Advanced Algorithms

Homework 3

Ari Feiglin

3.1 Exercise

In the Solitaire Memory Game, there are n pairs of cards distributed randomly and face down on a table. The single player plays each round by picking two (of the $2n$) cards from the table, and if they match he removes them, otherwise he returns them face down. The game ends when all the pairs are removed from the game. Suppose the player can remember the picture on every card he has seen.

(**1**) Show a deteriministic algorithm for the game with a competitive ratio of 2.

(**2**) Show a lower bound of $(2 - 2/n)$ for the competitive ratio of any deterministic algorithm for the game.

(**3**) Give a random algorithm which per round chooses the first card uniformly from the unseen cards, and show it has an expected competitive ratio of $1\frac{3}{4}$ against an oblivious adversary.

(**1**) The algorithm is just the naive one: for the first n rounds, uncover all the cards. Then the next n rounds is spent matching all the pairs remembered from the first rounds. The optimal algorithm matches a pair each round and so it takes n rounds, thus our competitive ratio is $\frac{2n}{n} = 2$, as required.

(**2**) We can assume the following: as soon as a pair is found, the player matches them; if no match is known, then the first available unknown card is chosen; and that the player starts from card 1. Then the cards can be arranged such that the player plays

$$(1,2)\ (3,1)\ (4,5)\ (5,3)\ (6,7)\ (7,4)\ (8,9)\ (9,6)\cdots$$

in general, the game is built of rounds of the form $(2k, 2k+1)\ (2k+1, 2k-2)$ for $k > 1$. This will occur $n-2$ times, and including the first two rounds gives $2(n-2)+2 = 2n-2$ rounds. Then we must make another round to match $2n$, to give us $2n-1$ rounds. Thus the competitive ratio is $(2 - 1/n)$ (since the optimal algorithm works in n rounds).

(**3**) Our algorithm will pick its first card at random, and if it has a match in a card already looked at, match it. Otherwise pick another card at random. For each match, there are three ways for it to be matched:

(**1**) The algorithm matched it immediately (so it picks a card not seen before, then its match).

(**2**) The algorithm matched it by finding a match at a later round (i.e. card A found in one round, then card B found as the first card in another round).

(**3**) The algorithm matched it by finding each card in separate rounds, then matching them in a third round (i.e. card A found in one round, then card B found as the second card in another round).

Note that for **i**, each card in the pair is flipped once. For **ii**, one is flipped once and one is flipped twice, so we can view this as flipping 1.5 times. For **iii**, both are flipped twice. Thus if A, B, C are the number of occurrences of each type of match, the total rounds are $A + 1.5B + 2C$, and we know $A + B + C = n$, so the total rounds are $n + 0.5B + C$. The probability of **iii** occurring is less than **ii** occurring, so the expected number of rounds is $\leq n + 1.5\,\mathbb{E}(B)$.

3.2 Exercise

Find a randomized algorithm for the random path search problem with a better competitive ratio than 9.

Our algorithm will first uniformly choose between ± 1 (let this be s), then it will make the following steps:

$$\left\{s, -2s, 4s, \ldots, s(-1)^i 2^i\right\}$$

Now suppose that the cow is found at $(-1)^n(2^n+\varepsilon)$, for $s=1$ the path taken will be

$$1,-2,\ldots,(-1)^n 2^n,(-1)^{n+1}2^{n+1},(-1)^n(2^n+\varepsilon)$$

So the time taken is

$$2(1+2+\cdots+2^{n+1})+2^n+\varepsilon = 2(2^{n+2}-1)+2^n+\varepsilon = 92^n+\varepsilon-2$$

And for $s=-1$ the path taken will be

$$-1,2,\ldots,-(-1)^n 2^n,(-1)^n 2^n+\varepsilon$$

So the time taken is

$$2(1+\cdots+2^n)+2^n+\varepsilon = 2(2^{n+1}-1)+2^n+\varepsilon = 52^n-2+\varepsilon$$

So the expected time is then

$$72^n+\varepsilon-1$$

The optimal time is $2^n+\varepsilon$, and so we have a competitive ratio of 7 (since this is the worst possible input, the analysis for $-(-1)^n(2^n+\varepsilon)$ is similar).

3.3 Exercise

Find a randomized algorithm for the rental ski problem with a better competitive adversary than shown in class.

Our algorithm will buy with probability p on day αM, and otherwise on day M. If we go skiing on day M then the expected price we pay is

$$p(\alpha M+M)+(1-p)(M+M) = M(\alpha p+p-2p+2) = M(p(\alpha-1)+2)$$

since the optimal algorithm in this case will just buy straight away, we have that

$$\mathbb{E}(C(M)) = (p(\alpha-1)+2)C_{\text{opt}}(M)$$

Otherwise if we go skiing on day αM, then the optimal thing to do is rent for αM days, and our expected price is

$$p(\alpha M+M)+(1-p)\alpha M$$

since with probability p we buy on day αM, and otherwise we wait for day M, but by then we will have gone skiing. Thus

$$\mathbb{E}(C(\alpha M)) = \alpha M\left(p\left(1+\frac{1}{\alpha}\right)+1-p\right) = \left(\frac{p}{\alpha}+1\right)C_{\text{opt}}(\alpha M)$$

So the competitive ratio is $\max\left\{p(\alpha-1)+2,\frac{p}{\alpha}+1\right\}$, which is minimized when they're equal:

$$p(\alpha-1)+2 = \frac{p}{\alpha}+1 \implies p\alpha^2+\alpha(1-p)-p=0$$

So

$$\alpha = \frac{p-1\pm\sqrt{5p^2-2p+1}}{2p}$$

In order for this to be positive, we must have

$$\alpha = \frac{p-1+\sqrt{5p^2-2p+1}}{2p}$$

Then the competitive ratio is

$$p(\alpha-1)+2 = \frac{p-1+\sqrt{5p^2-2p+1}}{2}-p+2 = \frac{3-p+\sqrt{5p^2-2p+1}}{2}$$

Which obtains a minimum at $p=0.4$ with a value of 1.8. And we compute $\alpha=0.5$.

So our algorithm will buy with probability 0.4 on day $M/2$ and 0.6 on day M. This has a competitive ratio of 1.8.

3.4 Exercise

The *periodicity* of a string S is the smallest p such that $S[i] = S[i+p]$ for all $i \leq n-p$. Make use of the KMP algorithm to find the periodicity of a string S.

Recall that KMP computes the following function in $\Theta(n)$ time:

$$\pi(q) = \max\{k < q \mid S[1,\ldots,k] \text{ is a suffix of } S[1,\ldots,q]\}$$

Now if $S = \xi^m$ where ξ is the period of S, then notice that ξ^{m-1} is the largest suffix of $S = \xi^m$. Thus $\pi(n) = n - |\xi|$, thus $n - \pi(n) = |\xi| = p$.

Now if $S = \xi^m\xi'$ where ξ' is a prefix of ξ, then the largest prefix of S which is a proper suffix of S is ξ', and so $\pi(n) = |\xi|'$. So our first step is to compute $\ell = \pi(n)$, then we focus on $P[1,\ldots,n-\ell]$ and compute $n - \ell - \pi(n-\ell)$. That is, we just return

$$|S| - \pi(|S|) - \pi(|S| - \pi(|S|))$$

3.5 Exercise

Give a linear algorithm, which given a string S, computes the longest substring of S which is also a palindrome.

Now, a substring palindrome is a substring $S[i:i+k)$ such such that $S[i:i+k)^r = S[i:i+k)$. We know that

$$S = S[0:i)\,S[i:i+k)\,S[i+k:n) \implies S^r = S[i+k:n)^r\,S[i:i+k)^r\,S[0:i)^r$$

Thus $S[i:i+k)^r = S^r[n-i-k:n-i)$. So we require $S[i:i+k) = S^r[n-i-k:n-i)$.