Introduction à l'algorithmique **Hiver 2018**

IFT2125-6001 TA: Stéphanie Larocque

Démonstration 9

À partir des corrigés de Maelle Zimmermann

1

Question: Topological sorting of the vertices of the next oriented and acyclic graph.

Solution: See jupyter notebook / python file for code

All you need to do is go through the graph in depth and number each top at the very end of its exploration, then reverse the order obtained.

The order `postnum` obtained after the depth course is: 24, 8, 12, 4, 6, 2, 3, 1.

2

Question: Calculate the expected number of rolls of two dice before getting a pair of 1. Ditto get two numbers that are 7.

Solution: This problem amounts to calculating the expectation of a geometrical law with parameter p. Indeed the geometric law considers the problem of repeating a proof whose probability of success is p, and the geometric random variable denotes the rank of the first success.

Let p be the probability of obtaining the desired event during a throw. The expected number of n throws is given by:

$$\begin{aligned} n &= 1 \cdot p + 2(1-p)p + 3(1-p)^2 p + ... \\ &= \sum_{k=1}^{\infty} k(1-p)^{k-1} p. \end{aligned}$$

Or we have for $x \in [0, 1)$:

$$f(x) = \sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \qquad \text{(série géométrique)}$$
$$f'(x) = \sum_{k=1}^{\infty} k x^{k-1} = \frac{1}{(1-x)^2}.$$

Comme $n = p \sum_{k=1}^{\infty} k(1-p)^{k-1}$, nous obtenons

$$n = \frac{p}{(1-(1-p))^2} = \frac{1}{p}.$$

Since the probability of getting a pair of 1 is given by $p = 1/36$, the expected number of throws is 36. Similarly, the probability of getting a pair of numbers that is 7 is $1/6$, so the expected number of throws is 6.

3

Question: Give a Monte Carlo algorithm to determine if two polynomials p and q on $\mathbb{Z}$, presented by arithmetic circuits, are equal. We can use without proof the fact that the degree of a polynomial is limited by the height of the circuit.

Solution: Consider the following example (image missing, see notes from class):

The circuits C_p et C_q respectively calculate the following polynomials:

$$p(x_1, x_2, x_3) = ((6 + (x_1x_2)) - (x_1x_2)(7 - x_3)) + 3(x_1x_2)(7 - x_3)$$
$$q(x_1, x_2, x_3) = (6 + (x_1x_3)) - 3(x_2 - x_3)(7x_2 - 2)$$

To determine if $p = q$, we ask $r = p - q$ and we test if r is zero. In other words, we test whether the circuit $C_r = C_p - C_q$ calculates the polynomial 0. To do this, we choose a random assignment of $x_1, x_2, ..., x_m$, that is to say a set of numbers $a_1, a_2, ..., a_m \in \mathbb{Z}$ and we evaluate $r(a_1, a_2, ..., a_m)$. If $r(a_1, a_2, ..., a_m) \neq 0$, then it is impossible for r to be the polynomial 0. On the other hand if $r(a_1, a_2, ..., a_m) = 0$, it is still possible that r is not polynomial 0, if we chose a root of r.

In the case where $m = 1$, that is to say that the polynomial is univariate, it is enough to choose a_1 in a set of size $k \cdot \deg(r)$ to have a probability of error at most $1/k. Indeed$, r $hasatmost$ $\deg(r)$ roots.

In these examples, btw, $\deg(r) issimplythedegreeofthepolynomial.$

In the multivariate case, the number of roots of r is no longer limited by $\deg(r)$. In fact, r can have an infinity of roots. Indeed, the polynomial $p(x_1, x_2, x_3) = x_1$ has an infinity of roots of the form $(0, x_2, x_3)$. We will see from a lemma that the strategy of the univariate case is still generalized to multivariate polynomials.

Let h_p and h_q be the height of C_p and C_q respectively. We give the following algorithm:

1. Poser $S = \{1, 2, ..., 4 \cdot \max(2^{h_p}, 2^{h_q})\}$
2. Choisir aléatoirement et indépendamment $a_1, a_2, ..., a_m \in$ uniforme(S)
3. Poser $r = p - q$
4. Si $r(a_1, a_2, ..., a_m) = 0$ alors retourner oui, sinon retourner non.

When the algorithm responds no, we are sure that p and q are not equal. When the algorithm responds yes, it is possible that p and q are not equal and that the algorithm

is wrong. To evaluate the probability that the algorithm is wrong in the multivariate case, we need the following lemma:

Lemme de Schwartz-Zippel: let $p \in \mathbb{Q}[x_1, x_2, ..., x_m]$ a nonzero polynomial. Let $S \subset \mathbb{Q}$ be a finite subset and $a_1, a_2, ..., a_m$ chosen in S uniformly and independently. So,

$$\Pr[p(a_1, a_2, ..., a_m) = 0] \leq \frac{\deg(p)}{|S|}. \tag{1}$$

Evidence: The lemma is demonstrated by induction on the number of variables.

Base Case (m=1): in this case, p is univariate and has at most $\deg(p)$ roots, so we get

$$\Pr[p(a_1) = 0] \leq \frac{\deg(p)}{|S|}$$

.

Induction Step: we can put each power of x_1 highlighted in p and get the next rewrite.

$$p(x_1, x_2, ...x_m) = \sum_{i=0}^{d} x_1^i p_i(x_2, ...x_m)$$

Since p is non-zero, there is at least one i such that p_i is non-zero. Let i be the largest such index, then by induction hypothesis:

$$\Pr[p_i(a_2, ...a_m) = 0] \leq \frac{\deg(p_i)}{|S|} \leq \frac{\deg(p) - i}{|S|} \tag{2}$$

If $p_i(a_2, ...a_m) \neq 0$, then $p(x_1, a_2, ..., a_m)$ is a one-variable polynomial (x_1) of degree i, because i is the largest degree of x_1 non-zero in p. Thus by induction hypothesis:

$$\Pr[p(a_1, a_2, ..., a_m) = 0 \| _i (a_2, ...a_m) \neq 0] \leq \frac{i}{S} \tag{3}$$

If we call A the event $p(a_1, a_2, ..., a_m) = 0$ and B the event $p_i(a_2, ...a_m) = 0$, then we have

$$\begin{aligned}
\Pr[A] &= \Pr[A|B]\Pr[B] + \Pr[A|B^c]\Pr[B^c] \\
&\leq \Pr[B] + \Pr[A|B^c] \\
&\leq \frac{\deg(p) - i}{|S|} + \frac{i}{|S|} \quad \text{par (2) et (3)} \\
&= \frac{\deg(p)}{|S|}.
\end{aligned}$$

Which concludes the proof.

Now consider the case where the algorithm above answers yes. Let w be the probability that the algorithm is wrong, in other words that $r(a_1, a_2, ...a_m) = 0$ knowing that r is non-zero. We have:

$$\begin{aligned}
w &\leq \frac{\deg(r)}{|S|} && \text{par (??)} \\
&\leq \frac{\max(\deg(p), \deg(q))}{|S|} && \text{car } \deg(r) \leq \max(\deg(p), \deg(q)) \\
&= \frac{\max(\deg(p), \deg(q))}{4 \cdot \max(2^{h_p}, 2^{h_q})} && \text{par déf. de } S \\
&\leq \frac{\max(\deg(p), \deg(q))}{4 \cdot \max(\deg(p), \deg(q))} && \text{car } \deg(p) \leq 2^{h_p} \text{ et } \deg(q) \leq 2^{h_q} \\
&= \frac{1}{4}
\end{aligned}$$

Thus the algorithm errs with probability at most $\frac{1}{4}$ when it answers yes.

Execution time analysis: We analyze the execution time of the algorithm according to the size of the input, ie according to the degree of p and q and according to the number of variables m. If you can draw a coin in $O(1)$, you can select a number a_i in S per time binary search $\log_2 |S| = \log_2(4 \cdot \max(2^{h_p}, 2^{h_q})) = 2 + \max(h_p, h_q)$ coin toss. The execution time of the random selection algorithm is therefore in $O(m \cdot \max(h_p, h_q))$. Evaluating the circuit r is done in polynomial time. The

Execution time of the algorithm is therefore polynomial in the size of the input.

4

Question: Give an algorithm that calculates the d edit distance, also called Levenshtein distance, between two words. More precisely, let u and v be two words, then $d(u, v)$ is the minimum number of letters that must be removed, inserted or substituted to go from u to V.

For example, we can go from *table* to *stage* in 3 operations:

$$\begin{aligned} table &\to tale && \text{(remove } b\text{)} \\ &\to stale && \text{(insert } s\text{)} \\ &\to stage && \text{(substuitute } l \text{ pour } g\text{)} \end{aligned}$$

Solution: We construct a D array of size $|u|+1 \times |v|+1$ such that $D[i][j]$ is the distance between the subwords $u[1...i]$ and $v[1...j]$.

Note first that $D[i][0] = i$ for all $0 \leq i \leq |u|$, and $D[0][j] = j$ for all $0 \leq j \leq |v|$. Indeed, for an empty word ϵ, we have $d(x, \epsilon) = d(\epsilon, x) = |x|$ because the minimal edition consists in the first case to erase all the letters x ($|x|$ deletions) and in the second case to add all the letters of x to the empty word ($|x|$ insertions). More generally, to pass from a word $u[1...i]$ to a word $v[1...j]$, there are three possibilities:

- remove $u[i]$ then go from $u[1...i-1]$ at $v[1...j]$
- Skip from $u[1...i]$ at $v[1...j-1]$ then insert $v[j]$
- Skip from $u[1...i-1]$ at $v[1...j-1]$ the insert $u[i]$ for $v[j]$ (if these letters are different)

This gives the rule:

$$D[i][j] = \min(\underbrace{D[i-1][j]+1}_{\text{suppression}}, \underbrace{D[i][j-1]+1}_{\text{insertion}}, \underbrace{D[i-1][j-1]+b_{i,j}}_{\text{substitution}})$$

or $b_{i,j} = 0$ si $u[i] = v[j]$, and 1 if not.

Here is an implementation which takes time in $\Theta(|u||v|)$:

```
def d_tab(u, v):
    D = [[0]*(len(v) + 1) for _ in range(len(u) + 1)]

    for i in range(len(u)+1):
        D[i][0] = i

    for j in range(len(v)+1):
        D[0][j] = j

    for i in range(1, len(u) + 1):
        for j in range(1, len(v) + 1):
            b = 1 if u[i] != v[j] else 0
            D[i][j] = min(D[i-1][j] + 1, D[i][j-1] + 1, D[i-1][j-1] + b)
    return D

def d(u, v):
    return d_tab(u, v)[-1][-1]
```

5

Question: Let $u = AGGAGGA$ and $v = AAGCTAAG$. Identify all alignments between u and v of cost $d(u, v)$, that is, all optimal alignments.

Solution: Le tableau des distances est:

	ϵ	A	A	G	C	T	A	A	G
ϵ	0	1	2	3	4	5	6	7	8
A	1	0	1	2	3	4	5	6	7
G	2	1	1	1	2	3	4	5	6
G	3	2	3	1	2	3	4	5	5
A	4	3	2	2	2	3	3	4	5
G	5	4	3	2	3	3	4	4	4
G	6	5	4	3	3	4	4	5	4
A	7	6	5	4	4	4	4	4	5

So the distance of Levenshtein between the two words is 5. to find which alignment and therefore which sequence of operations gives this cost, it is necessary to go up the table since the box |T [-1] [- 1] | following the rules of the algorithm.

Le chemin du haut correspond à l'alignement de séquence suivant:

A	A	G	C	T	A	A	G	-
-	A	G	-	G	A	G	G	A

which corresponds to making two insertions (A and C), two substitutions (G for T and G for A) and a deletion (A). In all we get 6 paths, and the 5 others correspond to the following sequence alignments:

A	A	G	C	T	A	A	G
A	G	G	A	G	G	A	-

A	A	G	C	T	A	A	G	-
A	G	G	-	-	A	G	G	A

A	A	G	C	T	A	A	G	-
A	-	G	-	G	A	G	G	A

A	A	G	C	T	A	A	G	-
-	A	G	G	-	A	G	G	A

A	A	G	C	T	A	A	G	-
A	-	G	G	-	A	G	G	A