1 Last remark on Knapsack (KT §6.4/ CLRS §16.2)

Running time:

Also note: more on knapsack and fractional knapsack in CLRS 16.2

2 Interval Scheduling/Activity Selection Problem (KT §6.1, CLRS §16.1)

Input: List of intervals $S =$

Goal: Find a subset

First attempt: Dynamic Programming

1. Subproblems: for any $i < j$, the optimal solution for intervals

2. Guess an interval

3. Recurrence:

Second attempt: Improved dynamic programming
Sort the activities by:

Guess whether

Subproblems:

Recurrence:

2.1 Greedy strategy

Maybe we don't need to try all possible activities? Can we identify an activity that is used in an optimal solution?

Ideas:

- Activity with the
- Shortest
- Activity intersecting

Turns out

Theorem. For any list of intervals S, there is an optimal solution that includes

Proof.

Running time.

Algorithm:

```
cur_fin =
For i = 1 to n:
    if
        print
        cur_fin =
```

3 Huffman codes

3.1 Intro to compression

- Normally use 8 bits per
  - What if our file uses ≤ 32 symbols?
- Some symbols are used
  - Maybe we can encode

Example

To make sure there is no ambiguity, use:

3.2 Prefix codes

Definition

Example

Example Suppose we have a file of 1 billion symbols with frequencies:

$$f_a = 0.32, \quad f_e = 0.25, \quad f_k = 0.2, \quad f_r = 0.18, \quad f_u = 0.05$$

Definition

It is convenient to model a prefix code as

Can this encoding be made more efficient?

Definition

Claim. The binary tree corresponding to an optimal prefix code is full.

Proof.

3.3 Greedy attempt 1: Shannon-Fano 1949

Create tree top-down, splitting S into

$$f_a = 0.32, \quad f_e = 0.25, \quad f_k = 0.2, \quad f_r = 0.18, \quad f_u = 0.05$$

3.4 Greedy attempt 2: Huffman encoding 1952

- *Observation 1.* Lowest frequency symbols should be
- *Observation 2.* The lowest level always contains
- *Observation 3.* The order in which items appear

Claim 1. There is an optimal prefix code with tree T^* where

Create tree bottom-up.

Example

$$f_a = 0.32, \quad f_e = 0.25, \quad f_k = 0.2, \quad f_r = 0.18, \quad f_u = 0.05$$

3.5 Algorithm

```
 if |S| = 2:
     return
Let y and z be
S’ =
Remove y and z from
Insert new
T’ =
T =
Return
```

Time complexity

- Naive implementation
- Use priority queue to store symbols

3.6 Proof of correctness/optimal

Claim 2. ABL(T) =

Proof.

Claim 3. The Huffman code achieves the minimum ABL of any prefix codes.

Proof.