Greedy Algorithms

❖ Optimization Problems:

Given a set of constraints and an optimization function. Solutions that satisfy the constrains are called feasible solutions. A feasible solution for which the optimization function has the best possible value is called an optimal solution.

❖ The Greedy Method:

Make the best decision at each stage, under some greedy criterion. A decision made in one stage is not changed in a later stage, so each decision should assure feasibility.

Note:

- **Greedy algorithm works only if the local optimum is equal to the global optimum.**
- **Greedy algorithm does not guarantee optimal solutions. However, it generally produces solutions that are very close in value (heuristics) to the optimal, and hence is intuitively appealing when finding the optimal solution takes too much time.**

Activity Selection Problem

Given a set of activities $S = \{ a_1, a_2, \ldots, a_n \}$ that wish to use a resource (e.g. a classroom). Each a_i takes place during a time interval $[s_i, f_i)$.

Activities a_i and a_j are *compatible* if $s_i \geq f_j$ or $s_j \geq f_i$ (i.e. their time intervals do not overlap).

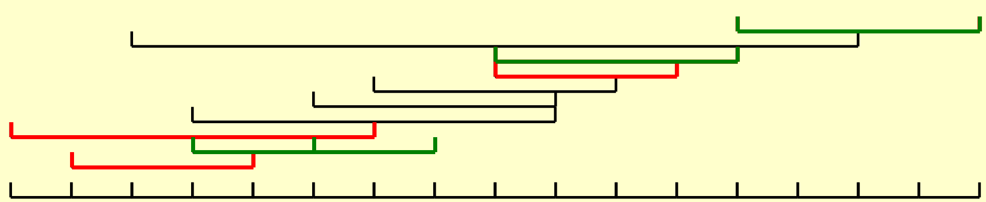
Select a maximum-size subset of mutually compatible activities.

Assume: $f_1 \leq f_2 \leq \ldots \leq f_{n-1} \leq f_n$

〖Example〗

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
	4	5	6	7	9	9	10	11	12	14	16

Discussion 12:
How can we be greedy?



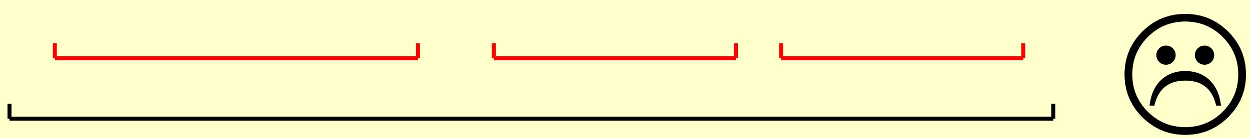
☞ **A DP Solution** $a_1 \;\; a_2 \;\; \ldots \underbrace{a_i \ldots a_k \;\; \ldots a_j}_{S_{ij}} \ldots \; a_n$

$$c_{ij} = \quad c_{ik} + c_{kj} + 1 \quad \texttt{if} \;\; S_{ij} \neq \Phi$$

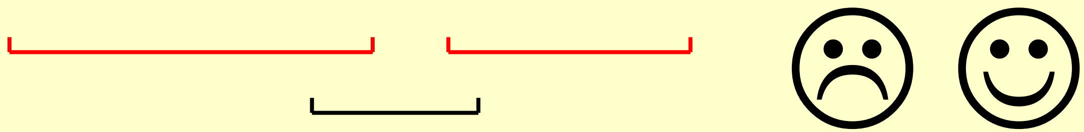
Can we be greedy?

O(N^2)

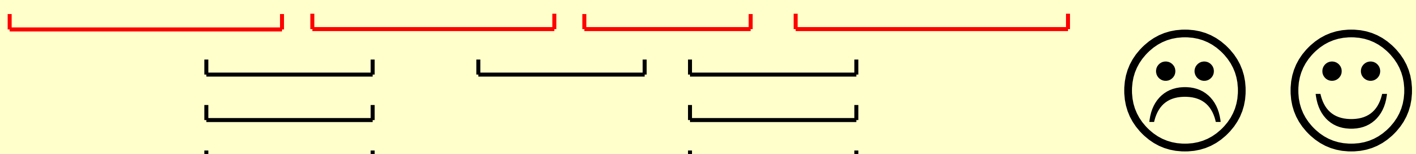
☞ **Greedy Rule 1: Select the interval which *starts earliest***
(but not overlapping the already chosen intervals)



☞ **Greedy Rule 2: Select the interval which is the *shortest*** **(but not overlapping the already chosen intervals)**

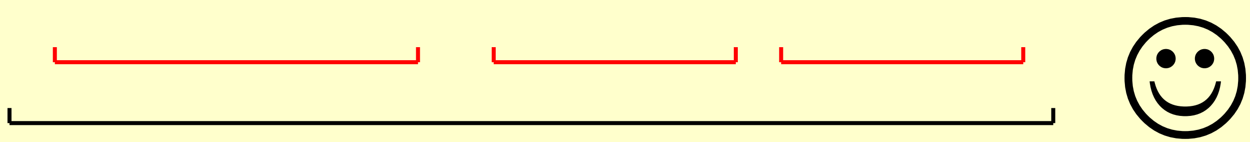


☞ **Greedy Rule 3: Select the interval with the *fewest conflicts* with other remaining intervals** **(but not overlapping the already chosen intervals)**



☞ **Greedy Rule 4: Select the interval which *ends first*** **(but not overlapping the already chosen intervals)**

Resource become free as soon as possible



Correctness:

① **Algorithm gives non-overlapping intervals**

② **The result is optimal**

【Theorem】 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 the optimal solution set, and a_{ef} is the activity in A_k with the earliest finish time.

If a_m and a_{ef} are the same, we are done! Else ……

replace a_{ef} by a_m and get A_k'.

Since $f_m \leq f_{ef}$, A_k' is another optimal solution. ∎

Implementation:

① **Select the first activity; Recursively solve for the rest.**

② **Remove tail recursion by iterations.** $O(N\log N)$

☞ Another Look at DP Solution

$$c_{1,j} = \begin{cases} 1 & \text{if } j = 1 \\ \max\{ c_{1,j-1}, c_{1,k(j)} + 1 \} & \text{if } j > 1 \end{cases}$$

where $c_{1,j}$ is the optimal solution for a_1 to a_j , and $a_{k(j)}$ is the nearest compatible activity to a_j that is finished before a_j.

If each activity has a weight …

$$c_{1,j} = \begin{cases} 1 & \text{if } j = 1 \\ \max\{ c_{1,j-1}, c_{1,k(j)} + w_j \} & \text{if } j > 1 \end{cases}$$

Q1: Is the DP solution still correct?

Q2: Is the Greedy solution still correct?

Elements of the Greedy Strategy

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.

Beneath every greedy algorithm, there is almost always a more cumbersome dynamic-programming solution

Huffman Codes – for file compression

〖Example〗 Suppose our text is a string of length 1000 that comprises the characters *a*, *u*, *x*, and *z*. Then it will take 8000 bits to store the string as 1000 one-byte characters.

We may encode the symbols as *a* = 00, *u* = 01, *x* = 10, *z* = 11. For example, *aaaxuaxz* is encoded as 0000001001001011. Then the space taken by the string with length 1000 will be 2000 bits + space for code table. /* $\lceil \log C \rceil$ bits are needed in a standard encoding where *C* is the size of the character set */

Notice that we have only 4 distinct characters in that string. Hence we need only 2 bits to identify them.

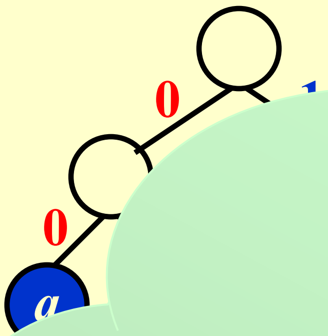
- **frequency** ::= number of occurrences of a symbol.

In string *aaaxuaxz*, $f(a) = 4$, $f(u) = 1$, $f(x) = 2$, $f(z) = 1$.

The size of the coded string can be reduced using variable-length codes, for example, *a* = 0, *u* = 110, *x* = 10, *z* = 111. ➡ 00010110010111

Note: If all the characters occur with the same frequency, then there are not likely to be any savings.

Representation of the original code in a binary tree /* trie */



- **If character C_i is at depth d_i and occurs f_i times, then the cost of the code = $\sum d_i f_i$.**

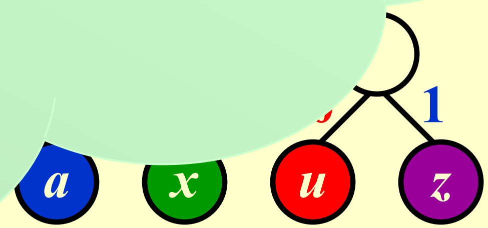
001011)

Now, with

$a = 0$, $u = 110$, $x = 10$, $z = 111$

and the string 00010110010111,

can you decode it?



Discussion 13: What must the tree look like if we are to decode unambiguously?

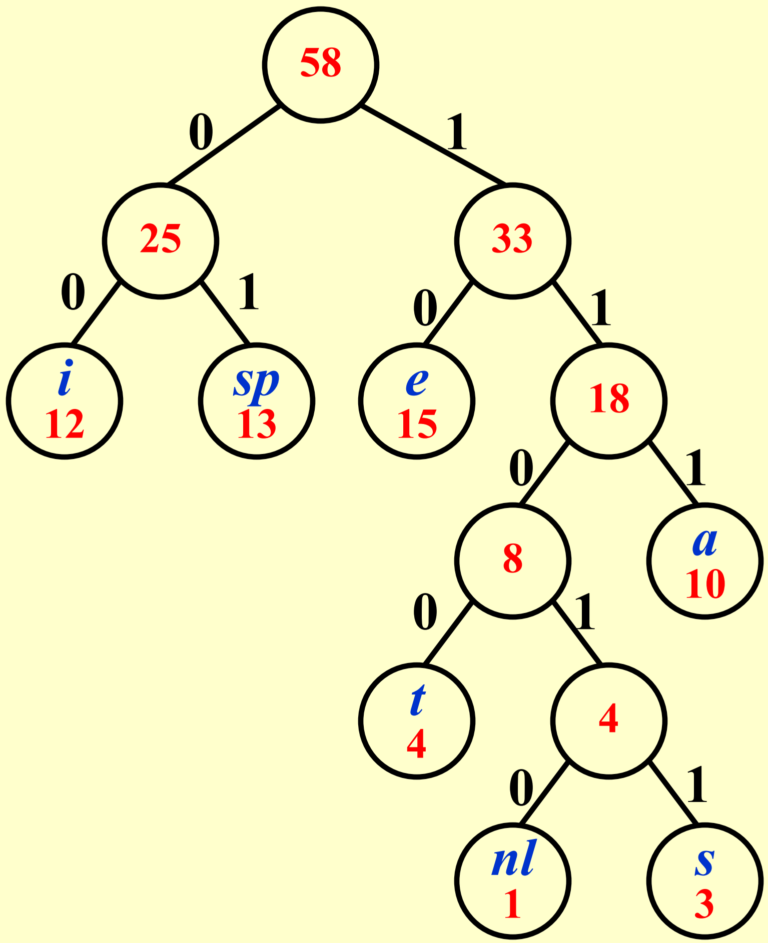
➢ Huffman's Algorithm (1952)

```
void Huffman ( PriorityQueue  heap[ ],  int  C )
{   consider the C characters as C single node binary trees,
    and initialize them into a min heap;
    for ( i = 1; i < C; i++ ) {
       create a new node;
       /* be greedy here */
       delete root from min heap and attach it to left_child of node;
       delete root from min heap and attach it to right_child of node;
       weight of node = sum of weights of its children;
       /* weight of a tree = sum of the frequencies of its leaves */
       insert node into min heap;
   }
}
```

$$T = O(\quad C \log C \quad)$$

【Example】

C_i	*a*	*e*	*i*	*s*	*t*	*sp*	*nl*
f_i	10	15	12	3	4	13	1



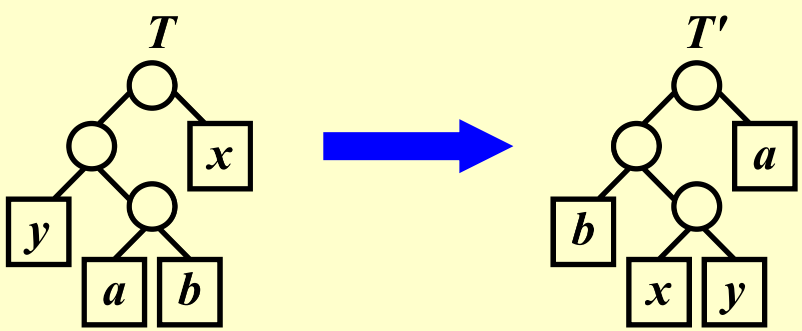
a : 111
e : 10
i : 00
s : 11011
t : 1100
sp : 01
nl : 11010

$$Cost = 3\times10 + 2\times15 + 2\times12 + 5\times3 + 4\times4 + 2\times13 + 5\times1 = 146$$

Correctness:

① The greedy-choice property

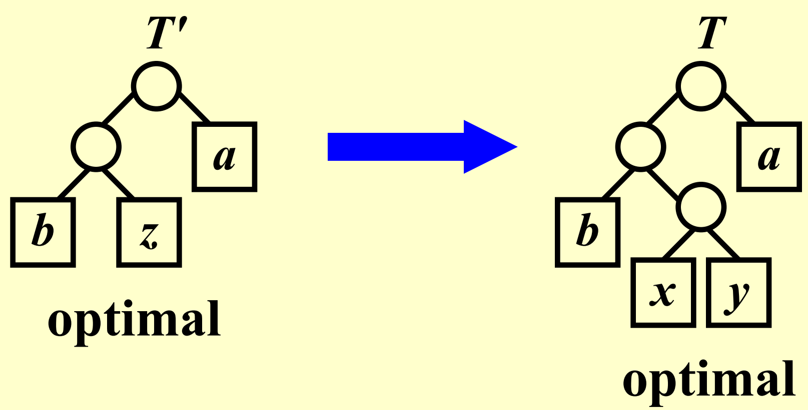
【Lemma】 Let C be an alphabet in which each character $c \in C$ has frequency $c.freq$. Let x and y be two characters in C having the lowest frequencies. Then there exists an optimal prefix code for C in which the codewords for x and y have the same length and differ only in the last bit.



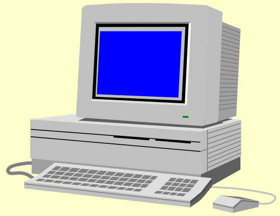
$$\text{Cost}(T') \le \text{Cost}(T)$$

② The optimal substructure property

【Lemma】 **Let C be a given alphabet with frequency $c.freq$ defined for each character $c \in C$. Let x and y be two characters in C with minimum frequency. Let C' be the alphabet C with a new character z replacing x and y, and $z.freq = x.freq + y.freq$. Let T' be any tree representing an optimal prefix code for the alphabet C'. Then the tree T, obtained from T' by replacing the leaf node for z with an internal node having x and y as children, represents an optimal prefix code for the alphabet C.**



By contradiction.



Research Project 4

Huffman Codes (26)

In 1953, David A. Huffman published his paper "*A Method for the Construction of Minimum-Redundancy Codes*", and hence printed his name in the history of computer science. As a professor who gives the final exam problem on Huffman codes, I am encountering a big problem: the Huffman codes are NOT unique. The students are submitting all kinds of codes, and I need a computer program to help me determine which ones are correct and which ones are not.

Detailed requirements can be downloaded from https://pintia.cn/

Reference:

Introduction to Algorithms, 3rd Edition: Ch.16, p. 415-437； ***Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. The MIT Press. 2009***