École Polytechnique Fédérale de Lausanne
Prof. Eisenbrand

April 8, 2025

**Discrete Optimization** (Spring 2025)

# Assignment 8

1) Let $a, e, k \in \mathbb{N}$ be three given natural numbers.

(a) Argue that $a^{2^k}$ can be computed using $\Theta(k)$ multiplications.

(b) How many bits ($\Theta$-notation) does $a^{2^k}$ have?

(c) Let $(e_0, \dots, e_l)$ be the bit representation of $e$, that is, $e = \sum_{i=0}^{l} e_i 2^i$ with $e_i \in \{0,1\}$ for $i = 0, \dots, l$. Complete the following algorithm by replacing each occurrence of three question marks (???) so that it computes $a^e$ using $O(l)$ many arithmetic operations.

```
$E = 1$
$S = a$
For ($i = 0$ to $l$)
  if ($e_i == 1$)
    $E = E \cdot ???$
  $S = ???$
return ???
```

(d) Show that for given $a, e, N \in \mathbb{N}$ one can compute $a^e (\mod N)$ in time polynomial in the binary encoding length of $a, e$ and $N$.

(e) Let $a, b, c, N \in \mathbb{N}$ be given and suppose that $N$ is a prime number. Show that $a^{b^c} (\mod N)$ can be computed in polynomial time in the binary encoding length of $a, b, c$ and $N$. You may use Fermat's little theorem: $a^N \equiv a (\mod N)$.

**Solution:**

(a) Start with $a(0) = a$, and compute $a(i) = a(i-1)a(i-1)$ for $i \in [k]$. By induction

$$a(k) = a^{2^{k-1}} a^{2^{k-1}} = a^{2^k}.$$

Thus, $a^{2^k}$ has been obtained by performing $k$ multiplications.

(b) This has $\log(a^{2^k}) = \Theta(2^k \log a)$ many bits.

(c) One has $E = E \cdot S$, $S = S \cdot S$ and return $E$.

(d) We use the same algorithm as in part (c), but do operations modulo $N$, in particular we replace $E = E \cdot S$ with $E = E \cdot S (\mod N)$ and $S = S \cdot S$ with $S = S \cdot S (\mod N)$. Observe that the modulo can be obtained by performing division with remainder. The correctness of the algorithm follows from the fact that for any $a, b \in \mathbb{N}$, $ab \equiv (a (\mod N))(b (\mod N))(\mod N)$. The algorithm performs $O(l)$ steps, where $l$ is the size of $e$, and in each step we perform division with remainder on $E \cdot S$ (or $S \cdot S$) and $N$: this takes time polynomial in the binary encoding length (size) of the input, in particular of $a, e, N$ (notice that the value of $A (\mod N)$ has size at most the size of $N$, irrespectively of the size of $A$). Hence the total number of operations performed is polynomial in the size of $a, e, N$.

(e) First, by applying (d) we can compute $e$ such that $b^c \equiv e (\bmod N-1)$ in time polynomial in the binary encoding length of $b, c, N$. Note that $e$ is bounded by $N$. Then by Fermat's little theorem $a^{N-1} \equiv 1 (\bmod N)$, we have $a^{b^c} \equiv a^e (\bmod N)$. Finally, by applying (d) again, we can compute $a^e (\bmod N)$ in time polynomial in the binary encoding length of $a, N$.

2) There are n types of animals, and you want to assign them to two stables. Unfortunately, some animals would eat other animals when left unattended. Therefore you need to assign the animals carefully. There are m relations of the form "u eats v", where u and v are animals. Find an $O(n+m)$ algorithm that decides whether there is an assignment of animals to the two stables such that no animal eats another one of the same stable, and outputs a feasible assignment.
*Hint: Breadth-first-search might be useful.*

**Solution:**
Consider the following directed graph $D = (V, A)$, where each node in $V$ corresponds to an animal, and we have an arc $(u, v)$ for each relation "u eats v". Observe an assignment of the animals to the two stables is feasible if and only if the corresponding partition of the nodes into sets $V_1$ and $V_2$ we have that $(u, v) \notin A$ for all $u, v \in V_1$ and $(u, v) \notin A$ for all $u, v \in V_2$. In other words, there is a feasible assignment if and only if the underlying undirected graph $G = (V, E)$, where $E := \{\{u, v\} : (u, v) \in A\}$ is bipartite. Consider the following algorithm:

```
$FULLBFS(G = (V, E))$
  $D[v] \leftarrow \infty \;\; \forall v \in V$
  for all $v \in V$
    if $D[v] = \infty$
      $D[v] \leftarrow 0$
      $BFS(G, v, D)$
  return $D$
```

We run this algorithm on the graph $G$. Let $D$ be the output. We define $V_1 := \{v \in V : D[v] \text{ is odd}\}$ and $V_2 := \{v \in V : D[v] \text{ is even}\}$. We claim that the assignment given by $V_1$ and $V_2$ is a feasible solution for our problem if and only if there exists a feasible solution. With the observation from above, it is sufficient to show that $V_1$ and $V_2$ are feasible if and only if $G$ is bipartite. Trivially if $V_1$ and $V_2$ are feasible, then $G$ is bipartite, since $V_1$ and $V_2$ give a bipartite partition of the nodes. Now assume that $V_1$ and $V_2$ are infeasible, i.e. there is an edge $e \in E$ such that $e \subseteq V_1$ (or $e \subseteq V_2$. In the latter case we swap the roles of V1 and V2). We need to show that G is not bipartite. As seen in the lecture, it is sufficient to show that G contains an odd cycle. Let $e = (u, w)$. Hence w is reachable from u and vice versa. Hence, u and w got their D- label assigned in the same run of BFS in Line 6. Thus there is a node v such that there is a v,u-path P of length $D[u]$ and a v, w -path P' of length $D[w]$. Since u and w both belong to V1, we have $D[u] \equiv D[w] \mod 2$, and therefore $D[u] + D[v]$ is even. Let $P \Delta P'$ be the set of edges that are contained in either P or P' (but not in both of them). Note that since $D[u] + D[v]$ is even, the number of arcs in this set even as well. Moreover $e$ is contained in neither of the paths (otherwise the algorithm would not have assigned u and w to the same Vi ). We conclude that $(P \Delta P') \cup \{e\}$ is an odd cycle in G.

3) Let $M_{2^k}$ be a matrix of order $n := 2^k$, where $k \in \mathbb{N}_{>0}$ such that it is recursively defined as follows:

$$M_{2^k} = \begin{pmatrix} M_{2^{k-1}} & M_{2^{k-1}} \\ M_{2^{k-1}} & -M_{2^{k-1}} \end{pmatrix}$$

and $M_1 = [1]$, a $1 \times 1$ matrix. Prove that $|\det(M_n)| = n^{n/2}$, i.e. that the Hadamard bound is tight.

**Solution:**
We prove the statement by induction on $k \in \mathbb{N}_0$ that $M_{2^k}^2 = 2^k I_{2^k}$. For $k = 0$ one has $M_0^2 = M_1 = [1] = 2^0 I_{2^0}$. Assume that the statement holds for $k$ and prove it for $k+1$. Then

$$M_{2^{k+1}}^2 = \begin{pmatrix} M_{2^k} & M_{2^k} \\ M_{2^k} & -M_{2^k} \end{pmatrix} \begin{pmatrix} M_{2^k} & M_{2^k} \\ M_{2^k} & -M_{2^k} \end{pmatrix} = \begin{pmatrix} 2M_{2^k}^2 & 0 \\ 0 & 2M_{2^k}^2 \end{pmatrix} = \begin{pmatrix} 2 \cdot 2^k I_{2^k} & 0 \\ 0 & 2 \cdot 2^k I_{2^k} \end{pmatrix} = 2^{k+1} I_{2^{k+1}}$$

where we have used the induction hypothesis in the second to last equality. By using the statement above we have that $\det(M_n^2) = \det(nI_n) = n^n$ so $|\det(M_n)| = n^{n/2}$.

4) The determinant of a matrix $A \in \mathbb{R}^{n\times n}$ can be computed by the recursive formula

$$\det(A) = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(A_{1j}),$$

where $A_{1j}$ is the $(n-1) \times (n-1)$ matrix that is obtained from $A$ by deleting its first row and j-th column. This yields the following recursive algorithm.
Input: $A \in \mathbb{R}^{n\times n}$
Output: $\det(A)$

```
if ($n = 1$)
return $a_{11}$
else
  $d := 0$
  for $j = 1, ..., n$
    $d := (-1)^{1+j} \det(A_{1j}) + d$
  return $d$
```

Let $A \in \mathbb{R}^{n\times n}$ and suppose that the $n^2$ components of $A$ are pairwise different.

(a) Suppose that $B$ is a matrix that can be obtained from $A$ by deleting the first k rows and k of the columns of A. How many (recursive) calls of the form $\det(B)$ does the algorithm create?

(b) How many different submatrices can be obtained from $A$ by deleting the first k rows and some set of k columns? Conclude that the algorithm remains exponential, even if it does not expand repeated subcalls.

**Solution:**
a) Let $i_1, ..., i_k$ be the indices of the columns of $A$ that were removed to obtain $B$. We have to count the number of nodes of the form $\det(B)$ in the recursion tree of the algorithm. Each node of the tree can be identified by its level (nodes of the form $\det(A_{ij})$ are at level i) and a sequence of column indices representing the columns of $A$ that are not columns of the submatrix called by the node. Hence the nodes of the form $\det(B)$ are at level $k$ of the tree, and their sequences are permutations of $i_1, ..., i_k$ (note that there is only one submatrix of $A$ equal to $B$ since all the entries are pairwise different). Hence there are $k!$ such nodes.
b) The number of such submatrices is clearly $\binom{n}{k}$, which is exponential for instance for $k = n/2$, we have that:

$$\binom{n}{n/2} = \frac{n \cdot \ldots (n/2+1)}{n/2 \cdot \ldots \cdot 1} \geqslant 2^{n/2}$$

such that even if the algorithm calls $\det(B)$ only once for any submatrix $B$, the algorithm remains exponential.