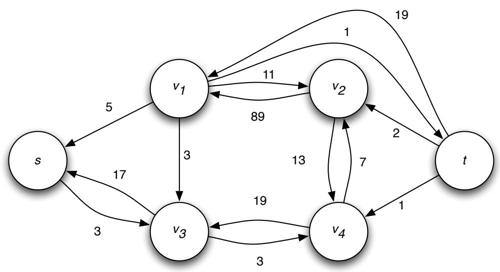
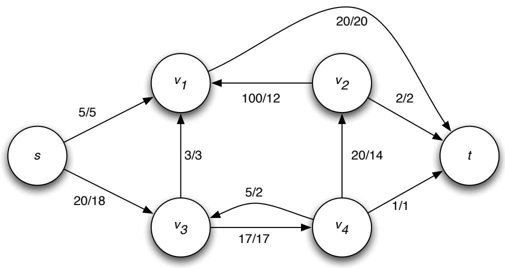
CMPSCI 611 FALL '09: MIDTERM SOLUTIONS

Answer 1. (1) 22 (2) 37 (3) The residual graph is:



(4) Nodes in p are s, v_3, v_4, v_2, v_1, t and new flow is:



(5) Max-Flow Min-Cut Theorem says that for any flow network and flow f, f is a maximum flow iff there exists an $s-t$ cut (A, B) with $|f| = C(A, B)$. Since the new flow has size 23 and the cut $(\{s, v_1, v_2, v_3, v_4\}, \{t\})$ has capacity 23, the new flow is a maximum flow.

Answer 2. (1) FALSE (2) FALSE (3) FALSE (4) TRUE

(1) By the cardinality theorem, it is sufficient to show that for all $A \subseteq E$, if $i, i' \in \mathcal{I}$ are maximal subsets of A then $|i| = |i'|$. For a given A, let k be the number of connected components of $G = (V, A)$. But any maximal subset of A has exactly $|V| - k$ edges.

(2) Let $\mathcal{I} = \{E' \subset E : (V, E') \text{ is an acyclic graph}\}$. We know that $(E, \mathcal{I})$ is a matroid from the last part of the question. Hence the greedy algorithm finds $i \in \mathcal{I}$ of maximum weight with respect to any positive weight function. Note that a maximum weight i is a spanning tree because i is acyclic and G is connected. Consider a new weight function $w'_e = \ln w_e$ and note that $w'_e > 0$ because $w_e > 1$ and that for any $i \in \mathcal{I}$, $w'(i) = \sum_{e \in i} w'_e = \ln(\prod_{e \in i} w_e)$. Running a greedy algorithm with weights w'_e returns an $i \in \mathcal{I}$ with $\ln(\prod_{e \in i} w_e)$ maximized. Since ln is a strictly increasing function, this i also maximizes $\prod_{e \in i} w_e$.

Answer 3. (1) 10 (2) $[0, 6, \infty, 4, 11, 9]$

(3) Run Dijkstra on G to find $\delta_G(v^*, v)$ for all $v \in V$. Reverse the orientation of every edge to create graph $G' = (V, E')$ and run Dijkstra to find $\delta_{G'}(v^*, u)$ for all $u \in V$. Note that $\delta_{G'}(v^*, u) = \delta_G(u, v^*)$. Since the shortest path from u to v in G via v^* consists of the shortest path from u to v^* followed by the shortest path from v^* to v we return $\delta_G(u, v^*) + \delta_G(v^*, v)$ as the length of the shortest path from u to v in G via v^*. Running time is $2D(n, m) + O(n^2) + O(m) = O(n^2)$.

Answer 4. (1) 2,10000,1,1

(2) For $1 \leq i < j \leq n$ and $j-i+1$ even define $S_{i,j}$ to be the maximum value that Alice can guarantee if it's her turn to pick and the remaining sequence is $s_i, \ldots, s_j$. Before the game starts Alice precomputes the $O(n^2)$ values of $S_{i,j}$ as follows. First compute $S_{i,i+1} = \max\{v_i, v_{i+1}\}$ for $1 \leq i \leq n-1$. Then, for $k = 4, 6, \ldots n$ and $1 \leq i \leq n+1-k$, compute

$$S_{i,i+k-1} = \max(v_i + \min(S_{i+2,i+k-1}, S_{i+1,i+k-2}), v_{i+k-1} + \min(S_{i+1,i+k-2}, S_{i,i+k-3})) \ .$$

The above equation follows because Alice's guaranteed score is the value she picks (either v_i or v_{i+k-1}) plus the value she can guarantee subsequently no matter what card Bob picks next. Note that this computation takes $O(n^2)$ time. While playing, when Alice has to pick from the cards $s_i, \ldots, s_j$, she picks s_i if

$$v_i + \min(S_{i+2,j}, S_{i+1,j-1}) \geq v_j + \min(S_{i+1,j-1}, S_{i,j-2})$$

and s_j otherwise. This decision takes $O(1)$ time given the precomputed information.

Answer 5. (1)

$$H_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \text{ and } H_2 v = \begin{bmatrix} 3 \\ -1 \\ -1 \\ 3 \end{bmatrix}$$

(2) For any length $n = 2^k$ vector $v = [v_1, \ldots, v_n]^T$, let $t(v) = [v_1, \ldots, v_{n/2}]^T$ and $b(v) = [v_{n/2+1}, \ldots, v_n]^T$ and note that

$$H_k v = \begin{pmatrix} H_{k-1}t(v) + H_{k-1}b(v) \\ H_{k-1}t(v) - H_{k-1}b(v) \end{pmatrix}$$

Hence the following divide and conquer algorithm can be used to compute $H_k v$:

```
If k=0: Return v
Compute t=H_{k-1} t(v)
Compute b=H_{k-1} b(v)
Return the vector w=[t+b, t-b]^T
```

The running time satisfies $T(n) = 2T(n/2) + O(n)$ and hence $T(n) = O(n \log n)$.

Answer 6. Let A_i and B_i denote the i-th entries of A and B. Let $A_{i,j} = A_i A_{i+1} \ldots A_j$ and $B_{i,j} = B_i B_{i+1} \ldots B_j$. Consider the following algorithm:

MEDIAN(n, A, B):
(1) If $n = 1$: return $\min(A_1, B_1)$
(2) If $A_{n/2} < B_{n/2}$: return MEDIAN$(n/2, A_{n/2+1,n}, B_{1,n/2})$
(3) If $A_{n/2} > B_{n/2}$: return MEDIAN$(n/2, A_{1,n/2}, B_{n/2+1,n})$

Correctness: Clearly if $n = 1$ then the smallest element from $A_1 \cup B_1$ is $\min(A_1, B_1)$. Otherwise if $A_{n/2} < B_{n/2}$ then we know that each element in $A_{1,n/2}$ is smaller than the n-th element of $A \cup B$ and that each element in $B_{n/2+1,n}$ is larger than the n-th element of $A \cup B$. Hence the $n/2$-th smallest element of $A_{n/2+1,n} \cup B_{1,n/2}$ is the n-th smallest element of $A \cup B$. Similarly, if $A_{n/2} > B_{n/2}$ then the $n/2$-th smallest element of $A_{1,n/2} \cup B_{n/2+1,n}$ is the n-th smallest element of $A \cup B$.

Running Time: At each recursion, the total number of elements being considered decreases by a factor of 2. Hence, the depth of the recursion is only $\log_2 n$ and at each stage the running time is $O(1)$. Hence, total running time is $O(\log n)$.