Algorithms

Dana Shapira
Lesson #2: Greedy Algorithms
Part - I

What is a Greedy Algorithm?

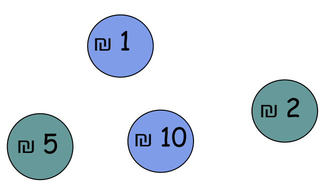
- Solves an optimization problem
- Optimal Substructure:
  - optimal solution contains in it optimal solutions to subproblems
- Greedy Strategy:
  - At each decision point, do what looks best "locally"
  - Top-down algorithmic structure
    - With each step, reduce problem to a smaller problem
- Greedy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

Coin Change

Problem –

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

Example: n= ₪ 38



Coin Change

```
S ← set of all coins
A ← 0
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?

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	₪ 1.0	10
Chocolate	₪ 2.0	1
Ice cream	₪ 2.5	4

- If *C*=4 kilos, what would you do?

The Fractional Knapsack Problem

- Problem -

  Given $G_1, G_2, \ldots, G_n$, 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 $\sum_{i=1}^{n} f_i w_i \le C$

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

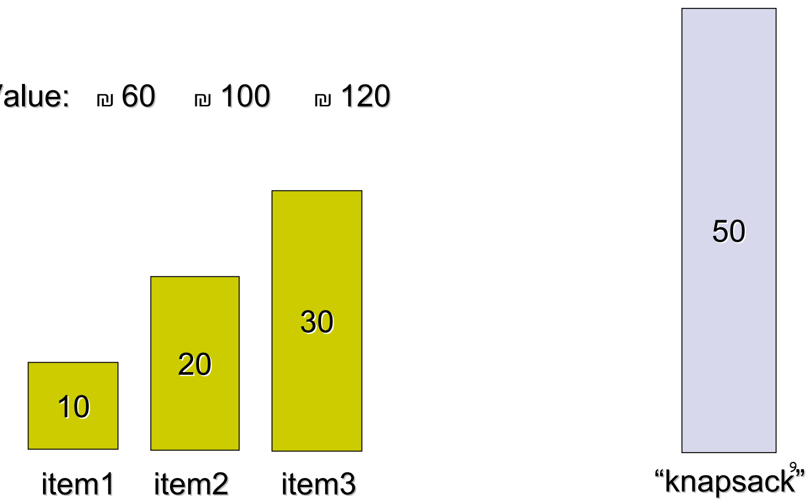
- Problem -

  Given $G_1, G_2, \ldots, G_n$, 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=0/1 is the fractional of G_i
- 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

Value: ₪ 60 ₪ 100 ₪ 120

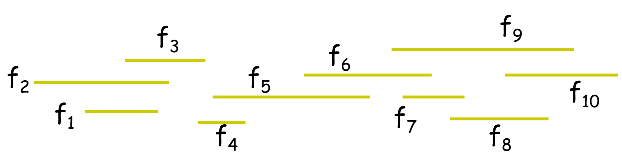


Activity-Selection

- 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 **<u>maximum-sized</u>** set of mutually compatible activities

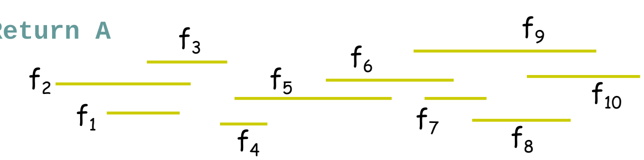
Activity-Selection

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



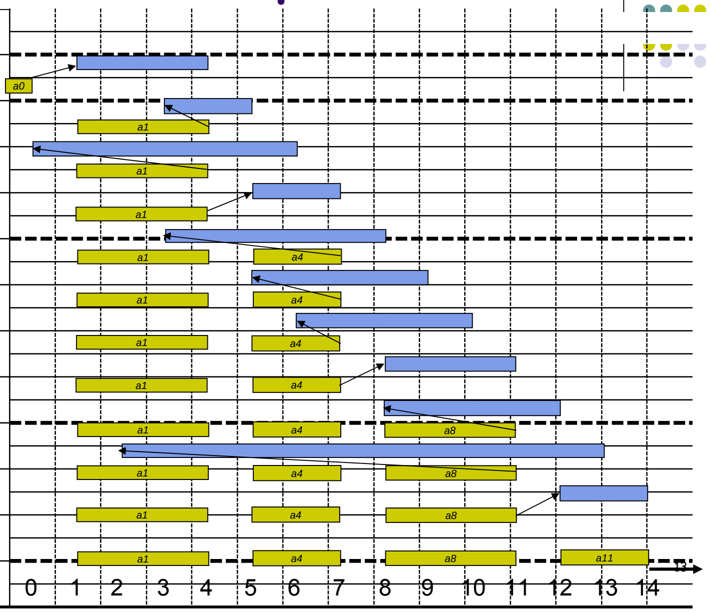
Activity-Selection

```
Greedy-Activity-Selector(s,f)
1. n ← length(s)
2. A ← {1}
3. j ← 1
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 Idea** - the activity problem satisfied
  - Greedy choice property.
  - Optimal substructure property.
- **Proof** -
  - Activity 1 has the earliest finish time – why? Suppose, $A \subseteq S$ is an optimal solution and let activities in A be ordered by finish time.
  - Suppose, the first activity in A is k.

Optimality of Activity-Selection

if k = 1, then A begins with a greedy choice and we are done.
if k > 1, we want to show that there is another solution B that begins with greedy choice, activity 1.
Let B = A - {k}∪{1}. Because $f_1 \leq f_k$, the activities in B are disjoint and since B has same number of activities as A, B is also optimal.

Once the greedy choice is made, the problem reduces to finding an optimal solution for the problem. If A is an optimal solution to the original problem S, then A′=A - {1} is an optimal solution to the activity-selection problem $S' = \{i \in S : s_i \geq f_1\}$.

why? Because if we could find a solution B′ to S′ with more activities then A′, adding 1 to B′ would yield a solution B to S with more activities than A, there by contradicting the optimality. □