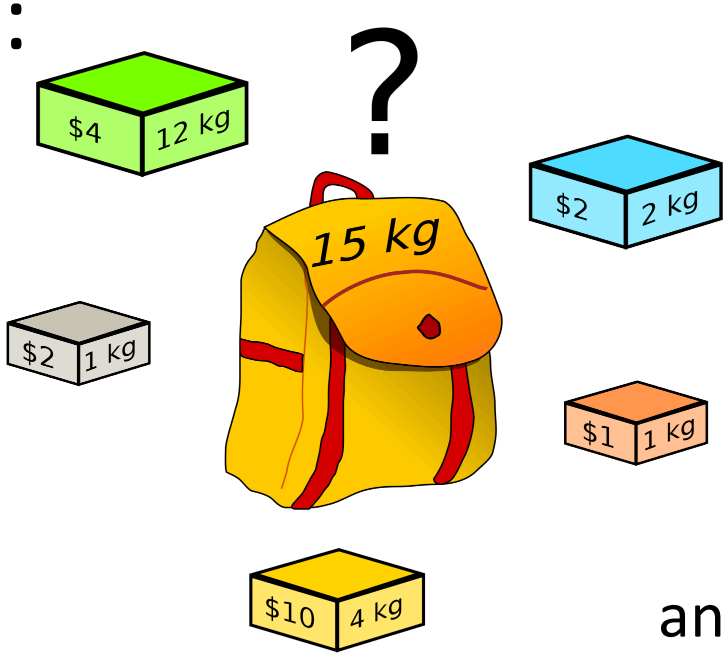
```
vector<int> LIS(n,1);
for (int i=1; i<n; i++)
  for (int j=0; j<i; j++)
    if (a[i]>a[j])
      LIS[i] = max(LIS[i], LIS[j]+1);
```

definition
recursion
target
base
order

Problem: 0-1 Knapsack

Given a capacity C and n items with weights and values, find the max value that fits.

Example:



ans = $15

definition
recursion
target
base
order

Problem: 0-1 Knapsack

Given a capacity C and n items with weights and values, find the max value that fits.

$f(i, c)$ = max possible value within capacity c when only considering the first i items.

$$ans = f(n, C)$$

$$f(i,c) = \begin{cases} f(i-1, c), & w[i] > c \\ \max\left\{\begin{matrix} f(i-1,c) \\ v[i] + f(i-1, c-w[i]) \end{matrix}\right\}, & w[i] \leq c \end{cases}$$

$$f(i, 0) = f(0, c) = 0$$

Time $O(nC)$

definition
recursion
target
base
order

Problem: 0-1 Knapsack

$$ans = f(n, C)$$

$$f(i,c) = \begin{cases} f(i-1,c), & w[i] > c \\ \max\left\{\begin{matrix} f(i-1,c) \\ v[i] + f(i-1, c-w[i]) \end{matrix}\right\}, & w[i] \le c \end{cases}$$

$$f(i,0) = f(0,c) = 0$$

```
vvi f(n+1, vi(C+1,0));
for (int i=1; i<=n; i++){
  for (int c=1; c<=C; c++){
    if (w[i]<=c)
      f[i][c] = max(f[i-1][c],v[i]+f[i-1][c-w[i]]);
    else
      f[i][c] = f[i-1][c];
  }
}
int ans = f[n][C];
```

definition
recursion
target
base
order

Improvements

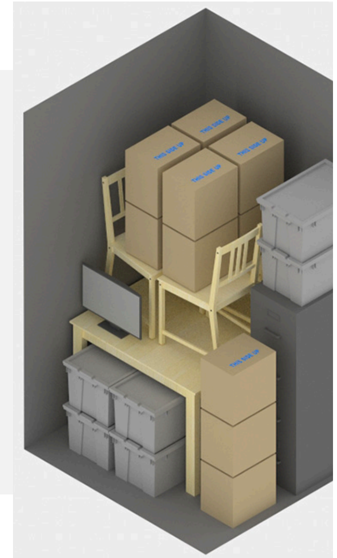
```
vvi f(n+1, vi(C+1,0));
for (int i=1; i<=n; i++){
  for (int c=1; c<=C; c++){
    if (w[i]<=c)
      f[i][c] = max(f[i-1][c],v[i]+f[i-1][c-w[i]]);
    else
      f[i][c] = f[i-1][c];
  }
}
int ans = f[n][C];
```

Accepted? Good!

TLE? Try to save time.

MLE? Try to save space.

Improvements: Space



```
vvi f(n+1, vi(C+1,0));
for (int i=1; i<=n; i++) {
  for (int c=1; c<=C; c++) {
    if (w[i]<=c)
      f[i][c] = max(f[i-1][c],v[i]+f[i-1][c-w[i]]);
    else
      f[i][c] = f[i-1][c]; }}
```

Observation: $f(i, ?)$ depends only on $f(i-1, ?)$

Don't store all of the previous i values!

```
vi f_cur(C+1,0); vi f_prev(C+1,0);
for (int i=1; i<=n; i++) {
  for (int c=1; c<=C; c++) {
    if (w[i]<=c)
      f_cur[c] = max(f_prev[c],v[i]+f_prev[c-w[i]]);
    else
      f_cur[i][c] = f_prev[i-1][c]; }
  f_prev = f_cur; f_cur.assign(C+1,0); }
```

Improvements: Time

```
vvi f(n+1, vi(C+1,0));
for (int i=1; i<=n; i++) {
  for (int c=1; c<=C; c++) {
    if (w[i]<=c)
      f[i][c] = max(f[i-1][c],v[i]+f[i-1][c-w[i]]);
    else
      f[i][c] = f[i-1][c]; }}
```

Observation: many states are not useful

Don’t compute all of them!

```
vvi memo(n+1, vi(C+1,-1));

int f(int i, int c) {
  if (memo[i][c] != -1) return memo[i][c];
  if (i == 0 || c == 0) return 0;
  if (w[i]<=c)
    return memo[i][c] = max(f(i-1,c), v[i]+f(i-1,c-w[i]));
  return memo[i][c] = f(i-1,c);
}
```

DP vs. memoization

```
int C = 15;
int n = 3;
vi v = {0,4,2,10};
vi w = {0,12,1,4};
```

DP

c

i

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
0	2	2	2	2	2	2	2	2	2	2	2	4	6	6	6
0	2	2	2	10	12	12	12	12	12	12	12	12	12	12	12

Memoization

c

i

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
0	2	2	2	2	2	2	2	2	2	2	2	2	4	6	6
0	2	2	2	10	12	12	12	12	12	12	12	12	12	12	12

DP vs. memoization

- Many times both work
- Sometimes one is more intuitive
- Sometimes one is better

	DP	memoization
direction	Bottom-up	Top-down
computation	Comprehensive (every subproblem is computed exactly once)	On-demand (subproblems are computed when required)
Pros	• Faster if many sub-problems are revisited (no overhead from recursive calls to calculate the same value) • Can save memory if each column only depends on the previous one (only save the last column, not the entire table)	• Faster if many sub-problems are not required (no need to fill in the entire table) • Easier to program when a good order is not obvious

Problem: Subset Sum

Given a number v and a set of numbers S,
is there a subset of S with sum equal to v?

Example: $v = 11, S = \{\boxed{2,}4,7,8\boxed{,9}\}$

ans = yes

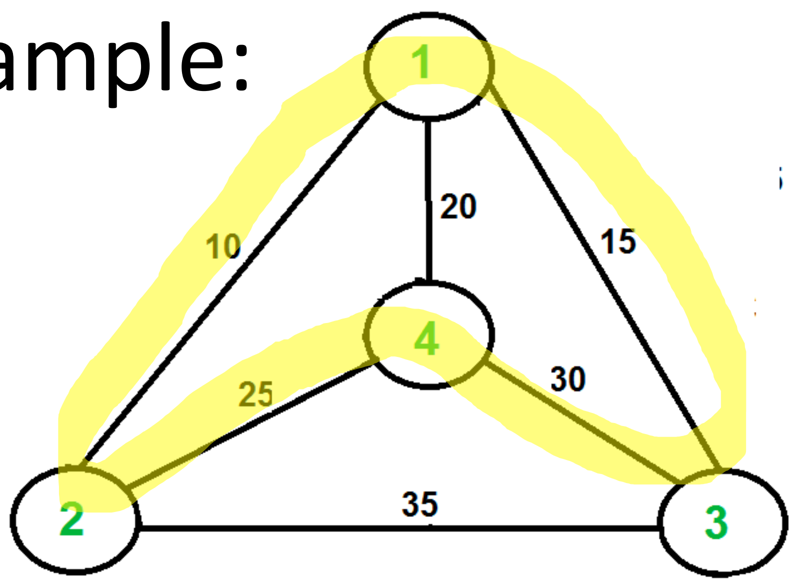
This is considered a variation on 0-1 Knapsack.

Problem: Traveling Salesman

Given a full weighted graph, find the min weight of a Hamiltonian cycle.

Given the distances between cities, find the min time for going to each city once and back.

Example:



ans = 80

definition
recursion
target
base
order

Problem: Traveling Salesman

Given a full weighted graph, find the min weight of a Hamiltonian cycle.

$f(v, visited)$ = the cost of going from v to 0 through all nodes that are not in $visited$

$$ans = f(0, \{0\})$$

$$f(v, visited) = \min_{u \notin visited} (g[v][u] + f(u, visited \cup \{u\})$$

$$f(v, \{1, \dots, n\}) = g[v][0]$$

Time $O(2^n n^2)$

definition
recursion
target
base
order

Reminder: Bitmasks

- Use an int to represent subsets:

$$i\text{th digit} = \begin{cases} 1, i \notin S \\ 0, i \in S \end{cases}$$

- Example: the set $\{0,1,4\}$ is represented by $19 = 010011_{(2)}$

Operations:

- $1 \ll i$ is the set $\{i\}$
- $(1 \ll i) - 1$ is the set $\{0,1, \dots, i-1\}$
- $x \mid y$ is the union $x \cup y$
- $x \mid (1 \ll i)$ is the set $x \cup \{i\}$
- $x \,\&\, y$ is the intersection $x \cap y$
- $x \,\&\, (1 \ll i))$ tests membership of i in x

Problem: Traveling Salesman

$$ans = f(0, \{0\})$$
$$f(v, visited) = \min_{u \notin visited} (g[v][u] + f(u, visited \cup \{u\})$$
$$f(v, \{1, \dots, n\}) = g[v][0]$$

```
vvi memo(n, vi(C+1,-1));

int f(int v, int visited) {
  if (memo[v][visited] != -1) return memo[v][visited];
  if (visited == (1<<n)-1) return g[v][0];
  int ans = INF;
  for (int u = 0; u < n; u++)
    if (!(visited & (1<<u)))
      ans = min(ans, g[v][u] + f(u,visited|(1<<u)));
  return memo[v][visited] = ans;
}

int ans = f(0,1);
```

definition
recursion
target
base
order

Extra Slides

Problem: max range sum

Given an array, find the max range sum.

Example: 4, -5, 4, -3, 4, 4, -4, 4, -5.

ans = 9

definition
recursion
target
base
order

Problem: max range sum

Given an array, find the max range sum.

$s[i]$ = the max range sum ending in (and including) index i

$$s[i] = \max(a[i], s[i-1] + a[i])$$

$$ans = \max\left(\max_i s[i], 0\right)$$

$$s[0] = a[0]$$

Time $O(n)$

definition
recursion
target
base
order

Problem: max range sum

$$s[i] = \max(a[i], s[i-1] + a[i])$$

$$ans = \max\left(\max_i s[i], 0\right)$$

$$s[0] = a[0]$$

```
vector<int> s(N, 0);
s[0]=a[0];
for (int i=1; i<N; i++)
    s[i] = max(a[i], s[i-1]+a[i]);
int ans = 0;
for (int i=0; i<N; i++)
    ans = max(ans, s[i]);
```

definition
recursion
target
base
order

Problem: Coin Change

Given a target V and coin values $c_1, \ldots, c_n$, find the min number of coins to form V.

Example: $V = 6, coins = \{4,3,1\}$

3+3=6 ans = 2

definition
recursion
target
base
order

Problem: Coin Change

Given a target V and coin values $c_1, \ldots, c_n$, find the min number of coins to form V.

$s[i]$= the number of coins needed for target i

$$ans = s[V]$$

$$s(i) = 1 + \min_{c \in coins} s(i - c)$$

$$s(0) = 0,\ s(< 0) = \infty$$

Time $O(Vn)$

definition
recursion
target
base
order

Problem: Coin Change

$$ans = s[V]$$
$$s(i) = 1 + \min_{c \in coins} s(i - c)$$
$$s(0) = 0,\ s(< 0) = \infty$$

```
vector<int> s(V+1, INF);
s[0]=0;
for (int i=1; i<=V; i++)
    for (int c : coins)
        if (i-c>=0)
            s[i] = min(s[i],s[i-c]);
int ans = s[V];
```

definition
recursion
target
base
order