Announcements

- Homework 4 due today
- Homework 5 online due next week

Last Time

- Dynamic Programming
- Longest Common Subsequence

Dynamic Programming

Our final general algorithmic technique:

1. Break problem into smaller subproblems.
2. Find recursive formula solving one subproblem in terms of simpler ones.
3. Tabulate answers and solve all subproblems.

Recursion

$LCSS(A_1A_2 \ldots A_n, B_1B_2 \ldots B_m) =$

$Max(LCSS(A_1A_2 \ldots A_{n-1}, B_1B_2 \ldots B_m),$

$LCSS(A_1A_2 \ldots A_n, B_1B_2 \ldots B_{m-1}),$

$[LCSS(A_1A_2 \ldots A_{n-1}, B_1B_2 \ldots B_{m-1})+1])$

[where the last option is only allowed if $A_n = B_m$]

Algorithm

```
LCSS(A_1A_2…A_n,B_1B_2…B_m)
  Initialize Array T[0…n,0…m]
    \\ T[i,j] will store LCSS(A_1A_2…A_i,B_1B_2…B_j)
  For i = 0 to n
    For j = 0 to m
      If (i = 0) OR (j = 0)
        T[i,j] ← 0
      Else If A_i = B_j
        T[i,j] ← max(T[i-1,j],T[i,j-1],T[i-1,j-1]+1)
      Else
        T[i,j] ← max(T[i-1,j],T[i,j-1])
  Return T[n,m]
```

O(nm) iterations

O(1)

Today

- Notes about design and analysis of dynamic programs
- Knapsack

Notes about DP

- General Correct Proof Outline:
  - Prove by induction that each table entry is filled out correctly
  - Use base-case and recursion

Notes about DP

- General Correct Proof Outline:
  - Prove by induction that each table entry is filled out correctly
  - Use base-case and recursion
- Runtime of DP:
  - Usually [Number of subproblems]x[Time per subproblem]

More Notes about DP

- Finding Recursion
  - Often look at first or last choice and see what things look like without that choice

More Notes about DP

- Finding Recursion
  - Often look at first or last choice and see what things look like without that choice
- Key point: Picking right subproblem
  - Enough information stored to allow recursion
  - Not too many

Problem: Knapsack

You are a burglar and are in the process of robbing a home. You have found several valuable items, but the sack you brought can only hold so much weight, what is the best combination of items to steal?

Problem: Knapsack

You are a burglar and are in the process of robbing a home. You have found several valuable items, but the sack you brought can only hold so much weight, what is the best combination of items to steal?

Alternative formulations:

- Packing for a trip
- Deciding what modules to put on a spacecraft

Specification

You have an available list of items. Each has a (non-negative integer) weight, and value. Your sack also has a capacity.

Specification

You have an available list of items. Each has a (non-negative integer) weight, and value. Your sack also has a capacity.

The goal is to find the collection of items so that:

1. The total weight of all the items is less than the capacity
2. Subject to 1, the total value is as large as possible.

Variations

There are two slight variations of this problem:

1. Each item can be taken as many times as you want.
2. Each item can be taken at most once.

Question: Knapsack

Given the knapsack problem below (only one copy of each item), what is the best set of items to take?

Item	Weight	Value
A	1	\$ 1
B	2	\$ 4
C	3	\$ 3
D	4	\$ 5

Capacity:
6

Question: Knapsack

Given the knapsack problem below (only one copy of each item), what is the best set of items to take?

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:
6

BD
Weight = 6
Value = $9

Greedy Algorithms Don’t Work

Greedy Algorithms Don’t Work

Capacity = 6

Most valuable item

Item	Weight	Value
A	6	$ 10
B	3	$ 9
C	3	$ 9

Greedy:
A = $10

Optimal:
B+C = $18

Greedy Algorithms Don't Work

Capacity = 6

Most valuable item

Item	Weight	Value
A	6	$ 10
B	3	$ 9
C	3	$ 9

Greedy:
A = $10

Optimal:
B+C = $18

Biggest Value/Weight

Item	Weight	Value
A	4	$ 5
B	3	$ 3
C	3	$ 3

Greedy:
A = $5

Optimal:
B+C = $6

Subproblems (multiple copies version)

What are our subproblems?

Subproblems (multiple copies version)

What are our subproblems?

- If you make one choice of an item to go into the bag, what is left?

Subproblems (multiple copies version)

What are our subproblems?

- If you make one choice of an item to go into the bag, what is left?
  - Remaining items must have total weight at most Capacity – Weight of item

Subproblems (multiple copies version)

What are our subproblems?

- If you make one choice of an item to go into the bag, what is left?
  - Remaining items must have total weight at most Capacity – Weight of item
  - Total value equals
    Value of item + Value of other items

Subproblems (multiple copies version)

What are our subproblems?

- If you make one choice of an item to go into the bag, what is left?
  - Remaining items must have total weight at most Capacity – Weight of item
  - Total value equals
    Value of item + Value of other items
  - Want to maximize value of other items subject to their weight not exceeding Capacity-Weight of chosen item

Subproblems (multiple copies version)

What are our subproblems?

- If you make one choice of an item to go into the bag, what is left?
  - Remaining items must have total weight at most Capacity – Weight of item
  - Total value equals
    Value of item + Value of other items
  - Want to maximize value of other items subject to their weight not exceeding Capacity-Weight of chosen item
- Subproblem: BestValue(Capacity’).

Recursion

What is BestValue(C)?

Recursion

What is BestValue(C)?

Possibilities:

- No items in bag
  - Value = 0

Recursion

What is BestValue(C)?

Possibilities:

- No items in bag
  - Value = 0
- Item i in bag
  - Value = BestValue(C-weight(i)) + value(i)

Recursion

What is BestValue(C)?

Possibilities:

- No items in bag
  - Value = 0
- Item i in bag
  - Value = BestValue(C-weight(i)) + value(i)

<u>Recursion:</u> BestValue(C) =

$$\text{Max}(0, \text{Max}_{wt(i) \leq C} (\text{val}(i)+\text{BestValue}(C-\text{wt}(i))))$$

Algorithm

```
Knapsack(Wt,Val,Cap)
  Create Array T[0…Cap]
  For C = 0 to Cap
    T[C] ← 0
    For items i with Wt(i) ≤ C
      If T[C] < Val(i)+T[C-Wt(i)]
        T[C] ← Val(i)+T[C-Wt(i)]
  Return T[Cap]
```

Algorithm

```
Knapsack(Wt,Val,Cap)
  Create Array T[0…Cap]
  For C = 0 to Cap
    T[C] ← 0
    For items i with Wt(i) ≤ C
      If T[C] < Val(i)+T[C-Wt(i)]
        T[C] ← Val(i)+T[C-Wt(i)]
  Return T[Cap]
```

O(Cap)
Subproblems

Algorithm

```
Knapsack(Wt,Val,Cap)
   Create Array T[0…Cap]
   For C = 0 to Cap
      T[C] ← 0
      For items i with Wt(i) ≤ C
         If T[C] < Val(i)+T[C-Wt(i)]
            T[C] ← Val(i)+T[C-Wt(i)]
   Return T[Cap]
```

O(Cap) Subproblems

O(#items) time/subproblem

Algorithm

```
Knapsack(Wt,Val,Cap)
  Create Array T[0…Cap]
  For C = 0 to Cap
    T[C] ← 0
    For items i with Wt(i) ≤ C
      If T[C] < Val(i)+T[C-Wt(i)]
        T[C] ← Val(i)+T[C-Wt(i)]
  Return T[Cap]
```

O(Cap) Subproblems

O(#items) time/subproblem

Runtime: O([Cap] [#Items])

Example

Item	Weight	Value
A	1	\$ 1
B	2	\$ 4
C	3	\$ 3
D	4	\$ 5

Capacity:

6

C	0	1	2	3	4	5	6
BestValue							

Example

Item	Weight	Value
A	1	\$ 1
B	2	\$ 4
C	3	\$ 3
D	4	\$ 5

Capacity:

6

C	0	1	2	3	4	5	6
BestValue	\$0						

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:
6

C	0	1	2	3	4	5	6
BestValue	$0	$1					

$0 or $1+$0

Example

Item	Weight	Value
A	1	\$ 1
B	2	\$ 4
C	3	\$ 3
D	4	\$ 5

Capacity:
6

C	0	1	2	3	4	5	6
BestValue	\$0	\$1	\$4				

\$0 or \$1+\$1 or \$4+\$0

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:
6

C	0	1	2	3	4	5	6
BestValue	$0	$1	$4	$5			

$0 or $1+$4 or $4+$1 or $3+$0

Example

Item	Weight	Value
A	1	\$ 1
B	2	\$ 4
C	3	\$ 3
D	4	\$ 5

Capacity:
6

C	0	1	2	3	4	5	6
BestValue	\$0	\$1	\$4	\$5	\$8		

\$0 or \$1+\$5 or \$4+\$4 or \$3+\$1 or \$5+\$0

Example

Item	Weight	Value
A	1	\$ 1
B	2	\$ 4
C	3	\$ 3
D	4	\$ 5

Capacity:
6

C	0	1	2	3	4	5	6
BestValue	\$0	\$1	\$4	\$5	\$8	\$9	

\$0 or \$1+\$8 or \$4+\$5 or \$3+\$4 or \$5+\$1

Example

Item	Weight	Value
A	1	\$ 1
B	2	\$ 4
C	3	\$ 3
D	4	\$ 5

Capacity:

6

C	0	1	2	3	4	5	6
BestValue	\$0	\$1	\$4	\$5	\$8	\$9	\$12

\$0 or \$1+\$9 or \$4+\$8 or \$3+\$5 or \$5+\$4

Example

Item	Weight	Value
A	1	\$ 1
B	2	\$ 4
C	3	\$ 3
D	4	\$ 5

Capacity:
6

C	0	1	2	3	4	5	6
BestValue	\$0	\$1	\$4	\$5	\$8	\$9	\$12

B

\$0 or \$1+\$9 or \$4+\$8 or \$3+\$5 or \$5+\$4

Example

Item	Weight	Value
A	1	\$ 1
B	2	\$ 4
C	3	\$ 3
D	4	\$ 5

Capacity:

6

C	0	1	2	3	4	5	6
BestValue	\$0	\$1	\$4	\$5	\$8	\$9	\$12

B B

\$0 or \$1+\$5 or \$4+\$4 or \$3+\$1 or \$5+\$0

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:
6

C	0	1	2	3	4	5	6
BestValue	$0	$1	$4	$5	$8	$9	$12

B B B

$0 or $1+$1 or $4+$0

Example

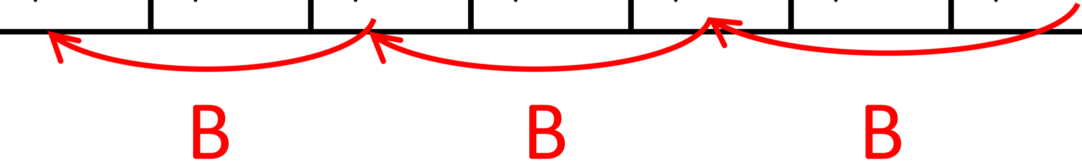
Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

B+B+B = $12

C	0	1	2	3	4	5	6
BestValue	$0	$1	$4	$5	$8	$9	$12



Non-Repeating Items

Let’s try this with non-repeating items.

Non-Repeating Items

Let’s try this with non-repeating items.

- If we put some item in the sack:
  - Other items must have total weight at most Capacity – Weight(chosen item)
  - Total value is value(other items)+value(chosen item)

Non-Repeating Items

Let's try this with non-repeating items.

- If we put some item in the sack:
  - Other items must have total weight at most Capacity – Weight(chosen item)
  - Total value is value(other items)+value(chosen item)
  - Chosen item cannot be picked again.

Non-Repeating Items

Let's try this with non-repeating items.

- If we put some item in the sack:
  - Other items must have total weight at most Capacity – Weight(chosen item)
  - Total value is value(other items)+value(chosen item)
  - Chosen item cannot be picked again.
- Recursion needs to keep track of remaining capacity and the item that cannot be used.

Attempt 1

Let's make subproblem

$\text{BestValue}_{\neq i}(\text{Cap})$ – the best value achievable without using item i that doesn't go over capacity.

Attempt 1

Let's make subproblem

$BestValue_{\neq i}(Cap)$ – the best value achievable without using item i that doesn't go over capacity.

Can we make a recursion with this?

Attempt 1

Let's make subproblem

$\text{BestValue}_{\neq i}(\text{Cap})$ – the best value achievable without using item i that doesn't go over capacity.

Can we make a recursion with this?

No!

Attempt 1

Let’s make subproblem

BestValue$_{\neq i}$(Cap) – the best value achievable without using item i that doesn’t go over capacity.

Can we make a recursion with this?

No!

After using item j, the remaining items cannot include i or j.

Attempt 2

BestValue excluding 2 items? No… recursive calls would need to exclude a 3rd and so on.

Attempt 2

BestValue excluding 2 items? No… recursive calls would need to exclude a 3rd and so on.

$\text{BestValue}_S(\text{Cap})$ – best value achievable using only items from S with total weight at most Cap.

Attempt 2

BestValue excluding 2 items? No… recursive calls would need to exclude a 3rd and so on.

$\text{BestValue}_S(\text{Cap})$ – best value achievable using only items from S with total weight at most Cap.

$BV_S(\text{Cap}) = \max_{i \in S}(\text{Val}(i) + BV_{S-i}(\text{Cap-Wt}(i)))$ [or 0]

Attempt 2

BestValue excluding 2 items? No… recursive calls would need to exclude a 3rd and so on.

$\text{BestValue}_S(\text{Cap})$ – best value achievable using only items from S with total weight at most Cap.

$BV_S(\text{Cap}) = \max_{i \in S}(\text{Val}(i) + BV_{S-i}(\text{Cap-Wt}(i)))$ [or 0]

We have a recursion!

Attempt 2

BestValue excluding 2 items? No… recursive calls would need to exclude a 3rd and so on.

$\text{BestValue}_S(\text{Cap})$ – best value achievable using only items from S with total weight at most Cap.

$BV_S(\text{Cap}) = \max_{i \in S}(\text{Val}(i) + BV_{S-i}(\text{Cap}-\text{Wt}(i)))$ [or 0]

We have a recursion!

Unfortunately, this is too slow. The number of subproblems is more than $2^{\#\text{items}}$.

Attempt 3

- Need to try something different.

Attempt 3

- Need to try something different.
- Imagine items coming along a conveyor belt. You decide one at a time whether or add to your sac.

Attempt 3

- Need to try something different.
- Imagine items coming along a conveyor belt. You decide one at a time whether or add to your sac.
- Last item: either add or don't.

Attempt 3

- Need to try something different.
- Imagine items coming along a conveyor belt. You decide one at a time whether or add to your sac.
- Last item: either add or don't.
  - Add: $\text{BestValue}_{\leq n-1}(\text{Cap}-\text{Wt}(n)) + \text{Val}(n)$

Attempt 3

- Need to try something different.
- Imagine items coming along a conveyor belt. You decide one at a time whether or add to your sac.
- Last item: either add or don't.
  - Add: $BestValue_{\leq n-1}(Cap-Wt(n)) + Val(n)$
  - Don't add: $BestValue_{\leq n-1}(Cap)$

Attempt 3

- Need to try something different.
- Imagine items coming along a conveyor belt. You decide one at a time whether or add to your sac.
- Last item: either add or don't.
  - Add: $\text{BestValue}_{\leq n-1}(\text{Cap-Wt}(n)) + \text{Val}(n)$
  - Don't add: $\text{BestValue}_{\leq n-1}(\text{Cap})$
- We only need subproblems of the form $\text{BestValue}_{\leq k}(\text{Cap})$.

Recursion

$\text{BestValue}_{\leq k}(\text{Cap})$ = Highest total value of items with total weight at most Cap using only items from the first k.

Recursion

$\text{BestValue}_{\leq k}(\text{Cap})$ = Highest total value of items with total weight at most Cap using only items from the first k.

<u>Base Case:</u> $\text{BestValue}_{\leq 0}(C) = 0$

Recursion

$\text{BestValue}_{\leq k}(\text{Cap})$ = Highest total value of items with total weight at most Cap using only items from the first k.

Base Case: $\text{BestValue}_{\leq 0}(C) = 0$

Recursion: $\text{BestValue}_{\leq k}(C)$ is the maximum of

1. $\text{BestValue}_{\leq k-1}(C)$

Recursion

$BestValue_{\leq k}(Cap)$ = Highest total value of items with total weight at most Cap using only items from the first k.

Base Case: $BestValue_{\leq 0}(C) = 0$

Recursion: $BestValue_{\leq k}(C)$ is the maximum of

1. $BestValue_{\leq k-1}(C)$
2. $BestValue_{\leq k-1}(C - Wt(k)) + Val(k)$
   [where this is only used if $Wt(k) \leq Cap$]

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
∅							
A							
AB							
ABC							
ABCD							

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:
6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A							
AB							
ABC							
ABCD							

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0						
AB							
ABC							
ABCD							

Example

Item	Weight	Value
A	1	\$ 1
B	2	\$ 4
C	3	\$ 3
D	4	\$ 5

Capacity:
6

Cap	0	1	2	3	4	5	6
Ø	\$0	\$0	\$0	\$0	\$0	\$0	\$0
A	\$0	\$1					
AB							
ABC							
ABCD							

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:
6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB							
ABC							
ABCD							

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0						
ABC							
ABCD							

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1					
ABC							
ABCD							

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4				
ABC							
ABCD							

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:
6

Cap	0	1	2	3	4	5	6
∅	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5			
ABC							
ABCD							

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

<u>Capacity:</u>

6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC							
ABCD							

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC	$0						
ABCD							

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:
6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC	$0	$1					
ABCD							

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC	$0	$1	$4				
ABCD							

Example

Item	Weight	Value
A	1	\$ 1
B	2	\$ 4
C	3	\$ 3
D	4	\$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
∅	\$0	\$0	\$0	\$0	\$0	\$0	\$0
A	\$0	\$1	\$1	\$1	\$1	\$1	\$1
AB	\$0	\$1	\$4	\$5	\$5	\$5	\$5
ABC	\$0	\$1	\$4	\$5			
ABCD							

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:
6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC	$0	$1	$4	$5	$5		
ABCD							

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC	$0	$1	$4	$5	$5	$7	
ABCD							

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC	$0	$1	$4	$5	$5	$7	$8
ABCD							

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC	$0	$1	$4	$5	$5	$7	$8
ABCD	$0						

Example

Item	Weight	Value
A	1	\$ 1
B	2	\$ 4
C	3	\$ 3
D	4	\$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
Ø	\$0	\$0	\$0	\$0	\$0	\$0	\$0
A	\$0	\$1	\$1	\$1	\$1	\$1	\$1
AB	\$0	\$1	\$4	\$5	\$5	\$5	\$5
ABC	\$0	\$1	\$4	\$5	\$5	\$7	\$8
ABCD	\$0	\$1					

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC	$0	$1	$4	$5	$5	$7	$8
ABCD	$0	$1	$4				

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC	$0	$1	$4	$5	$5	$7	$8
ABCD	$0	$1	$4	$5			

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:
6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC	$0	$1	$4	$5	$5	$7	$8
ABCD	$0	$1	$4	$5	$5		

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC	$0	$1	$4	$5	$5	$7	$8
ABCD	$0	$1	$4	$5	$5	$7	

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC	$0	$1	$4	$5	$5	$7	$8
ABCD	$0	$1	$4	$5	$5	$7	$9

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC	$0	$1	$4	$5	$5	$7	$8
ABCD	$0	$1	$4	$5	$5	$7	$9

D

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC	$0	$1	$4	$5	$5	$7	$8
ABCD	$0	$1	$4	$5	$5	$7	$9

D

Example

Item	Weight	Value
A	1	\$ 1
B	2	\$ 4
C	3	\$ 3
D	4	\$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
∅	\$0	\$0	\$0	\$0	\$0	\$0	\$0
A	\$0	\$1	\$1	\$1	\$1	\$1	\$1
AB	\$0	\$1	\$4	\$5	\$5	\$5	\$5
ABC	\$0	\$1	\$4	\$5	\$5	\$7	\$8
ABCD	\$0	\$1	\$4	\$5	\$5	\$7	\$9

B

D

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

Cap	0	1	2	3	4	5	6
∅	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC	$0	$1	$4	$5	$5	$7	$8
ABCD	$0	$1	$4	$5	$5	$7	$9

B

D

Example

Item	Weight	Value
A	1	$ 1
B	2	$ 4
C	3	$ 3
D	4	$ 5

Capacity:

6

B+D = $9

Cap	0	1	2	3	4	5	6
Ø	$0	$0	$0	$0	$0	$0	$0
A	$0	$1	$1	$1	$1	$1	$1
AB	$0	$1	$4	$5	$5	$5	$5
ABC	$0	$1	$4	$5	$5	$7	$8
ABCD	$0	$1	$4	$5	$5	$7	$9

B

D

Runtime

Runtime

- Number of Subproblems: O([Cap] [#items])

Runtime

- Number of Subproblems: O([Cap] [#items])
- Time per subproblem O(1)
  - Only need to compare two options.

Runtime

- Number of Subproblems: O([Cap] [#items])
- Time per subproblem O(1)
  - Only need to compare two options.
- Final runtime O([Cap][#items]).