abhi shelat

- Scheduling
- Cache problem

3.15.2016
Scheduling
<table>
<thead>
<tr>
<th>Course</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>sy333</td>
<td>2</td>
<td>3.25</td>
</tr>
<tr>
<td>en162</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>ma123</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>cs4102</td>
<td>3.5</td>
<td>4.75</td>
</tr>
<tr>
<td>cs4402</td>
<td>4</td>
<td>5.25</td>
</tr>
<tr>
<td>cs6051</td>
<td>4.5</td>
<td>6</td>
</tr>
<tr>
<td>sy333</td>
<td>5</td>
<td>6.5</td>
</tr>
<tr>
<td>cs1011</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
problem statement

\[(a_1, \ldots, a_n) \leq n \text{ activities}\]
\[(s_1, s_2, \ldots, s_n) \text{ start times}\]
\[\rightarrow (f_1, f_2, \ldots, f_n) \text{ finish times} \quad (\text{sorted}) 
  \quad s_i < f_i\]

find largest subset of activities \(C = \{a_i\}\) such that

(\text{compatible})

for any two \(a_i, a_j \in C\) s.t. \(i < j\)

\[f_i < s_j\]
problem statement

\[(a_1, \ldots, a_n)\]
\[(s_1, s_2, \ldots, s_n)\]
\[(f_1, f_2, \ldots, f_n) \text{ (sorted)}\]
\[s_i < f_i\]

find largest subset of activities \(C=\{a_i\}\) such that

\[a_i, a_j \in C, i < j\]
\[f_i \leq s_j\]
problem statement

\((a_1, \ldots, a_n)\)

\((s_1, s_2, \ldots, s_n)\)

\((f_1, f_2, \ldots, f_n)\) (sorted) \(s_i < f_i\)
$\text{Best}_n = \text{maximal number of activities that can occur before event } n.$
dynamic programming

\[ \text{Best}_{2n} = \max \left\{ 1 + \text{Best}_{\text{start}(2n)}, \text{Best}_{2n-1} \right\} \]
\[ \text{BEST}_{f_n} = \max \begin{cases} \text{BEST}_{s_n} + 1 & \text{in: } a_n \\ \text{BEST}_{e_t} & \text{out: } a_n \end{cases} \]
**greedy solution:**

**definition:**

\[ \text{SOLTN}_{i,j} = \text{maximal # of activities that occur between events } i \text{ and } j. \]

**goal:** \[ \text{SOLTN}_{0,2n} \]
greedy solution:

SOLTN_{i,j}

goal: SOLTN_{0,2n}
greedy solution:

Claim: the first action to finish in $e[i,j]$ is always part of some $\text{soltN}_{i,j}$, an optimal solution for period $[i,j]$. 

$\Rightarrow$ $a_i$ is always part of some $\text{soltN}_{0,2n}$.
proof: Consider $\text{SOLUTN}_{i,j}$ and let $a^*$ be the first activity to finish in $[i,j]$. 

1. If $a^* \in \text{SOLUTN}_{i,j}$, then the claim follows.
2. Suppose $a^* \notin \text{SOLUTN}_{i,j}$. Let activity $a$ be the first activity to finish in $\text{SOLUTN}_{i,j}$.

$a^*$ finishes before $a$, i.e., $f_{a^*} \leq f_a$ by hypothesis.

So therefore,

$$S = \text{SOLUTN}_{i,j} - \lfloor a \rfloor + \lfloor a^* \rfloor$$

then $|S| = |\text{SOLUTN}_{i,j}|$ and so $S$ is optimal too.

Therefore the lemma follows.
greedy solution:

algorithm: find first event to finish. add to solution. remove conflicting events. continue.
greedy solution:

algorithm:
find first event to finish. add to solution.
remove conflicting events.
continue.
greedy solution:

algorithm: find first event to finish. add to solution. remove conflicting events. continue.
greedy solution:

algorithm: find first event to finish. add to solution. remove conflicting events. continue.
Greedy solution:

Algorithm:
find first event to finish. add to solution.
remove conflicting events.
continue.
greedy solution:

algorithm:
- find first event to finish. add to solution.
- remove conflicting events.
- continue.
greedy solution:

algorithm: find first event to finish. add to solution. remove conflicting events. continue.
running time

**algorithm:** find first event to finish. add to solution.
remove conflicting events.
continue.

\[ \Theta(n) \]

\[
(f_1, f_2, \ldots, f_n) \text{ (sorted)} \quad s_i < f_i
\]
caching
Cache hit

Cache: slow → fast

CPU:
- load r2, addr a
- store r4, addr b

Main memory: slow

Virtual memory
question:

0) How can we manage a cache in order to minimize the number of cache misses.

1) Simplify by assuming that we know all memory accesses before hand.

3) Cache is fully associating, line size = 1.
problem statement

input: \( k \) - cache size, \( d_1, d_2, d_3, \ldots, d_n \) - memory access pattern.

output: schedule for the cache

cache is
problem statement

input: \( K \), the size of the cache \\
\( d_1, d_2, \ldots, d_m \) memory accesses

output: schedule for that cache that minimizes # of cache misses while satisfying requests

  cache is fully associative, line size is 1
contrast with reality

1. Caches are not fully associative

2. ***the manager does not know***
   the future access pattern
Belady evict rule

“Evict the item from the cache that is accessed farthest in the future.”

→ Using the FF rule results in an optimal schedule.
example

```
cache

a  b  c
a b c d a d e a d b a e c e a
```

```
"evict c for d."
```

memory access pattern
example

cache

```
    a
   / |
  a  b |
 /     |
 a b c d
```

```
    a
   / |
  a  b |
 /     |
 a b c d
```

```
    d
   / |
  a  b |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```

```
    e
   / |
  a  d |
 /     |
 a b c d
```
example

cache

```
  a
  b
  c
  d  e  d
```

```
  a  b  c  d  a  d  e  a  d  b  a  e  c  e  a
```

"evict d for b"
Example cache:

- a b c d a d e a d b a e c e a
- a b c d a d e a d b a e c e a

- Boldly eviction rule.
  - 4 cache misses

Another schedule:

Not 4 cache misses.
Surprising theorem

- Belady eviction rules lead to an optimal schedule.
Schedule for access pattern \(d_1, d_2, \ldots, d_n\): either the ‘NOP’ operation or the ‘evict x far y’ operation @ each step 1\(\ldots\)n. 
- At step \(i\), element \(d_i\) must be in the cache.

Lazy Reduced schedule: Schedule for which the operation “Evict x far y” only occurs at a step \(i\) if \(y = d_i\).
Exchange lemma

Let reduced schedule $S$ agree with schedule $S_{ff}$ for the first $j$ operations. There exists a reduced schedule $S'$ that agrees with $S_{ff}$ on the first $j$ operations and

$$\#\text{misses}(S') \leq \#\text{misses}(S).$$
**Exchange Lemma:**

Let $S$ be a reduced schedule that agrees with $S_{ff}$ on the first $j$ items. There exists a reduced schedule $S'$ that agrees with $S_{ff}$ on the first $j+1$ items and has the same or fewer #misses as $S$. 
Optimal schedule.

\( S_0 \xrightarrow{\text{lemma}} S_1 \xrightarrow{\text{lemma}} S_2 \rightarrow S_{\text{ff}} \)

\[ \#\text{misses}(S^*) = \#\text{misses}(S_{\text{ff}}) \]

\( S_1 \) agrees with \( S_{\text{ff}} \) on 1 operation

\[ \#\text{misses}(S_1) = \#\text{misses}(S^*) = \#\text{misses}(S_2) \]
Proof of Lemma

Let $S$ be a reduced sched that agrees with $S_{ff}$ on the first $j$ items.

There exists a reduced sched $S'$ that agrees with $S_{ff}$ on the first $j+1$ items and has the same or fewer #misses as $S$.

Proof: Since $S$ agrees with $S_{ff}$ on the first $j$ operations, then the state of the cache at operation $j+1$ will be the same. Let $d$ be the address accessed at operation $j+1$. 
Proof of lemma

State of the cache after J operations under the two schedules.

\[ \text{easy case 1} \]
\[ \text{Spse } d \in \text{cache. Then } S' = S \text{ b/c both } S \text{ and } S_{ff} \text{ issue "Ndp."} \]
\[ \Rightarrow S' \text{ agrees with } S_{ff} \text{ and has } \#\text{misses}(S) \]

\[ \text{easy case 2} \]
\[ \text{Spse } d \notin \text{cache, but both } S \text{ and } S_{ff} \text{ "evict } e \text{ for } d" \]

Same arguments apply from case 1.
Proof of lemma

case 3

d & cache, S evicts e. Sff evicts f. 

We need to construct a schedule $S'$ that satisfies

- $S'$ agrees with Sff on $j^{th}$
- $S'$ has same # of misses as $S$
Timeline

$S_f$  

$S'$  

$S$  

The first operation in $S$ that involves either $e$ or $f$.  

Copy from $S$.  

Proof of lemma

Let access $t$ be the first operation in $S$ after jet $j$ that involves either $e$ or $f$.

Either $t = e$

$t = f$

$t = g + e + f$
Proof of lemma

what if \(d=e\)?

\[ S \]

S must load e. "Evict x for e"
by this is the first operation after \(j+1\) that involve e or f

\[ S' \]

S' can issue the operation
"evict x for f"
as a result, S' and S will have the same state of the cache and therefore the same # of misses.

Return Reduce (S')
Proof of lemma

what if $g=f$ ?

Cannot happen. Because $Sff$ uses the "farthest in the future" rule, so $f$ cannot be accessed before $e$. 

Proof of lemma

what if $g$ is neither $e$ nor $f$?

"evict $f$ for $t$"

"evict $e$ for $t$"

caches will be the same again.
What have we shown

Let $S$ be a reduced sched that agrees with $S_{ff}$ on the first $j$ items. There exists a reduced sched $S'$ that agrees with $S_{ff}$ on the first $j+1$ items and has the same or fewer #misses as $S$. 
Let $S$ be a reduced sched that agrees with $S_{ff}$ on the first $j$ items. There exists a reduced sched $S'$ that agrees with $S_{ff}$ on the first $j+1$ items and has the same or fewer #misses as $S$. 

$S^*$  

$S_{ff}$