Algorithmique

Pierre Aboulker, Paul Jeanmaire, Tatiana Starikovskaya

28 septembre 2023

Table des matières

Première partie

Cours 1 - 28/09

1 Organisation

Mail Tatiana : `starikovskaya@di.ens.fr` Homeworks are 30% of the final grade, final (theory from lecture) Textbooks :

- *Introduction to Algorithms* - Cormen, Leiserson, Riverst, Stein
- *Algorithms on strings, trees, and sequences* - Gusfield
- *Approximation Algorithms* - Vazirani
- *Parametrized Algorithms* - Cygan, Fomin, Kowalik, Lokshtanov, Marx, Pilipczuk, Pilipczuk, Saurabh

2 Introduction

Algorithm take Inputs and give an output.

Open Problem 1 (Mersenne Prime). *Find a new prime of form* $2^n - 1$

Algorithms do not depend on the language. Algorithms should be simple, fast to write and efficient. Word RAM model : Two Parts : one with a constant number of registers of w bits with direct access, and one with any number of registers, only with indirect access (pointers). Allows for elementary operations : basic arithmetic and bitwise operations on registers, conditionals, goto, copying registers, halt and malloc. To index the memory storing input of size n with n words, we need register length to verify $w \geq \log n$ Algorithms can always be rewritten using only elementary operations. Complexity :

— $Space(n)$ is the maximum number of memory words used for input of size n

— $Time(n)$ is the maximum number of *elementary* operations used for input of size n

Complexity Notations :

— $f \in \mathcal{O}(g)$ if $\exists n_0 \in \mathbb{N}, c \in \mathbb{R}_+,\ f(n) \leq c \cdot g(n),\ \forall n \geq n_0$

— $f \in \Omega(g)$ if $\exists n_0 \in \mathbb{N}, c \in \mathbb{R}_+,\ f(n) \geq c \cdot g(n),\ \forall n \geq n_0$

— $f \in \Theta(g)$ if $\exists n_0 \in \mathbb{N}, c_1, c_2 \in \mathbb{R}_+,\ c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n),\ \forall n \geq n_0$

3 Data Structures

3.1 Introduction

Way to store elements of a data base that is created to answer frequently asked queries using pre-processing. We care about space used, construction, query and update time. Can be viewed as an algorithm, which analysed on basics. Containers are basic Data Structures, maintaining the following operations :

1. Random Access : given i, access e_i
2. Access first/last element
3. Insert an element anywher
4. Delete any element

3.2 Array

An array is a pre-allocated contiguous memory area of a *fixed* size. It has random access in $\mathcal{O}(1)$, but doesn't allow insertion nor deletion.

Linear Search : given an integer x return 1 if $e_i = x$ else 0.

Algorithme 1 Linear Search in an Array.
Complexity : Time = $\mathcal{O}(n)$ | Space = $\mathcal{O}(n)$

Input$x$

3.3 Doubly Linked List

Memory area that does not have to be contiguous and consists of registers containing a value and two pointers to the previous and next elements. It has random access in $\mathcal{O}(n)$, access/insertion/deletion at head/tail in $\mathcal{O}(1)$.

3.4 Stack and Queue

Stack : Last-In-First-Out data structure, abstract data structure. Access/insertion/deletion to top in $\mathcal{O}(1)$.

Open Problem 2 (Optimum Stack Generation)**.** *Given a finite alphabet Σ and $X \in \Sigma^n$. Find a shortest sequence of stack operations push, pop, emit that prints out X. You must start and finish with an empty stack. Current best solution is in $\tilde{\mathcal{O}}(n^{2.8603})$.*

Queue : First-In-First-Out abstract data structure. Access to front, back in $\mathcal{O}(1)$, deletion and insertion at front and back in $\mathcal{O}(1)$.

Algorithme 2 Insertion in a Doubly Linked List
Complexity : $\mathcal{O}(1)$

Input L, x
$x.next \leftarrow L.head$
if $L.head \neq NIL$ **then**
  $L.head.prev \leftarrow x$
end if
$L.head \leftarrow x$
$x.prev = Nil$

4 Approaches to algorithm design

Solve smalle sub-problems to solve a large one.

4.1 Dynamic Programming

Break the problem into many closely related sub-problems, memorize the result of the sub-problems to avoid repeated computation

Examples :

Algorithme 3 Recursive Fibonacci Numbers
Complexity : Exponential

RFibo(n) :
Input n
if $n \leq 1$ **then**
  return n
end if
return RFibo($n-1$) + RFibo($n-2$)

Algorithme 4 Dynamic Programming Fibonacci Numbers
Time = $\mathcal{O}(n)$ | Space = $\mathcal{O}(n)$

Input n
$Tab \leftarrow zeros(n)$ ▷ $zeros(n)$ returns a n-array of zeros.
$Tab[0] \leftarrow 0$
$Tab[1] \leftarrow 1$
for $i \leftarrow 2$ to n **do**
  $Tab[i] = Tab[i-1] + Tab[i-2]$
end for
return Tab[n]

Levenshtein Distance between two strings can be computed in $\mathcal{O}(mn)$ instead of exponential time. Based on `https://arxiv.org/pdf/1412.0348.pdf`, this is the best one can do. RNA folding : retrieving the 3D shape of RNA based on their representation as strings. Currently, we know it is possible to find $\mathcal{O}(n^3)$, in $\tilde{\mathcal{O}}(n^{2.8606})$ and if *SETH* is true, there is no $\mathcal{O}(n^{\omega-\varepsilon})$. We know $\omega \in [2, 2.3703)$

Open Problem 3. *Is there a better Complexity for RNA folding ? What is the true value of* ω *?*

Knapsack problem : An optimization problem with bruteforce complexity $\mathcal{O}(2^n)$.

4.2 Greedy Techniques

Problems solvable with the greedy technique form a subset of those solvable with DP. Problems must have the optimal substrcture property. Principle : choosing the best at the moment.

Algorithme 5 Knapsack : Dynamic Programming
Time = $\mathcal{O}(nW)$ | Space = $\mathcal{O}(nW)$

```
Input W, w, v                                  ▷ Capacity, weight and values vectors.
KP = zeros(n, W)
for i ← 0 to n do
    KP[i, 0] = 0
end for
for w ← 0 to W do
    KP[0, w] = 0
end for
for i ← 0 to n do
    for w ← 0 to W do
        if w < w_i then
            KP[i, w] ← KP[i − 1, w]
        else
            KP[i, w] = max { KP[i − 1, w]
                           { KP[i − 1, w − w_i] + v_i
        end if
    end for
end for
return KP[n, W]
```

Example : The Fractional Knapsack Problem
Algorithm : Iteratively select the greatest value-per-weight ratio.

Théorème 4.2.1. *This algorithm returns the best solution, in time $\mathcal{O}(n \log n)$*

By contradiction. Suppose we have $\frac{v_1}{w_1} \geq \ldots \geq \frac{v_n}{w_n}$. Let $ALG = p = (p_1, \ldots, p_n)$ be the output by the algorithm and $OPT = q = (q_1, \ldots, q_n)$ be optimal.
Assume that $OPT \neq ALG$, let i be the smallesst index such $p_i \neq q_i$. There is $p_i > q_i$ by construct. Thus, there exists $j > i$ such that $p_j < q_j$. We set $q' = (q_1^{'}, \ldots, q_n^{'}) = (q_1, \ldots, q_{i-1}, q_i + \varepsilon, q_{i+1}, \ldots, q_j - \varepsilon\frac{w_i}{w_j}, \ldots, q_n)$
$q^{'}$ is a feasible solution : $\sum_{i=1}^{n} q_i^{'} \cdot w_i = \sum_{i=1}^{n} q_i w_i \leq W$
Yet, $\sum_{i=1}^{n} q_i^{'} \cdot v_i > \sum_{i=1}^{n} q_i \cdot v_i$, ce qui contredit la ■

Deuxième partie

Devoir 1