CS 170 Second Midterm 5 April 2007

NAME (1 pt): _____

TA (1 pt): _____

Name of Neighbor to your left (1 pt): _____

Name of Neighbor to your right (1 pt): _____

Instructions: This is a closed book, closed calculator, closed computer, closed network, open brain exam, but you are permited a 1 page, double-sided set of notes, large enough to read without a magnifying glass.

You get one point each for filling in the 4 lines at the top of this page. Each other question is worth the amount shown.

Write all your answers on this exam. If you need scratch paper, ask for it, write your name on each sheet, and attach it when you turn it in (we have a stapler).

1	
2	
3	
Total	

2

(1) Scheduling (30 points).

- You're an engineer planning the construction of a large bridge. There are constraints between the tasks involved. For instance, a portion of road cannot be attached to the suspensions until these are in place, or until the road itself is built. Let each task be represented as a node in a graph, where a directed edge joins A to B if task A must be completed before B begins.

  Give a condition on this graph for you to be able to order the tasks so as to satisfy all the constraints.

  Answer. The graph must be a DAG.

  Give a method to check this condition.

  Answer. Run Depth First Search. The graph is a DAG if and only if no back-edge is found.

  Give a method to find an ordering of the tasks that satisfies the constraints if one exists.

  Answer. Run DFS and order the nodes by decreasing post *number.*

- In fact in a real schedule, tasks can happen simultaneously, unless a constraint forces us to finish one before beginning the other.

  To represent task duration, let the length of edge A→B be the duration of A. (In other words, every outgoing edge from A must have the same length.) Let S ("Start") and F ("Finish") be special tasks of length 0, that must happen respectively before and after all other tasks. For instance they can represent the contract signature and the inauguration.

  An important concept in scheduling is the *critical path*, that is a sequence of tasks $S, A_1, \ldots A_k, F$ such that $A_i \to A_{i+1}$ and the length of the path is equal to that of the shortest possible schedule. We call this length the *construction time*.

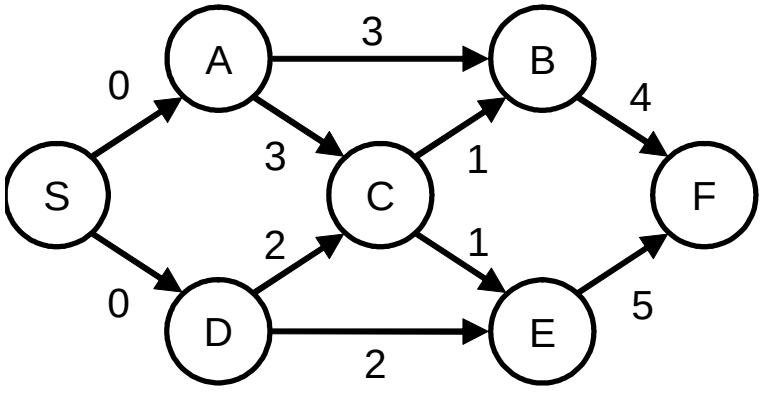

Figure 1: A list of tasks (nodes), constraints (edges) and task durations (edge lengths).

In Fig. 1, find the critical path and the construction time.

Answer. The critical path is the longest path from S to F. Here it is $S \to A \to C \to E \to F$, of length 9.

Give a method to find the critical path automatically on a given graph (hint: this method uses the property of this graph that you found in the first part of this problem).

Answer. The best method is to adapt the "dags-shortest-path" method to find the longest path instead. Find a linearization order as in the third question. Iterate through each node u in linearized order, calling update() on each edge from u, except update is now:
update(u,v): dist(v) = max(dist(v),dist(u)+l(u,v))
and the array dist() is initialized to $-\infty$*, or to* 0 *since all durations are positive. We can call this method "dags-longest-path".*

- What we really want is not just the construction time, but an entire schedule, specifying for each task when to start it. **How can you use the intermediate results of your algorithm to output a start time for each task (each node)? Show that this schedule is indeed valid, i.e. it does not violate any constraint.**

  Answer. The array element dist(u) *is the length of the longest path from S to u. We can use it as start time. To prove that this gives us a valid schedule, look at a constraint* $u \rightarrow v$*. The start time of u is* dist(u)*, and the update equation gives us* dist$(v) \geq$dist$(u)+l(u,v)$ *so the start time of v is after the end time of u.*

- If each task requires a team of workers, **show how to compute the number of teams we need to hire, i.e. the maximum number of tasks that will be executed in parallel**. For example, if all tasks take unit time and $A \rightarrow B$, $A \rightarrow C$, $B \rightarrow D$, $C \rightarrow D$, then answer is 2 teams, because B and C can be done in parallel.

  Answer. Now that we have a start_time and end_time for each task where start_time is the longest path to the vextex, and end_time = start_time + task_duration (lengths of outgoing edges), we can form records of the form (start_time,+1), $(end_time, -1)$ *and sort all the records by their first entry yielding the list* $(t_1, s_1), (t_2, s_2), ...(t_n, s_n)$ *where* $t_1 \leq t_2 \leq ... \leq t_n$ *and each* s_i *is +1 or -1. If there are ties (multiple equal* t_i*), then put all the records with* $s_i = -1$ *before the records with* $s_i = +1$*. Now do*

```
parallel_tasks = 0;
max_parallel_tasks = 0;
for i = 1 to n
    parallel_tasks = parallel_tasks + s_i
        // increases by 1 when a new task starts
        // decreases by 1 when one ends
    max_parallel_tasks = max(max_parallel_tasks, parallel_tasks)
end
```

(1) Scheduling (30 points).

- Planning the construction of a large building is a challenging engineering task, including dealing with constraints among the tasks involved. For instance, a portion of roof cannot be attached to the building until supports are in place, or until the roof itself is available. Let each task be represented as a node in a graph, where a directed edge joins A to B if task A must be completed before B begins.

  Give a condition on this graph for you to be able to order the tasks so as to satisfy all the constraints.

  Answer. The graph must be a DAG.

  Give a method to check this condition.

  Answer. Run Depth First Search. The graph is a DAG if and only if no back-edge is found.

  Give a method to find an ordering of the tasks that satisfies the constraints if one exists.

  Answer. Run DFS and order the nodes by decreasing post *number.*

- In fact in a real schedule, tasks can happen simultaneously, unless a constraint forces us to finish one before beginning the other.

  To represent task duration, let the length of edge A→B be the duration of A. (In other words, every outgoing edge from A must have the same length.) Let S ("Start") and F ("Finish") be special tasks of length 0, that must happen respectively before and after all other tasks. For instance they can represent the contract signature and the inauguration.

  An important concept in scheduling is the *critical path*, that is a sequence of tasks $S, A_1, \dots A_k, F$ such that $A_i \to A_{i+1}$ and the length of the path is equal to that of the shortest possible schedule. We call this length the *construction time*.

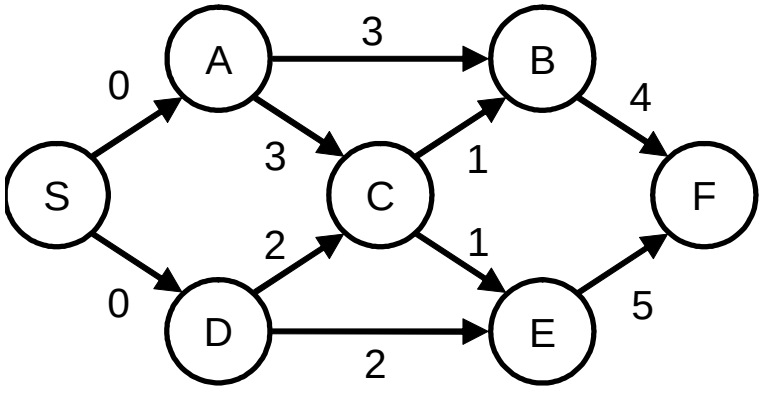

Figure 2: A list of tasks (nodes), constraints (edges) and task durations (edge lengths).

In Fig. 2, find the critical path and the construction time.

Answer. The critical path is the longest path from S to F. Here it is $S \to A \to C \to E \to F$, of length 9.

Give a method to find the critical path automatically on a given graph (hint: this method uses the property of this graph that you found in the first part of this problem).

Answer. The best method is to adapt the "dags-shortest-path" method to find the longest path instead. Find a linearization order as in the third question. Iterate through each node u in linearized order, calling update() on each edge from u, except update is now:
update(u,v): dist(v) = max(dist(v),dist(u)+l(u,v))
and the array dist() is initialized to $-\infty$*, or to* 0 *since all durations are positive. We can call this method "dags-longest-path".*

- What we really want is not just the construction time, but an entire schedule, specifying for each task when to start it. **How can you use the intermediate results of your algorithm to output a start time for each task (each node)? Show that this schedule is indeed valid, i.e. it does not violate any constraint.**

  Answer. The array element dist(u) *is the length of the longest path from S to u. We can use it as start time. To prove that this gives us a valid schedule, look at a constraint* $u \to v$*. The start time of u is* dist(u)*, and the update equation gives us* dist$(v) \geq$dist$(u)+l(u,v)$ *so the start time of v is after the end time of u.*

- If each task requires a team of workers, **show how to compute the number of teams we need to hire, i.e. the maximum number of tasks that will be executed in parallel**. For example, if all tasks take unit time and $A \to B$, $A \to C$, $B \to D$, $C \to D$, then answer is 2 teams, because B and C can be done in parallel.

  Answer. Now that we have a start_time and end_time for each task where start_time is the longest path to the vextex, and end_time = start_time + task_duration (lengths of outgoing edges), we can form records of the form (start_time,+1), $(end_time, -1)$ *and sort all the records by their first entry yielding the list* $(t_1, s_1), (t_2, s_2), ...(t_n, s_n)$ *where* $t_1 \leq t_2 \leq ... \leq t_n$ *and each* s_i *is +1 or -1. If there are ties (multiple equal* t_i*) then put all the records with* $s_i = -1$ *before those with* $s_i = +1$*. Now do*

```
parallel_tasks = 0;
max_parallel_tasks = 0;
for i = 1 to n
    parallel_tasks = parallel_tasks + s_i
        // increases by 1 when a new task starts
        // decreases by 1 when one ends
    max_parallel_tasks = max(max_parallel_tasks, parallel_tasks)
end
```

(2) (20 points) Let F be a file containing symbols a_1,...,a_n, where a_i appears f_i times. Suppose $f_1 = f_2 = 1$ and $f_i = f_{i-1} + f_{i-2}$ for $i > 2$ (a Fibonacci sequence). We want to use Huffman encoding to compress F. Determine an optimal encoding of each symbol a_i.

Answer. The algorithm for Huffman encoding creates a priority queue of nodes (representing sets of symbols) ordered by increasing frequency of appearance (the sum of all the frequencies of symbols in the set). Initially the priority queue contains (a_1, f_1), ... , (a_n, f_n). *We claim that at step* i *of the algorithm the two lowest frequency sets removed from the priority queue are* (a_{i+1}, f_{i+1}) *and* $(\{a_1, ..., a_i\}, f_1 + \cdots + f_i)$, *where* $f_1 + \cdots + f_i = f_{i+2} - 1 \geq f_{i+1}$. *We prove this by induction. The base case is* $i = 1$, *and this is clearly true. Now suppose it is true for* i, *we must show it true for* $i + 1$. *Then at step* i, *we remove these two sets, merge them to form* $\{a_1, ..., a_{i+1}\}$ *with frequency* $f_{i+1} + f_{i+2} - 1 = f_{i+3} - 1$, *where we have used the definition of the Fibonacci sequence. The two other lowest frequency entries in the queue are* (a_{i+2}, f_{i+2}) *and* (a_{i+3}, f_{i+3}). *Since* $f_{i+2} \leq f_{i+3} - 1 < f_{i+3}$, *the two lowest frequency items on the queue at step* $i + 1$ *are* (a_{i+2}, f_{i+2}) *and* $(\{a_1, ..., a_{i+1}\}, f_{i+3} - 1)$, *as claimed.*

The resulting tree created by the Huffman encoding algorithm is therefore a chain, with a_1 *and* a_2 *being leaves at the the bottom (level* n*), and* a_k *being the only leaf at level* $n - k + 2$, *for* $k > 2$ *(the root being at level 1).*

So one possible optimal encoding is for a_1 *to get symbol* $0 \cdots 0$ *(*$n - 1$ *zeros), and* a_k *to get symbol* $0 \cdots 01$ *(*$n - k$ *zeros) for* $1 < k \leq n$.

(2) (20 points) Let G be a file containing symbols b_1,...,b_m, where b_i appears c_i times. Suppose $c_1 = 1$, $c_2 = 2$ and $c_i = c_{i-1} + c_{i-2}$ for $i > 2$ (a Fibonacci sequence). We want to use Huffman encoding to compress G. Determine an optimal encoding of each symbol b_i.

Answer. The algorithm for Huffman encoding creates a priority queue of nodes (representing sets of symbols) ordered by increasing frequency of appearance (the sum of all the frequencies of symbols in the set). Initially the priority queue contains (b_1, c_1), ... , (b_m, c_m). *We claim that at step* i *of the algorithm the two lowest frequency sets removed from the priority queue are* (b_{i+1}, c_{i+1}) *and* $(\{b_1, ..., b_i\}, c_1 + \cdots + c_i)$, *where* $c_1 + \cdots + c_i = c_{i+2} - 2 \geq c_{i+1}$. *We prove this by induction. The base case is* $i = 1$, *and this is clearly true. Now suppose it is true for* i, *we must show it true for* $i + 1$. *Then at step* i, *we remove these two sets, merge them to form* $\{b_1, ..., b_{i+1}\}$ *with frequency* $c_{i+1} + c_{i+2} - 2 = c_{i+3} - 2$, *where we have used the definition of the Fibonacci sequence. The two other lowest frequency entries in the queue are* (b_{i+2}, c_{i+2}) *and* (b_{i+3}, c_{i+3}). *Since* $c_{i+2} < c_{i+3}$ *and* $c_{i+3} - 2 < c_{i+3}$, *the two lowest frequency items on the queue at step* $i + 1$ *are* (b_{i+2}, c_{i+2}) *and* $(\{b_1, ..., b_{i+1}\}, c_{i+3} - 1)$, *as claimed.*

The resulting tree created by the Huffman encoding algorithm is therefore a chain, with b_1 *and* b_2 *being leaves at the the bottom (level* m*), and* b_k *being the only leaf at level* $m - k + 2$, *for* $k > 2$ *(the root being at level 1).*

So one possible optimal encoding is for b_1 *to get symbol* $0 \cdots 0$ *(*$m - 1$ *zeros), and* b_k *to get symbol* $0 \cdots 01$ *(*$m - k$ *zeros) for* $1 < k \leq m$.

(3) (20 points) Let $p(n)$ be the number of ways you can write the positive integer n as a sum of positive integers. For example, 3 can be written as 3, $2+1$ and $1+1+1$, so $p(3)=3$. (Note that $2+1=1+2$ is only counted once, i.e. the order of summands doesn't matter.) Give a dynamic programming algorithm for computing $p(n)$. Hint: Start with a dynamic programming algorithm for the slightly different function $p(n,k)$ = the number of ways you can write n as a sum of positive integers less than or equal to k. You should include an update formula for $p(n,k)$ (with justification), a program for filling in the values of $p(n,k)$, a bound on the running time (using $O()$), and how to compute $p(n)$ from the function $p(n,k)$.

Answer. The base case for $p(n,k)$ is $p(n,1)=1$ (since $n=1+1+\cdots+1$ can only be written one way). We will also need $p(0,0)=1$ for convenience. The update formula is $p(n,k)=p(n,k-1)+p(n-k,\min(k,n-k))$, since n can either be written not using k ($p(n,k-1)$ ways) or using k ($p(n-k,\min(k,n-k))$ ways). The reason for having $\min(k,n-k)$ instead of just k is that $n-k$ cannot be written using numbers any larger than $n-k$. We can now compute $p(n,k)$ using the following program (where N is the largest value of n that we are interested in).

$p(0,0)=1$
for $n=1$ *to* N, $p(n,1)=1$, *end for*
for $n=2$ *to* N
    for $k=2$ *to* n
        $p(n,k)=p(n,k-1)+p(n-k,\min(k,n-k))$

The cost of this algorithm is $O(N^2)$. Finally, $p(n)=p(n,n)$.

(3) (20 points) Let $Q(m)$ be the number of ways you can write the positive integer m as a sum of positive integers. For example, 4 can be written as 4, $3+1$, $2+2$, $2+1+1$ and $1+1+1+1$, so $Q(4) = 5$. (Note that $3+1 = 1+3$ is only counted once, i.e. the order of summands doesn't matter.) Give a dynamic programming algorithm for computing $Q(m)$. Hint: Start with a dynamic programming algorithm for the slightly different function $Q(r, m)$ = the number of ways you can write m as a sum of positive integers less than or equal to r. You should include an update formula for $Q(r, m)$ (with justification), a program for filling in the values of $Q(r, m)$, a bound on the running time (using $O()$), and how to compute $Q(m)$ from the function $Q(r, m)$.

Answer. The base case for $Q(r, m)$ is $Q(1, m) = 1$ (since $m = 1 + 1 + \cdots + 1$ can only be written one way). We will also need $Q(0, 0) = 1$ for convenience. The update formula is $Q(r, m) = Q(r-1, m) + Q(\min(r, m-r), m-r)$, since m can either be written not using r ($Q(r-1, m)$ ways) or using r ($Q(\min(r, m-r), m-r)$ ways). The reason for having $\min(m-r, r)$ instead of just r is that $m-r$ cannot be written using numbers any larger than $m-r$. We can now compute $Q(r, m)$ using the following program (where M is the largest value of m that we are interested in).

$Q(0, 0) = 1$
for $m = 1$ *to* M, $Q(1, m) = 1$, *end for*
for $m = 2$ *to* M
  for $r = 2$ *to* m
    $Q(r, m) = Q(r-1, m) + Q(\min(r, m-r), m-r)$

The cost of this algorithm is $O(M^2)$. Finally, $Q(m) = Q(m, m)$.