Honors Algorithms/Written Qualifying Exam

Wednesday, Dec. 21, 2011

This is a $3\frac{1}{2}$ hour examination.
All questions carry the same weight.
Answer all six questions.

- Please *print* your name on the sticky note attached to the outside envelope, and nowhere else.
- Please *do* not write your name on the examination booklets.
- Please answer each question in a separate booklet, and *number* each booklet with that question number.
- Read the questions carefully. Keep your answers brief. Assume standard results, except when asked to prove them.
- When you have completed the exam, please reinsert the booklets back into the envelope.

Problem 1 **[10 points]** *New numbered exam booklet PLEASE*

a) Consider the following problem.

- Input: A target bound K and an undirected graph $G = (V, E)$ with n vertices and m edges.
- Question: Is there a subset W of $3K$ vertices such that there are exactly $3K$ edges in E that have both endpoints in W, and these edges form K disjoint triangles?

- Prove that this problem is NP-Complete.
  The reduction can use any standard problem that is known to be NP-Complete.

Solution

a) First, this problem is in NP: I can write a program to check the solution: it counts the number of triangles, checks that all of its edges are in E, verifies that they are disjoint, and checks to see if there are any other edges in E that have both endoints in the set of $3K$ vertices.

b) We reduce Independent Set to this problem. The IS problem is: given an undirected graph $H = (W, F)$ and target K, is there a set of K vertices in W such that no pair of these vertices are connected by an edge in F?

The reduction is: for each w in W, insert w, w_1 and w_2 into V, and edges (w, w_1), (w, w_2), (w_1, w_2) into E. Also insert F into E. It is clear that a solution to IS gives a solution to this problem. To see that a solution to this problem gives a solution to IS, note that each triangle can have no more than two vertices that lie outsde of W, so the it suffices to select from each solution triangle a vertex that belongs to W to get independent set of at least K vertices.

b) Consider the following problem.

- Input: An undirected graph $G = (V, E)$ with $V = \{1, 2, 3, \ldots, n\}$, and a bound K.
- Question: Is there a subset S of vertices from V where each edge in E has at least one endpoint in S, and the sum of the vertex number-names in S is no more than K?

- Prove that this problem is NP-Complete.
  `Hint:` The difficulty with this problem is that as defined, far more vertices with small number-names can be used in a cover than with large number-names.

Solution

a) First, this problem is in NP: I can write a program to check the solution: it verifies that the solution is a vertex cover, and adds up the vertex numbers to see if the sum is at most K.

b) We reduce Vertex Cover to this problem. The VC problem is: given an undirected graph $H = (W, F)$ and target T, is there a set S of K vertices in W where each edge in F has at least one endpoint in S.

The idea is: Let $|W| = n$. Given H, create G as follows: it has n^2 isolatied edges with left endpoint vertices named 1 through n^2 and right endpoint vertices named $n^2 + 1$ through $2n^2$. And it contains a copy of H with vertex numbers $2n^2 + 1$ through $2n^2 + n$. Set $K = 1 + 2 + \ldots + n^2 + T(2n^2 + n)$. The solution must find at most T vertices in W because $(T + 1)(2n^2 + 1) > 2Tn^2 + 2n^2 > T(2n^2 + n)$. So a solution to this problem for G is a solution for H. Likewise, a solution for H gives a solution for this thing on G.

Problem 2 **[10 points]** *New numbered exam booklet PLEASE*

Let G be an undirected graph with edge weight function w, and m edges $e_1 \dots e_m$ in sorted according to their weight: $w(e_1) \leq w(e_2) \leq \dots \leq w(e_m)$. Imagine you just ran the some MST algorithm on G and it output an MST T of G. Now suppose that somebody changes the weight of a single edge e_i from $w(e_i)$ to some other value w'. For each of the following 3 scenarios, describe (at a high level) the fastest algorithm you can think of to transform the original MST T of G into a new (and correct) MST T' of G after the edge weight change. Be sure to justify the correctness of your answers, and to express your running time as a function of m and n.

a) Suppose e_i is in the MST T and $w' < w(e_i)$ (so the weight of the MST edge e_i is decreased).

Solution

T is an MST solution T'. So the answer runs in constant time. Correctness: all cuts are the same for each tree, and the same edges are min cost connectors across those cuts.

b) Suppose e_i is not in T and $w' < w(e_i)$ (so the weight of a non-MST edge e_i is decreased).
`Hint:` Compute the unique shortest path in T between the two end-points of e_i.

Solution

Orient the tree via a DFS so that each vertex (other than the root) has an parent. Now it is easy to trace the cycle formed by conceptually adding the edge e_i to T. If w' is cheaper than some edge on the cycle, remove the most expensive edge on the cycle and keep e_i in the new solution tree T'. Otherwise the solution is the original T.

Time: linear. Correctness: If T' is T there is nothing to prove. Otherwise call the excised edge e_j. Imagine running a min cut based MST algorithm where you omit any cut that would select the new e_i with cost w'. (You do this by selecting e_i when it is the min cost connector, but conceptually refrain from inserting it in the MST.) This will build two subtrees of altogether $n - 2$ edges. Each edge in T minus e_j is a min cost connector across its cut, and each edge in T minus e_j will qualify as such a min cost connector. So use those edges. Now complete the construction by adding the cheapest connector across the remaining cut. Two contenders are e_i and e_j but there cannot be any other connector (besides e_i) of cost less than $w(e_j)$, since e_j belonged to T. So e_i is the final edge to add.

c) Suppose e_i is in T and $w' > w(e_i)$ (so the weight of the MST edge e_i is increased).

Solution

Remove e_i and trace the two resulting subtrees to identify the vertices on each side of the cut. Partition E into the edges connecting opposite sides of the cut, and those that are useless. Select the cheapest of the connecting edges. If it is cheaper that w' use it. If not, the tree is unchanged.

Time $\Theta(|E|)$. Correctness: T is built from min-cut cost connectors.

Problem 3 **[10 points]** *New numbered exam booklet PLEASE*

You are given a collection of equations of the form variable = variable + constant
For example:

$$a = d + 2$$
$$a = b + 4$$
$$c = f - 1$$
$$a = f + 3$$
$$c = a - 2$$
etc.

For notational simplicity, you can assume that the variables are $a_1, a_2, \ldots, a_n$, and the equations are formulated as the m triples $(D[i], E[i], F[i])$ for $i = 1, 2, 3, \ldots, m$. The meaning of the triple $(D[i], E[i], F[i])$ is that equation i reads: $a_{D[i]} = a_{E[i]} + F[i]$.
Present an $O(m + n)$-time to determine if the system has a solution.
`Hints::` Note that this system can be modeled as a graph. Moreover, if there is a solution, then you can find another solution by adding the same constant to all of the variables. This means that if there is a solution, then there is also a solution where – for example – a_1 is zero. In addition, please note that the system might define a graph that is not connected.

Solution

Create a directed graph of n vertices where vertex i represents the variable a_i. For each triple $(D[i], E[i], F[i])$ insert a directed edge from vertex $D[i]$ to vertex $E[i]$ with length $F[i]$.

There are many ways to do the consistency check. One might check the strong components and then check the overall DAG. But the easiest way is to double the edges by assigning the two edges $(D[i], E[i])$ with cost $F[i]$ and $(E[i], D[i])$ with cost $-F[i]$ to the graph. So let G be this graph, which is just a set of disjoint subgraphs that are strongly connected. The consistency check is as follows.

```
Global : $Consistent$;
procedure Driver();
  $Consistent \leftarrow$ TRUE;
  mark all vertex costs Nil;
  foreach vertex $v$ do
    if $v.cost$ is Nil then
      $v.cost \leftarrow 0$;
      $DFS(v)$
    endif
  endfor;
  print($Consistent$)
end_Driver;

procedure DFS(w);
  foreach vertex $u$ in $Adj[w]$ do
    if $u.cost$ is Nil then
      $u.cost \leftarrow w.cost + Edgecost(w, u)$;
      $DFS(u)$
    elseif $u.cost \neq w.cost + Edgecost(w, u)$ then
      $Consistent \leftarrow$ FALSE
    endif
  endfor
end_DFS;
```

It is not difficult to create the adjacency lists for G in $\Theta(m + n)$ time. The edge costs can then be stored in a companion adjacency edge cost array so that the k^{th} element in $Ecost[j]$ is the edge cost of (j, h), where h is the k^{th} entry in $Adj[j]$. With this construction, total time to construct G and to run the algorithm is $\Theta(|V| + |E|)$, which is $\Theta(m + n)$.

Problem 4 [10 points] *New numbered exam booklet PLEASE*

Let $A[1..n]$ be an array of n floating point numbers.

a) Describe a linear-time algorithm that determines if more than $n/2$ of the elements have the same value.
Hint: Imagine seeing the data in sorted order (even though you cannot sort the data in linear time). Now use the Blum Floyd Pratt Rivest Tarjan fast selection (or order statistics or rank selection) algorithm to find a candidate element that could appear more than half of the time in A.

Solution

Use the BFPRT algorithm to find the median. Now collect all values equal to the median. If that count is more than $n/2$ report yes. Otherwise report no.

b) Adapt your argument to the determine if more than $n/3$ of the elements have the same value.
`Hint:` You might need multiple passes of the BFPRT algorithm.

Solution

Use the BFPRT algorithm to find an element of rank $\frac{n}{3}$. Now collect all values equal to that value. If that count is more than $n/3$ report yes.

If not, use the BFPRT algorithm to find an element of rank $\frac{2n}{3}$. Now collect all values equal to that value. If that count is more than $n/3$ report yes. If not, report no.

c) Describe a fast algorithm that determines if there is an element that appears in A more than n/C times for any input parameter C.
Your target operation count is $O(C \cdot n)$.

Solution

For k sequencing from 1 to $C-1$ and while no yes has been reported do this:

> Use the BFPRT algorithm to find an element of rank $\frac{kn}{C}$. Now collect all values equal to that value. If that count is more than n/C report yes.

If no yes has been reported then report no.

d) Adapt your solution for c) to present, for any $C = 2^k$, an informal, very high level description of an algorithm that prints all of the elements that appear more than n/C times, and which runs in $O(n+n\log C)$ time.

Solution

Let A be the set of n numbers.

```
    Global YesNo;
    $YesNo \leftarrow no$;
    $many(A, k, n/C)$;
    print(YesNo);
```

```
procedure $many$(S,k,count)
    if $k$ is 0 OR YesNo is $yes$ then return endif;
    Use the BFPRT algorithm to find the median $\mu$ of $S$;
    Collect three sets: $W$ conains the elements $< \mu$, $X$ the values $= \mu$
            and $Y$ the values $> \mu$;
    if $|X| > count$ then $YesNo \leftarrow yes$
    else
        $many(W, k-1, count)$;
        if YesNo is no then $many(Y, k-1, count)$ endif
    endif
end_$many$;
```

Problem 5 **[10 points]** *New numbered exam booklet PLEASE*

a) The baby sitting assignment problem

You are the CTO of a baby sitting company, and have a request to baby sit n children one day. You can hire any number of baby sitters for a fixed cost of B dollars per baby sitter. Also, you can assign an arbitrary number of children $i \geq 1$ to any baby sitter. However, each parent will only pay the amount $p[i]$ if her child is taken care of by a baby sitter who looks after i children. For example, if $n = 7$ and you hire 2 baby sitters with one looking after 3 children and the other managing 4 children, then your total profit is $3p[3] + 4p[4] - 2B$.

Given $B, n, p[1], \ldots, p[n]$, your job is to assign children to baby sitters in a way that maximizes your total profit. So you want to find an optimal number k and an optimal partition $n = n_1 + \ldots + n_k$ so as to maximize revenue $R = n_1 \cdot p[n_1] + \ldots + n_k \cdot p[n_k] - k \cdot B$.

i) Present a recursive high-level program specification to find the maximum revenue R based on the input data B, n, and $P[1..n]$.
Note: the technical difficulty with this problem occurs when n is not a multiple of the "most profitable" number of children to assign to each baby sitter. For this general case, you will not find a convenient formula or simpler solution than that provided by dynamic programming.

Solution

Let $Best[j]$ be the solution to the subproblem for j children. Then

$$Best[j] = \begin{cases} 1; & \text{if } j = 0; \\ \max_{0 \leq h < j}\{Best[h] + p[j-h] - B\}, & \text{if } j > 0. \end{cases}$$

ii) Analyze – with a brief justification – the running time of your procedure when augmented with a look-up table to ensure that each subproblem is solved just once. Use $\Theta(\cdot)$ notation.

Solution

The operation count is quadratic because it takes $\Theta(j)$ time to check the j outcomes needed to compute the answer for location j of the look-up table when locations 1 through $j - 1$ are already filled.

b) Arithmetic to the max

You are given a sequence of positive floating point numbers. Insert $+$'s, $*$'s, and parentheses so as to get the largest possible expression.

For example, suppose the sequence is $1.5\ \ 2.0\ \ 2.5\ \ 1.0$.

Options include:

$$(1.5 * 2.0) + (2.5 * 1.0) = 5.5$$
$$1.5 + 2.0 + 2.5 + 1.0 = 7.0$$
$$1.5 * 2.0 * 2.5 * 1.0 = 7.5$$
$$(1.5 + 2.0) * (2.5 + 1.0) = 12.25$$

Present a high-level recursive program specification to compute the largest possible expression for the data $A[1..n]$.

Solution

Let $S[i,k]$ be the solution to the subproblem as defined on $A[i..k]$. Then

$$S[i,k] = \begin{cases} A[i] & \text{if } i = k; \\ \max_{i \le j < k} \{max\{(S(i,j) + S(j+1,k)),\ (S(i,j) * S(j+1,k))\}\} & \text{if } i < k. \end{cases}$$

Problem 6 [10 points] *New numbered exam booklet PLEASE*

For the two languages below, present a proof that shows if each is a CFL.

`Hints:` Recall that the recognizer can be nondeterministic, and when applying a pumping lemma, pick an initial string wisely, and be ready to pump in any direction.

a)
- Let $L_a = \{a^i b^j c^k : i,j,k\ in\ \{1,2,3,\cdots\}\ and\ (i \ne j\ or\ i \ne k)\}$.
- Present a proof that determines if L_a is a CFL.

Solution

The language is a CFL. We simply guess if it is j or k that is not equal to i. Then we push a token onto the stack for every a that is read, and pop the stack as the guessed symbol b or c appears. When done, we want the stack to be non-empty after or to have already been emptied before the last of the guessed symbols is read. We also check to ensure that the string belongs to $a^+b^+c^+$. So the language is a CFL.

b)
- Let $L_b = \{a^i b^j c^{\max\{i,j\}} : i,j\ in\ \{1,2,3,4,\cdots\}\}$.
- Present a proof that determines if L_b is a CFL.

Solution

This language is not a CFL. To prove that, we use a pumping lemma.

The $uvxyz$ pumping lemma says: for any CFL L_b, there is a count n such that: if s is in L_b and $|s| > n$, then we can write $s = uvxyz$ where $|vy| > 0$, and uv^ixy^iz is in L_b for $i = 0, 1, 2, \cdots$.

Let $s = a^n b^n c^n$. It is in L_b by definition. So $s = uvxyx$ for some strings u, v, x, y, and z where the pumping lemma is applicable. It is clear that v must belong to a^+ as otherwise uv^ixy^iz cannot belong to L_b. And for the same reason, y must belong to either b^+ or c^+. But then uv^0xy^0z cannot belong to L_b because its number of as will be less than the number of bs if y is in c^+, or less than the number of cs if y is in b^+.