CMPN302: Algorithms Design and Analysis

Lecture 05: Greedy Algorithms

Ahmed Hamdy

Computer Engineering Department

Cairo University

Fall 2018

Greedy algorithms

- A ***greedy algorithm*** always makes the choice that looks best at the moment.
- It makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.
- In some problems, can achieve optimal solutions.
- For Other problems, achieves near-optimal solutions fast.

Greedy algorithms

- ***Activity selection algorithm***
- Find max subset of non-overlapping tasks

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

$$f_1 \leq f_2 \leq f_3 \leq \cdots \leq f_{n-1} \leq f_n$$

- Solution: $\{a_1, a_4, a_8, a_{11}\}$

- *What is the hint behind ordering based on finish time??*

Greedy algorithms

- ***Activity selection algorithm***
- Using dynamic programming, the recurrence:

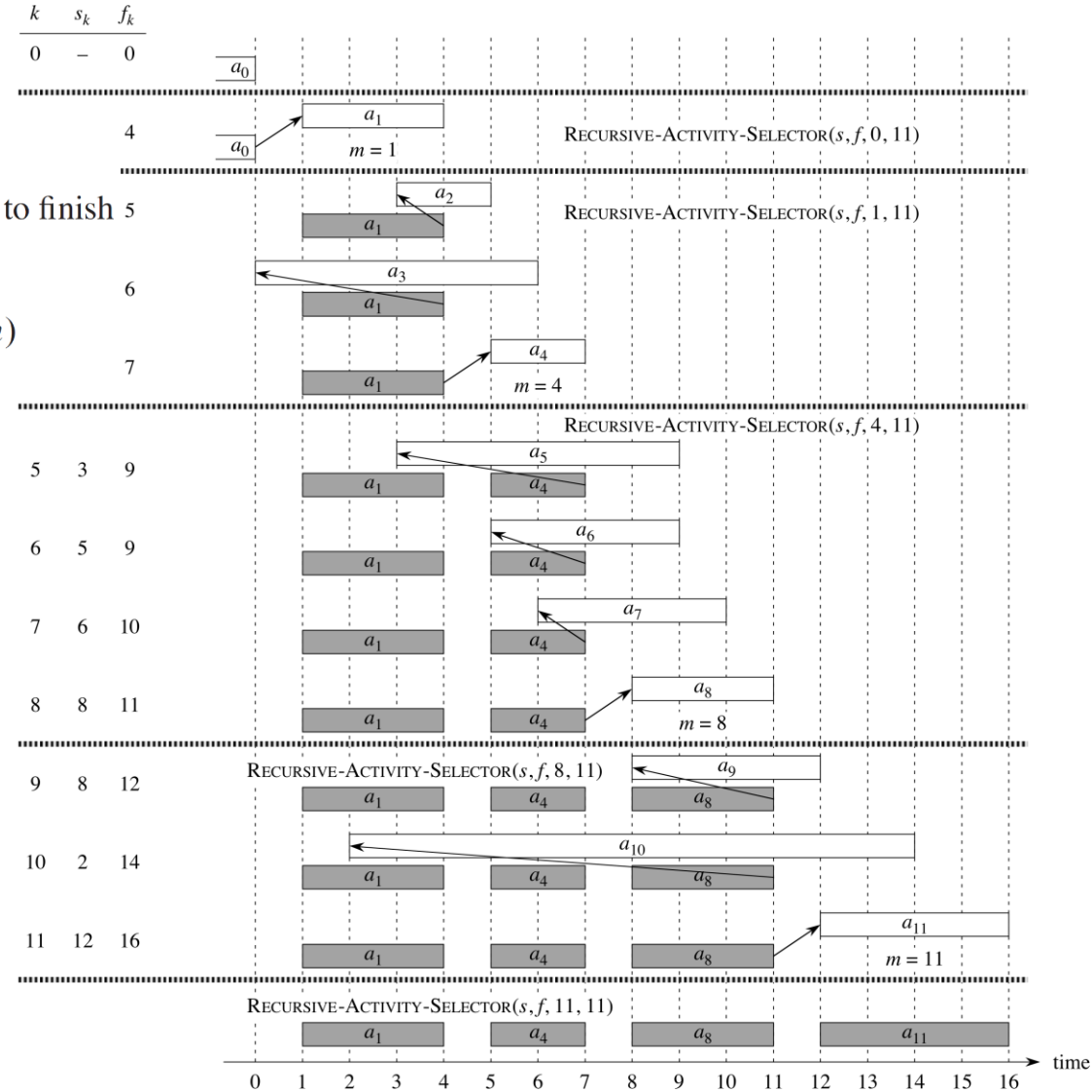
$$c[i, j] = c[i, k] + c[k, j] + 1$$

- Can greedy algorithms achieve the optimal solution too in a simpler way??

Greedy algorithms

- ***Activity selection algorithm***

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]        // find the first activity in S_k to finish
3        m = m + 1
4  if m ≤ n
5        return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅
```



- Exercise:
  - Recursive to Iterative

Greedy algorithms

- ***Activity selection algorithm***
- Why the greedy solution is optimal

Theorem 16.1

Consider any nonempty subproblem S_k, and let a_m be an activity in S_k with the earliest finish time. Then a_m is included in some maximum-size subset of mutually compatible activities of S_k.

Proof Let A_k be a maximum-size subset of mutually compatible activities in S_k, and let a_j be the activity in A_k with the earliest finish time. If $a_j = a_m$, we are done, since we have shown that a_m is in some maximum-size subset of mutually compatible activities of S_k. If $a_j \neq a_m$, let the set $A'_k = A_k - \{a_j\} \cup \{a_m\}$ be A_k but substituting a_m for a_j. The activities in A'_k are disjoint, which follows because the activities in A_k are disjoint, a_j is the first activity in A_k to finish, and $f_m \leq f_j$. Since $|A'_k| = |A_k|$, we conclude that A'_k is a maximum-size subset of mutually compatible activities of S_k, and it includes a_m. ■

Greedy algorithms

1. Cast the optimization problem as one in which we make a choice and are left with one subproblem to solve.
2. Prove that there is always an optimal solution to the original problem that makes the greedy choice, so that the greedy choice is always safe.
3. Demonstrate optimal substructure by showing that, having made the greedy choice, what remains is a subproblem with the property that if we combine an optimal solution to the subproblem with the greedy choice we have made, we arrive at an optimal solution to the original problem.

Greedy vs Dynamic programming

- ***0-1 knapsack problem***
  - Thief trying to max value when picking from n items (each worth v_i and weight w_i) while limited to total weight W.

- ***Fractional knapsack problem***
  - Same as above but can take fractions of items

Greedy vs Dynamic programming

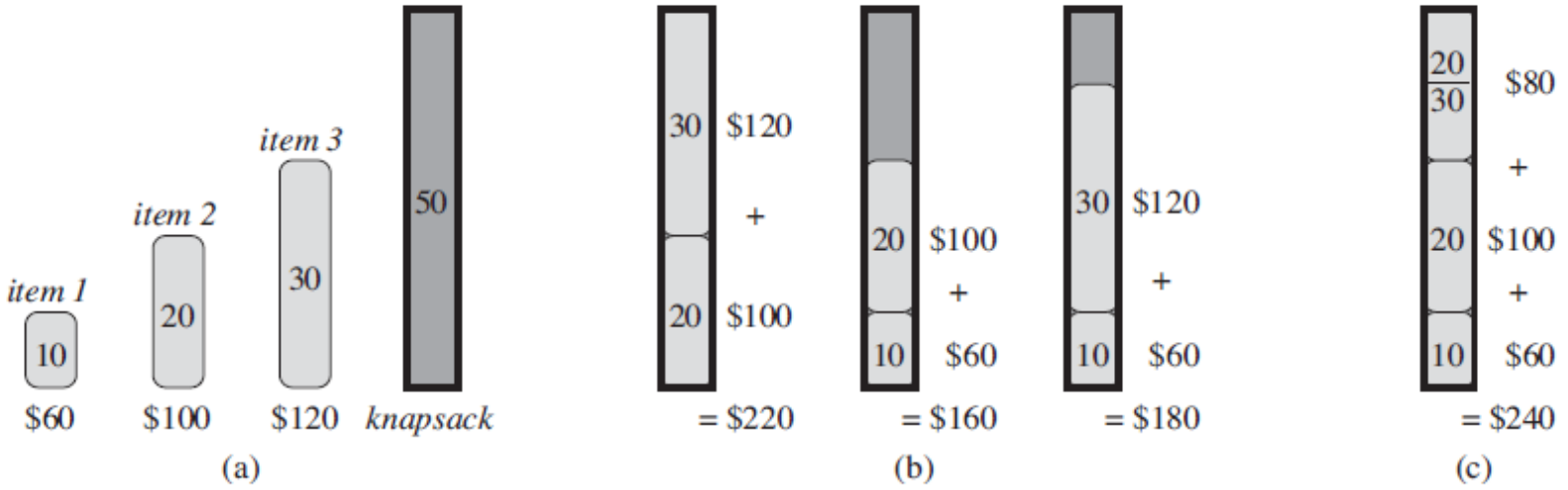


Figure 16.2 An example showing that the greedy strategy does not work for the 0-1 knapsack problem. **(a)** The thief must select a subset of the three items shown whose weight must not exceed 50 pounds. **(b)** The optimal subset includes items 2 and 3. Any solution with item 1 is suboptimal, even though item 1 has the greatest value per pound. **(c)** For the fractional knapsack problem, taking the items in order of greatest value per pound yields an optimal solution.

Greedy algorithms

- ***Huffman codes***
- Used for compression of data
- Can save 20% to 90%
- Example:

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Figure 16.3 A character-coding problem. A data file of 100,000 characters contains only the characters `a`–`f`, with the frequencies indicated. If we assign each character a 3-bit codeword, we can encode the file in 300,000 bits. Using the variable-length code shown, we can encode the file in only 224,000 bits.

- How to parse “1000101111001101”?

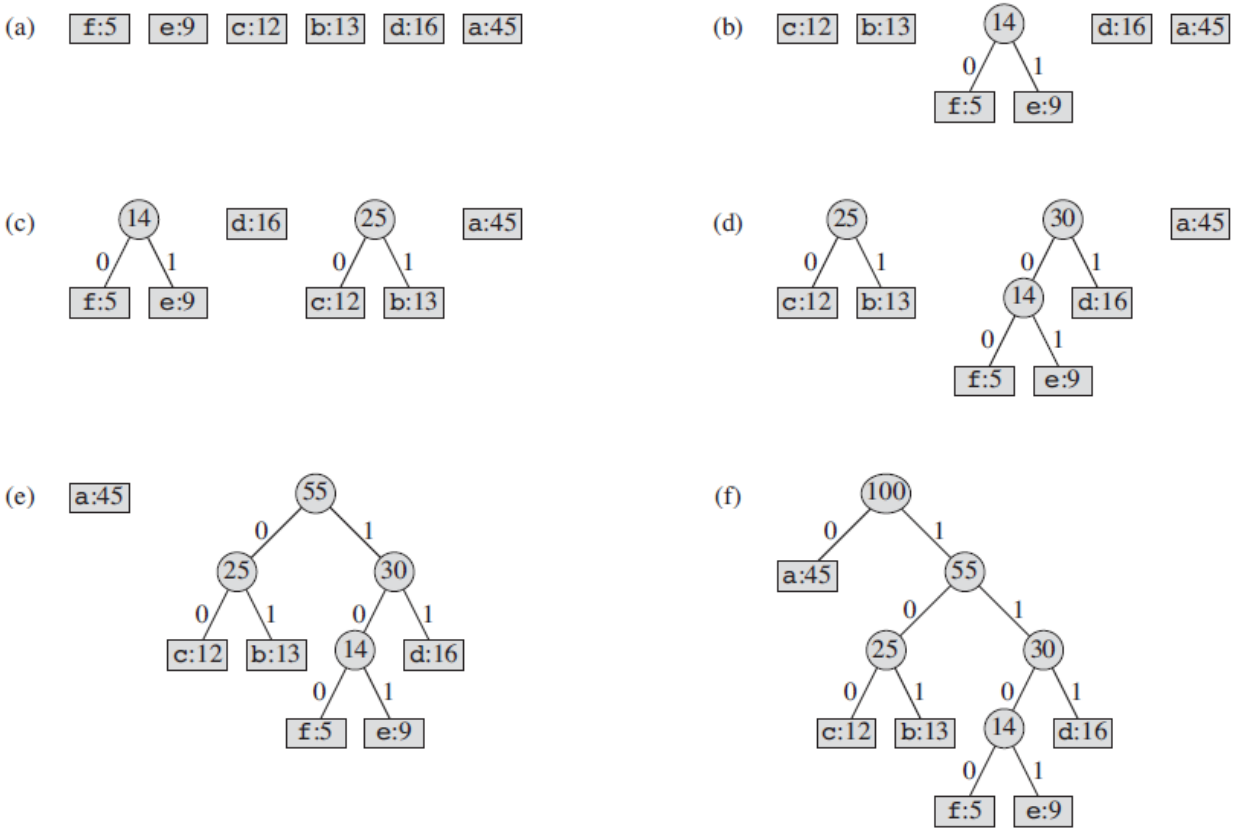
Greedy algorithms

- ***Huffman codes***
- Prefix codes:
  - Set of codewords where no codeword is a prefix of the other
  - Why?

Greedy algorithms

- ***Huffman codes***

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4      allocate a new node z
5      z.left = x = EXTRACT-MIN(Q)
6      z.right = y = EXTRACT-MIN(Q)
7      z.freq = x.freq + y.freq
8      INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return t
```



Complexity?

Figure 16.5 The steps of Huffman's algorithm for the frequencies given in Figure 16.3. Each part shows the contents of the queue sorted into increasing order by frequency. At each step, the two trees with lowest frequencies are merged. Leaves are shown as rectangles containing a character and its frequency. Internal nodes are shown as circles containing the sum of the frequencies of their children. An edge connecting an internal node with its children is labeled 0 if it is an edge to a left child and 1 if it is an edge to a right child. The codeword for a letter is the sequence of labels on the edges connecting the root to the leaf for that letter. **(a)** The initial set of $n = 6$ nodes, one for each letter. **(b)**–**(e)** Intermediate stages. **(f)** The final tree.