CS584
HW#4

Due: November 5

1. (12 points) Schedule a processor for a time between t=0 and $t=\delta$ to satisfy requests $\{(\sigma_1,\tau_1),...,(\sigma_n,\tau_n)\}$, where each request satisfies $0 \le \sigma_i \le \tau_i \le \delta$, $\tau_i \in \mathbb{Z}^+$, for $1 \le i \le n$, and each request (σ_i,τ_i) has a *value* $v_i \in \mathbb{R}^+$. Requests may not be pre-empted and may not overlap in time. The *value* of a schedule is the sum of the values of the requests that are granted. We want to find a schedule of maximal value.

a Show how to compute the value of an optimal schedule in time in $O(n \lg n)$.

b Assume that an oracle will tell the value of an optimal schedule, and the oracle will reply in time in $O(1)$. Show how to construct an optimal schedule given access to such an oracle.

2. (16 points) Suppose a digraph $G=(V,E)$ is represented as an adjacency matrix A, where

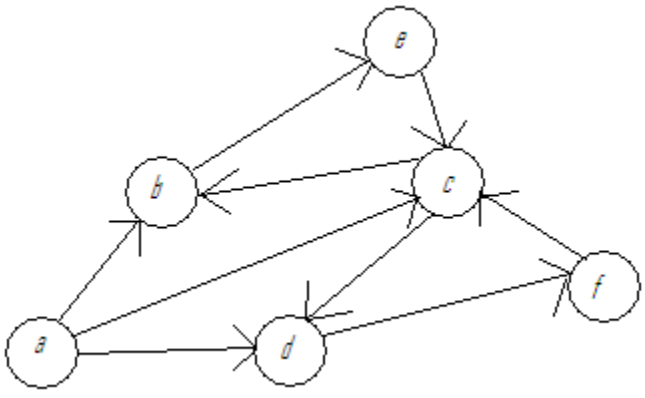
$$A[i,j]=\begin{cases}1, \text{if edge } v_i v_j \in E\\ 0, \text{if edge } v_i v_j \notin E\end{cases}$$

a Describe an algorithm to accept as input

- array A,
- $l \in \mathbb{N}$ (note that $0 \in \mathbb{N}$),
- $v_i, v_j \in V$

and your algorithm should return the number of distinct directed paths from v_i to v_j of length exactly l. Your algorithm should work in time in $O(ln^3)$.

When your algorithm is invoked with l=0 and $v_i = v_j$ it should return 1. When invoked with l=0 and $v_i \neq v_j$ it should return 0. When invoked for digraph



with $l=3$, $v_i = d$ and $v_j = b$ it should return 1 because of the path d, f, c, b. And finally, when invoked with $l=4$, $v_i = a$ and $v_j = c$ it should return 2 because of the paths a,c,d,f,c and a,c,b,e,c.

b Do the same thing as in part ***a*** exact that your algorithm should return the number of distinct directed paths from v_i to v_j of length at most l. When invoked with $l = 4$, $v_i = a$ and $v_j = c$ it should return 5 because of the paths

a,c
a, b, d, c
a, d, f, c
a,c,d, f,c
a,c,b,e,c

3. (10 points) Let a country's currency be coins worth c_1¢, c_2¢,…, c_n¢. We seek an algorithm which accepts as input $(c_1,...,c_n;x)$ and which gives as output a **minimal** number of coins, drawn from $(c_1,...,c_n)$, such that the sum of the values of the coins is x¢. So, for example, for $(1,5,10,25,50;156)$ the answer would be $(50,50,50,5,1)$.

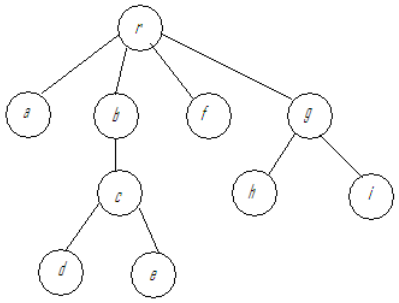
a One algorithm is

GREED$(c_1,...,c_n,x)$
if x>0 **then** {let c_i be max$(c_1,...,c_n)$ such that $c_i \le x$
give c_i
GREED$(c_1,...,c_n,x-c_i)$

GREED works for $(1,5,10,25,50;x)$ for any x. Give an instance of the problem, $(c_1,...,c_n;x)$, for which GREED does not work.

b Give an algorithm which works for any $(c_1,...,c_n;x)$. The time complexity of your algorithm should be in $O(nx)$.

4. (8 points) An *independent set* of vertices of a graph $G=(V,E)$ is a set of vertices such that there does not exist an edge between any pair of vertices of the set. As stated in **Problem *34-1*** on page 1018 of our text, finding a maximum independent set of a graph is very difficult. However, many problems which are difficult in graphs become easier if we restrict the graph to be a tree. Describe an algorithm, with time complexity in $O(m+n)$, to find a maximum independent set in a tree. So MAXINDEPENDENTSET(r) would return $\{a,b,d,e, f,h,i\}$ on



CS584
HW#4 SOLUTIONS

1. ***a*** We start by HEAPSORTing $\{(\sigma_1,\tau_1),...,(\sigma_n,\tau_n)\}$ on $\tau_1,...,\tau_n$. Without loss of generality, we assume that $\tau_1 \le \tau_2 \le ... \le \tau_n$ We compute the statistic $f[j]$, which is the value of an optimal schedule for $\{(\sigma_1,\tau_1),...,(\sigma_j,\tau_j)\}$. For any interval (σ_j,τ_j) we note that either it is **not included** in an optimal schedule, in which case $f[j] = f[j-1]$, or it **is included** in the optimal schedule, in which case $f[j] = \max_{\substack{0 \le i < j \\ \tau_i \le \sigma_j}} (f[i] + v_j)$.

$f[0] \leftarrow 0$
$\tau_0 \leftarrow 0$
for $j \leftarrow 1$ **to** n **do**
    BINARYSEARCH to find $\max_{0 \le i < j} (\tau_i \le \sigma_j)$
    $f[j] \leftarrow \max(f[j-1], f[i] + v_j)$
return $f[n]$

b We use the oracle to return values $f[j]$. The following program will **print** the intervals of an optimal schedule.

RETURNSCEDULE(*f*, *n*)
    if $n \ge 1$ **then if** $f[n] = f[n-1] + v_n$ **then print** (σ_n,τ_n)
    RETURNSCEDULE(*f*, *n*-1)

2. ***a*** We compute $n \times n$ arrays Num^{λ} such that $Num^{\lambda}[i, j]$ is the number of paths from v_i to v_j of length exactly λ. The idea is to have v_k iterate over all possible immediate predecessors of v_j on the path.

for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** n **do**
        if $i=j$ **then** $Num^0[i, j] \leftarrow 1$ **else** $Num^0[i, j] \leftarrow 0$
for $\lambda \leftarrow 1$ **to** l **do**
    for $i \leftarrow 1$ **to** n **do**
        for $j \leftarrow 1$ **to** n **do**
            $Num^{\lambda}[i, j] \leftarrow 0$
            for $k \leftarrow 1$ **to** n **do**
                if $A[i, j] = 1$ **then** $Num^{\lambda}[i, j] \leftarrow Num^{\lambda}[i, j] + Num^{\lambda-1}[i, k]$
return $Num^{l}[i, j]$

b **for** $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** n **do**
        if $i=j$ **then** $Num^0[i, j] \leftarrow 1$ **else** $Num^0[i, j] \leftarrow 0$
for $\lambda \leftarrow 1$ **to** l **do**
    for $i \leftarrow 1$ **to** n **do**
        for $j \leftarrow 1$ **to** n **do**
            $Num^{\lambda}[i, j] \leftarrow Num^{\lambda-1}[i, j]$
            for $k \leftarrow 1$ **to** n **do**
                if $A[i, j]=1$ **then** $Num^{\lambda}[i, j] \leftarrow Num^{\lambda}[i, j]+Num^{\lambda-1}[i, k]$
return $Num^{l}[i, j]$

3. ***a*** For $(1,4,6;8)$ GREED will return $(6,1,1)$ although the answer is $(4,4)$.

b For $0 \le m \le x$ we let $\kappa(m)$ denote the minimum number of coins to give m¢. If this minimum number of coins includes a c_i¢ coin, then, by the Optimality Principle, we use $\kappa(m-c_i)$ coins to give (m-c_i)¢.

$$\kappa(m)=1+\min_{1\le i\le n}\{\kappa(m-c_i)\}$$

In the dynamic programming formulation, it is understood that $\kappa(m)=0$ if $m \le 0$.

for $m \leftarrow 1$ **to** x **do**
    $\kappa[m] \leftarrow \infty$
    for $i \leftarrow 1$ **to** n **do**
        if $m>c_i$ **then** $\kappa(m)=\min(\kappa(m),1+\kappa(m-c_i))$
PRINTANSWER(κ, x)

PRINTANSWER(κ, m)
    if $m>0$ **then**
    $i \leftarrow 1$
    repeat
        if $m>c_i \wedge \kappa[m]=1+\kappa[m-c_i]$
        then {Print c_i
            return PRINTANSWER$(\kappa, m-c_i)$ }
        else $i \leftarrow i+1$

4. For each node *v* in the tree, we compute $\iota(v)$, the maximum number of independent nodes in the tree rooted at *v*. We compute $\iota(v)$ from the bottom up in the tree, so that when computing $\iota(v)$ we have already computed $\iota(w)$ for all children and grandchildren *w* of *v*. We note that if *v* is in a maximum independent set of the tree rooted at *v*, then none of its children is in the maximum independent set. The dynamic programming recurrence is

$$\iota(v) = \max\left\{ \sum_{\text{children } w \text{ of } v} \iota(w),\ 1 + \sum_{\text{grandchildren } w \text{ of } v} \iota(w) \right\}$$