BLG 336E – Analysis of Algorithms II
Recitation IV

Ezgi Yıldırım

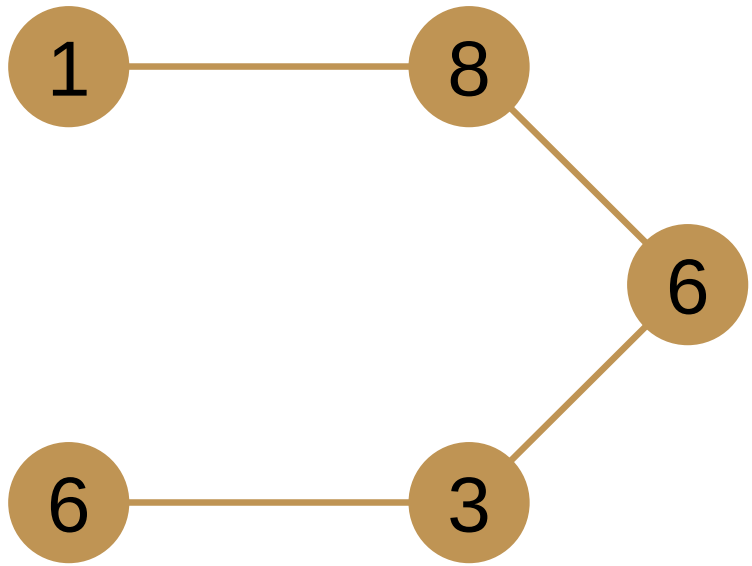
Istanbul Technical University – Department of Computer Engineering

12.04.2016

- Find an independent set in a path G whose total weight is as large as possible.

Independent Set

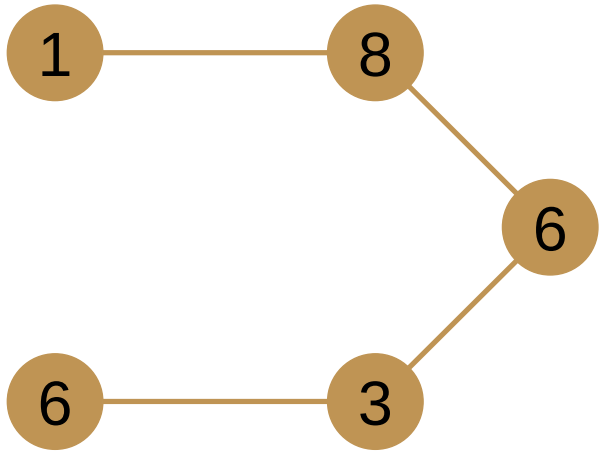
Subset of nodes such that no two of them are adjacent (connected by an edge).



- **Attempt 1:** Odds vs. evens

```
Let S_1 be the set of all v_i where i is an odd number
Let S_2 be the set of all v_i where i is an even number
(Note that S_1 and S_2 are both independent sets)
Determine which of S_1 or S_2 has greater total weight,
  and return this one
```

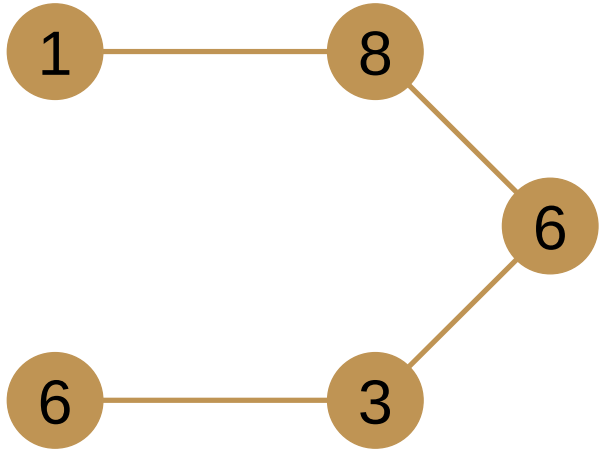
- $1 + 6 + 6$ vs. $8 + 3$
- Yields 13 which is sub-optimal.



- **Attempt 2:** Heaviest-first

```
Start with S equal to the empty set
While some node remains in G
  Pick a node v_i of maximum weight
  Add v_i to S
  Delete v_i and its neighbors from G
Endwhile
Return S
```

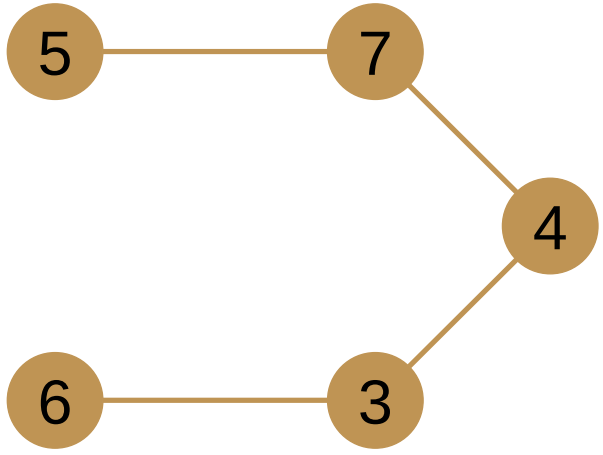
- Take 8, remove first 3 nodes then take 6.
- Yields 14 which is optimal.



- **Attempt 2:** Heaviest-first

```
Start with S equal to the empty set
While some node remains in G
  Pick a node v_i of maximum weight
  Add v_i to S
  Delete v_i and its neighbors from G
Endwhile
Return S
```

- What about now?
- Yields 13 which is sub-optimal.



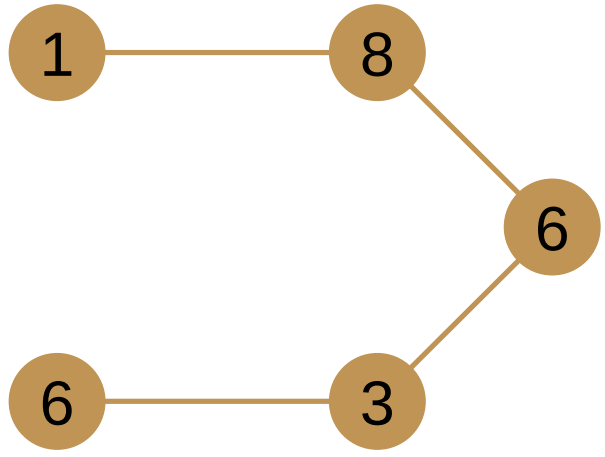
- **Attempt 3:** Dynamic Programming

- S_i: Independent set on $v_1, v_2, ...v_i$
- X_i: Weight of S_i

- $X_0 = 0$
- $X_1 = w_1$
- $i > 1$:
  1. v_i does not belong to $S_i \rightarrow X_i = X_{i-1}$
  2. v_i belongs to $S_i \rightarrow X_i = w_i + X_{i-2}$

- **So:** $X_i = max(X_{i-1}, w_i + X_{i-2})$

- **Attempt 3:** Dynamic Programming



- $X_0 = 0$ $S = ()$
- $X_1 = 1$ $S = (v_1)$
- $X_2 = max(X_1, w_2 + X_0) = max(1, 8 + 0) = 8$ $S = (v_2)$
- $X_3 = max(X_2, w_3 + X_1) = max(8, 6 + 1) = 8$ $S = (v_2)$
- $X_4 = max(X_3, w_4 + X_2) = max(8, 3 + 8) = 11$ $S = (v_2, v_4)$
- $X_5 = max(X_4, w_5 + X_3) = max(11, 6 + 8) = 14$ $S = (v_2, v_5)$

Knapsack problem

Given some items, pack the knapsack to get the maximum total value. Each item has some weight and some value. Total weight that we can carry is no more than some fixed number W.
So we must consider weights of items as well as their values.

Item #	Weight	Value
1	1	8
2	3	6
3	5	5

Knapsack problem

- Given a knapsack with maximum capacity *W*, and a set *S* consisting of *n* items
- Each item *i* has some weight w_i and benefit value b_i (all w_i and *W* are integer values)
- Problem: How to pack the knapsack to achieve maximum total value of packed items?

Knapsack problem

- Problem, in other words, is to find

$$\max \sum_{i \in T} b_i \text{ subject to } \sum_{i \in T} w_i \leq W$$

- The problem is called a "*0-1*" problem, because each item must be entirely accepted or rejected.

0-1 Knapsack Algorithm

```
for w = 0 to W
   V[0,w] = 0
for i = 1 to n
   V[i,0] = 0
for i = 1 to n
   for w = 0 to W
         if w_i <= w // item i can be part of the solution
               if b_i + V[i-1,w-w_i] > V[i-1,w]
                     V[i,w] = b_i + V[i-1,w- w_i]
               else
                     V[i,w] = V[i-1,w]
         else V[i,w] = V[i-1,w]  // w_i > w
```

Running time

```
for w = 0 to W
   V[0,w] = 0
for i = 1 to n
   V[i,0] = 0
for i = 1 to n
   for w = 0 to W
         < the rest of the code >
```

O(W)

Repeat *n* times

O(W)

What is the running time of this algorithm?

O(n*W)

Remember that the brute-force algorithm takes $O(2^n)$

Example

Let's run our algorithm on the following data:

n = 4 (# of elements)
W = 5 (max weight)
Elements (weight, benefit):
(2,3), (3,4), (4,5), (5,6)

Example (2)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1						
2						
3						
4						

```
for w = 0 to W
        V[0,w] = 0
```

Example (3)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

```
for i = 1 to n
        V[i,0] = 0
```

Example (4)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0				
2	0					
3	0					
4	0					

i=1

b_i=3

w_i=2

w=1

w-w_i =-1

if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else **V[i,w] = V[i-1,w]** // w_i > w

Example (5)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	**3**			
2	0					
3	0					
4	0					

i=1

b_i=3

w_i=2

w=2

w-w_i =0

if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w] // w_i > w

Example (6)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	**3**		
2	0					
3	0					
4	0					

i=1

b_i=3

w_i=2

w=3

w-w_i =1

if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w] // w_i > w

Example (7)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	**3**	
2	0					
3	0					
4	0					

i=1

b_i=3

w_i=2

w=4

w-w_i =2

if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w] // w_i > w

Example (8)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	**3**
2	0					
3	0					
4	0					

i=1

b_i=3

w_i=2

w=5

w-w_i =3

if w_i <= w // item i can be part of the solution

  if b_i + V[i-1,w-w_i] > V[i-1,w]

    V[i,w] = b_i + V[i-1,w- w_i]

  else

    V[i,w] = V[i-1,w]

else V[i,w] = V[i-1,w] // w_i > w

Example (9)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	**0**				
3	0					
4	0					

i=2

b_i=4

w_i=3

w=1

w-w_i =-2

if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else **V[i,w] = V[i-1,w]** // w_i > w

Example (10)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	**3**			
3	0					
4	0					

i=2

b_i=4

w_i=3

w=2

$w-w_i$ =-1

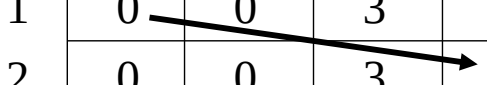
if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else **V[i,w] = V[i-1,w]** // w_i > w

Example (11)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	**4**		
3	0					
4	0					



i=2

b_i=4

w_i=3

w=3

w-w_i =0

if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w] // w_i > w

Example (12)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	**4**	
3	0					
4	0					

i=2

b_i=4

w_i=3

w=4

w-w_i =1

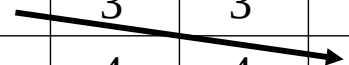
if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w] // w_i > w

Example (13)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	**7**
3	0					
4	0					



i=2

b_i=4

w_i=3

w=5

w-w_i =2

if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w] // w_i > w

Example (14)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	**0**	**3**	**4**		
4	0					

i=3

b_i=5

w_i=4

w= 1..3

```
if w_i <= w // item i can be part of the solution
      if b_i + V[i-1,w-w_i] > V[i-1,w]
         V[i,w] = b_i + V[i-1,w- w_i]
      else
         V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w]  // w_i > w
```

Example (15)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	**5**	
4	0					

i=3

b_i=5

w_i=4

w= 4

w- w_i=0

if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w] // w_i > w

Example (16)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0					

i=3

b_i=5

w_i=4

w= 5

w- w_i=1

if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w] // w_i > w

Example (17)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	**0**	**3**	**4**	**5**	

i=4

b_i=6

w_i=5

w= 1..4

if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else **V[i,w] = V[i-1,w]** // w_i > w

Example (18)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	**7**

i=4

b_i=6

w_i=5

w= 5

w- w_i=0

if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w] // w_i > w