Analysis and Design of Algorithms. Greedy Algorithms

Juan Gutiérrez

September 2019

These are algorithms that build a solution by choosing the best option locally. They are easy to design, but the difficult part is proving that the algorithm returns the optimal solution in the long run.

1 Disjoint Intervals (Unweighted)

Let $[s_1, f_1]$, $[s_2, f_2]$, …, $[s_n, f_n]$ be a sequence of closed intervals on the line. Two intervals are *compatible* if they do not overlap.

Problema Max-Disjoint-Intervals. Given a sequence of closed intervals on the line, find a subset of pairwise compatible intervals of maximum size.

Some possible greedy approaches to solve the problem:

- Select a compatible interval that starts earliest (smallest s_i)

  Does not work if the interval is very large:

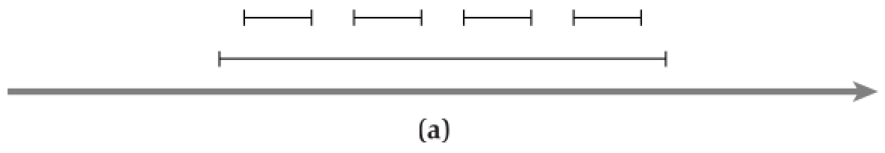


Figure 1: Taken from the book Kleinberg, Algorithm Design

- Select a compatible interval with smallest size (smallest $t_i - s_i$)

  Does not work in some cases:

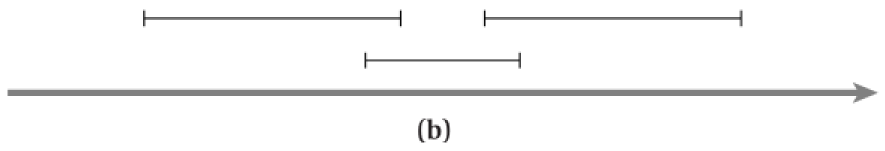


Figure 2: Taken from the book Kleinberg, Algorithm Design

- Select a compatible interval with fewer intersections

  It is harder to find a counterexample, but it also does not always work:

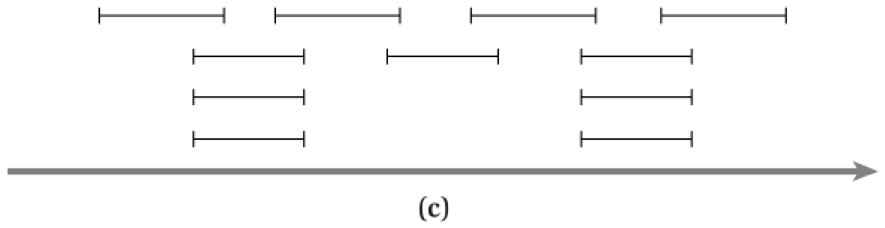


Figure 3: Taken from the book Kleinberg, Algorithm Design

An idea that **does** work: take the compatible interval **with the smallest finishing point**.

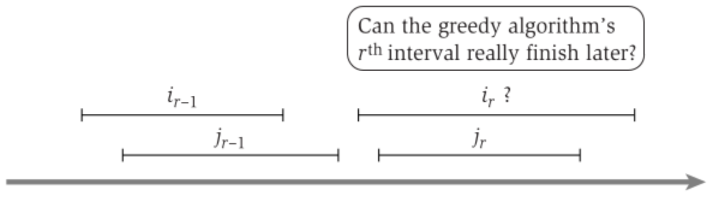

Can the greedy algorithm's r^{th} interval really finish later?
i_{r-1}
i_r ?
j_{r-1}
j_r

Figure 4: Taken from the book Kleinberg, Algorithm Design

2 A General Technique for Proofs

If we want to prove that our algorithm is correct, it suffices to prove two things.

1. *Greedy choice:* we must prove that there always exists an optimal solution that contains the greedy choice.

2. *Optimal substructure*: we must prove that the subsolution left is optimal for the subproblem left by the greedy choice.

If these two points are proven, I prove that my greedy algorithm is correct. Let us return to the disjoint intervals problem to prove it using this technique.

2.1 Proof for Disjoint Intervals

Let us recall the disjoint intervals problem.

Let $[s_1, f_1]$, $[s_2, f_2]$, $\cdots$, $[s_n, f_n]$ be a sequence of closed intervals on the line. Two intervals are *compatible* if they do not overlap.

Problema Max-Disjoint-Intervals. Given a sequence of closed intervals on the line, find a subset of pairwise compatible intervals of maximum size.

Greedy choice: choose the interval with the smallest f_i.

We propose the recursive algorithm:

Recibe: a set $\mathcal{I} = \{[s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]\}$ of intervals, sorted in increasing order by finishing point

Devuelve: a subset of pairwise compatible intervals of maximum size

```
MAX-DISJOINT-INTERVALS-REC($\mathcal{I}$)
1: if $\mathcal{I} = \emptyset$
2:     return $\emptyset$
3: $\mathcal{I}' = \mathcal{I} \setminus \{[s_i, f_i] : s_i \leq f_1\}$
4: return $\{[s_1, f_1]\} \cup$ MAX-DISJOINT-INTERVALS-REC($\mathcal{I}'$)
```

Lema 2.1 (Greedy Choice)**.** *There exists an optimal solution to the problem that contains the interval* $[s_1, f_1]$.

Proof. Let X be an optimal solution to the problem. If X contains $[s_1, f_1]$ then we have nothing to prove. Suppose then that $[s_1, f_1] \notin X$. Let $[s_j, f_j]$ be an interval in X with the smallest f_j value. Let $X' = (X \setminus \{[s_j, f_j]\}) \cup \{[s_1, f_1]\}$. We will show that X' is a solution to the problem. For this, it suffices to show that $[s_1, f_1]$ is compatible with any interval in $X \setminus \{[s_j, f_j]\}$.

Let $[s_k, f_k]$ be any interval in $X \setminus \{[s_j, f_j]\}$. Note that

$$f_1 \leq f_j < s_k,$$

therefore $[s_1, f_1]$ is compatible with $[s_k, f_k]$.

Since X' is a solution to the problem and $|X| = |X'|$, we conclude that X' is an optimal solution to the problem that contains $[s_1, f_1]$. □

Lema 2.2 (Optimal Substructure)**.** *If X is an optimal solution to the problem that contains $[s_1, f_1]$, then $X \setminus \{[s_1, f_1]\}$ is an optimal solution to the subproblem left by the greedy choice.*

Proof. Let $\mathcal{I}$ be the collection of intervals of the original problem. Let $\mathcal{I}'$ be the collection of intervals after applying the greedy choice, that is

$$\mathcal{I}' = \{[s_k, f_k] : f_1 < s_k\}.$$

Suppose for the sake of contradiction that $X' = X \setminus \{[s_1, f_1]\}$ is not an optimal solution for $\mathcal{I}'$. Then there exists a solution Y' for $\mathcal{I}'$ with $|Y'| > |X'|$. But in that case $Y = Y' \cup \{[s_1, f_1]\}$ is a solution for $\mathcal{I}$ with size $|Y| = |Y'| + 1 > |X'| + 1 = |X|$, a contradiction. $\square$

With these two lemmas in hand, we can prove the correctness of the algorithm.

Teorema 2.1. *The algorithm* MAX-DISJOINT-INTERVALS-REC *performs as requested.*

Proof. By induction on $|\mathcal{I}|$. If $|\mathcal{I}| = 0$, then $\mathcal{I} = \emptyset$ and the algorithm returns $\emptyset$, which is correct. Suppose then that $|\mathcal{I}| > 0$. Since $|\mathcal{I}'| \leq |\mathcal{I}|$, by the induction hypothesis, in line 4, the algorithm returns an optimal solution X' for $\mathcal{I}'$. We must demonstrate that $X = X' \cup \{[s_1, f_1]\}$ is an optimal solution for $\mathcal{I}$.

By Lemma 2.1 (Greedy Choice), there exists an optimal solution containing $[s_1, f_1]$. Let $Y = \{[s_1, f_1]\} \cup Y'$ be this solution. By Lemma 2.2 (Optimal Substructure), we have that Y' is optimal for the subproblem. Then $|Y'| = |X'|$ and therefore

$$|Y| = |Y'| + 1 = |X'| + 1 = |X|.$$

Then, since X has the same size as Y, X is an optimal solution to the problem. $\square$

Although it is true that the previous Theorem correctly proves the correctness of the proposed greedy algorithm, from here on, for the problems we see, we can assume that it suffices to prove only the two lemmas: greedy choice and optimal substructure.

Next, let's see another example.

Ejercicio 2.1. *Describe an efficient algorithm that, given a set $\{a_1, a_2, \ldots, a_n\}$ of points on the line, such that $a_1 \leq a_2 \leq \ldots a_n$, determines a minimum set of unit length intervals that contains all points. Justify that your algorithm is correct using the properties of greedy choice and optimal substructure.*

Solution: we will divide our solution into 4 steps.

1. Greedy choice: Select $[a_1, a_1 + 1]$.

2. Recursive algorithm.
   Recibe: A set $A = \{a_1, \ldots, a_n\}$ of points on the real line such that $a_1 \leq \cdots \leq a_n$
   Devuelve: A set of unit intervals of minimum size covering A
   GREEDY-SEGMENTS(A)

```
1: if $A = \emptyset$
2:     return $\emptyset$
3: $A' = \{a_i \in A : a_i > a_1 + 1\}$
4: return $\{[a_1, a_1 + 1]\} \cup$ GREEDY-SEGMENTS($A'$)
```

3. Proof of the greedy choice.

   Lema 2.3 (Greedy Choice)**.** *There exists an optimal solution that contains* $[a_1, a_1 + 1]$.

   Proof. Let X be an optimal solution. If $[a_1, a_1 + 1] \in X$ then there is nothing left to show. Suppose then that $[a_1, a_1 + 1] \notin X$. Since X is a solution, there exists an interval $[p, p + 1] \in X$ that contains a_1, that is $p \leq a_1 \leq p + 1$. We will show that $X' = X \setminus \{[p, p+1]\} \cup \{[a_1, a_1 + 1]\}$ is also a solution to the problem.

   For this, we must prove that every $a_i \in A$ is in some interval of X'. Let $a_i \in A$. If $a_i \notin [p, p+1]$ then it is in some interval of $X \setminus \{[p, p+1]\}$. Said interval is also in X' and therefore a_i is covered by some interval of X'. If $a_i \in [p, p + 1]$ then

   $$a_1 \leq a_i \leq p + 1 \leq a_1 + 1,$$

   and therefore $a_i \in [a_1, a_1 + 1]$. Then X' is a solution and since $|X| = |X'|$, then X' is an optimal solution. □

4. Proof of optimal substructure

   Lema 2.4 (Optimal Substructure)**.** *If X is an optimal solution for A that contains $\{[a_1, a_1 + 1]\}$, then $X \setminus \{[a_1, a_1 + 1]\}$ is an optimal solution for A'.*

   Proof. Suppose for the sake of contradiction that $X' = X \setminus \{[a_1, a_1 + 1]\}$ is not optimal for A'. Then there exists a solution Y' for A' such that $|Y'| < |X'|$. Then $Y = Y' \cup \{[a_1, a_1 + 1]\}$ is a solution for A. But $|Y| = |Y'| + 1 < |X'| + 1 = |X|$, a contradiction. □

3 Fractional Knapsack

Let us recall the knapsack problem.

Problema Integer-Knapsack. Given a set $\{1, 2, \ldots, n\}$ of items each with a natural weight w_i, a natural value v_i, and a natural number W, find a subset of items whose sum of values is the largest possible, but less than or equal to W.

The fractional problem allows choosing "fractions of items" in each choice.

Problema Fractional-Knapsack. Given a set $\{1, 2, \ldots, n\}$ of items each with a natural weight w_i, a natural value v_i, and a natural number W, find a vector of rationals between 0 and 1 $(x_1, x_2, \ldots, x_n)$ that maximizes $\sum_{i=1}^{n} x_i v_i$ subject to the constraint $\sum_{i=1}^{n} x_i w_i \leq W$.

Example: suppose that $W = 50, n = 5$, $w = [40, 30, 20, 10, 20]$, $v = [840, 600, 400, 100, 300]$. Then $x = [1, 1/3, 0, 0, 0]$ is a feasible solution for the problem, since $1 \cdot 40 + 1/3 \cdot 30 + 0 \cdot 20 + 0 \cdot 10 + 0 \cdot 20 = 50 \leq 50$.

Greedy choice: always choose the items with the highest value-to-weight ratio. We can assume that $v_1/w_1 \leq v_2/w_2 \leq \cdots \leq v_n/w_n$. We have the following algorithm.

Recibe: An instance v, w, W of the Fractional-Knapsack problem
Devuelve: An optimal solution for said instance

```
FractionalKnapsack-Greedy($v, w, W$)
1: for $j = n$ to 1
2:     if $w[j] \leq W$
3:         $x_j = 1$
4:         $W = W - w[j]$
5:     else
6:         $x_j = W/w[j]$
7:         $W = 0$
8: return $x$
```

Note that this algorithm does not solve the integer knapsack case; in that case, dynamic programming must be applied.

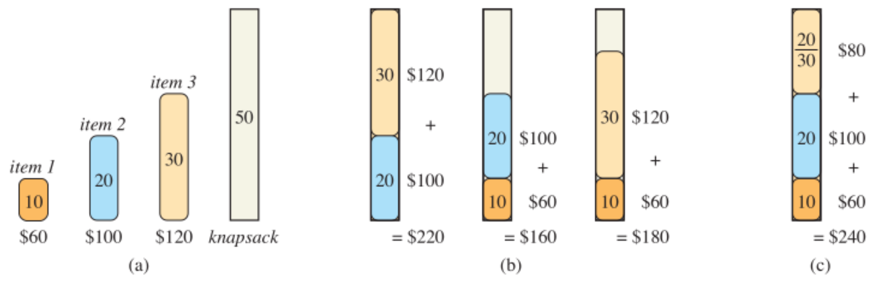


Figure 5: Taken from the book Cormen, Introduction to Algorithms

Next, we will demonstrate that our algorithm is correct.

Lema 3.1 (Greedy Choice)**.** *There exists an optimal solution* $(x_1, x_2, \ldots, x_n)$ *to the problem such that* $x_n = \min\{1, W/w_n\}$*.*

Proof. Let $\alpha = \min\{1, W/w_n\}$. Let $(y_1, y_2, \ldots, y_n)$ be an optimal solution to the problem **such that $\mathbf{y_n}$ is maximal**.

If $y_n = \alpha$ then there is nothing to prove. Suppose then that $y_n \neq \alpha$.

Since y is an optimal solution, it holds that $y \cdot w \leq W$. Therefore, there exists an $i \in \{1, \ldots, n-1\}$ such that $y_i > 0$.

Let

$$\delta = \min\{y[i], (\alpha - y[n])\frac{w[n]}{w[i]}\}$$

and

$$\beta = \delta\frac{w[i]}{w[n]}.$$

Note that $\delta, \beta > 0$.

Define x according to

$$x[j] = \begin{cases} y[j] & \text{if } j \notin \{i, n\} \\ y[i] - \delta & \text{if } j = i \\ y[n] + \beta & \text{if } j = n \end{cases}$$

Note that

$$x \cdot w = y \cdot w - \delta w[i] + \beta w[n] = y \cdot w - \delta w[i] + \delta w[i] = y \cdot w.$$

Also,

$$\begin{aligned} x \cdot v &= y \cdot v - \delta v[i] + \beta v[n] \\ &= y \cdot v - \delta v[i] + \delta\frac{w[i]}{w[n]}v[n] \\ &= y \cdot v + \delta(\frac{w[i]}{w[n]}v[n] - v[i]) \\ &= y \cdot v + \delta w[i](\frac{v[n]}{w[n]} - \frac{v[i]}{w[i]}) \\ &\geq y \cdot v \end{aligned}$$

We conclude that $xv \geq yv$, and since y is optimal, x is also optimal. But $x_n > y_n$, a contradiction to the choice of y. □

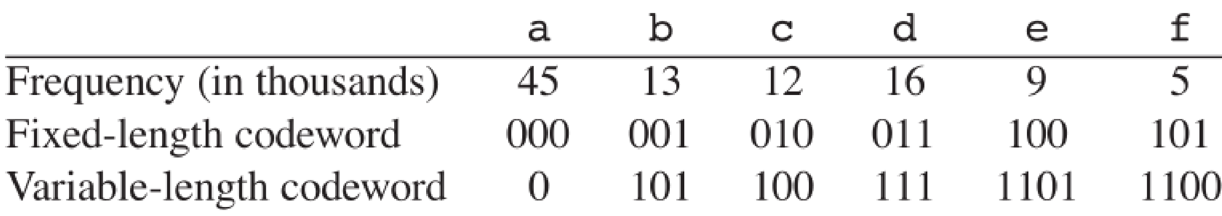
Lema 3.2 (Optimal Substructure)**.** *If $(x_1, x_2, \ldots, x_n)$ is an optimal solution to the problem with $x_n = \min\{1, W/w_i\}$, then $(x_1, x_2, \ldots, x_{n-1})$ is an optimal solution to the subproblem left with $W = W - x_n w_n$.*

Proof. Suppose for the sake of contradiction that this is not the case. Let $(y_1, y_2, \ldots, y_{n-1})$ be an optimal solution to the subproblem with maximum weight $W - x_n w_n$ that chooses the first $n-1$ items. Then $(y_1, y_2, \ldots, y_{n-1}, x_n)$ is a feasible solution to the original problem with a value greater than $x \cdot v$, a contradiction. □

4 Huffman Codes

It is a character encoding that allows compacting text files. That is, transforming a file of characters into a sequence of bits.

Idea: use few bits for the most frequent characters, and more bits for the rarer ones.



	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

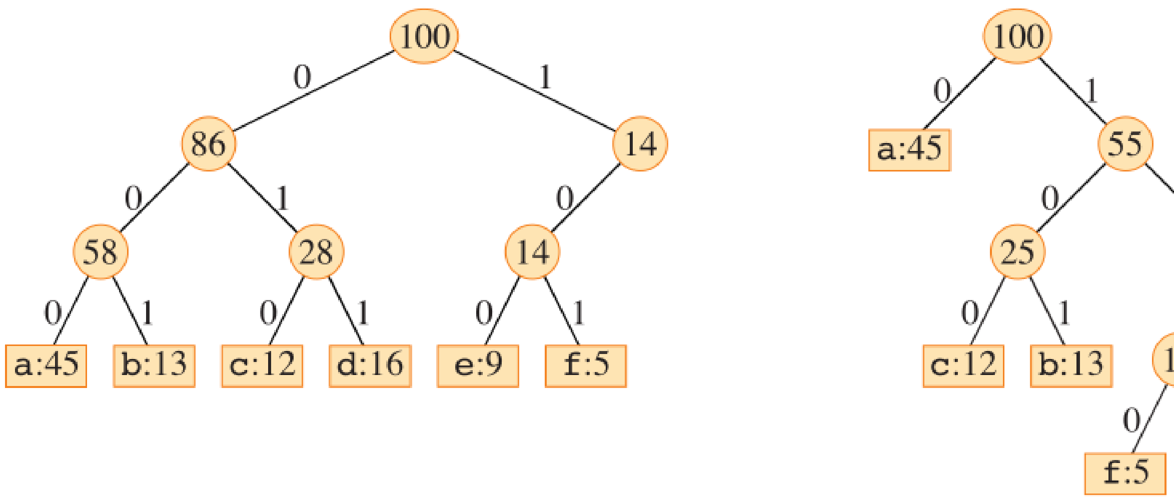


Figure 6: Taken from the book Cormen, Introduction to Algorithms

Binary Character Codes

Given a set C of characters, a *code table* for C is a bijection between C and some set of bit sequences. We call the sequence corresponding to a character the *character code*.

A code table is *prefix-free* if for any pair of characters x and y, the code of x is not a prefix of the code of y.

The example in the previous figure is prefix-free. In said example, the string $abacafe$ is encoded by 01010100011001101.

File Encoding

A *file* is a sequence of characters. The set of characters is the *alphabet* of the file.

The *weight* of a character in the file is the frequency (number of occurrences) of c, denoted by $p(c)$. Note that the number of characters in the file is equal to $\sum_{c \in C} p(c)$.

Problema Compression-Problem. Given a file of characters, find a prefix-free code table that produces an encoded file of minimum size.

A code tree for a set C of characters is a binary tree where each leaf corresponds to an element of C and each internal node has exactly two children.

Let $d(c)$ be the depth of character c. Then the total number of bits used in the encoding is

$$\sum_{c \in C} d(c)p(c).$$

Therefore, the previous problem is equivalent to finding a code tree whose sum of weights times depth is as small as possible.

Huffman Trees

Let $S = \{1, 2, \ldots, n\}$. A *Huffman tree* with respect to S is any collection Π of subsets of S that satisfies the following properties.

1. for each X and each Y in Π, we have that $X \cap Y = \emptyset$, or $X \subseteq Y$ or $Y \subseteq X$,
2. $S \in \Pi$,
3. $\{\} \notin \Pi$,
4. every non-minimal element in Π is the union of two other elements in Π.

The elements of Π are called *nodes.* The node S is called the *root.* The minimal nodes are called *leaves.* All other nodes are called *internal.*

Example: Let $S = \{1, 2, 3, 4, 5, 6\}$, and $\Pi = \{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\},$
$\{2, 3\}, \{5, 6\}, \{4, 5, 6\}, \{2, 3, 4, 5, 6\}, \{1, 2, 3, 4, 5, 6\}\}$.

If X, Y and $X \cup Y$ are nodes of the tree, we say that X and Y are *children* of $X \cup Y$ and that $X \cup Y$ is the *parent* of X and Y.

An *ancestor* of X is any node I such that $X \subseteq I$. If $I \neq X$ then I is a *proper ancestor.* In the example, $\{2, 3, 4, 5, 6\}$ is the parent of $\{4, 5, 6\}$.

The *depth* of a node X is the number of proper ancestors of X, denoted by $d(X)$. In the example, $d(\{2, 3\}) = 2$.

If X and Y are leaves, children of the same parent, we say that they are *sibling* leaves. Note that if X and Y are sibling leaves, then $\Pi - \{X, Y\}$ is also a Huffman tree. We say that a Huffman tree has *unitary* leaves if each of its leaves has only one element.

Minimum Weight Huffman Trees

A *weighting* of a set S is an assignment of numerical weights to the elements of S. Given a weighting p_i for each $i \in S$, and $X \subseteq S$, we denote by $p(X)$ the sum $\sum_{i \in X} p_i$. We will say that $p(X)$ is the *weight* of X.

Example: In the previous example, I could assign $p(1) = 45$, $p(2) = 13$, $p(3) = 12$, $p(4) = 16$, $p(5) = 9$, $p(6) = 5$.

The *weight* of a Huffman tree $p(\Pi)$, denoted by $p(\Pi)$, is the sum of the weights of the nodes that are not the root, that is

$$p(\Pi) = \sum_{X \in \Pi - \{S\}} p(X).$$

In the example, $p(\Pi) = p(\{1\}) + p(\{2\}) + p(\{3\}) + p(\{4\}) + p(\{5\}) + p(\{6\}) + p(\{2,3\}) + p(\{5,6\}) + p(\{4,5,6\}) + p(\{2,3,4,5,6\}) = 45 + 13 + 12 + 16 + 9 + 5 + 25 + 14 + 30 + 55 = 224$.

Ejercicio 4.1. *Prove that* $p(\Pi) = \sum_{X \in \Gamma} p(X)d(X)$*, where* Γ *is the set of leaves of* Π.

Problema Min-Weight-Huffman. Given a partition Γ and a set S with a weighting, find a Huffman tree of minimum weight among those that have Γ as leaves.

In the example, $S = \{1,2,3,4,5,6\}, \Gamma = \{\{1\},\{2\},\{3\},\{4\},\{5\},\{6\}\}, p(1) = 45, p(2) = 13, p(3) = 12, p(4) = 16, p(5) = 9, p(6) = 5$.

Huffman's Algorithm

Recibe: A set S, a weighting p of S and a partition Γ of S
Devuelve: An optimal Huffman tree (with minimum weight) that has Γ as set of leaves

```
HUFFMAN($S, p, \Gamma$)
1: if $|\Gamma| = 1$
2:     return $\Gamma$
3: Let $X$ be an element in $\Gamma$ with minimum weighting
4: $\Gamma = \Gamma - X$
5: Let $Y$ be an element in $\Gamma$ with minimum weighting
6: $\Gamma = \Gamma - Y$
7: $\Gamma = \Gamma \cup \{X \cup Y\}$
8: return $\{X, Y\} \cup$ HUFFMAN($S, p, \Gamma$)
```

Example: let $S = \{1, 2, \ldots, 6\}$, $p(1) = 45, p(2) = 13, p(3) = 12, p(4) = 16, p(5) = 9, p(6) = 5$ and $\Gamma = \{\{1\},\{2\},\{3\},\{4\},\{5\},\{6\}\}$.

The algorithm produces the tree $\{\{1\},\{2\},\{3\},\{4\},\{5\},\{6\},\{6,5\},\{6,5,4\},\{2,3\},\{2,3,4,5,6\},\{1,2,3,4,5,6\}\}$. Its weight is 224.

Ejercicio 4.2. *Run Huffman's algorithm for* $S = \{1,2,3,4,5,6\}$ *with unitary leaves and weighting* $p(1) = p(2) = p(3) = p(4) = p(5) = p(6)$. *Give also a non-optimal Huffman tree for that same weighting.*

Correctness of Huffman's Algorithm

We will show the properties of greedy choice and optimal substructure.

Lema 4.1. *(Greedy Choice) Let X and Y be two leaves with minimum weight. There exists an optimal Huffman tree Π that has X and Y as sibling leaves of maximum depth.*

Proof. Let Π be an optimal Huffman tree with set of leaves Γ. Let V, W be two sibling leaves of maximum depth. Then, $d(X), d(Y) \leq d(V) = d(W)$. Assume without loss of generality that $p(V) \leq p(W)$. Then

$$p(X), p(Y) \leq p(V) \leq p(W).$$

Now we will swap the positions of X and V in Π, producing a new Huffman tree Π'. More precisely, Π' is defined according to:

- for each ancestor I of X that is not an ancestor of V, change I to $(I-X)\cup V$,
- for each ancestor J of V that is not an ancestor of X, change J to $(J-V)\cup X$,

Note that Π' is a Huffman tree with set of leaves Γ.
Note also that

$$\begin{aligned}
p(\Pi') &= p(\Pi) - \sum_{F\in\Gamma} p(F)d(F) + \sum_{F\in\Gamma'} p(F)d'(F) \\
&= p(\Pi) - (p(X)d(X) + p(V)d(V)) + (p(X)d'(X) + p(V)d'(V)) \\
&= p(\Pi) - (p(X)d(X) + p(V)d(V)) + (p(X)d(V) + p(V)d(X)) \\
&= p(\Pi) + (p(X) - p(V))(d(V) - d(X)) \\
&\leq p(\Pi)
\end{aligned}$$

We note that the Huffman tree Π' is better than or equal to Π and therefore is also an optimal solution. We repeat the same process with Y and W obtaining a tree Π'' that is also optimal. Furthermore, Π'' contains X and Y as sibling leaves of maximum depth.

□

Lema 4.2. *(Optimal Substructure) Let Π be an optimal Huffman tree for Γ that has X and Y as sibling leaves of maximum depth. Then $\Pi' = \Pi - \{X, Y\}$ is an optimal Huffman tree for $\Gamma' = \Gamma - \{X, Y\} \cup \{X \cup Y\}$.*

Proof. Suppose for the sake of contradiction that there exists a solution Φ' for Γ' with weight less than Π'. Then $\Phi = \Phi' \cup \{X, Y\}$ is a solution with lower weight for the problem Γ, a contradiction. □

Implementation with Priority Queue

To implement Huffman's algorithm, we can make use of a priority queue. Recall that a priority queue can perform insertion and extraction of a minimum element in $O(\lg n)$.

Recibe: A set S, a weighting p of S

Devuelve: An optimal Huffman tree (with minimum weight) that has the elements of S as set of leaves

```
HUFFMAN-PRIORITYQUEUE($S, p$)
 1: $n = |S|$
 2: $Q$ = INIT-PQ()
 3: for $i = 1$ to $n$
 4:     $z.weight = p(i)$
 5:     $z.left = NIL$
 6:     $z.right = NIL$
 7:     INSERT-PQ($Q, z$)
 8: for $i = 1$ to $n - 1$
 9:     $x$ = EXTRACTMIN-PQ($Q$)
10:     $y$ = EXTRACTMIN-PQ($Q$)
11:     $z.left = x$
12:     $z.right = y$
13:     $z.weight = x.weight + y.weight$
14:     INSERT-PQ($Q, z$)
15: return EXTRACTMIN-PQ($Q$)
```

The running time is $O(n \lg n)$ if the priority queue is implemented as a heap.