3/4

CS 1510
Algorithm Design

Greedy Algorithms

Problems 1 and 2

Due August 27, 2014

Buck Young and Rob Brown

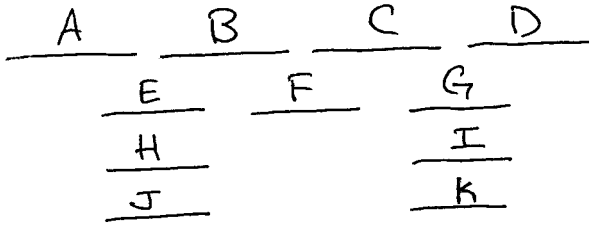
3/4

Problem 1

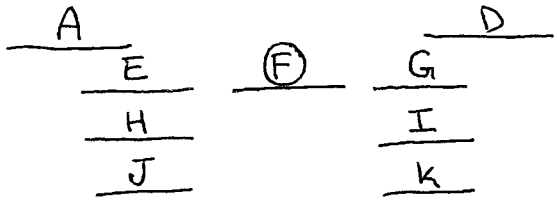
Theorem: That the given algorithm X is correct for all inputs in S.

Proof: This theorem will be proven incorrect by way of counter-example.

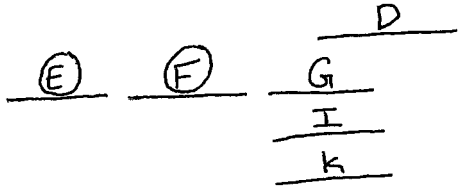
Consider these inputs:



Here we clearly see that the maximum cardinality subset of S is 4, given by the set {A, B, C, D}. This is optimal. Algorithm X, however, would choose the interval which overlaps the fewest number of other intervals first (choosing F). At this point, we throw out B and C:



At this point, we would choose any other interval (as they are all tied in overlaps) - let's choose E and toss out the overlaps A, H, and J:



Finally, we choose any other interval again (as they are all again tied in number of overlaps). Let's choose G and throw out D, I, and K:

Here we see that algorithm X has given us a subset with a cardinality of 3 (one less than optimal).

⇒ Thus, algorithm X is **not** correct for all inputs and our theorem is proven false.

Problem 2

(a) This algorithm does solve the Interval Coloring Problem.

Suppose we have a function G that returns $\{R_1, R_2, ..R_m\}$ where R_i is a set of intervals assigned to a given room according to the optimal algorithm discussed in class.

From this, we know that each room assignment R_i a) contains no overlapping intervals and b) has the maximum possible cardinality. Since each set has the maximum possible cardinality, and

$$|\bigcup_{i=1}^{n} R_i| = \sum_{i=1}^{n} |R_i|$$

is a mathematical fact, we know that the rooms as a whole have the maximum possible cardinality (ie, number of classes assigned). Since the maximum number of classes have been assigned to the current set of rooms, clearly the minimum number of rooms have been used. Not true

(b) This algorithm does solve the Interval Coloring Problem.

From a high-level, intuitive standpoint – the **only** justification for creating a new room is upon an overlap. This is an important interaction when we also consider the given hint (the maximum number of interval overlaps is equivalent to the lower bound for the number of rooms needed).

Let's dive a little deeper and consider any given interval I $\in$ S. The only instance where I is added to a **new** room is when it overlaps with an interval in **every** other room. Clearly, rooms are only added when absolutely necessary. So let's consider the very last room to be added. This room was created with an interval that must overlap an interval in every other room - its number of overlaps is equal to the total number of rooms. Thus, given the hint and this interaction, we can see that the algorithm is indeed optimal.

4 3/4

CS 1510
Algorithm Design

Greedy Algorithms

Problems 4 and 5

Due August 29, 2014

Buck Young and Rob Brown

1 18

Problem 4

(a)

Input: Locations x_1, x_2, ..., x_n of gas stations along a highway.

Output: Stoppage time at each station which minimizes the total time which you are stopped for gas.

Theorem: The given "next-station" algorithm (NS) is correct / optimal.

Proof: Assume to reach a contradiction that there exists an input on which NS produces an unacceptable output.

Let S_i be the stoppage time at each station
& NS(I)=$\{S_{\alpha(1)}, S_{\alpha(2)}, ..., S_{\alpha(n)}\}$ be the output of NS on input I
& OPT(I)=$\{S_{\beta(1)}, S_{\beta(2)}, ..., S_{\beta(n)}\}$ be the optimal solution that agrees with NS(I) the most number of steps

NS(I)	$S_{\alpha(1)}$	$S_{\alpha(2)}$	...	$S_{\alpha(k)}$	...
OPT(I)	$S_{\beta(1)}$	$S_{\beta(2)}$	...	$S_{\beta(k)}$	...

Let x_k be the first stop of the trip upon which NS(I) differs from OPT(I)

By the definition of NS, filling up for any time less than $S_\alpha(k)$ would not get you to the next station and since $S_{\alpha(k)} \neq S_{\beta(k)}$, then we know that $S_{\beta(k)} > S_{\alpha(k)}$

Let time t be defined as the difference between those stoppage times, t = $S_{\beta(k)}$ - $S_{\alpha(k)}$

Therefore $S_{\beta(k)} = S_{\alpha(k)} + t$:

NS(I)	$S_{\alpha(1)}$	$S_{\alpha(2)}$	...	$S_{\alpha(k)}$	...
OPT(I)	$S_{\beta(1)}$	$S_{\beta(2)}$	...	$S_{\alpha(k)}+t$	...
OPT')(I)	$S_{\beta(1)}$	$S_{\beta(2)}$	...	$S_{\alpha(k)}$	$S_{\beta(k+1)}+t$	...

Let OPT'(I) = OPT(I) with time t shifted to the next stop:

$$S'_k = S_k - t$$
$$S'_{k+1} = S_{k+1} + t$$

Clearly, we haven't changed the total stoppage time and we know that we can reach the next stop by the definition of NS. Furthermore we add the difference to the stop at x_{k+1}, so we will undoubtably be able to reach the station at x_{k+2} – and all future stations.

∴ We have reached a contradiction because OPT' agrees with NS for one additional step despite OPT being defined as agreeing with NS for the most number of steps.

(b)

Input: Locations x_1, ..., x_n of gas stations along a highway.

Output: Stoppage time at each station which minimizes the total time which you are stopped for gas.

Theorem: The given "fill-up" algorithm (FU) is not correct or optimal.

Proof:

Consider the input I $= \{x_1, x_2\}$ where $x_2 - x_1 = \frac{1}{2}\frac{C}{F}$

Note that to reach x_2 from x_1 it would take $F(x_2 - x_1) = F(\frac{1}{2}\frac{C}{F}) = \frac{1}{2}C$ liters of gas

It depends on where x_1 is

At x_1 the tank is filled to capacity according to FU, taking $\frac{C}{r}$ minutes. But it is possible and correct to reach x_2 using only $\frac{1}{2}C$ liters, making the optimal time $\frac{1}{2}\frac{C}{r}$ minutes.

$\therefore$ FU is not correct or optimal.

Problem 5

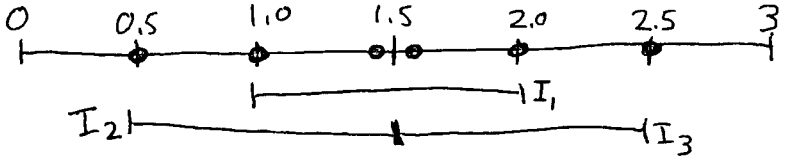
(a)
Input: A set of n points on the real line A where $a_i \in \mathbb{R}$.

Output: Minimum cardinality collection S of unit intervals that cover every point in A.

Theorem: The given "most points" algorithm (MP) is incorrect.

Proof:

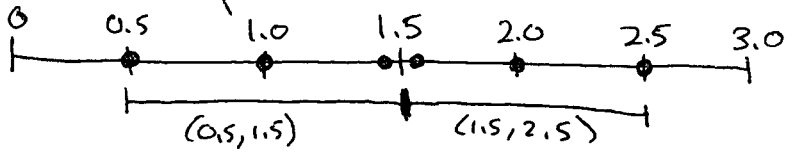
Consider the input P = {0.5, 1.0, 1.49, 1.51, 2.0, 2.5}



Clearly the interval $I_1 = (1, 2)$ covers the most points in P, so MP adds I_1 to S. Now MP has no choice but to add separate intervals $I_2 = (0.5, 1.5)$ and $I_3 = (1.5, 2.5)$ to cover the remaining two points (0.5 and 2.5, respectively).

Thus MP has arrived at S = $\{I_1, I_2, I_3\}$

Now consider S = {(0.5, 1.5), (1.5, 2.5)}



Clearly OPT(P) = S, $|OPT(P)| < |MP(P)|$, and MP is not optimal for P.

$\therefore$ MP is not correct.

(b)

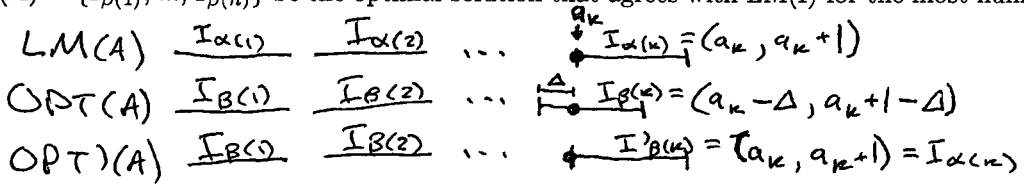
Input: A set of n points on the real line A where $a_i \in \mathbb{R}$.

Output: Minimum cardinality collection S of unit intervals that cover every point in A.

Theorem: The given "left-most" algorithm (LM) is correct / optimal.

Proof: Assume to reach a contradiction that there exists an input on which LM produces an unacceptable output.

Let I_i be unit-intervals
& LM(A) = $\{I_{\alpha(1)}, ..., I_{\alpha(n)}\}$ be the output of LM on input I
& OPT(A) = $\{I_{\beta(1)}, ..., I_{\beta(n)}\}$ be the optimal solution that agrees with LM(I) for the most number of steps



Let a_k be the first point where LM(I) differs from OPT(I)

By the definition of LM, we know that the interval which covers a_k is $I_{\alpha(k)} = (a_k, a_k + 1)$

In order to define $I_{\beta(k)}$, we should make a few observations:

- All points to the left of a_k are covered by some interval in OPT (by the definition of LM) and a_k is not covered
- The interval $I_{\beta(k)}$ can not start to the right of a_k (because a_k would not be covered) and $I_{\alpha(k)} \neq I_{\beta(k)}$
- Therefore, $I_{\beta(k)}$ must start to the **left** of a_k in OPT

So for some distance Δ, the interval in OPT which covers point a_k is $I_{\beta(k)} = (a_k - \Delta, a_k + 1 - \Delta)$

Let OPT' = OPT with $I_{\beta(k)}$ shifted to the right by Δ:

$$I'_{\beta(k)} = I_{\beta(k)} + (\Delta, \Delta) = (a_k, a_k + 1) = I_{\alpha(k)}$$

As stated earlier, every point a_i where i < k has been covered by some previous interval, therefore we can safely move the interval $I_{\beta(k)}$ to the right by a distance Δ. This move may cause a potential overlap to the right, however this overlapping is inconsequential

Further note that OPT' is still optimal because it does not create any additional intervals and covers all the points

$\therefore$ We have reached a contradiction because OPT' agrees with LM for one additional step despite OPT being defined as agreeing with LM for the most number of steps.

/

4/6

CS 1510
Algorithm Design

Greedy Algorithms

Problems 6 and 7

Due September 3, 2014

Buck Young and Rob Brown

11/14

Problem 6

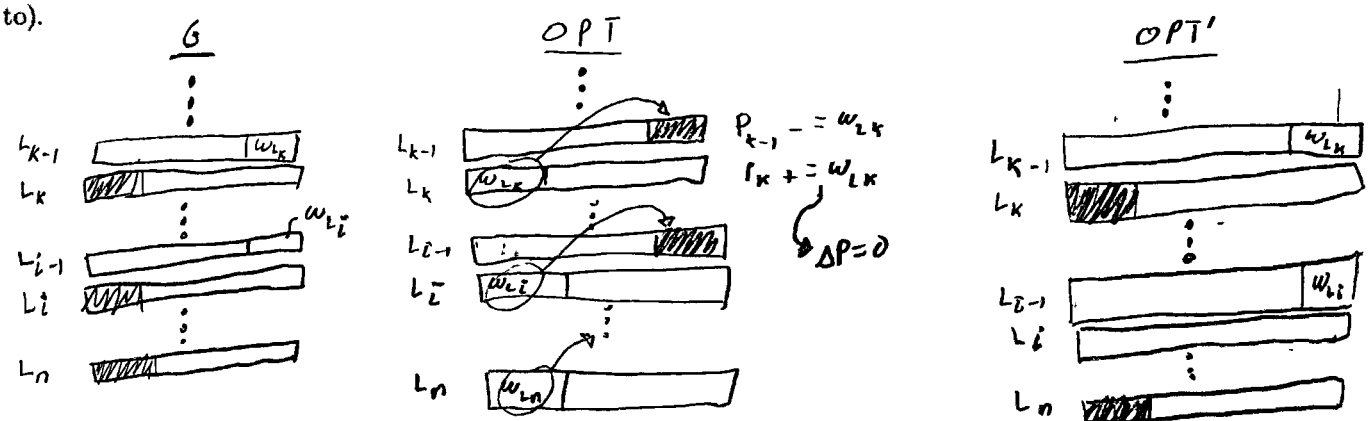
(a)

Input: An ordered sequence of words $S = \{w_1, w_2, ..., w_n\}$ where the length of the i^{th} word is w_i.

Output: Total penalty P as described by the problem statement (ΣP_i for each $P_i = K_i - L$) such that P is minimized.

Theorem: The given greedy algorithm G is correct.

Proof: Assume to reach a contradiction that there exists some input I on which G produces an unacceptable output. Let OPT be the optimal solution that agrees with G for the most number of steps. Let the first point of disagreement be the word w_{L_k} on line k. The only way OPT can disagree with G is if OPT moves w_{L_k} to a new line and G does not (obviously G will not fail to place a word on a line that it is able to).



Let OPT' be a modified version of OPT, such that the first word on each new line $(w_{L_k}, w_{L_{k+1}}, ..., w_{L_n})$ is moved to the previous line if possible. The logic governing this movement from the i^{th} line to the $(i-1)^{th}$ line is as follows.

1: **for** $i \geq k$ **do**
2: **if** $w_{L_i} \leq (L - K'_{i-1})$ **then**
3: **if** w_{L_i} is the ONLY word on the LAST line **then**
4: remove the final line, as the word fits on the previous line
5: P'_i changed from $L - w_i$ to 0
6: $P'_{i-1} -= w_i$
7: $\Delta P \leq 0$
8: **else**
9: Move w_i from L_i to L_{i-1}
10: $P_i += w_i$
11: $P_{i-1} -= w_i$
12: $\Delta P = 0$
13: **end if**
14: **else**
15: Cannot move w_{L_i} up
16: $\Delta P = 0$
17: **end if**
18: **end for**

Preferred English is fine; Do not use code

Thus, any state that OPT' can reach will result in $\Delta P = P' - P \leq 0$. This tells us that $OPT' \leq OPT$, and we have that OPT' is correct. Additionally, we know that OPT' can fit w_k on the prior line, since G made this selection. So for at least one word we are able to make OPT' more like G. Now OPT' is both correct and agrees with G for one step more than OPT.

$\therefore$ OPT' contradicts the definition of OPT, and G is correct.

(b) Input: An ordered sequence of words $S = \{w_1, w_2, ..., w_n\}$ where the length of the i^{th} word is w_i.

Output: Total penalty P as described by the problem statement (the maximum of the line penalties) such that P is minimized.

Theorem: The given greedy algorithm G correctly solves this problem.

Proof: Consider OPT as a counter-example to prove G incorrect.

L=6

G

1) b, y, e, b, y, e, $P_1 = \emptyset$

2) a, m, $P_2 = 4$

$\rightarrow P_G = 4$

OPT

1) b, y, e, $P_1 = 3$

2) b, y, e, a, m, $P_2 = 1$

$\rightarrow P_{OPT} = 3$

Clearly $P_G > P_{OPT}$. Since we want to minimize P, OPT is correct.

$\therefore$ Greedy algorithm G is not correct by way of this counter-example.

Problem 7

Algorithm: If a page is not in fast memory and an eviction must occur, evict the page that does not occur again or whose next use will occur farthest in the future.

Input: A sequence I = $\{P_1, P_2, ..., P_n\}$ of page requests

Output: Minimum cardinality of a sequence of evicted pages E

Theorem: The "farthest in the future" algorithm F is correct.

Proof: Assume to reach a contraction that there exists input I on which F produces unacceptable output.

Let F(I)=$\{P_{\alpha(1)}, P_{\alpha(2)}, ..., P_{\alpha(n)}\}$ be the output of F on input I
& OPT(I)=$\{P_{\beta(1)}, P_{\beta(2)}, ..., P_{\beta(n)}\}$ be the optimal solution that agrees with F(I) the most number of steps

Consider the first point of disagreement k between OPT and F

$F(I) = \{P_1, P_2, ..., P_{\alpha k}, ...\}$
$OPT(I) = \{P_1, P_2, ..., P_{\beta k}, ...\}$
$OPT)(I) = \{P_1, P_2, ..., P_{\alpha k}, ...\}$

Let OPT' equal OPT with $P_{\alpha(k)}$ chosen at k instead of $P_{\beta(k)}$

Clearly OPT' is more similar to F (for one additional step)

To show that OPT' is still correct, consider the four possible cases for input I:

Case 1

$I = \{P_1, P_2, ..., P_n\}$

In this case, neither $P_{\alpha(k)}$ nor $P_{\beta(k)}$ exist in the input after time-step k. Therefore $|E|$ and $|E'|$ do not change.

Case 2

$I = \{P_1, P_2, ..., P_{\beta k}, ... P_n\}$

In this case, only $P_{\beta(k)}$ exists in the input after time-step k. Since $P_{\beta(k)}$ is needed at some point in the future (and OPT has evicted it at time-step k), OPT is guaranteed to need at least one additional eviction. Therefore $|E| > |E'|$.

Case 3

$I = \{P_1, P_2, \overset{k}{\cdots}, P_{\beta k}, \cdots, P_{\alpha k}, \cdots, P_n\}$

In this case, both $P_{\alpha(k)}$ and $P_{\beta(k)}$ exist in the input after time-step k. Therefore $|E|$ must account for an eviction to bring $P_{\beta(k)}$ back into fast memory and $|E'|$ must account for an eviction to bring $P_{\alpha(k)}$ back into fast memory.

it is not clear in this case what OPT' is or why it's better.

Case 4

$I = \{P_1, P_2, \overset{k}{\cdots}, P_{\alpha k}, \cdots, P_n\}$

In this case, only $P_{\alpha(k)}$ exists in the input after time-step k. By the definition of F, if $P_{\beta(k)}$ did not exist in the input after time-step k then F would have chosen it. Therefore this is an impossible state.

In all cases, $|E| \geq |E'|$. Therefore (since we are minimizing) OPT $\leq$ OPT'

$\therefore$ We have reached a contradiction because OPT' agrees with F for one additional step (and still produces a correct output) despite OPT being defined as agreeing with F for the most number of steps.

8/8

CS 1510
Algorithm Design

Greedy Algorithms

Problems 9 and 10

Due September 5, 2014

Buck Young and Rob Brown

19/22

Problem 9

(a)

Input: A number n of skiers with heights $p_1, ..., p_n$, and n skies with heights $s_1, ..., s_n$.

Output: The minimal average difference between skier height p_i and ski height $s_{\alpha(i)}$: $\frac{1}{n}\sum_{i=1}^{n}|p_i - s_{\alpha(i)}|$

Theorem: The given "minimized height difference first" algorithm MF is correct.

Proof: Consider a "counter-example" algorithm CE to prove MF is incorrect.

Let T be the total height difference (and $\frac{T}{n}$ be the average total height difference), & H_i be the height difference between a person p_i and a chosen ski $s_{\alpha(i)}$.

Input: Skies = {10, 12, 18, 20}
Skiers = {16, 20, 22, 25}

MF

$H_1 = |P_2 - S_4| = |20 - 20| = 0$
$H_2 = |P_1 - S_3| = |16 - 18| = 2$
$H_3 = |P_3 - S_2| = |22 - 12| = 10$
$H_4 = |P_4 - S_1| = |25 - 10| = 15$

$\frac{T_{MF}}{n} = 27 \div 4 = (6.75)$

$\frac{T_{MF}}{n} = 6.75$

CE

$H_1 = |P_1 - S_1| = |16 - 10| = 6$
$H_2 = |P_2 - S_2| = |20 - 12| = 8$
$H_3 = |P_3 - S_3| = |22 - 18| = 4$
$H_4 = |P_4 - S_4| = |25 - 20| = 5$

$\frac{T_{CE}}{n} = 23 \div 4 = 5.75$

$\frac{T_{CE}}{n} = 5.75$

Clearly $\frac{T_{MF}}{n} > \frac{T_{CE}}{n}$ and the problem wants us to minimize $\frac{T}{n}$.

Also – let OPT be the optimal solution to this problem – we see that MF < CE ≤ OPT.

∴ The given algorithm MF is sub-optimal and not correct by way of this counter-example.

(b)

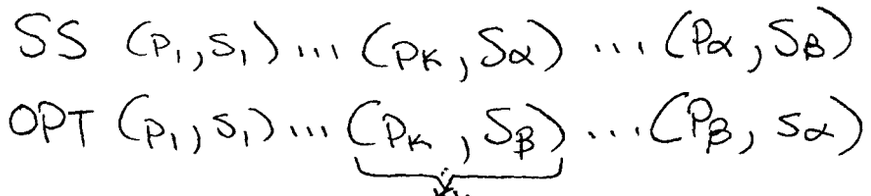
Input: A number n of skiers with heights $p_1, ..., p_n$, and n skies with heights $s_1, ..., s_n$.

Output: The minimal average difference between skier height p_i and ski height $s_{\alpha(i)}$: $\frac{1}{n}\sum_{i=1}^{n}|p_i - s_{\alpha(i)}|$

Theorem: The given "shortest-skier to shortest-ski" algorithm SS is correct.

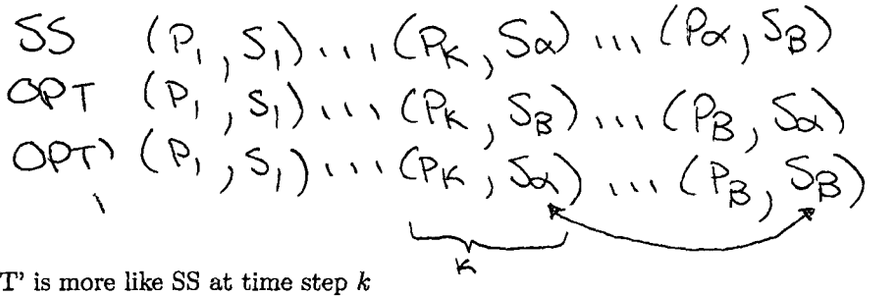
Proof: Assume to reach a contradiction that there exists an input on which SS produces an unacceptable output.

Let SS(I) be the output of SS on input I
& OPT(I) be the optimal solution which agrees with SS(I) the most number of steps



Let k be the first point in time where the pairing of skier to ski in SS(I) differs from the pairing in OPT(I)

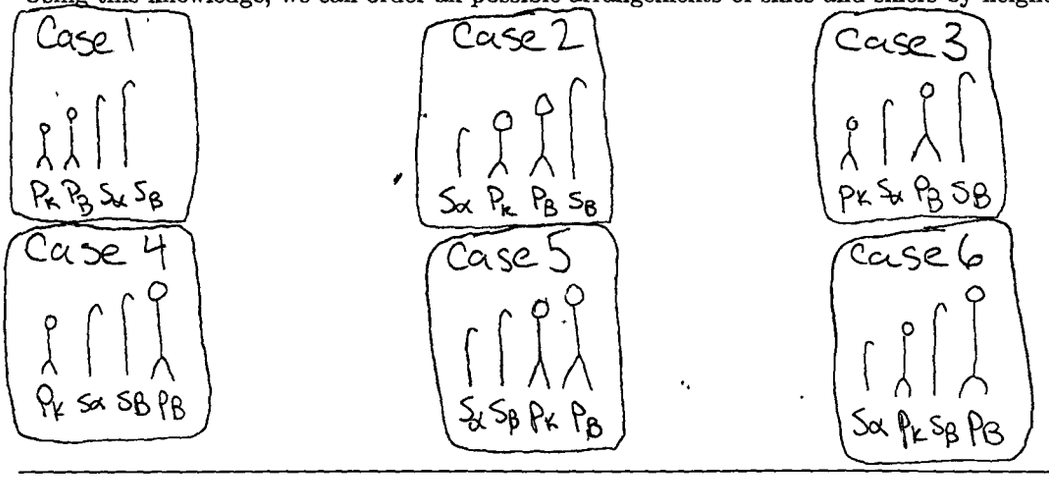
Let OPT'(I) = OPT with s_β and s_α swapped.
Note: p_α could be the same person as p_β but they do not have to be



Clearly OPT' is more like SS at time step k
Ultimately, we want to show that the average height difference between OPT and OPT' has not gotten worse (in order to prove that OPT' is still an acceptable solution).

First we make a few simple observations: by definition of SS, we know that $p_k < p_\beta$ and that $s_\alpha < s_\beta$

Using this knowledge, we can order all possible arrangements of skies and skiers by height:



Thus we have 6 clear cases and for each we must show that the average height difference between OPT and OPT' has not gotten worse. We can do this by using an inequality: $|p_k - s_\alpha| + |p_\beta - s_\beta| \leq |p_k - s_\beta| + |p_\beta - s_\alpha|$ (the height difference of OPT' is $\leq$ the height difference of OPT)

If we can prove this inequality valid for each of our six cases, we will have proven that OPT' is no worse than OPT. We may remove the absolute values in the inequality by simply reordering the terms to always produce a positive value (based on which of the terms is larger on a case-by-case basis)...

Case 1 : $p_k \leq p_\beta \leq s_\alpha \leq s_\beta$

$$\begin{array}{l} s_\alpha - p_k + s_\beta - p_\beta \leq s_\beta - p_k + s_\alpha - p_\beta \\ \underline{-s_\alpha + p_k - s_\beta + p_\beta \qquad -s_\beta + p_k - s_\alpha + p_\beta} \\ 0 \leq 0 \end{array}$$

Check.

Case 2 : $s_\alpha \leq p_k \leq p_\beta \leq s_\beta$

$$\begin{array}{l} p_k - s_\alpha + s_\beta - p_\beta \leq s_\beta - p_k + p_\beta - s_\alpha \\ \underline{+s_\alpha - s_\beta \qquad -s_\beta \qquad\qquad + s_\alpha} \\ p_k - p_\beta \leq -p_k + p_\beta \\ 2p_k \leq 2p_\beta \\ p_k \leq p_\beta \end{array}$$

Check.

Case 3 : $p_k \leq s_\alpha \leq p_\beta \leq s_\beta$

$$\begin{array}{l} s_\alpha - p_k + s_\beta - p_\beta \leq s_\beta - p_k + p_\beta - s_\alpha \\ \underline{+p_k - s_\beta \qquad -s_\beta + p_k} \\ s_\alpha - p_\beta \leq p_\beta - s_\alpha \\ 2s_\alpha \leq 2p_\beta \\ s_\alpha \leq p_\beta \end{array}$$

check.

Case 4 : $p_k \leq s_\alpha \leq s_\beta \leq p_\beta$

$$\begin{array}{l} s_\alpha - p_k + p_\beta - s_\beta \leq s_\beta - p_k + p_\beta - s_\alpha \\ \underline{+p_k - p_\beta \qquad\qquad +p_k - p_\beta} \\ s_\alpha - s_\beta \quad 2s_\alpha \leq 2s_\beta \quad s_\beta - s_\alpha \\ s_\alpha \leq s_\beta \end{array}$$

Check.

Case 5 : $s_\alpha \leq s_\beta \leq p_k \leq p_\beta$

$$\begin{array}{l} p_k - s_\alpha + p_\beta - s_\beta \leq p_k - s_\beta + p_\beta - s_\alpha \\ \underline{-p_k + s_\alpha - p_\beta + s_\beta \qquad -p_k + s_\beta - p_\beta + s_\alpha} \\ 0 \leq 0 \end{array}$$

check.

Case 6 : $s_\alpha \leq p_k \leq s_\beta \leq p_\beta$

$$p_k - s_\alpha + p_B - s_B \leq s_B - p_k + p_B - s_\alpha$$
$$+ s_\alpha - p_B \qquad\qquad - p_B + s_\alpha$$

$$p_k - s_B \leq s_B - p_k$$
$$2p_k \leq 2s_B$$
$$p_k \leq s_B \quad \text{check.}$$

Our inequality is valid for every case, thus we have shown that the height difference between OPT and OPT' has not gotten any worse.

$\therefore$ We have reached a contradiction because OPT' is still a valid solution and it agrees with SS for one additional step despite OPT being defined as agreeing for the most number of steps.

Problem 10

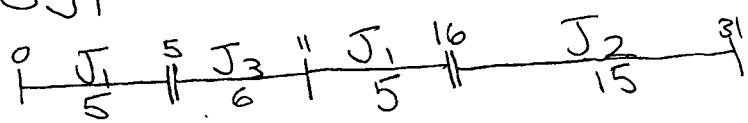
(SJF)
Input: A collection of jobs $J_1, ..., J_n$, where the ith job is a tuple (r_i, x_i) of non-negative integers specifying the release time r and size of the job x.
Output: A preemptive feasible schedule on one processor that minimizes the total completion time $\sum_{i=1}^{n} C_i$
Theorem: That the given "shortest job first" algorithm SJF is correct.

Proof: Consider a "counter-example" algorithm CE to prove SJF is incorrect.

Let C be the total completion time and C_i be the completion time for job i.

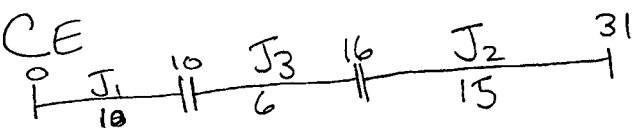
Input: $\{J_1=(0,10), J_2=(3,15), J_3=(5,6)\}$

SJF



$C_1 = 16$
$C_2 = 31$
$C_3 = 11$
$C_{SJF} = 58$

CE



$C_1 = 10$
$C_2 = 31$
$C_3 = 16$
$C_{CE} = 57$

Clearly $C_{SJF} > C_{CE}$ and the problem requires us to minimize C.

Also – let OPT be the optimal solution to this problem – we see that SJF < CE ≤ OPT.

∴ The given algorithm SJF is sub-optimal and not correct by way of this counter-example.

(SRPT)

Input: A collection of jobs $J_1, ..., J_n$, where the ith job is a tuple (r_i, x_i) of non-negative integers specifying the release time r and size of the job x.

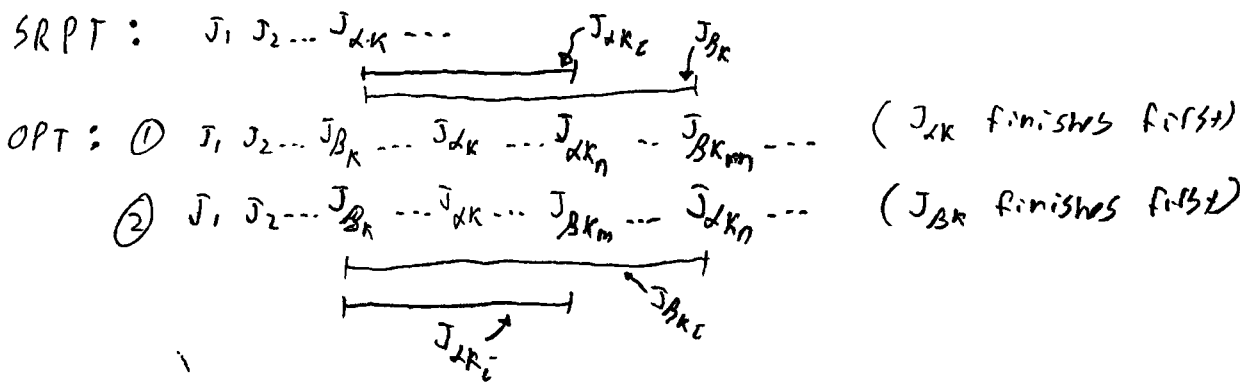
Output: A preemptive feasible schedule on one processor that minimizes the total completion time $\sum_{i=1}^{n} C_i$.

Theorem: The given "shortest remaining processing time" algorithm SRPT is correct.

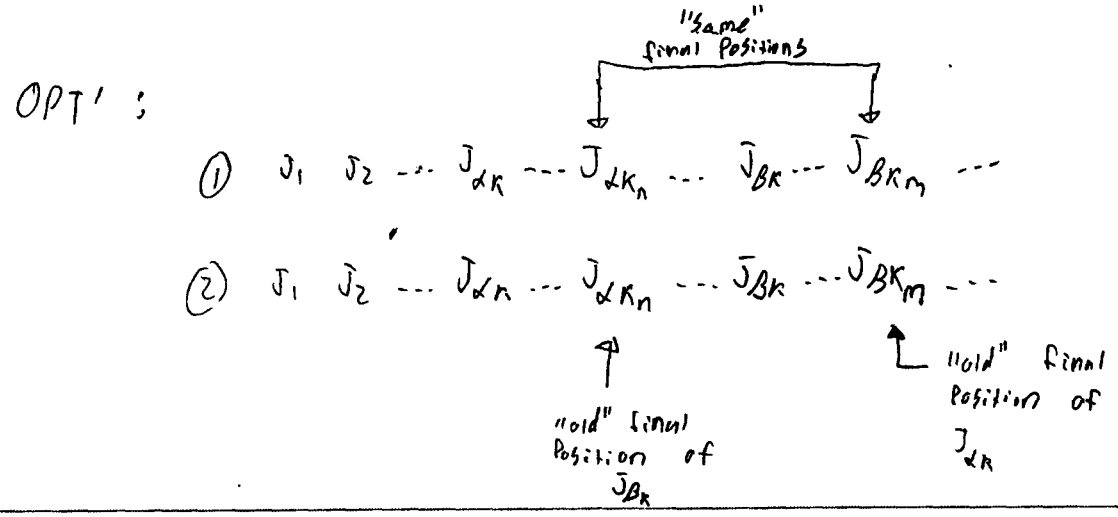
Proof:

Suppose, for the sake of reaching a contradiction, SRPT produces unacceptable output for some input I. Let $SRPT(I) = \{J_{a1}J_{a2}, ..., J_{an}\}$ be the output of SRPT, and and $OPT(I) = \{J_{a1}J_{a2}, ...J_{an}\}$ be the output of some optimal algorithm which agrees with SRPT for the most number of steps. For each $J_i \in$ OUTPUT, J_i identifies a job that executes for a single unit of time and r_i is the remaining number of time units required for job completion. [redacted].

Consider the output of these two algorithms. Let k be the first point of disagreement, where SRPT schedules J_{ak} which completes at J_{ak_n} and OPT schedules $J_{\beta k}$ which completes at $J_{\beta k_m}$. This disagreement can take the two following forms.



Let OPT' be OPT with each $S_{ak_i} \in \{S_{ak_1}, ..., S_{ak_n}\}$ swapped with $S_{\beta i} \in \{S_{\beta k_1}, ..., S_{\beta k_m}\}$ until every $S_{\beta i}$ appears AFTER every S_{ai} in the schedule. Note that $r_{ak} = |\{S_{ak}, ..., S_{akc}\}|$ and $r_{\beta k} = |\{S_{\beta k}, ..., S_{\beta kc}\}|$ and $r_{ak} \leq r_{\beta k}$. Consider OPT' for the two general forms above



CASE 1

$C'_\alpha \leq C_\alpha$ since $J_{\alpha k_C}$ no longer needs to wait for any $J_{\beta i}$, and $C'_\beta = C_\beta$ since $r_\alpha \leq r_\beta$ (and $J_{\beta k_C}$ is therefore never swapped). Thus $C'_\alpha + C'_\beta \leq C_\alpha + C_\beta$. Since no intervals were inserted/created (only swapped) OPT' has not affected any other completion times, and this translates into $\sum_{i=1}^{n} C_i$ being minimized for OPT'.

CASE 2

$C'_\alpha \leq C_\beta$ (in words, $J_{\alpha k} \epsilon OPT'$ completes at or before $J_{\alpha k} \epsilon OPT$) since all $J_{\alpha k_i}$ are swapped until they appear before all $J_{\beta k_i}$; and $C'_\beta = C_\alpha$ since the position of $J_{\alpha k_n}$ in OPT is swapped with some $J_{\beta k_i}$ to become the new completion time of $J_{\beta k}$ in OPT'). Thus $C'_\alpha + C'_\beta \leq C_\alpha + C_\beta$, and as in CASE 1 since no intervals were created/inserted, this translates into $\sum_{i=1}^{n} C_i$ being minimized for OPT'.

Clearly, OPT' is more like SRPT than OPT. We have also demonstrated the correctness of OPT' in the two feasible scenarios of disagreement between OPT and SRPT.

$\therefore$ We observe a contradiction implying the correctness of SRPT.

CS 1510
Algorithm Design

Greedy Algorithms & Dynamic Programming

Problems 12, 13 and 1

Due Monday September 8, 2014

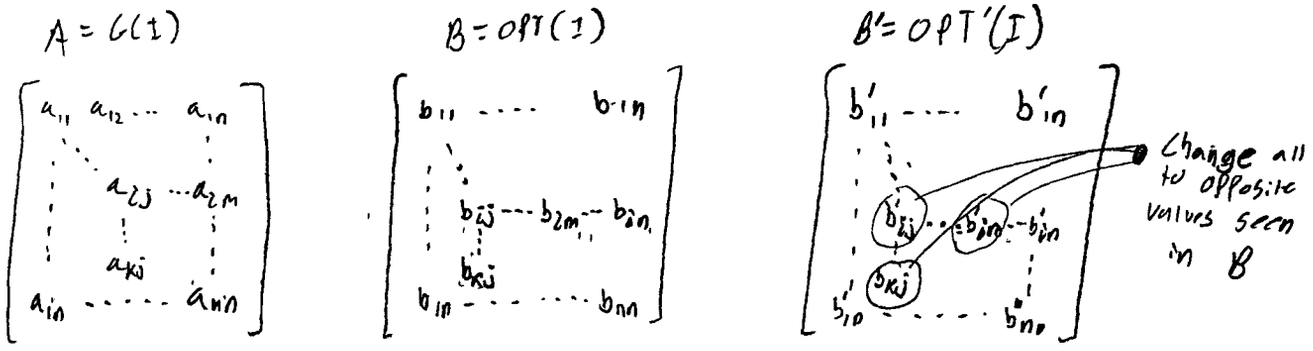
Buck Young and Rob Brown

Problem 12

Input: Positive integers $r_1, r_2, ..., r_n$ and $c_1, c_2, ..., c_n$
Output: An n by n matrix with 0/1 entries such that for all i the sum of the i^{th} row in a is r_i and the sum of the i^{th} column is c_i.
Theorem: The greedy algorithm (G) given in problem 12 is correct.

Proof: Suppose for the purpose of reaching a contradiction that there exists some input I consisting of two sets $\{r_1, r_2, ..., r_n\}$ and $\{c_1, c_2, ..., c_n\}$ such that G produces unacceptable output. Let $G(I) = A$ be the matrix that G produces, and let $OPT(I) = B$ be the matrix that some optimal algorithm which agrees for the most number of steps produces.

Consider the first pair of indices (i, j) where the indexed values of A disagree with those of B. In other words, consider (i, j) such that $\forall (x, y) \leq (i, j) : a_{ij} = b_{ij}$.



Since $a_{ij} = b_{ij}$, and both A and B consist entirely of 0/1 values, we have two cases:

1) $a_{i,j} = 0$ and $b_{i,j} = 1$
2) $a_{i,j} = 1$ and $b_{i,j} = 0$

Consider case 1. Since $b_{i,j} = 1$ and $(\sum_{w=1}^{n} b_{i,w}) = r_i$ we know that in some column (m) of B's i^{th} row there exists $b_{i,m} = 0$ where $b_{i,m} \neq a_{i,m}$ (ie, $a_{i,m} = 1$). A is constructed so that the a_i^{th} row has exactly r_i 1's. If OPT gives B a 1 at (i, j), it must compensate later by NOT having a 0 at some later point. This is the value we call $b_{i,m}$. Likewise, since OPT must produce $(\sum_{h=1}^{n} b_{h,j}) = c_j$, there must be some $b_{k,j}$ in the j^{th} column of B such that $b_{k,j} = 0$ and $a_{k,j} = 1$.

By the same logic, for case 2 we can deduce that there is some column (m) in B's i^{th} row where $b_{i,m} = 1$ and $a_{i,m} = 0$; likewise, there is some row k in B's j^{th} column where $b_{k,j} = 1$ and $a_{k,j} = 0$.

Let OPT'(I) = OPT(I) = B with the values of $b_{i,j}$, $b_{i,m}$, and $b_{k,j}$ changed to their opposite values.

CASE 1

$$\sum_{w=1}^{n} b'_{i,w} = r_i = r_i - b_{i,j} - b_{i,m} + !(b_{i,m}) + !(b_{i,j}) = r_i - 1 - 0 + 0 + 1 = r_i$$

$$\sum_{h=1}^{n} b'_{h,j} = c_j = c_j - b_{i,j} - b_{k,j} + !(b_{k,j}) + !(b_{i,j}) = c_j - 1 - 0 + 0 + 1 = c_j$$

CASE 2

$$\sum_{w=1}^{n} b'_{i,w} = r_i = r_i - b_{i,j} - b_{i,m} + !(b_{i,m}) + !(b_{i,j}) = r_i - 0 - 1 + 1 + 0 = r_i$$

$$\sum_{h=1}^{n} b'_{h,j} = c_j = c_j - b_{i,j} - b_{k,j} + !(b_{k,j}) + !(b_{i,j}) = c_j - 0 - 1 + 1 + 0 = c_j$$

We have thus demonstrated that, if OPT was indeed correct to begin with (ie, respective sums equaled c_j ad r_i), these modifications do not effect the correctness of OPT'. It is also evident that OPT' is more like G for one additional step.

$\therefore$ We have reached a contradiction because OPT' agrees with G for one additional step despite OPT being defined as agreeing for the most number of steps.

Problem 13

(a)

Input: A set of n jobs j – each job is described by an integer release time and deadline (r_j, d_j) – each of which must run for one unit of time. Additionally at each time interval t $\in [0, T)$, the machine may be ON or OFF at the time interval $(t, t+1)$. Let T equal the maximum deadline – $\max(d_j)$ for all j.

Output: Whether or not a feasible sequence was found in which every job ran for one unit of time at some interval after r_j and before d_j.

Algorithm: Run the job with the earliest deadline first:

while $t < T$ **do**
  if the machine is on at time t **then**
    for each job that has not yet run && is released – $(r_j \leq t)$ **do**
      Run the job with the earliest deadline – $(\min(d_j)$ for all $j)$ – (break ties arbitrarily)

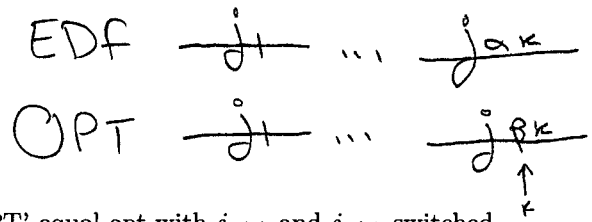
if all jobs in the input I have run by time T **then**
  return FEASIBLE
else
  return NOT FEASIBLE

Theorem: The described "earliest deadline first" algorithm EDF is correct.

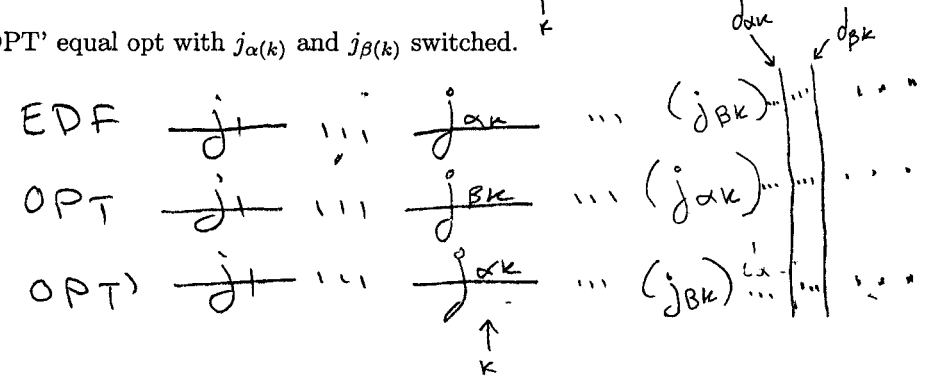
Proof: Assume to reach a contradiction that there exists an input on which EDF produces an unacceptable output.

Let EDF(I) = $\{j_{\alpha(1)}, ..., j_{\alpha(n)}\}$ be the output of EDF on input I
& OPT(I) = $\{j_{\beta(1)}, ..., j_{\beta(n)}\}$ be the optimal solution which agrees with EDF(I) for the most number of steps

Let k be the first point of disagreement between EDF and OPT



Let OPT' equal opt with $j_{\alpha(k)}$ and $j_{\beta(k)}$ switched.



Clearly OPT' is more like EDF. In order to prove that it still produces an acceptable output, consider the following four cases. Also note that by the definition of EDF, we know that the deadline $d_{\alpha(k)}$ is earlier

than (or tied with) the deadline $d_{\beta(k)}$. Stated another way: $d_{\alpha(k)} \leq d_{\beta(k)}$

Case 1: Both EDF and OPT determined the solution was FEASIBLE

Since we know $d_{\alpha(k)} \leq d_{\beta(k)}$, we know that $d_{\beta(k)}$ is after the scheduled $j_{\alpha(k)}$ in OPT. Therefore we can easily switch $j_{\alpha(k)}$ and $j_{\beta(k)}$ and both jobs will complete before their deadlines. Therefore an acceptable solution is still produced.

Case 2: Both EDF and OPT determined the solution was NOT FEASIBLE

Since a NOT FEASIBLE solution was determined, we know that some job was unable to be scheduled before its deadline. Therefore switching $j_{\beta(k)}$ and $j_{\alpha(k)}$ (or scheduling one and not the other) should have no affect on the final outcome and an acceptable solution is still produced.

Case 3: EDF determined FEASIBLE but OPT determined NOT FEASIBLE

3

This state is impossible by the definition of OPT. If EDF was able to create a FEASIBLE solution, then OPT is guaranteed to also produce a FEASIBLE solution, by definition.

Case 4: EDF determined NOT FEASIBLE but OPT determined FEASIBLE

In this case, we can do the switch using what we know about the deadlines ($d_{\alpha(k)} \leq d_{\beta(k)}$) and OPT' still produces an acceptable solution.

All in all, OPT' always produces an acceptable solution.

$\therefore$ We have reached a contradiction because OPT' is more like EDF for one more step even though OPT was defined as being most like EDF.

Problem 13

(b)

Input: A set of n jobs j – each job is described by an integer release time and deadline (r_j, d_j) – each of which must run for one unit of time. Let T equal the maximum deadline – $\max(d_j)$ for all j. Additionally, a positive integer L is given which defines the length of an ON state.

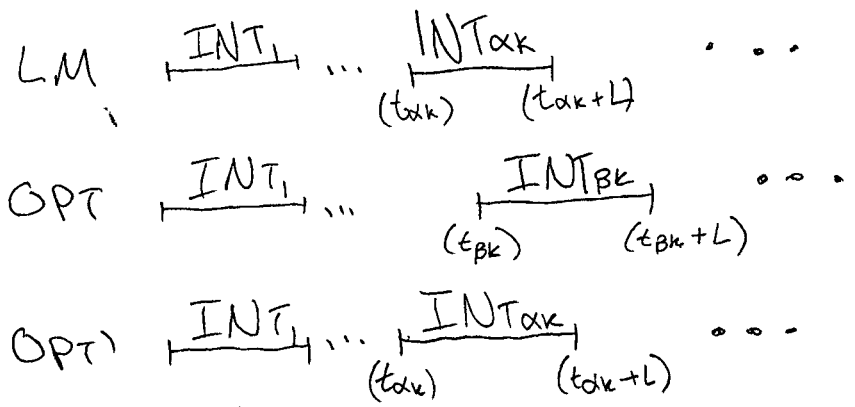
Output: The minimum number k times that the machine must turn ON in order to feasibly schedule all jobs within the intervals of length L.

Algorithm: While there are schedulable jobs in the input set, place an interval of length L at the left-most point t where that interval $(t + L)$ contains the most schedulable jobs $\leq L$. Then schedule jobs within that interval according to the earliest deadline first. } Not correct
(Note that a schedulable job is one that has been released and is not past it's deadline.)

Theorem: The described "left most" algorithm LM is correct.

Proof: Assume to reach a contradiction that there exists an input I on which LM produces and unacceptable output.

Let INT_1, ..., INT_n be the created intervals of length L starting at t_1, ..., t_n
& LM(I) = $\{INT_{\alpha(1)}, ..., INT_{\alpha(n)}\}$ be the output of LM on input I
& OPT(I)= $\{INT_{\beta(1)}, ..., INT_{\beta(n)}\}$ be the optimal solution which agrees with LM the most number of steps
& k be the first point of disagreement between LM and OPT



Note: we know that $t_{\alpha(k)} < t_{\beta(k)}$, by the definition of LM
This is tricky because it also chooses interval based on most # of jobs

Let OPT' equal OPT with $INT_{\beta k}$ shifted to the left such that it starts at $t_{\alpha(k)}$ and is, of course, of length L – and therefore becomes (indistinguishable from) $INT_{\alpha(k)}$.

Clearly OPT' is more like LM for one additional interval.

A few observations about the shift:
1) OPT clearly hasn't missed scheduling a job, by the definition of OPT (therefore the space from $t_{\alpha(k)}$ to $t_{\beta(k)}$ is void of a completely unscheduled job in OPT) } This is not clear since the interval changed.

2) OPT has not created an interval that scheduled more jobs than $INT_{\alpha(k)}$, by the definition of LM

Thus, shifting $INT_{\beta(k)}$ to the left so that it becomes $INT_{\alpha(k)}$ in OPT' will still produce an acceptable output.

$\therefore$ We have reached a contradiction because OPT' agrees with LM for one additional step despite OPT being defined as agreeing with LM for the most number of steps.

Problem 1

(a)

```
procedure T(int n)
    if n < 0 then
        return -1
    if n == 0 or n==1 then
        return 2;
    SUM = 0
    for 1 ≤ i ≤ n − 1 do
        SUM+=T(i)*T(i-1)
    return SUM
```

Not true.

Recursive algorithms which only reduce the input by a constant size each call are inherently exponential. Here, we have an algorithm which does this (ie, only decreases the input size by a constant size each call), and additionally creates this many more recursive calls at each point in the recursion. Clearly, it is exponential.

(b)

```
procedure T(int n)
    if n < 0 then
        return -1
    if n == 0 or n==1 then
        return 2;
    T[0] = 2, T[1] = 2
    for 2 ≤ i ≤ n do
        SUM = 0
        for 1 ≤ j ≤ n − 1 do
            SUM+=T[j]*T[j-1]
        T[i] = SUM
    return T[n]
```

Need english descriptions of algorithms as well

Clearly this algorithm is $\mathcal{O}(n^2)$. This is achieved by starting at the base ($n = 0, n = 1$) and computing the the next value based on the sum of all the previous; as opposed to starting at the top and recomputing everything at each step closer to the base.

(c)

```
procedure T(int n)
    if n < 0 then
        return -1
    if n == 0 or n==1 then
        return 2;
    T[0] = 2, T[1] = 2, T[2] = 4
    for 3 ≤ i ≤ n do
        T[i] = T[i-1]*T[i-2] + T[i-1]
    return T[n]
```

Clearly this algorithm is $\mathcal{O}(n)$.

/

G 8/14

CS 1510
Algorithm Design

Greedy Algorithms & Dynamic Programming

Problems 16, 2, and 3

Due Wednesday September 10, 2014

Buck Young and Rob Brown

32/48

Problem 16

(a)

Input: A linear graph of n nodes (ie, a linked list of length n) and set of k packets $p_1, p_2, ..., p_k$ where each packet $p_i = (A, r_p, s_p, t_p)$ where A is the packet name, r_p is the packet release time, s_p is the packet start node, and t_p is the packet's target node.

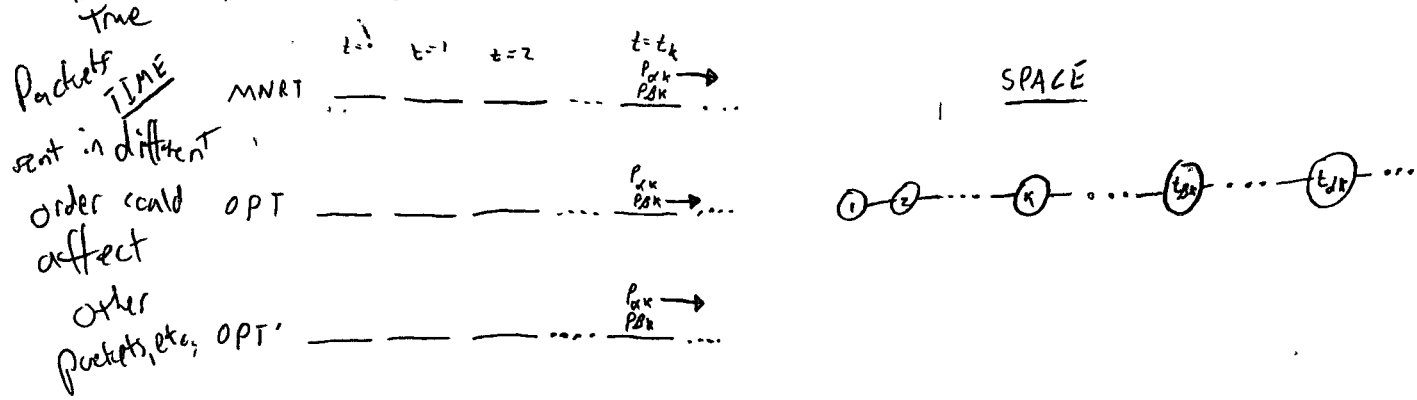
Output: The order of "sent" packets for each of the n nodes such that the maximum time that a packet takes to leave the network is minimized (ie, the network is cleared of all packets as quickly as possible).

Algorithm: For for the i^{th} node, take the the j^{th} packet from its packet-list such that the number of remaining nodes $NR_j = t_{pj} - i$ is the maximum of all packets in the node's packet-list. Put p_j in i's "sent" list for this time-step, and move p_j to node $i+1$.

Proof: The given "Most Remaining Nodes First" (MRNF) algorithm is correct.

Suppose for the sake of reaching a contradiction that there exists some input I on which MNRF produces unacceptable output. Let MNRF(I) be the output of MNRF on this input, and let OPT(I) be the output of some optimal algorithm which agrees with MNRF for the most number of steps.

Consider a given point where the output of MNRF disagrees with that of OPT. We define a "disagreement" to be any point where OPT sends a packet from a given node that violates the "Most Remaining Nodes First" principle. Note, however, that the only important differences are those which occur on the last packet to leave the network (in time), since we are concerned solely with the minimization of this value. Thus, consider this point to be the first disagreement and label this the k^{th} step.

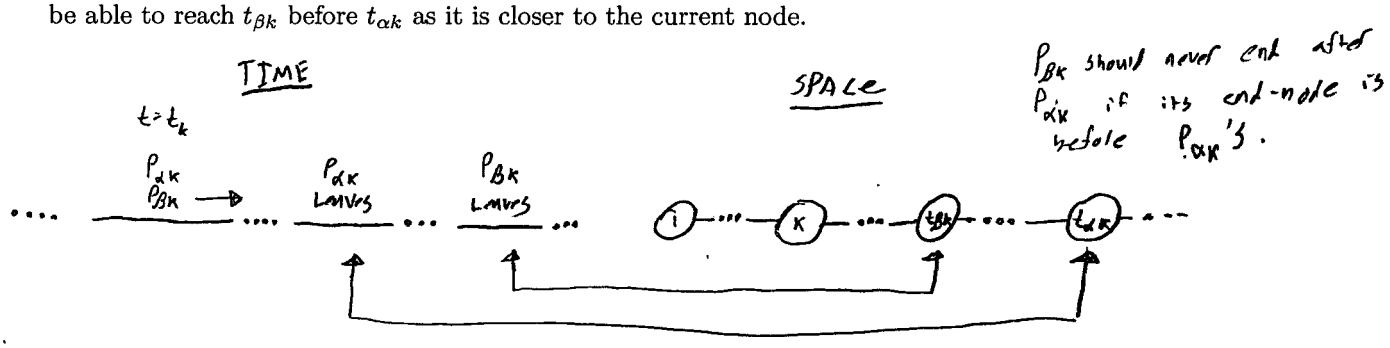


Let OPT' be some algorithm such that its output is that of OPT, but where $p_{\alpha k}$ is sent at the point of disagreement instead of $p_{\beta k}$.

Note that if neither $p_{\beta k}$ or $p_{\alpha k}$ are the last packets to leave the network in the schedule produced by OPT, then this single modification can be made without repercussion.

Now consider the more interesting cases where one of these two packets is the final packet to leave the network according to the schedule of OPT. If $p_{\alpha k} \in OPT$ is the last packet to leave the network, then we have that $MNRF \leq OPT$ since $p_{\alpha k} \in MNRF$ is defined to be the last packet to leave the system (ie, both OPT and MNRF have $p_{\alpha k}$ leaving the system last). OPT choosing to hold this packet up can only have detrimental effects on its time in the system. Thus $MNRF \leq OPT$.

In the case where $p_{\beta k} \in OPT$ is the last packet out the schedule produced by OPT, we have the following two diagrams of the system (one for space, one for time). There are $t_{\alpha k} - t_{\beta k}$ nodes between the exit node of $p_{\alpha k}$ and $p_{\beta k}$, and we know that the exit node of $p_{\alpha k}$ comes later later in the network than that of $p_{\beta k}$ by the definition of $MNRF$. We also know that $p_{\beta k}$ takes longer to leave the network than $p_{\alpha k}$ (and in fact takes the longest time to leave the network). This should never be, since an optimal algorithm would be able to reach $t_{\beta k}$ before $t_{\alpha k}$ as it is closer to the current node.



So in this case $OPT' \geq OPT$ and we can safely make our exchange (or simply raise a contradiction on the definition of OPT).

$\therefore$ for all covered cases $OPT \leq OPT'$ and OPT' is clearly more similar to $MNRF$ than OPT $\square$

Problem 16

(b)

For this problem, a first-in first-out (FIFO) algorithm would be optimal. That is, we can queue up the packets as they are released and move them along the edge in the order they are queued. Ties would be broken arbitrarily.

Problem 2

The following iterative, array-based, bottom-up algorithm will return the longest sequence S that is a subsequence of three strings A, B, and C in polynomial (n^3) time.

```
string x, y, z, result = ""
string LCS[A.length][B.length][C.length] = "" // Initialize to empty strings

// Traverse from 1 to size length (leave all a=0, b=0, c=0 as empty strings)
for a = 1 to A.length:
    for b = 1 to B.length:
        for c = 1 to C.length:

            if A[a-1] == B[b-1] == C[c-1]: // Access the strings from 0 to length-1
                LCS[a][b][c] = LCS[a-1][b-1][c-1] + A[a-1] // Add the character to the solution

                if LCS[a][b][c].length > result.length: // Store iff longest substring
                    result = LCS[a][b][c]
            else:
                x = LCS[a-1][b][c]
                y = LCS[a][b-1][c]
                z = LCS[a][b][c-1]

                // Find the maximum substring amongst x, y, and z and add it to the solution
                if x.length > y.length && x.length > z.length:
                    LCS[a][b][c] = x
                elif y.length > x.length && y.length > z.length:
                    LCS[a][b][c] = y
                else:
                    LCS[a][b][c] = z // All equal or z.length is greatest

return result
```

Problem 3

The following iterative, array-based, bottom-up algorithm will compute a table for finding the shortest common super-sequence of two strings A and B.

```
SCSS[A.length][B.length] = min(A.length, B.length) // Initialize to shortest string size

for a = 1 to A.length:
    for b = 1 to B.length:
        if A[a-1] == B[b-1]: // Access strings from 0 to length-1
            SCSS[a][b] = SCSS[a-1][b-1] - 1
        else:
            SCSS[a][b] = min( SCSS[a-1][b], SCSS[a][b-1] )
```

(a)

		Z	X	Y	Y	Z	Z
	6	6	6	6	6	6	6
Z	6	5	5	5	5	5	5
Z	6	5	5	5	5	4	4
Y	6	5	5	4	4	4	4
X	6	5	4	4	4	4	4
Z	6	5	4	4	4	3	3
Y	6	5	4	3	3	3	3

(b) The length of the SCSS will be the smallest number in the table plus min(A.length, B.length).

(c) Unfortunately, the created algorithm doesn't seem to be correct – as a backwards trace is confusing and inconsistent. However, we were hoping for something along the lines of: when the letters are unequal, travel towards min(vertical up, horizontal left) and write the letter depending on the direction / when the letters are equal travel diagonally and write the letter.