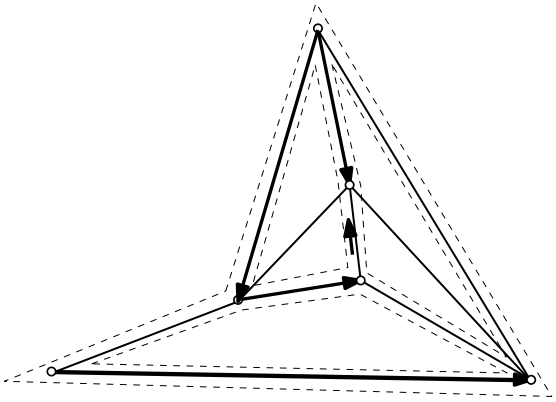
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute of Theoretical Computer Science
Mohsen Ghaffari, Angelika Steger, David Steurer, Emo Welzl, Peter Widmayer

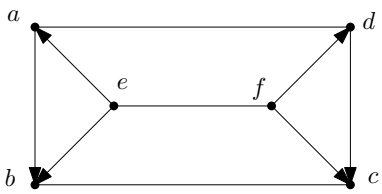
Algorithms, Probability, and Computing **Solutions KW49** **HS18**

Solution 1

(a) There are six nice cycles (four of length 4 and two of length 6). The orientation of one edge (the one which is oriented in the following figure, but not on the exercise sheet) is determined by this. Two more edges can be oriented arbitrarily (not part of any nice cycle), and there are two possibilities to orient the remaining three edges (fixing the orientation of any one of those edges determines the orientations of the other two).



(b) An orientation of a graph $H = (V_H, E_H)$ is Pfaffian if every cycle is oddly oriented, where a cycle C with vertex set V_C is nice if $H[V_H \setminus V_C]$ has a perfect matching. Since F has no even length cycles (and therefore no nice cycles) the given orientation is a Pfaffian orientation of F.



Observe that G has nice cycles $(abcd)$, $(aefd)$ and $(bcfe)$. W.l.o.g. we orient $\{a, d\}$ as (a, d), i.e., from a to d, (the other case is symmetric). Then for the orientation of $(aefd)$ to be odd we must orient $\{e, f\}$ as (f, e). Now for $(abcd)$ to be oddly oriented we must orient $\{b, c\}$ as (c, b). Then $(bcfe)$ is not oddly oriented.

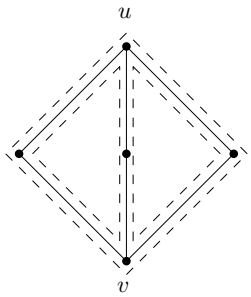


Figure 1: $K_{2,3}$ with u and v; all even cycles indicated with dashed lines.

(c) First, consider the graph $K_{2,3}$. Let u and v denote the two vertices in the smaller partition (i.e., the maximal independent set of size 2) of $K_{2,3}$ (cf. Figure 1). We have three even cycles in $K_{2,3}$ every pair of which shares a common u-v-path. If the two edges on such a path are oriented in the same direction, they contribute an even number of edges in the opposite direction to every oriented cycle that contains them (cf. Figure 2, left); otherwise, i.e. if they are oriented in differing directions, they contribute an odd number (cf. Figure 2, right). In our case, every

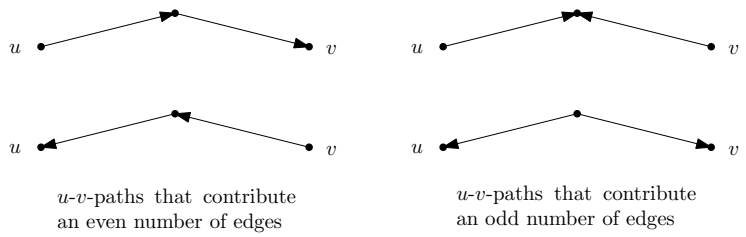


Figure 2: Parity of paths

cycle consists of two such paths. If we want a cycle to be oddly oriented, one of the two paths it consists of needs to contribute an even number of edges oriented in the opposite direction, the other an odd number of oddly oriented edges. But out of the three u-v-paths in $K_{2,3}$, at least two must have the same parity (by the pigeon-hole principle). Hence there is a cycle consisting of these two paths of equal parity that is not oddly oriented. This proves that we cannot orient a $K_{2,3}$ in such a way that all even cycles in it are oddly oriented, and hence we cannot do so in any graph that contains a $K_{2,3}$ (because the property that all even cycles are oddly oriented is preserved with respect to taking subgraphs). □

(d) If we add a vertex w to $K_{2,3}$ such that we obtain $K_{3,3}$, each of the three cycles considered in (c) becomes a nice cycle, as now the one vertex not covered by cycle C has an edge to w, so $K_{3,3} \setminus V_C$ has a perfect matching. But we have already shown in (c) that not all of the three cycles in $K_{2,3}$ can be oddly oriented at the same time. It follows that in every orientation of $K_{3,3}$, there is a nice cycle that is not oddly oriented. □

(e) We know from Lemma 5.5 that if every nice cycle in G is oddly oriented in $\vec{G}$, then

$\det(A_S(\vec{G})) = \mathrm{pm}(G)^2$, i.e. $\vec{G}$ is Pfaffian. What about the other direction?

Assume that $\vec{G}$ is Pfaffian and that there is a nice cycle C_E in G that is evenly oriented in $\vec{G}$. We want to show that under these assumptions, $\det(A_S(\vec{G})) \neq \mathrm{pm}(G)^2$. We know (cf. last part of proof of Lemma 5.4,) that $\det(A_S(\vec{G})) = |IE|$ iff for all $\pi \in IE$, $\mathrm{sign}(\pi)a(\pi) = 1$. And as $|IE| = \mathrm{pm}(G)^2$, it is sufficient to show that there is a $\pi \in IE$ with $\mathrm{sign}(\pi)a(\pi) \neq 1$ to reach our goal.

We can construct a permutation π with the desired property as follows: Let $V(C_E) = \{i_1, \ldots, i_k\}$ be the vertices of C_E numbered in the order of their appearance on the cycle. Put $c_E : i_1 \mapsto i_2 \mapsto \ldots \mapsto i_k \mapsto i_1$ into π. Furthermore, choose a perfect matching P on $G\,[V(G) \backslash V(C_E)]$, which is possible as C_E is a nice cycle. For every edge $f = \{i, j\} \in E(P)$, put the cycle $c_f : i \mapsto j \mapsto i$ into π.

It is easy to check that $\pi \in IE$, and thus it holds for π that

$$\mathrm{sign}(\pi)a(\pi) = \prod_{c \text{ cycle in } \pi} (-a[c]).$$

As noted thereafter, for all 2-cycles c_f, $-a[c_f] = 1$. So $\mathrm{sign}(\pi)a(\pi) = 1$ iff $a[c_E] = -1$. But we know that $a[c] = -1$ if the undirected counterpart C of c is oddly oriented in $\vec{G}$, which is by assumption not the case for c_E. Hence we have found a π with $\mathrm{sign}(\pi)a(\pi) \neq 1$. It follows that under the assumptions made (i.e., that there is a nice cycle in G that is evenly oriented in $\vec{G}$), $\det(A_S(\vec{G})) \neq \mathrm{pm}(G)^2$. We can conclude that in order for $\vec{G}$ to be Pfaffian, every nice cycle in G must be oddly oriented in $\vec{G}$. □

Solution 2

We provide a strategy which decreases the number of remaining items in each phase until we only have the item with the maximum value left. Let k_t for $t \geq 1$ denote the number of remaining items at the end of the t-th phase. In the first phase, we do a simple step of comparing $n/2$ consecutive pairs to reduce the number of remaining items to at most $n/2$; thus, $k_1 = n/2$. In the t-th phase for $t \geq 2$ we partition the remaining k_{t-1} items into ${k_{t-1}}^2/n$ many groups of size n/k_{t-1}. Then, we apply the algorithm from the lecture on each group separately. Note that we require $(n/k_{t-1})^2$ processors for each group, which implies that

$$\frac{k_{t-1}^2}{n} \cdot \frac{n^2}{k_{t-1}^2} = n$$

processors are needed. Furthermore, recall that each of these phases can be done in $\mathcal{O}(1)$ time-steps. Therefore, by using only n processors in the t-th phase we drop the number of remaining items from k_{t-1} to k_t in $\mathcal{O}(1)$ time-steps. We have $k_t = \frac{k_{t-1}^2}{n}$ for $t \geq 2$, where $k_1 = n/2$. Now, we prove that $k_t = \frac{n}{2^{2^{t-1}}}$ for $t \geq 1$ by induction. The base case

holds because $k_1 = n/2 = n/2^{2^0}$. As the induction hypothesis, assume that $k_t = \frac{n}{2^{2^{t-1}}}$ for some $t \geq 1$. For the inductive step, we have

$$k_{t+1} = \frac{k_t^2}{n} \overset{\text{I.H.}}{=} \frac{(\frac{n}{2^{2^{t-1}}})^2}{n} = \frac{n}{2^{2^t}}.$$

Furthermore, $\frac{n}{2^{2^{t-1}}} \leq 1$ for $t \geq \log\log n + 1$. Thus, after at most $\log\log n + 1$ phases, we are left with only one item, namely the maximum one. Since each phase requires $\mathcal{O}(1)$ time-steps, the run time of this algorithm is in $\mathcal{O}(\log\log n)$.

Solution 3

Based on Brent's principle, if an algorithm does x total work and has depth t (i.e., critical path of length t), then using p processors, this algorithm can be run in $x/p + t$ time. In this setting, we have $x = \mathcal{O}(n)$ and $t = \mathcal{O}(\log n)$. Thus, to get an algorithm which runs in $\mathcal{O}(\log n)$, we need $\Omega(n/\log n)$ processors.

In the Parallel Prefix Problem, as the input we are given an array A of length n and we want to compute an array B of length n that contains all the prefix sums, i.e., $B[j] = \sum_{i'=1}^{j} A[i']$ for all $j \in \{1, \cdots, n\}$. We provide an algorithm which solves this problem with $n/\log n$ processors and runs in $\mathcal{O}(\log n)$ time-steps. Assume that we have $n/\log n$ processors $p_1, \cdots, p_{n/\log n}$. We divide the array into $n/\log n$ sub-arrays $A_1, \cdots, A_{n/\log n}$ of $\log n$ consecutive elements in the array. The processor p_i is supposed to be responsible for the sub-array A_i. First, each of the processors will compute the prefix sum for the last element in its sub-array (with respect to the sub-array) which is doable in $\mathcal{O}(\log n)$ time-steps; that is, at the end of this phase $B[i\log n] = \sum_{i'=(i-1)\log n+1}^{i\log n} A[i']$ for $i \in \{1, \cdots, n/\log n\}$. Now, we consider only values of $B[j]$ which correspond to the last element of the sub-arrays, i.e., $B[i\log n]$. This reduces the size of the problem to $n/\log n$. We know how to solve the problem on these $n/\log n$ elements by applying the method from the lecture, in $\mathcal{O}(\log n)$ time-steps. Note that this is doable since the number of processors and elements are equal. After this phase, $B[i\log n]$ for $i \in \{1, \cdots, \frac{n}{\log n}\}$ has our desired prefix sum, meaning $B[i\log n] = \sum_{i'=1}^{i\log n} A[i']$. Now, each processor computes its related values in $\mathcal{O}(\log n)$. More precisely, processor p_i for $i \geq 1$, set $B[(i-1)\log n + j'] = B[(i-1)\log n] + \sum_{i'=(i-1)\log n+1}^{(i-1)\log n+j'} A[i']$ for $j' \in \{1, \cdots, \log n - 1\}$, where we assume $B[0] = 0$. Note this can be done in $\mathcal{O}(\log n)$ time-steps. Thus, we have an algorithm which needs $n/\log n$ processors and solves the Parallel Prefix Problem in $\mathcal{O}(\log n)$ time-steps.