GREEDY: Solved Excercise n. 2 pag. 185

- **Input:** **n** items, the cost of **j** at round **t** is
  $100 * r(j)^t$, with $r(j) > 1$ and $r(j) \# r(j')$
- **PROBLEM**: Find a purchase scheduling
  S = Permutation of **J ={1,…,n}** with **total minimum cost**

- **Example**: Consider the Permutation $P^<$ of J s.t. $r(1) < r(2) < \ldots < r(n)$, then the total cost is

$$C(P) = 100 * \sum r(t)^t$$

THM. The Greedy permutation $P^>$ (decreasing sched.) is OPTIMAL

Proof (Exchange Argument). Consider any $P \# P^> \rightarrow$ there exists t in P :
$r(t) < r(t+1)$ and its cost comp. is $100*(r(t)^t + r(t+1)^{t+1})$

Let's **swap** the order of thes two items and show their cost strictly **decrease**!

GOAL: to prove that, since $1 < r(t) < r(t+1)$ (***), it holds

$100*(r(t+1)^t + r(t)^{t+1}) < 100*(r(t)^t + r(t+1)^{t+1})$ (**)

Trivial since from (***) and (**) $\rightarrow$ $r(t)^t\,(r(t) - 1) < r(t+1)^t\,(r(t+1) - 1)$

4.8 Huffman Codes

These lecture slides are supplied by Mathijs de Weerd

Data Compression

Q. Given a text that uses alphabet S of 32 symbols, how can we encode this text in **bits?**

Q. Some symbols (**e, t, a, o, i, n**) are used far **more often** than others. How can we use this to reduce our encoding?

Q. How do we know when the next symbol begins?

Ex. c(a) = 01
c(b) = 010
c(e) = 1

What is 0101?

Data Compression

Q. Given a text that uses alphabet S of 32 symbols, how can we encode this text in **bits**?

A. We can **encode** 2^5 different symbols using a **fixed** length of ***5*** bits per symbol.

C: S → $\{0,1\}^5$ This is called fixed length encoding.

Q. Some symbols ***(e, t, a, o, i, n)*** are used far **more often** than others. How can we use this to reduce our encoding?

A. Encode these characters with **fewer** bits, and the others with **more**bits.

Q. How do we know when the next symbol begins?

A. Use a separation symbol (like the pause in Morse), or make sure that there is no **ambiguity** by ensuring that **no code is a prefix of another one**.

Ex. of *non Prefix* Code

What is 0101?

c(a) = 01

c(b) = 010

c(e) = 1

Prefix Codes

Definition. A prefix code for a set S is a function c that maps each $x \in S$ to $\{0,1\}^*$ in such a way that

For any $x, y \in S$, $x \neq y$, $c(x)$ is not a prefix of $c(y)$.

Ex. c(a) = 11
c(e) = 01
c(k) = 001
c(l) = 10
c(u) = 000

Q. What is the meaning of 1001000001 ?

Suppose **frequencies** are known in a text of **1G**:

$f_a=0.4$, $f_e=0.2$, $f_k=0.2$, $f_l=0.1$, $f_u=0.1$

Q. What is the ***size*** of the encoded text?

Prefix Codes

Definition. A prefix code for a set S is a function c that maps each $x \in S$ to 1s and 0s in such a way that for $x, y \in S$, $x \neq y$, c(x) is not a prefix of c(y).

Ex. c(a) = 11
c(e) = 01
c(k) = 001
c(l) = 10
c(u) = 000

Q. What is the meaning of 1001000001 ?
A. "leuk"

Suppose frequencies are known in a text of 1G:
$f_a=0.4$, $f_e=0.2$, $f_k=0.2$, $f_l=0.1$, $f_u=0.1$

Q. What is the size of the encoded text
(ordering is not relevant!)

A. $2*f_a + 2*f_e + 3*f_k + 2*f_l + 4*f_u = 2.4G$

(Saving w.r.t. fixed length code is 0.6 G)

Optimal Prefix Codes

Definition. The average bits per letter of a prefix code ***c*** is the **sum** over all symbols of:

(its frequency) * (the number of bits of its encoding):

$$ABL(c) = \sum_{x \in S} f_x \cdot |c(x)|$$

Optimization Problem:

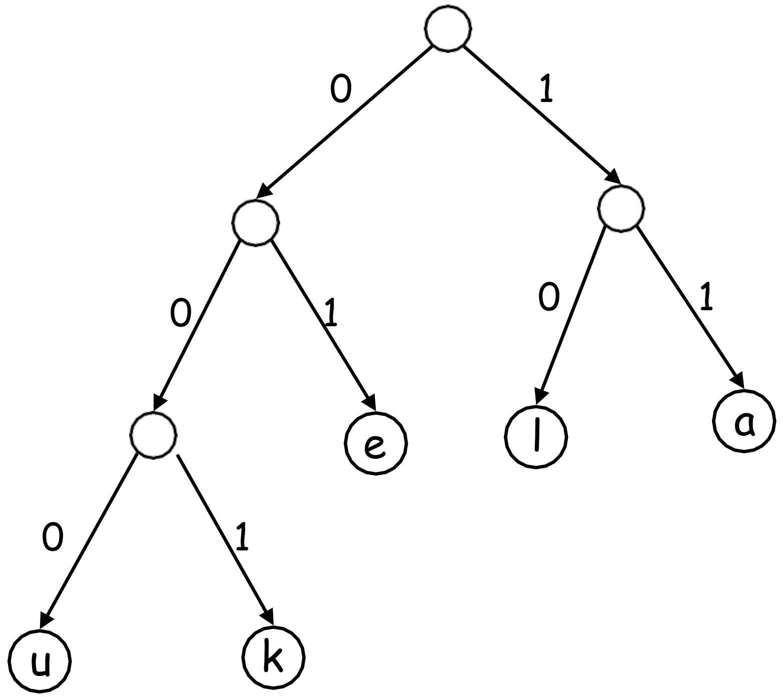
Input. A finite alphabet S with symbol freq. $\{f_x : x \in S\}$

Goal: find a **prefix code c** that has the ***lowest*** possible ***average bits*** per letter.

We can model a code in a ***binary tree***...

Representing Prefix Codes using Binary Trees

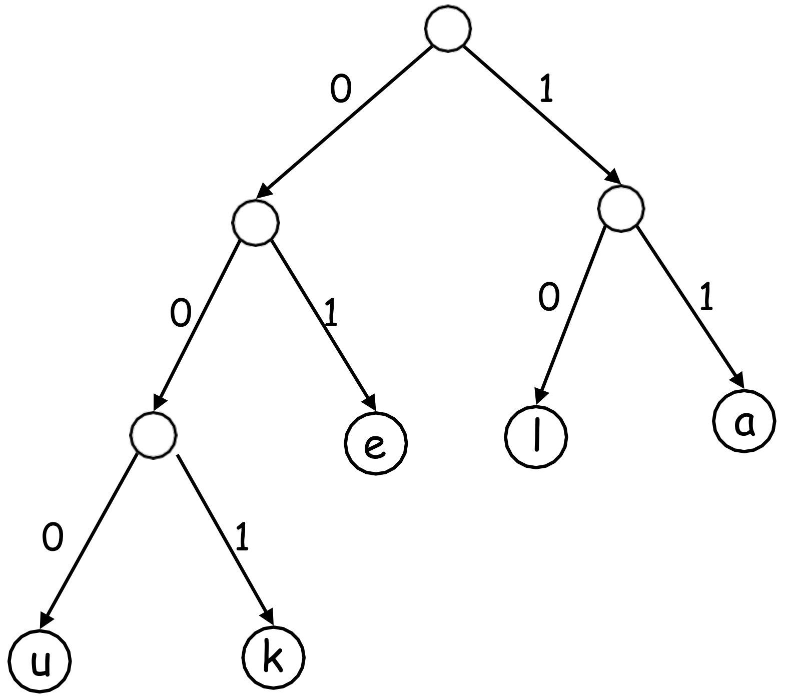
Ex. c(a) = 11
c(e) = 01
c(k) = 001
c(l) = 10
c(u) = 000



Q. How does the tree of a prefix code look?

Representing Prefix Codes using Binary Trees

Ex. c(a) = 11
c(e) = 01
c(k) = 001
c(l) = 10
c(u) = 000



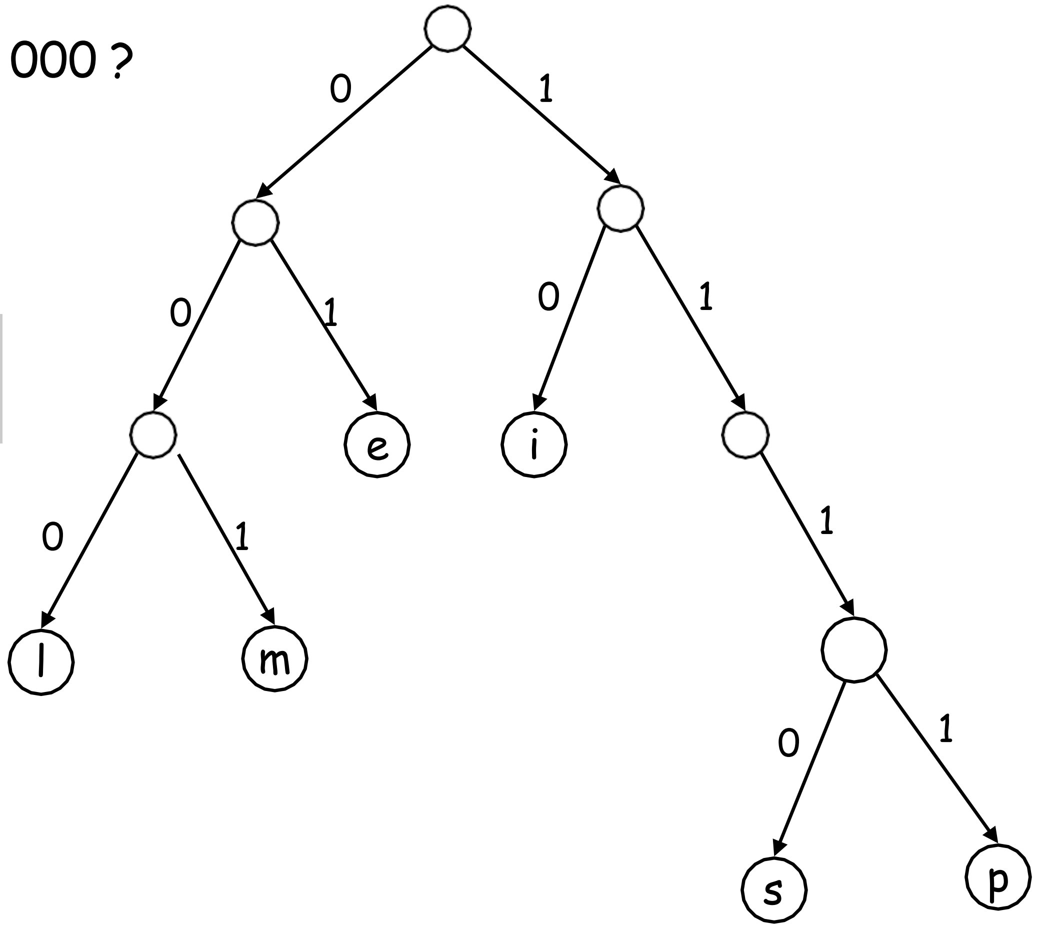
Q. How does the **tree** of a **prefix code** look?

A. Only the ***leaves*** have a ***label***.

Proof. An encoding of ***x*** is a prefix of an encoding of ***y*** **iff** the path of ***x*** is a prefix of the path of ***y***.

Representing Prefix Codes using Binary Trees

Q. What is the meaning of
1110 10 001 1111 01 000 ?



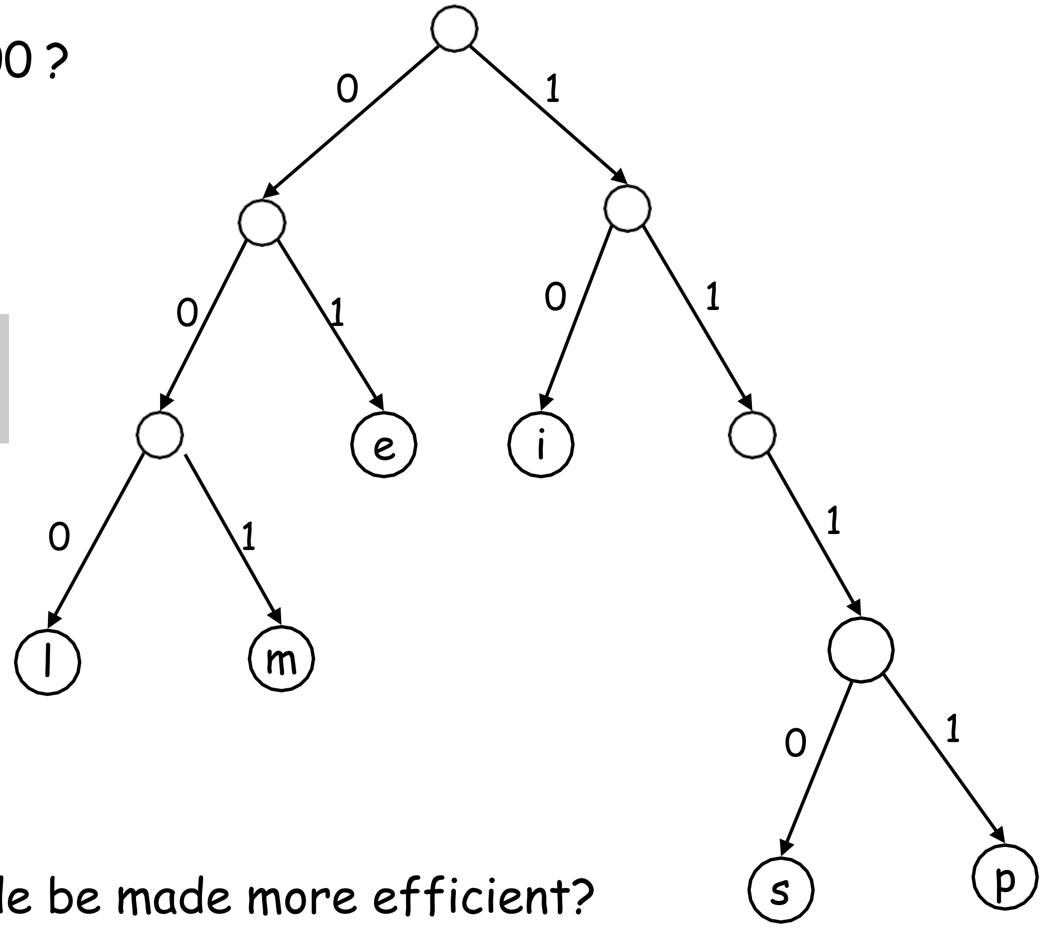
$$ABL(T) = \sum_{x \in S} f_x \cdot \text{depth}_T(x)$$

Representing Prefix Codes using Binary Trees

Q. What is the meaning of
111010001111101000 ?

A. "simpel"

$$ABL(T) = \sum_{x \in S} f_x \cdot \text{depth}_T(x)$$



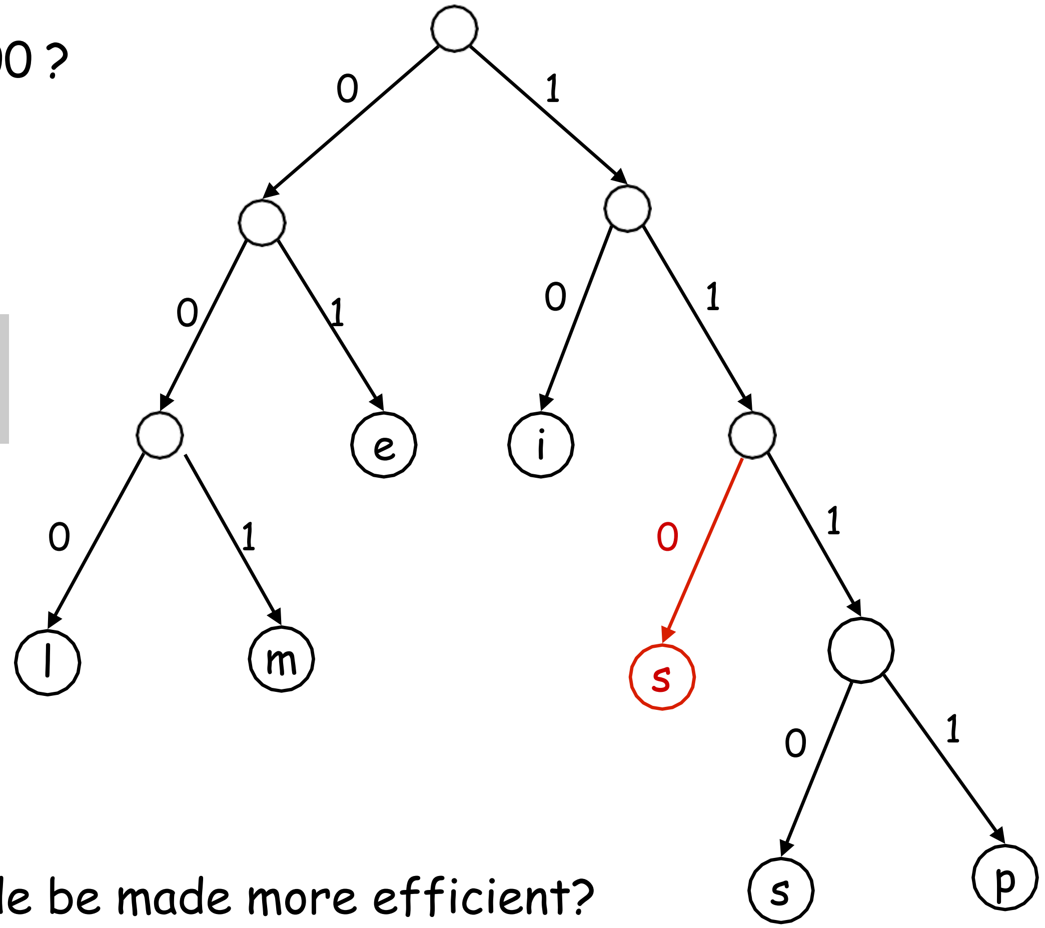
Q. How can this prefix code be made more efficient?

Representing Prefix Codes using Binary Trees

Q. What is the meaning of
111010001111101000 ?

A. "simpel"

$$ABL(T) = \sum_{x \in S} f_x \cdot \text{depth}_T(x)$$



Q. How can this prefix code be made more efficient?

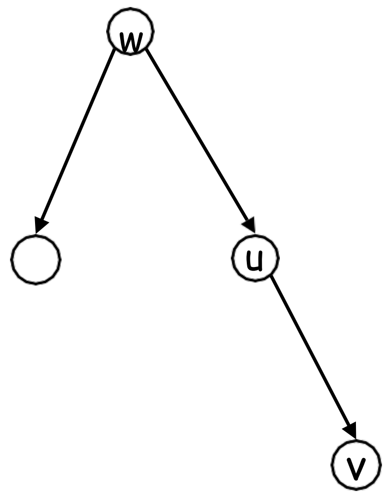
A. Change encoding of p and s to a shorter one.

This tree is now **full**.

Representing Prefix Codes using Binary Trees

Definition. A tree is full if every node that is not a leaf has two children.

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



Representing Prefix Codes using Binary Trees

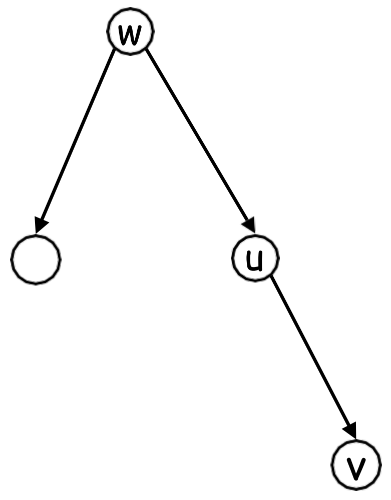
Definition. A tree is full if every node that is not a leaf has two children.

Claim. The binary tree corresponding to the optimal prefix code is full. Proof. (by contradiction)

- Suppose T is binary tree of optimal prefix code and is not full.

- This means there is a node **u** with only one child **v**.
- **Case 1**: **u** is the root; delete **u** and use **v** as the **root**

- **Case 2:** **u** is not the root
  - Let **w** be the parent of **u**
  - Delete **u** and make **v** be a child of **w** in place of **u**



- In both cases the number of bits needed to encode any leaf in the subtree of **v** is **decreased**. The rest of the tree T is not affected.
- Clearly this new tree *T'* has a smaller ABL than T. **Contradiction.**

Optimal Prefix Codes: *False Start*

Q. Where should letters be placed with a high frequency in the tree of an **optimal** prefix code ?

Optimal Prefix Codes: False Start

Q. Where in the tree of an optimal prefix code should letters be placed with a high frequency?

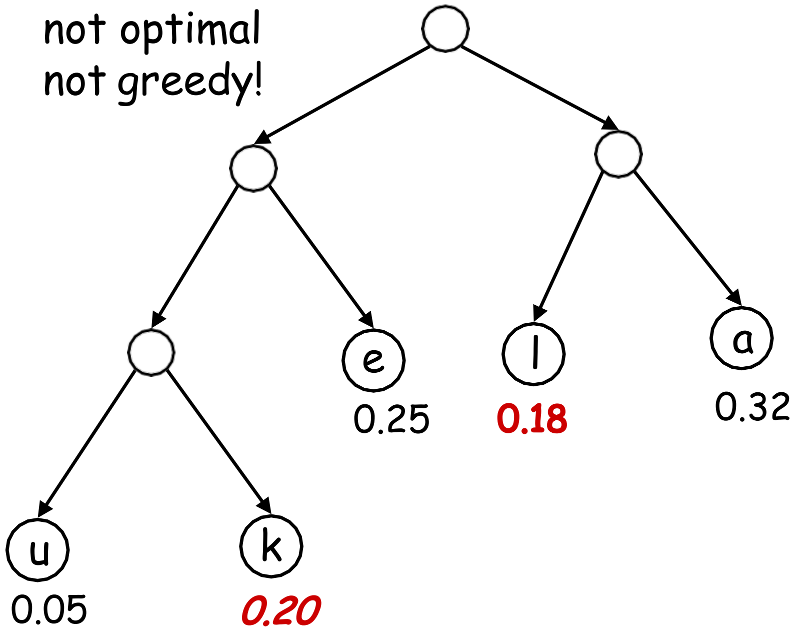
A. Near the top! Use recursive structure of trees.

Greedy template. Create tree top-down, split S into two sets S_1 and S_2 with (almost) ***equal frequencies***. Recursively build tree for S_1 and S_2.

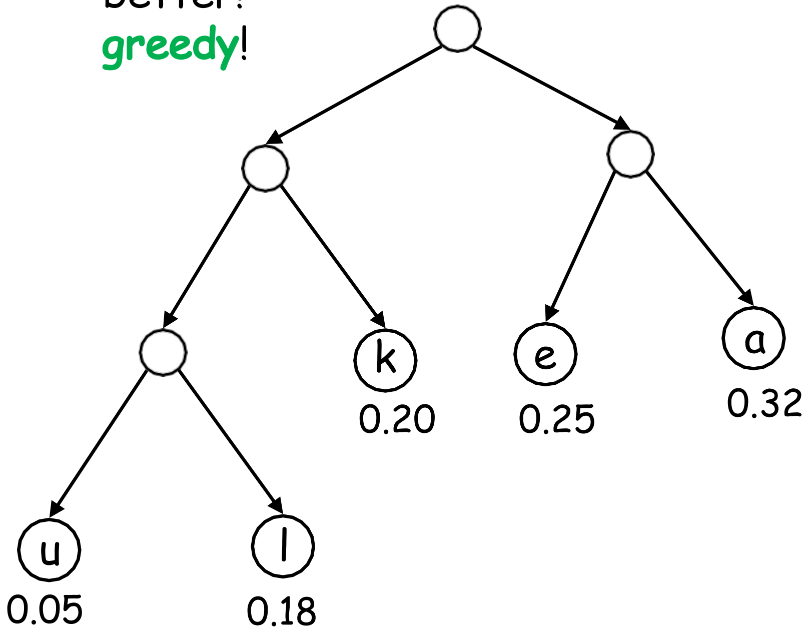
[Shannon-Fano, 1949]

$f_a=0.32$, $f_e=0.25$, $f_k=0.20$, $f_l=0.18$, $f_u=0.05$

S-F is
not optimal
not greedy!



better!
greedy!



End of Part I (Data Compression)

MAIN CONCEPTS

- Def. of Prefix CODES
- Def. of the correspond. OPTIMIZATION PROBLEM
- Equivalence between Prefix CODES and Labeled TREES
- The Shannon Fano Algorithm

EXCERCISE/TEST.

- Write the Pseudo-Code of the S-F algorithm and run it over some worst-case instances that show it is not optimal.

- Analyze its complexity time.