1. Online Algorithm

Online algorithm is one, that does not have the entire input from the beginning. Because it does not know the whole input, an online algorithm sometimes makes decisions that may later turn out not to be optimal.

In contrast, an offline algorithm is given the whole input data from the beginning.

For example, suppose, we have a large cluster of processors. We have to decide how many processor we need to allocate for our programs and where to allocate them. This problem is known as resource allocation problem, which is heavily online, because it does not know about all the jobs ahead of time and has to figure out the allocation over time.

General characteristics of an online algorithm are following :

- Algorithm is unware of all inputs ahead of time.
- It has to maintain a feasible solution always.
- It can make very few changes over time(cannot entirely reoptimize each time).
- It has to be competitive i.e., it has to be reasonable for underlying objective functions.

A famous example on online algorithm is online paging problem, which we will describe now.

2. Paging

A cache memory has k number of slots, i.e., it can hold k pages. Whenever there is a request for a new page which is not currently present in cache,it has to evict some old page if the cache is full and insert the requested page in place of evicted page.

So, we have to figure out which page to be evicted. As we don't know about the future pages that are going to be requested, this problem is well motivated for online algorithm.

2.1. Example

Suppose there are three possible pages A, B, C and only two cache slots. The sequence of page requests are known : $AABCAABBBCACC$. Now, the question is how to maintain the cache so that as few page evictions as possible.

Suppose, initially the cache was empty.

So, after arriving of A and B, the cache looks like

A	B

When C arrives, we will evict B. Since, we know that A will be going to be requested before B in future, we will intuitively evict B

A	C

Similarly when B comes, evict A

B	C

and, when A comes, evict B

A	C

So, in this replacement policy there are 3 evictions in total.

Our goal is to use a cache replacement policy that minimizes the total number of evictions.

In the above example, we have used the optimal algorithm to minimize the number of evictions, that is, "Farthest into the future" replacement algorithm. In this algorithm, we evict the page which is not going to be used in near future.

But this model is flawed because we cannot know the future requests in advance.

So, we want an online algorithm that will minimize the number of evictions even without knowing the future page requests.

2.2. Least Recently Used Algorithm and Competitive Ratio

LRU works on the idea that pages that have been most heavily used in the past few instructions are most likely to be used heavily in the next few instructions too. So it evicts the page that has been unused for the longest time.

Now, how can we measure the goodness of online algorithm?

"Competitive ratio" is a commonly used measure to analyze online algorithms, in which the performance of an online algorithm is compared to the performance of an optimal offline algorithm. The competitive ratio of is defined as the worst-case ratio over all possible input sequences.

$$\text{competitive ratio} = \frac{\text{number of evictiones of online algorithm}(\sigma)}{\text{number of evictions of optimal offline algorithm}(\sigma)}$$

where σ is the input sequence of requests.

2.3. Competitive Ratio of LRU

Suppose the cache has k slots, which are initially empty.

Consider the input sequence $\sigma = 1, 2, 3, \ldots, k, k+1, 1, 2, 3, \ldots, k$.

After arriving of first k inputs, cache looks like :

1	2	3	...	k

When $k+1$ arrives, 1 will be evicted,because 1 is unused for longest time.

~~1~~ k+1	2	3	...	k

When 1 arrives, 2 will be evicted :

k+1	~~2~~ 1	3	...	k

Similarly, all the k pages will be evicted one by one.

k+1	1	~~3~~ 2	...	~~k~~ k-1

And, when the last input k arrives, $k+1$ will be evicted.

~~k+1~~ k	1	2	...	k-1

So, the total number of evictions using LRU is $k+1$.

Now, if we use optimal algorithm for the same input sequence, the cache will look like :

1	2	3	...	~~k~~ k+1

1	2	3	...	~~k+1~~ k

Hence, the number of evictions is 2 in this case.

Theorem 20.1. $\frac{k}{2} \leq$ *Competitive ratio of LRU* $\leq k$.

Proof. Exercise. □

2.4. Randomization of LRU

2.4.1. Marking Algorithm

Suppose cache has already k pages in it. Initially,keep each of the pages unmarked. When a new page request comes, do the following steps :

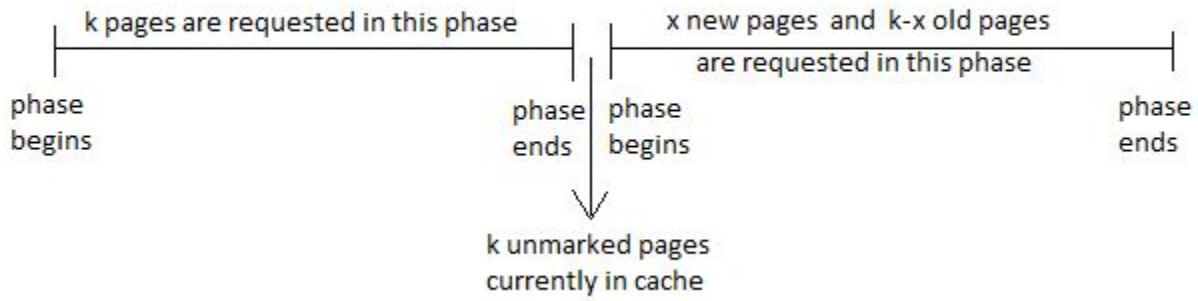
(a) If the requested page is already in cache, serve it, and mark it.

(b) If the requested page is not in the cache, bring it in, and mark it, and evict a random unmarked page.

(c) If all the pages are marked in the cache, reset them to unmarked.

Lemma 20.2. *Competitive ratio of marking algorithm is* $O(\log_2 k)$.

Proof. Cache has k slots. Consider the two phases of page requests.



Fact 20.3. *Marking algorithm is hurt most when the old pages are going to be requested after the requests of new pages.*

If the old pages are requested before the new pages, they will immediately be marked, as the old pages are already present in the cache. So, it does not make any new eviction of page.

So, in the worst case scenario, old pages are going to be requested after the requests of new pages.

Therefore, the expected number of cache evictions

$$\leq x + \frac{x}{k} + \frac{x}{k-1} + \frac{x}{k-2} + \ldots + 1 \approx x \log_2 k$$

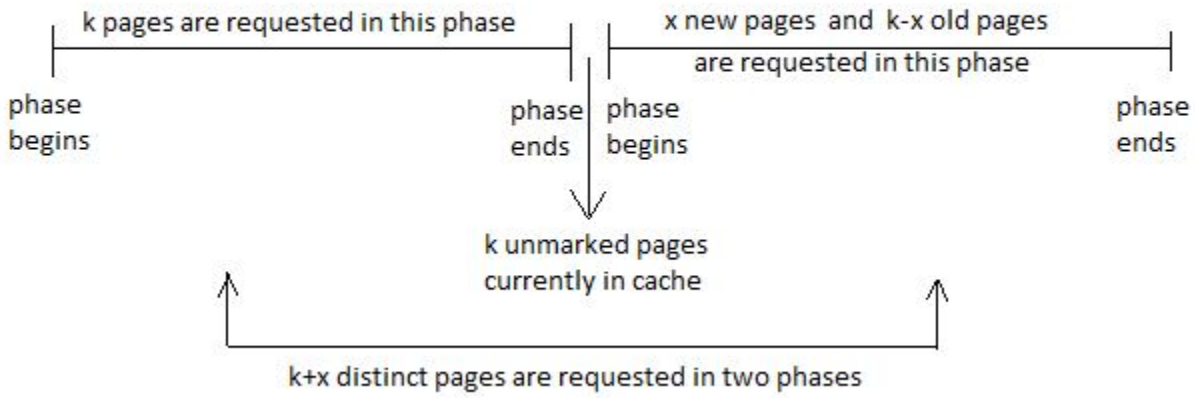
The first term of the expression is x, because x number of new page requests causes x number of evictions in the new phase.

The second term of the expression is $\frac{x}{k}$. When the first old page request comes, if it is not present in the cache, it evicts some other old page and enters into the cache. The probability that this first old page was thrown out of the cache by the x new pages $= \frac{x}{k}$. Hence the expected number of evictions is also $\frac{x}{k}$.

Now, because the first old page has already occupied its slot in the cache, the probability for the second old page will be calculated out of remaining $k-1$ slots. Hence, the probability that the second old page was thrown out of the cache by the x new pages $= \frac{x}{k-1}$. So, the expected number of evictions is also $\frac{x}{k-1}$.

Similarly, for the third old page request expected number of eviction is $\frac{x}{k-2}$ and so on.

Now, we have to check number of evictions in case of optimal offline algorithm.



So, in case of optimal algorithm, there are $k+x$ number of distinct page requests across two phases and it has to evict atleast x number of pages across these two phases. So, we can say, the expected number of evictions in one phase $\geq \frac{x}{2}$.

Hence, the competitive ratio of marking algorithm

$$= \sum \frac{\mathbb{E}[\#\text{evictions of marking algorithm in current phase}]}{\#\text{evictions of optimal algorithm in current phase}} \leq 2 \log_2 k = O(\log_2 k).$$

□

So, we have noticed that the marking algorithm performs exponentially better than LRU algorithm according to competitive ratio.

Exercise.

Any randomized algorithm has competitive ratio $\geq \Omega(\log_2 k)$.

3. Online Set Cover

3.1. Set Cover Problem

Given a set of elements(called the universe of elements) and a collection of sets whose union equals to the universe. Our goal is to pick the minimum number of sets to cover the universe.

Proposed algorithms:

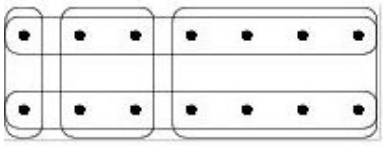
Algorithm 1 : Greedy

In greedy approach, at each step we pick the set that contains largest number of uncovered elements.

The greedy algorithms achieve a approximation ratio of $O(\log_2 n)$.

Here is an example of greedy algorithm that achieves approximation ratio of $\log_2 n/2$.

The universe consists of n elements. The set system consists of k pairwise disjoint sets $S_1, \ldots, S_k$ with sizes $2, 4, 8, \ldots, 2^k$ respectively, where $n = 2^{(k+1)} - 2$ and two more disjoint sets T_1, T_2, each of which contains half of the elements from each S_i. An example of such an input for k=3 is in the picture :



The greedy algorithm picks the sets $S_k, \ldots, S_1 (\approx \log_2 n$ sets), in that order while the optimal algorithm picks only T_1 and T_2.

Hence, the approximation ratio becomes $\log_2 n/2$.

We have noticed that greedy is inherently an offline algorithm. It needs the knowledge of all the elements in advance to decide which is the best set to pick.

Algorithm 2 : LP Rounding

For every set $s \in S$ we are taking a variable x_s.

We have to minimize the number of sets. So, Linear programming can be formulated as following:

$$min \sum_{s \in S} x_s$$

Subject to:

$$\sum_{s:e \in S} x_s \geq 1, \forall e(\text{elements in universe})$$

$$x_s \geq 0, \forall s \in S$$

We can compute the optimal fractional solution from the above equations and then round it to get an integer solution.

3.2. What is online set cover problem?

In case of online set cover we don't know the entire input at once. More specifically, we don't know what elements need to be covered before hand. So, we need to maintain a good set cover for the elements that have currently shown up.

For example, suppose, we have potential m locations in a city where airtel can install their cell towers. The reach of a cell tower is always fixed(say, it has a radius r).

Initially, the city is unpopulated and airtel has not installed any cell tower yet.

When a new colony is built, airtel installs a cell tower to cover that colony,if it is already not covered. Moreover, if a tower is already installed in a location it cannot be uninstalled.
Similarly, in case of set cover problem, the algorithm knows all the M possible sets at time-0.

At time-1 : element e_1 arrives and algorithm picks a set that covers e_1.

At time-2 : element e_2 arrives. If e_2 is not already covered by the existing sets, algorithm picks a set that covers e_2.

$\vdots$

At time-T : element e_T arrives. If e_T is not already covered by the existing sets, algorithm picks a set that covers e_T.

This algorithm cannot be reoptimized, i.e., once a set is included in the solution we can not remove it from the solution.

How good is our algorithm?

We can measure the goodness of a set cover algorithm using competitive ratio.

$$\text{competitive ratio} = \frac{\mathbb{E}[\#\text{sets choosen by online algorithm for a input sequence } \sigma\ ]}{\#\text{sets choosen by optimal algorithm for same } \sigma}$$

Lets see a bad example using greedy algorithm.

Candidate Greedy Algorithm(CG):

When a new element arrives, if it is not covered by any existing set, choose a set which covers most of the elements including the new element.

Now, suppose sets are $S_0 = \{e_1, e_2, e_3, e_4, e_5\}$, $S_1 = \{e_1, e_{100}, \ldots, e_{199}\}$,$S_2 = \{e_2, e_{200}, \ldots, e_{299}\}$, $S_3 = \{e_3, e_{300}, \ldots, e_{399}\}$, $S_4 = \{e_4, e_{400}, \ldots, e_{499}\}$, $S_5 = \{e_5, e_{500}, \ldots, e_{599}\}$ and the input sequence $\sigma = e_1, e_2, e_3, e_4, e_5$.

So, our CG algorithm chooses the sets S_1, S_2, S_3, S_4, S_5 in this order to cover the elements e_1, e_2, e_3, e_4, e_5, while optimal algorithm chooses only one set S_0 to cover all these elements.

So, our CG algorithm fails here, because it includes many unnecessary elements.

Alternate Meta algorithm

In this algorithm, we maintain a utility(or confidence) of sets. Every time, a new element arrives, we will update the confidence/utility of every set and will include the most useful set.

This Meta algorithm can be broken up into two parts. In one part, we will solve the fractional algorithm which will maintain the confidence of the sets. In the other part we will apply the rounding algorithm to get integer solution.

1. Fractional Algorithm

In this part we are going to maintain an online Liner Programming solution. The online LP can be written as follows :

$$min \sum_{s \in S} x_s$$

Subject to:

$\sum_{s:e \in S} x_s \geq 1$, $\forall e$, where e is the currently arrived element.

When we will use the above online LP in our fractional algorithm, we should remember the following two conditions on every iteration

(a) New element adds new constraint,

(b) Want the x_s values to be monotonically non-decreasing(once a set is included in the solution we can not remove it from the solution).

The Dual of the above online LP can be written as follows :

$$min \sum_{arrived\ e} y_e$$

Subject to:

$\sum_{e \in S} y_e \leq 1$, $\forall sets\ s \in S$

Now, the steps of fractional algorithm are described below :

Initialize all $x_s = \frac{1}{m}$, where m is the number of sets and initial LP value $= \sum x_s = \sum \frac{1}{m} = 1$.

When new element e arrives,

while($\sum x_s < 1$)

update $x_s = 2x_s$ if s covers e.

also update $y_e = y_e + 1$ for analysis purpose.

endwhile

Hence, if a set is repeatedly useful, its x_s value jumps up to 1 very quickly.

We will see the details of this fractional algorithm and the rounding algorithm in next lecture.