Algorithms

Lesson #2:
Greedy Algorithms

(Based on slides by Prof. Dana Shapira)

1

What is a Greedy Algorithm?

- Solves an optimization problem
- Optimal Substructure:
  - optimal solution contains in it optimal solutions to subproblems
- Greedy Strategy:
  - At each decision point, do what looks best "locally"
  - Top-down algorithmic structure
    - With each step, reduce problem to a smaller problem
- Greedy Choice Property:
  - "locally best" = globally best

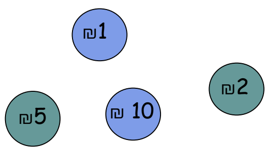
2

Coin Change

Problem -

Give change for *n* Shekels using the minimum number of coins.

Example: n= ₪38



3

Coin Change

```
S ← set of all coins
A ← 0    //Number of used coins
while n>0{
      C ← maximum coin in S
      A ← A +⌊n/C⌋
      n ← n - ⌊n/C⌋·C
      S ← S-{C}
}
```

- Can you think of an example where Greedy is not optimal?

4

The Fractional Knapsack Problem

Problem –

You have a knapsack which can only contain certain weight C of goods.
With this weight capacity constraint, you want to maximize the values of the goods you can put in the knapsack.

Example:

	Total value	Total weight (kilos)
Candy	₪50	10
Chocolate	₪40	5
Ice cream	₪30	5

- If C=14 kilos, what would you do?

The Fractional Knapsack Problem

- Problem –
  Given $G_1, G_2, \dots, G_n$, each G_i with weight w_i and value v_i.
  Maximize the profit out of the goods you can put in the knapsack with capacity C.
- Let f_i ($0 \le f_i \le 1$) be the fractional of G_i one would put in the knapsack.
- Maximize $\sum_{i=1}^{n} f_i v_i$
- Subject to $\sum_{i=1}^{n} f_i w_i \le C$

General concepts:
- Feasible solution
- Optimal solution

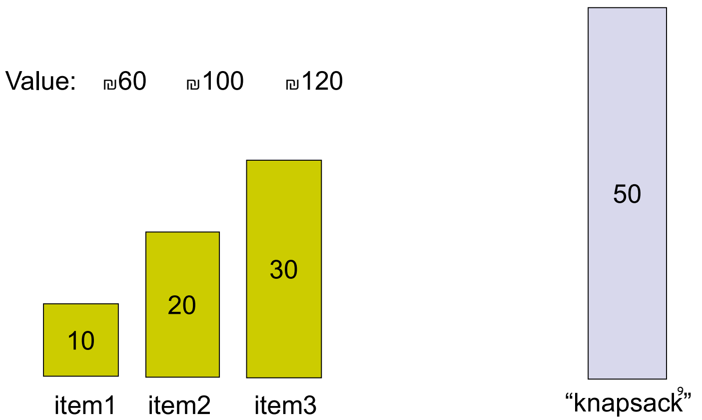
The Fractional Knapsack Problem

```
S ← set of all vi/wi
while C>0
  i ← index of maximum value in S
  S ← S-{vi/wi}
  if (wi<C)
      print('wi Kilos of item i were taken')
      C ← C - wi
  else
      print('C Kilos of item i were taken')
      C ← 0
```


The Integer Knapsack Problem

- Problem –
  Given $G_1, G_2, \dots, G_n$, each G_i with weight w_i and value v_i.
  Maximize the profit out of the goods you can put in the knapsack with capacity C.
- This time $f_i \in \{0,1\}$ (can't cut goods in the middle)
- Maximize $\sum_{i=1}^{n} f_i v_i$
- Subject to $\sum_{i=1}^{n} f_i w_i \le C$
- Can you think of an example where Greedy is not optimal?

The Integer Knapsack Problem

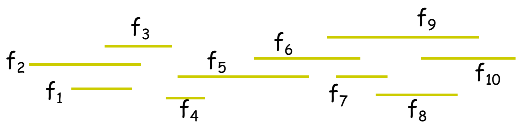


Activity Selection

- Problem –
  Given S={1,2,…,n} of n activities
  - Each activity i has:
    - start time: s_i
    - finish time : f_i
    - $s_i < f_i$
  - Activities i, j are compatible iff non-overlapping:
    $[s_i, f_i)$ $[s_j, f_j)$
  - Objective:
    - select a **maximum-sized** set of mutually compatible activities

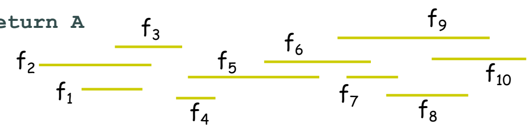
Activity Selection

- Solution
  Order the activities by increasing finishing time, i.e., $f_1 \le f_2 \le \ldots \le f_n$



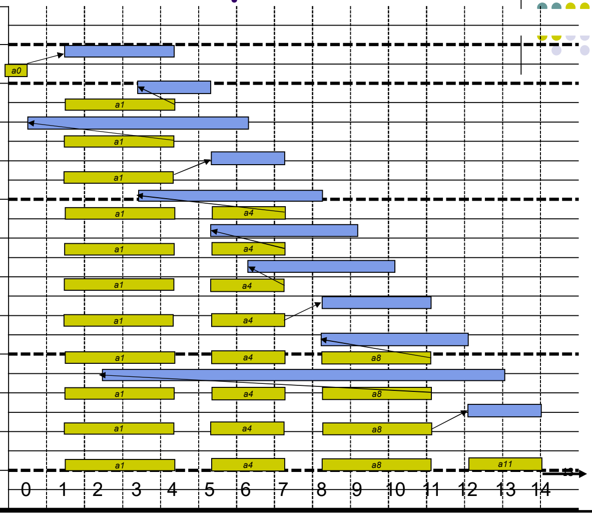
Activity Selection

```
Greedy-Activity-Selector(s,f)
1. n ← length(s)
2. A ← {1}    //Set of selected activities
3. j ← 1      //Last selected activity
4. for i = 2 to n
5.   if s_i ≥ f_j
6.      then A ← A ∪ {i}
7.           j ← i
8. Return A
```



Example

k	s_k	f_k
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14
12	∞	-



Optimality of Activity Selection

Theorem: **Algorithm GREED-ACTIVITY-SELECTOR is optimal**

Proof:

```
Greedy-Activity-Selector(s,f)
n ← length(s)
A ← {1}     //Set of selected activities
j ← 1       //Last selected activity
for i = 2 to n
   if sᵢ ≥ fⱼ
       then A ← A ∪ {i}
               j ← i
Return A
```

Loop invariant: Here we have a feasible solution for activities 1,…,*i*-1 with:
- |*A*| maximal
- *j* minimal for this |*A*|

The loop invariant is proven by induction on *i*

Weighted Activity Selection

Now each activity also has a ***weight*** w_i

We want to maximize the sum of weights

(Before, all activities had w_i=1 -- unweighted)

Does greedy work here?

How do we do greedy?

- Choose minimum finishing time, as before…
- Choose maximum weight…