Due: Thu Nov 17, in class.

HOMEWORK with SOLUTION, prepared by Instructor and T.A.s

INSTRUCTIONS:

• Please write clearly, and economically. Use notations we introduce in lecture notes and in lectures.

• If you cannot write clearly, then consider typesetting the solutions (not recommended in general because this takes too much time).

• AGAIN: you get 2 bonus points just for writing your name in the correct format: [LAST-NAME-in-CAPS] [comma] [First-Name]. E.g., ”Y AP, Chee K.” or ”Y AP, Chee” are good, but ”Yap, Chee” or ”Chee Yap” or ”Yap Chee K.” are no good.

1. (5 Points, Radius and Diameter)
One of the scratched problems in hw5 is about the radius and diameter of a connected bigraph $G$. For the current problem, we want you to prove a nice observation of your fellow classmate, Zhao Jin: assuming $G$ is also acyclic, then

$$\text{radius}(G) = \lceil \text{diameter}(G)/2 \rceil.$$  

(1)

REMARKS: This shows that for acyclic connected bigraphs, you can reduce the computation of radius to the computation of diameter. But could you reduce the computation of diameter to radius?

Recall that hw5 showed

$$\text{radius}(G) \leq \text{diameter}(G) \leq 2\text{radius}(G).$$  

(2)

Both inequalities in (2) are tight: for all $n$ there are graphs $H_n$ where diameter($H_n$) = radius($H_n$) = $n$ and also graphs $G_n$ where $n$ = diameter($G_n$) = 2radius($G_n$). Thus, (1) tells us that the graphs $G_n$ cannot be acyclic.

SOLUTION To show (1), we need to prove a lower and an upper bound on radius($G$). From (2), we have

$$2\text{radius}(G) \geq \text{diameter}(G),$$

which is equivalent to $\text{radius}(G) \geq \text{diameter}(G)/2$. Recall our trick: if $x \geq y$ and $x$ is integer, then this is equivalent to the apparently stronger statement, $x \geq \lceil y \rceil$. Thus we conclude that $\text{radius}(G) \geq \lceil \text{diameter}(G)/2 \rceil$. This gives our lower bound on radius($G$).

Next, to show an upper bound on radius($G$), we must exploit the fact that $G$ is acyclic. Let $m$ be the “median” element in the unique path from $u$ to $v$. Wlog, assume $\delta(m, u) = \lceil \delta(u, v)/2 \rceil = \lceil \text{diameter}(G)/2 \rceil$ and $\delta(m, v) = \lceil \delta(u, v)/2 \rceil$. We claim that

$$\text{radius}(m, G) \leq \lceil \text{diameter}(G)/2 \rceil.$$  

(3)

If this claim is false, then there is some $x$ such that $\delta(m, x) > \lceil \text{diameter}(G)/2 \rceil$. Since $G$ is connected and acyclic, either (i) $\delta(x, u) = \delta(x, m) + \delta(m, u)$ or (ii) $\delta(x, v) = \delta(x, m) + \delta(m, v)$. To see this, let $p$ be the point on the unique path from $u$ to $v$ such that $\delta(x, u) = \delta(x, p) + \delta(p, u)$ and $\delta(x, v) = \delta(x, p) + \delta(p, v)$. If $m$ between $p$ and $v$ then (i) holds; otherwise (ii) holds. If (i), we conclude that $\delta(x, u) > \lceil \text{diameter}(G)/2 \rceil + \lceil \text{diameter}(G)/2 \rceil \geq \text{diameter}(G)$. If (ii), we conclude that $\delta(x, v) > \lceil \text{diameter}(G)/2 \rceil + \lceil \text{diameter}(G)/2 \rceil = \text{diameter}(G)$. Both yields a contradiction. Thus (3) is true. But (3) implies that radius($G$) $\leq$ radius($m, G$) $\leq \lceil \text{diameter}(G)/2 \rceil$, which is our desired upper bound on radius($G$).
REMARK: For completeness, let us prove (2) (from the scratched problem). Suppose $\delta(u, v) = \text{diameter}(G)$ and $c$ is a center. Then clearly, $\delta(u, v) \geq \delta(c, w)$ for all $w$. Hence $\delta(u, v) \geq \text{radius}(c, G) = \text{radius}(G)$, which is one half of (2).

Next we prove the other half of (2),

\[ 2\text{radius}(G) \geq \text{diameter}(G). \quad (4) \]

We have $\delta(u, v) \leq \delta(c, u) + \delta(c, v)$ (triangular inequality), and so $\delta(u, v) \leq 2\text{radius}(c, G) = 2\text{radius}(G)$. This proves (2).

To show the tightness of (2): For any natural number $n$, let $G_n$ be the “ring graph” on vertices $V = \{1, 2, \ldots , 2n\}$ and the edge set is $E = \{i-i+1 : i = 1, \ldots , 2n-1\} \cup \{2n-1\}$. Clearly,

\[ \text{diameter}(G_n) = \text{radius}(G_n) = n. \]

Let $H_n$ be the “line graph” on vertices $V = \{1, 2, \ldots , n+1\}$ and edges $E = \{i-i+1 : i = 1, \ldots , n\}$. Clearly,

\[ \text{diameter}(H_n) = n, \text{radius}(H_n) = \lceil n/2 \rceil. \]

2. (12+12 Points, $(a, b)$-search trees)

We consider the tradeoffs in using one of the following schemes to organize the nodes of an $(a, b)$-search tree: (i) an array, (ii) a singly-linked list, (iii) a doubly-linked list, (iv) a balanced binary search tree.

Consider a specific numerical example: block size is $4096 = 2^{12}$ bytes, and each block pointer is 4 bytes, and each key 6 bytes. A local pointer within the block uses 12 bits, but for simplicity treat this as two bytes. Please be sure to note other information you need in a node, such as a parent pointer, the degree, etc.

(a) What is the maximum value of the parameter $b$ under each of the schemes (i)-(iv)? Be sure to show your calculations. The lecture notes has some discussion of these issues.

(b) What is the worst case time to search for a key in an $(a, b)$-search tree with two million items? We need one number (not expression) for each of the four schemes as answer: the unit for each number is CPU cycles (or “CPUC”).

MAKE THESE ASSUMPTIONS: Each disk I/O takes 1000 CPU cycles. If searching for a key takes $O(\log n)$ or $O(n)$ CPU time, always assume that “4” is the constant in big-Oh notation. E.g., searching for a key in a balanced BST with $n = 100$ keys takes $4\log n = 4 \times 7 = 28$ CPUC. Searching for in a list of length $n = 100$ takes $4n = 400$ CPUC. The root of the search tree is always in main memory, so you never need to read or write the root. Assume $a = \lceil (b + 1)/2 \rceil$. You can use calculators, but I encourage you to make calculator-free simplified estimates whenever possible (e.g., log base 2 of one million is 20).
SOLUTION  (a) Regardless of which method (i)-(iv) is used, we need to keep track of the degree of the node (2 bytes), and parent pointer (4 bytes). Note that knowing the degree is useful to check if a node is overfull or underfull. So the available memory in each block is $4096 - 6 = 4090$.

(i) Array representation. With $b$ children, we need $10b$ bytes. So $10b \leq 4090$. So we can choose $b = 409$.

Before providing the solution for (ii)-(iv), we note a common feature of these three solutions: you want to divide the space in a block into a set of nodes of some fixed size. Each node is either used or free. When you insert into your data structure (linked list, doubly-linked list or BST), you need to get a free node. When you delete, you need to create a free node. Thus, you must manage the set of free nodes, and the usual way is to put them together into a linked list called a FREELIST. In this FREELIST, you can use one of the local pointers in the nodes to get to the next free node. Therefore, the management of FREELIST has no impact on the size of your nodes. But you need to allocate 2 bytes in the block to point to the beginning of the FREELIST; so each block now has 4088 bytes.

(ii) Singly-linked list. Each node of the linked list needs 12 bytes: 4 bytes for a block pointer, 6 bytes for the key, and 2 bytes for the next node local pointer. Thus the number of $12b \leq 4088$. So the largest possible choice is $b = 340$.

(iii) Doubly-linked list. Each node needs 14 bytes, because we need an extra local pointer compared to (ii). Thus the maximum value of $b$ is $\lceil 4088/14 \rceil = 292$.

(iv) Balanced binary search tree. For our purposes, this can be assumed to be an AVL tree. Each node needs 16 bytes: this is two bytes more than (iii) because each node needs a left-, right- and parent pointer. So the number of bytes is 16, and a simplistic answer for this part is that $b = \lceil 4088/16 \rceil = 255$. But this solution is too optimistic. Let us see why. Each AVL node needs to maintain two bits of balance information (we might as well allocate one byte to this. A more important issue is this: when you lookup a key $K$, you get back the closest key $K_i$ in the BST. We may have $K < K_i$ or $K \geq K_i$. Thus, you need to get to either $P_i$ or $P_{i-1}$ where $P_i$’s are the pointers to children nodes of the $(a,b)$-search tree. Suppose you store $P_i$ with $K_i$. But how to you get to $P_{i-1}$? The simplest solution is to assume that your AVL nodes also has a predecessor pointer (these are local pointers). But in order to maintain predecessor pointers, we also need to maintain successor pointers. That means that each AVL node needs, not 16 bytes, but 21 bytes (1 byte for balance info, four bytes for succ/pred pointers). So $b = \lceil 4088/21 \rceil = 194$.

(b) Two million $2 \times 10^6$ is roughly $2^{21}$. For worst case, I will that the degree of the root is 2 and the degree of each internal node is $a$ in an $(a,b)$-search tree. So the depth of our search tree is $1 + \lceil \log_{10} b \rceil$. You can take $\log_{10} b = 20$.

(i) Array representation: $a = 205$ and height is $1 + \lceil 20/ \log(205) \rceil = 1 + [2.59] = 4$. So the height is 4 and the time to reach the leaf takes 4 reads, or 4000 CPUC. In each node, we need to search the keys in an array to determine the child pointer. We can use binary search in an array, and time is $4 \lceil \log a \rceil = 4 \times 8 = 32$ CPUC. We need to do this search on four nodes, so this takes $4 \times 32 = 128$ CPUC. So answer is 4000 + 128 = 4128 CPUC.

(ii) Singly-linked list. $a = 170$, and height is $1 + \lceil 20/ \log(170) \rceil = 1 + [2.69] = 4$. In each node, we need to search the keys in the linked list to determine the child pointer, and time is linear, $4a = 820$ CPUC. We need to do this search on four nodes, so this takes $4 \times 820 = 3280$ CPUC. So answer is 4000 + 3280 = 7280 CPUC.

(iii) Doubly-linked list. $a = 146$, and height is $1 + \lceil 20/ \log(146) \rceil = 1 + [2.77] = 4$. So the answer is the same in (i).

(iv) Balanced BST. $a = 194/2 = 97$, and height is $1 + \lceil 20/ \log(97) \rceil = 1 + [3.02] = 5$. Now, searching is a Balanced BST of height $\lceil \log(97) \rceil$ times time $4 \times \lceil \log(97) \rceil = 28$ CPUC. Doing this at five nodes takes time $5 \times 28 = 140$ CPUC. So answer is 5000 + 140 = 5140 CPUC.

CONCLUSION? the array solution is the fastest. Balanced BST comes next.
3. (0 Points – do not hand in)
Do part (b) of the previous problem, but for the Insert and Delete operations.

4. (5 Points, General Bin Packing)
Recall that the general bin packing problem can be solved in time to $O((n/e)^{n+1/2})$ (see Lemma 2 of Lect.V, p.4) We ask you to improve this to $O((n/e)^{n-(1/2)})$.
HINT: Repeat the trick which saved us a factor of $n$ in the first place. Fix two weights $w_1, w_2$. We need to consider two cases: either $w_1, w_2$ belong to the same bin or they do not.

**SOLUTION** The idea is to fix two weights $w_0$ and $w_1$. There are two possibilities: the two weights either belong to the same bin, or they do not.
(CASE 1: Same Bin) In this case, we generate all $(n-2)!$ permutations of $w_2, \ldots, w_n$, and for each of these permutations, we append $w_1, w_2$ and we solve linear bin packing problem.
(CASE 2: Different Bins) In this case, we take each of the $(n-2)!$ permutations, and prepend $w_1$ at the front, and append $w_2$ at the back of the permutation. Then we solve linear bin-packing on this instance.
Taking the best among these $2 \times (n-2)!$ solutions, we obtain the optimum. Thus the complexity of this solution is $O(n(n-2)!)$ = $O((n-1)!)$ time.
Comments: Food for thought: we can keep shaving off constant factors of $n$ in this way. But what about shaving off non-constant factors of $n$?

5. (3 Points, 2-Car Loading Policies)
We consider two policies for loading a front and rear car. Let us introduce a useful concept: if the total weight of the riders loaded in a car is $W$, then we say the car has residual capacity of $M - W$. Thus an empty car has residual capacity of $M$, and a completely full car has residual capacity of 0. If a new rider has weight at most the residual capacity of a car, then we say the rider fits into that car. Here are two possible loading policies for $G_2$ :

- **First Fit Policy**: Load each rider into the front car if it fits, otherwise load into the rear car if it fits. If it fits neither car, dispatch the front car.
- **Best Fit Policy**: Load each rider into the car with the “best fit”, i.e., the car with the minimum residual capacity that fits the rider. If it fits neither car, then dispatch the front car.

Let $G_2(w)$ and $G_{2}'(w)$ denote (respectively) the number of cars used when loading according to the First Fit and Best Fit Policies. Note that the text describes only the First Fit Policy, and proves that $G_2(w) \leq G_1(w)$.
(a) Show an example where $G_2(w) > G_{2}'(w)$.
(b) Show an example where $G_2(w) < G_{2}'(w)$.

**SOLUTION** Assume $M = 40$.
(a) Let $w = (20, 22, 8, 19)$. Then $G_2(w) = 3 > 2 = G_{2}'(w)$.
(b) Let $w = (20, 22, 8, 35, 13)$. Then $G_2(w) = 3 < 4 = G_{2}'(w)$.
Comments:

6. (0 Points, Sorted linear bin packing)
This problem seems too hard, so I decided to scratch it. Below is the modified the problem which you can try to solve for extra credit. (Please note do not submit partial solutions for partial credit.)
Here is the greedy algorithm for this problem. It is for the general bin packing, NOT linear bin packing. Wlog, assume the bin capacity is 1 and the weights are sorted in decreasing order and are $< 1$.

**Greedy General Bin Packing Algorithm**

**Input:** $w = (w_1, \ldots, w_n)$ where $1 > w_1 \geq w_2 \geq \cdots \geq w_n$.

**Output:** Bins $B_1, \ldots, B_m$ containing the weights in $w$.
1. Let the bins to be filled be $B_1, B_2, B_3, \ldots$, initially empty.
2. For $i = 1, \ldots, n$,
   1. place $w_i$ into the first bin $B_j$ that it fits into.
3. Output all the non-empty bins, say $(B_1, B_2, \ldots, B_m)$.
7. (2 Point, Coin Change Problem) with parameters

(a) Give a simple \(O(n^2)\) implementation of this algorithm. We want you to write this in very DETAILED pseudo code, rather close to actual code in Java or C++, say.

(b) Assume that this algorithm uses \(m > \text{Opt}(w)\) many bins. There is a smallest index \(i (2 \leq i \leq n)\) such that \(w_i\) is placed into bin \(B_j\) where \(j = \text{Opt}(w) + 1\). Let \(i_0\) denote this index. Show that \(w_{i_0} \leq 1/3\).

(c) Show that \(n - i_0 \leq \text{Opt}(w)\) where \(i_0\) is the critical index of part(b).

(d) Conclude that \(m \leq 1.5 \cdot \text{Opt}(w)\).

7. (2 Point, Coin Change Problem)

Fix a currency system \(D = (1, d_2, \ldots, d_m)\). Recall the basic definitions for coin changing problems found in p.7 (Lecture V). For any \(x\) let \(G(x) = (s_1, \ldots, s_m)\) be the greedy solution and \(\text{Opt}(x) = (s'_1, \ldots, s'_m)\) is the optimum solution. Suppose \(x\) is the smallest counter example to the canonicity of \(D\). Show that

\[s_i \cdot s'_i = 0 \quad \text{(for all } i = 1, \ldots, m)\]  

SOLUTION

Suppose not. Say \(s_i \cdot s'_i > 0\). Then we can replace \(x\) by \(x' = x - d_i\) to get a strictly smaller counter example.

Why? Clearly, \(G(x')\) is obtained from \(G(x)\) by subtracting 1 from the \(i\)-th component. Also, \(|\text{Opt}(x')| \leq |\text{Opt}(x)| - 1\). We know that \(|\text{Opt}(x)| < |G(x)|\). Thus \(|\text{Opt}(x')| \leq |\text{Opt}(x)| - 1 < |G(x)| - 1 = |G(x')|\). This shows that \(x'\) is also a counter example.

Comments:

8. (5+0+2 Points, Coin Changing Problem)

How do you prove that the US currency system is canonical? We will provide an inductive approach here. REMARK: Part(b) seems hard, and is considered extra credit work. Do NOT hand it in unless you have a good solution.

Suppose \(D = (1, d_2, \ldots, d_m)\) is a canonical currency system \((m \geq 1)\). We look at extensions of \(D\):

(a) If \(D' = (1, d_2, \ldots, d_m, d_{m+1})\) extends \(D\) with a denomination \(d_{m+1} := qd_m\) for some \(q \geq 2\), then \(D'\) is called a Type A extension of \(D\). Show that Type A extensions of \(D\) are canonical.

(b) Assume \(D'\) is a Type A extension of \(D\) as in part(a). Let \(D'' = (1, d_2, \ldots, d_m, d_{m+1}, d_{m+2})\) extend \(D'\) with a denomination \(d_{m+2} = ad_{m+1} + bd_m\) where \(a, b\) are non-negative integers. Call \(D''\) a Type B extension of \(D\) with parameters \(a, b, q\).

Prove that the Type B extension \(D''\) of a canonical system \(D\) is also canonical, provided if \(q \leq 3\) and \(a \geq 2\).

Example: let \(D = (1, 5)\), which is canonical. Then \(D' = (1, 5, 10)\) is a Type A extension of \(D\). Then \(D'' = (1, 5, 10, 25)\) is a Type B extension of \(D'\).

(c) Conclude that the US currency system comprising the notes $100, $50, $20, $10, $5, $1 and coins 25¢, 10¢, 5¢, 1¢ is canonical.

SOLUTION

(a) Suppose \(x\) is the minimum counter example for \(D'\). Write the greedy \(D'\)-solution for \(x\) as \(G_{D'}(x) = (s'_1, \ldots, s'_m, s'_{m+1})\). Clearly, \(s'_{m+1} \geq 1\). If \(\text{Opt}_{D'}(x) = (s'_1, \ldots, s'_m, s'_{m+1})\) then by the property in the previous problem, we know that \(s'_{m+1} = 0\). Thus, we can view \(\text{Opt}_{D'}(x)\) as a \(D\)-solution for \(x\). Since \(D\) is canonical, this proves that \(|\text{Opt}_{D'}(x)| \geq |G_{D'}(x)|\). Thus,

\[
|G_{D'}(x)| > |\text{Opt}_{D'}(x)| \\
\geq |G_D(x)| \\
= \left| (s'_1, \ldots, s'_{m-1}, s'_m, s'_{m+1}) \right| + (q-1)s'_{m+1} \\
= |G_{D'}(x)| + (q-1)s'_{m+1} \quad \text{as noted above} \\
> |G_D(x)| + (q-1)s'_{m+1}
\]

which is a contradiction.
(cont'd)
(b) We can repeat the first part of the argument in part (a), using \( x \) as a minimum counter example. Again, we can regard \( \text{Opt}_{D'}(x) \) as a \( D' \)-solution for \( x \), and hence \(|\text{Opt}_{D'}(x)| \geq |G_{D'}(x)|\) by the canonicity of \( D' \) (by part(a)). Moreover, we know the form of \( G_{D'}(x) \) which is closely related to \( G_{D'}(x) \) because both are greedy solutions. If \( G_{D'}(x) = (s''_1, \ldots, s''_{m+1}, s''_{m+2}) \) and \( G_{D'}(x) = (s'_1, \ldots, s'_{m+1}) \). From the fact that
\[
d_{m+2} = ad_{m+1} + bd_m = (aq + b)d_m
\]
we may assume that \( b \leq q - 1 \) and also conclude:
\[
s'_i = \begin{cases} s'_i & \text{if } i \leq m - 1 \\ (s'_m + bs'_{m+2}) \mod q & \text{if } i = m \\ (s'_{m+1} + as'_{m+2}) + [(s''_m + bs''_{m+2})/q] & \text{if } i = m + 1 \end{cases}
\]
(6)

Suppose we can show that
\[
s'_m + s'_{m+1} \geq s''_m + s''_{m+1} + s''_{m+2}.
\]
(7)

Then we would obtain a similar contradiction as before:

\[
\begin{aligned}
|G_{D'}(x)| &> |\text{Opt}_{D'}(x)| \\
&= |G_{D'}(x)| & x \text{ is a counter example} \\
&= |(s'_1, \ldots, s'_{m-1}, s'_m)| & D' \text{ is canonical by part(a), so } G_{D'}(x) \text{ is optimal } D'-solution \\
&\geq |(s''_1, \ldots, s''_{m+1}, s''_{m+2})| & \text{definition of } G_{D'}(x) \\
&= |G_{D'}(x)| & \text{contradiction}.
\end{aligned}
\]

In fact, our conditions \( q \leq 3 \) and \( a \geq 2 \) is designed just to ensure (7). To see why, observe that \( q \leq 3 \) ensures that \( s''_m \leq q - 1 \leq 2 \) (a property of greedy solutions). To simplify notations, let us rewrite \((s_m, s_{m+1}, s_{m+2})\) as \((u, v, w)\). Also, \( u', u'' \) stands for \( s'_m, s''_m \), etc. So (7) becomes
\[
u' + v' \geq u'' + v'' + w''.
\]

But from (6), we have \( u' = (u'' + bw'') \mod q \) and \( v' = [(u'' + bw'')/q] + v'' + aw'' \). Therefore, (7) becomes
\[
\begin{aligned}
(u'' + bw'') \mod q + [(u'' + bw'')/q] + v'' + aw'' &\geq u'' + v'' + w'' \\
(u'' + bw'') \mod q + [(u'' + bw'')/q] + (a - 1)w'' &\geq u''
\end{aligned}
\]
(8)

As noted above, \( u'' \leq q - 1 \). But under our assumptions, \( q = 2 \) or \( q = 3 \). Therefore, \( u'' \leq 2 \). If \( u'' \leq 1 \), then the inequality (8) is clearly true since we know \( u'' \geq 1 \) and \( a \geq 2 \). So we henceforth assume \( u'' = 2 \) (and so \( q = 3 \)). If \( u'' \geq 2 \), again the inequality holds. If \( u'' = 1 \), then (8) reduces to
\[
(2 + b) \mod 3 + [(2 + b)/3] + (a - 1) \geq 2.
\]

As noted above, we may assume \( b \leq q - 1 \). We may check that this inequality holds for \( b = 0, 1, 2 \). This concludes the proof.

(c) Note that the US currency system is obtained by a sequence of Type A or Type B extensions, starting from \( D = (1) \). All but two of these extensions are Type A. The exceptions are (i) when we add 25¢ and (ii) when we add $50. If we include the two dollar bill, then the five dollar bill represents yet another Type B extension. Remarkably, all these Type B extensions have the same parameters of \( q = 2, a = 2 \) and \( b = 1 \).
9. (8 Points, Sorted linear bin packing)
(a) Show a currency system that is complete and canonical but does not have uniqueness. HINT: you need not consider more than 3 denominations.
(b) Show that the binary system \( D = (1, 2, 4, \ldots, 2^n) \) is a canonical system that is also unique.

**SOLUTION**
(a) Consider \( D'' = (1, 2, 3) \). This is canonical because \( D = (1) \) is a canonical system, and \( D'' \) is a Type B extension of \( (1) \) with parameters \((q, a, b) = (2, 1, 1)\). See the above question for terminology of Type B extension. Although \( a = 1 \), the same proof in the above exercise can be adapted to show that if \( q = 2 \) and \( a = 1 \), then \( D'' \) is canonical. Basically, we have to show that \( u' + v' \geq u'' + v'' + w'' \) (using notations in that proof). But we also have non-uniqueness: \( G_D(4) = (1, 0, 1) \) but another optimal solution for 4 is \((0, 2, 0)\).
(b) Clearly, \( D = (1, 2, 4, \ldots, 2^{m-1}, 2^m) \) is a Type A extension of \( (1, 2, 4, \ldots, 2^{m-1}) \). So \( D \) is canonical by induction. Uniqueness follows from the fact that natural numbers has a unique binary representation.

**Comments:**

10. (3+4 Points)
We gave four different greedy criteria for the activities selection problem. Be sure to make your examples as small and simple as possible (we will take off points for unnecessary complications).
(a) Show that the other three criteria are suboptimal.
(b) Actually, each of the four criteria has an inverted version in which we sort in decreasing order (break ties arbitrarily). Prove that one of these is optimal, and give counter examples to the other three.

We gave four different greedy criteria for the activities selection problem.
(a) We know one of them is optimal, but do not if sorting in increasing order degree-of-conflict is optimal. Show that the two of the other three criteria are suboptimal. EXTRA CREDIT: provide a single set of activities that serves as strong counter example to each of three criteria. It is possible to find a single counter example with 9 activities.
(b) Naturally, each of the above four criteria has an inverted version in which we sort in decreasing order. Again, one of these inverted criteria is actually optimal, and the other three suboptimal. Prove the optimality of one, and provide counter examples for the other three.

![Figure 1: Counter Example for Conflict Degree](image)
(a-i) Increasing finish times. This was proved optimal.
(a-ii) Increasing start times. The ordering of activities in our original example (beach, swim, tennis, movie1, movie2) provides counter example, since we would pick the suboptimal \{beach, movie2\}.
(a-iii) Increasing duration. Suppose in the original example, we make tennis last from 2 : 20 − − 3 : 20, then we would pick the suboptimal \{tennis, movie\}.
(a-iv) Increasing conflict degree. A weak counter example is shown in Figure 1(I): using our criterion, we would pick the middle interval, and then we can pick at most two more (e.g., the green intervals). But the optimal solution is the four red intervals. Note that this is a weak counter example. But if you duplicate two of the blue intervals, you can get a strong counter example. It turns out that there is a single instance that provide counter examples to (a-ii, a-iii, a-iv). Thanks to Jinil Jang for this example.
Consider the 9 intervals (A,B,C,D,E,F,G,H,I) in Figure 1(II): the optimal solution has 4 intervals (A,B,C,D). But using any of the other three criteria, you only get three intervals. (a-ii) If you sort by increasing start times, you will pick E. This allows you to choose at most two more intervals.
(a-iii) If you sort by increasing duration times, you will choose F. That implies you can choose at most two more intervals on either side of F. (a-iv) Using the degree of conflict criterion, you will again choose interval F.
(b) The optimal criteria when you sort the start times in decreasing order. Proving that this is optimal can follow the proof for the optimality of (a-i) in the text. Suppose the greedy solution using decreasing start times finds \(k\) compatible activities, and these are their start times:

\[ s_1 \geq s_2 \geq \cdots \geq s_k \]

Suppose the optimal solution found \(\ell\) compatible activities with start times

\[ s'_1 \geq s'_2 \geq \cdots \geq s'_\ell. \]

It is easy to prove by induction that \(s_i \geq s'_i\) for each \(i = 1, \ldots, \ell\). Now suppose the greedy method is suboptimal. Then \(k < \ell\). Then look at \(s'_{k+1}\). We have \(s_k \geq s'_{k} \geq s'_{k+1}\). Clearly, the activity represented by \(s'_{k+1}\) is compatible with the greedy solution. That means the greedy algorithm would have chosen this interval as well, contradiction.
The other three are suboptimal, but counter examples here are very easy to come by, so we omit it.

Comments: