Advanced Algorithms 2024 14/10 2024

Sample Solutions 04

Lecturer: Johannes Lengler *Teaching Assistant: Andor Vari-Kakas*

1 Online Edge Coloring

Given a fixed set of vertices V, a set of edges $E \subseteq V \times V$ arrives over time and upon arrival of each edge, we should color it with one of colors $\{1, 2, \ldots, q\}$. This is the permanent color of that edge and cannot be changed later. The coloring should be a proper coloring at all times, i.e., no two edges that share an endpoint should receive the same color. Suppose that we are given the guarantee that at all times, the maximum degree of any node is at most Δ. Notice that by Vizing's theorem, the offline algorithm can color the edges using just $\Delta + 1$ colors.

(A) Devise an online algorithm that computes a $(2\Delta - 1)$-edge-coloring.

Solution: Consider the greedy online coloring algorithm, i.e., the algorithm that always colors the next incoming edge with the smallest possible color.
Observe now that the highest color that this algorithm will use for an edge $\{u, v\}$ will occur if all edges incident to either vertex have different colors. The algorithm will then use $deg(u) + deg(v) - 1$ colors to color all edges incident to these two vertices. We can therefore conclude that the algorithm uses at most $2\Delta - 1$ colors, namely if both incident vertices have degree Δ and already use $2\Delta - 2$ different colors.

(B) More interestingly, prove that any deterministic online edge-coloring algorithm requires at least $2\Delta - 1$ colors, i.e., no deterministic online algorithm can get a competitive ratio better than 2.

Solution: We will show that there exists a sequence of edge arrivals such that any deterministic algorithm will have to use at least $2\Delta - 1$ colors, for any $\Delta \geq 1$.
The adversary will start by sending edges to create vertex disjoint stars, where the central vertex in each star has degree $\Delta - 1$. By doing that, one of two cases will occur first. Either there will be Δ different stars that all use the same edge colors, or there will be so many stars that do not use the same colors, that the algorithm will have already used at least $2\Delta - 1$ colors. After at most $\binom{2\Delta-2}{\Delta-1}(\Delta - 1) + 1$ stars have been added, at least one of the cases will have occurred. This holds as each distinct star can occur at most $\Delta - 1$ times, and there are $\binom{2\Delta-2}{\Delta-1}$ possible differently colored stars, using at most $2\Delta - 2$ colors. Suppose the algorithm did not yet use $2\Delta - 1$ colors. It then follows that there must be Δ stars that use exactly the same colors. The adversary can then add an additional Δ edges, connecting a new vertex to each of the identically colored stars. By doing that, another Δ colors will need to be introduced, which, with the already used $\Delta - 1$ colors, in each of the stars, brings the total number of colors used to $\Delta + \Delta - 1 = 2\Delta - 1$, as claimed.

2 Hungry Cow

Consider the following hungry cow problem—a cow stands on the x-axis at the origin, and is looking for a nice patch of yummy green grass, which it knows exists somewhere on the x-axis at some integer distance $d \geq 1$ either to the left or to the right of the origin. Neither the distance nor the side are known to the cow. Devise a 9-competitive algorithm for the cow with respect to the distance it needs to travel to get to the food.

Solution: Let $X_i := (-2)^i$, for every $i \geq 0$. Consider the strategy in which the cow moves directly towards the point X_i at every iteration i, i.e., starting at $X_{-1} := 0$, the cow goes from the point X_{i-1} directly towards the point X_i, at the i^{th} iteration, for $i \geq 0$. We make the following two remarks. First, note that, for all $i > 0$, the distance $d(i)$ traversed by the cow at the i^{th} iteration is equal to

$$d(i) = |X_i - X_{i-1}| = |(-2)^i - (-2)^{i-1}| = 2^i + 2^{i-1} = 3 \cdot 2^{i-1}. \tag{1}$$

Second, note that $X_{2i} = 2^{2i}$ and that $X_{2j+1} = -2^{2j+1}$, for every $i, j \geq 0$. With this in mind we consider the following two cases, where x denotes the location of the patch of grass:

Case 1: $x > 0$.

Suppose that $x > 0$. If $x = 1$ then the cow will get to x directly, so from now on we assume that $x > 1$. Let $k \geq 0$ be the unique integer such that

$$X_{2k} = 2^{2k} < x \leq 2^{2k+2} = X_{2k+2}. \tag{2}$$

Note that the cow will reach the desired destination x at some point during the $(2k+2)^{th}$ iteration. By the above remarks the distance $D(x)$ that the cow needs to travel in order to get to point x is given by

$$\begin{aligned}
D(x) &= 1 + \left(\sum_{i=1}^{2k+1} d(i)\right) + |x - X_{2k+1}| \\
&= 1 + \left(\sum_{i=1}^{2k+1} d(i)\right) + x + 2^{2k+1} \\
&= 1 + \left(\sum_{i=1}^{2k+1} 3 \cdot 2^{i-1}\right) + x + 2^{2k+1} \quad \text{(by equation (1))} \\
&= 1 + \left(3 \sum_{i=0}^{2k} 2^i\right) + x + 2^{2k+1} \\
&= 1 + 3 \cdot \frac{2^{2k+1} - 1}{2 - 1} + x + 2^{2k+1} \quad \left(\text{since } \textstyle\sum_{i=0}^{n} z^i = \frac{z^{n+1}-1}{z-1} \text{ for } z \neq 1\right) \\
&= 1 + 4 \cdot 2^{2k+1} - 3 + x \\
&\leq 4 \cdot 2^{2k+1} + x \\
&= 8 \cdot 2^{2k} + x \\
&\leq 9x. \quad \text{(by (2))}
\end{aligned} \tag{3}$$

Hence, the cow will need to travel at most 9 times the minimum distance x required to reach its destination.

Case 2: $x < 0$.

Suppose that $x < 0$. It is straightforward to check that if $x \in \{-1, -2\}$, then our cow is 9-competitive, hence assume that $x < -2$. Let $k > 0$ be the unique integer such that

$$|X_{2k-1}| = 2^{2k-1} < |x| \leq 2^{2k+1} = |X_{2k+1}|. \tag{4}$$

The cow will reach the desired destination x at some point during the $(2k+1)^{th}$ iteration. Similarly as case 1 the distance $D(x)$ that the cow needs to travel in order to get to point

x is given by

$$\begin{aligned}
D(x) &= 1 + \left(\sum_{i=1}^{2k} d(i)\right) + |x - X_{2k}| \\
&= 1 + \left(\sum_{i=1}^{2k} d(i)\right) + |x| + 2^{2k} \\
&= 1 + \left(\sum_{i=1}^{2k} 3 \cdot 2^{i-1}\right) + |x| + 2^{2k} \quad \text{(by equation (1))} \\
&= 1 + \left(3 \sum_{i=0}^{2k-1} 2^{i}\right) + |x| + 2^{2k} \\
&= 1 + 3 \cdot \frac{2^{2k} - 1}{2 - 1} + |x| + 2^{2k} \quad (\text{since } \textstyle\sum_{i=0}^{n} z^i = \frac{z^{n+1}-1}{z-1} \text{ for } z \neq 1) \\
&= 1 + 4 \cdot 2^{2k} - 3 + |x| \\
&\leq 4 \cdot 2^{2k} + |x| \\
&= 8 \cdot 2^{2k-1} + |x| \\
&\leq 9|x|. \quad \text{(by (4))} \qquad (5)
\end{aligned}$$

Therefore we conclude that the proposed algorithm is 9−competitive.

3 Optimal Offline Algorithm for Paging

Devise a polynomial-time algorithm for computing the optimal offline solution for the paging problem. Prove its correctness and analyze its time complexity.

Solution:

Claim 1. *The offline scheduling algorithm that always evicts the element that, among all elements, will be requested the farthest in the future solves the cache scheduling problem optimally, i.e., has the minimum number of cache misses.*

For an eviction schedule S, define $S(i)$ to be the state of the cache at the time of the i-th request. Furthermore, let $S[i, j]$ be the sequence of cache states according to the schedule, between the i-th and j-th request.

Let S_A be a schedule created by the algorithm. We will use the following claim to construct a sequence of schedules S_i, and use that sequence to show that any schedule will have at least as many cache misses as the algorithm. In particular, an optimal schedule can not be better than the algorithm's schedule.

Claim 2. *Let $i \in [n]$, where n is the number of requests. Suppose that for the schedule S_i it holds that $S_i[1, i] = S_A[1, i]$, then there exists a schedule S_{i+1} such that $S_{i+1}[1, i+1] = S_A[1, i+1]$ and S_{i+1} has at most as many cache misses as S_i.*

Proof. Consider the i-th request, and call the requested element x. There are then 3 cases to consider.

If x is already in $S_A(i)$ and $S_i(i)$, we can simply define $S_{i+1} := S_i$, as then it also holds that $S_{i+1}[1, i + 1] = S_A[1, i + 1]$, and furthermore, it certainly has the same amount of cache misses as S_i.

If x is not in $S_A(i)$ and $S_i(i)$, and both of them evict the same element, we can again define $S_{i+1} := S_i$, which, by the same reasoning as before satisfies the conditions.

Suppose now that x is not in $S_i(i)$ and $S_A(i)$, and S_i evicts a different element than S_A. Call the elements evicted a and b, respectively.
In this case, we have to do a bit more analysis. First off, we define S_{i+1} such that $S_{i+1}[1, i+1] = S_A[1, i+1]$. The following steps will be defined according to the remaining requests.
At this point, it holds that a is in S_{i+1}, and b is in S_i, but otherwise they are identical. We will preserve the property P that at all times until b is requested next, S_{i+1} and S_i differ in at most one element, and after b is requested (and the request processed), they will be identical again.

For each request before b is requested, S_{i+1} will not evict anything if the requested item is in the cache, and evict the same element as S_i if the requested item is not in the cache of S_{i+1}. If, in the latter case, the item S_i evicts is not in the cache of S_{i+1}, then S_{i+1} simply evicts a, and, assuming Property P, S_i and S_{i+1} have completely identical cache content again, and from then on S_{i+1} will behave identical to S_i. As, in this case, S_{i+1} has at most as many cache misses as S_i, Claim 2 follows. Hence, assume this case does not occur. In particular, this implies that if S_i evicts b, then the request must have been a as otherwise (assuming Property P) S_{i+1} would have incurred a cache miss and could not have evicted the same element as S_i. But, in this case, S_{i+1} and S_i are identical after the request, and Claim 2 follows as well. Hence, we will also assume that b is not evicted by S_i before b is requested.

It is easy to check that the above definition of S_{i+1} indeed preserves Property P until b is requested. Note that a will always be requested before b, by definition of the algorithm. Hence, at some point before b is requested, S_i will incur a cache miss whereas S_{i+1} does not, and thus in total S_{i+1} will incur strictly fewer cache misses than S_i, until b is requested. Now, when b is requested, S_{i+1} simply evicts the element that is not contained in the cache of S_i, and since our above observations ensure that S_i contains b, S_i and S_{i+1} are again identical after the request of b. From now on, S_{i+1} will behave exactly like S_i. This proves the second part of Property P. Since S_{i+1} incurs strictly fewer cache misses than S_i before the request of b, potentially one more when b is requested, and the same number after the request of b, the claim follows. $\square$

By construction of the claim, it follows that $S_n = S_A$.

Consider an optimal schedule S_O which has fewer cache misses than S_A. It is clear that at some point S_O will evict a different element than S_A. By the previous claim, however, we can conclude that the number of cache misses that S_n incurs is at most the number of cache misses that S_O has. Therefore, we get a contradiction, showing that S_A indeed minimizes the number of cache misses, thereby proving Claim 1.