BU CLA CS 530: Analysis of Algorithms
Spring 1999

Selected Solutions for Problem Set 4

Problem 1.

(a) Exercise 13 in Handout 2. The time complexity of TEST_2 is $\Theta(n^2)$. If the test in line 2 of TEST_1 is changed to "$X(AB) = XC$", the time complexity of TEST_2 is $\Theta(M(n))$ where $M(n)$ is the time to multiply two $n \times n$ integer matrices. We assume that $M(n)$ is at most $\mathcal{O}(n^3)$ (the complexity of the classical algorithm for matrix multiplication) and at least $\Omega(n^2)$ (the complexity of inspecting all entries in both matrices A and B).

(b) Exercise 14 in Handout 2. This is equivalent to testing whether $AB = I_n$ where I_n is the $n \times n$ identity matrix.

Problem 2.

Suppose files $f_1, f_2, \dots, f_n$ have lengths $m_1, m_2, \dots, m_n$, respectively. Let $i_1, i_2, \dots, i_n$ be a permutation of $1, 2, \dots, n$. Suppose we store the files in the order $f_{i_1}, f_{i_2}, \dots, f_{i_n}$. To retrieve the kth file f_{i_k} on the tape costs $\sum_{j=1}^{k} m_{i_j}$. The cost of retrieving all n files is therefore:

(†) $\sum_{k=1}^{n} \sum_{j=1}^{k} m_{i_j} = \sum_{k=1}^{n} (n-k+1) m_{i_k}$

To see why the left-hand side of (†) equals its right-hand side, consider the table:

$$\begin{array}{rl}
f_{i_1}: & m_{i_1} \\
f_{i_2}: & m_{i_1} + m_{i_2} \\
f_{i_3}: & m_{i_1} + m_{i_2} + m_{i_3} \\
 & \vdots \\
f_{i_{n-1}}: & m_{i_1} + m_{i_2} + m_{i_3} + \cdots + m_{i_{n-1}} \\
f_{i_n}: & m_{i_1} + m_{i_2} + m_{i_3} + \cdots + m_{i_{n-1}} + m_{i_n}
\end{array}$$

The left-hand side of (†) adds this table by rows. If the total is computed by columns, we get:

$$n \cdot m_{i_1} + (n-1) \cdot m_{i_2} + (n-2) \cdot m_{i_3} + \cdots + 2 \cdot m_{i_{n-1}} + m_{i_n} = \sum_{k=1}^{n} (n-k+1) m_{i_k}$$

which is the right-hand side of (†). The greedy algorithm picks the files $f_1, f_2, \dots, f_n$ in order of increasing sizes (or non-decreasing sizes, if distinct files can have the same size). It remains to show that this is the minimum-cost permutation.

Claim: Any permutation in nondecreasing order of the m_i's has minimum cost.

Proof: Let $\Pi = (i_1, i_2, \dots, i_n)$ be a permutation of $1, 2, \dots, n$ that is not in nondecreasing order of the m_i's. We prove that the corresponding cost is not minimum. Because $m_{i_1}, m_{i_2}, \dots, m_{i_n}$ is not in decreasing order, there is $j \in \{1, \dots, n_1\}$ such that $m_{i_j} > m_{i_{j+1}}$. Let Π' be permutation Π

with i_j and i_{j+1} swapped. The cost of permutation Π is

$$\begin{aligned} C(\Pi) &= \sum_{k=1}^{n}(n-k+1)m_{i_k} \\ &= \Big(\sum_{k=1}^{j-1}(n-k+1)m_{i_k}\Big) + (n-j+1)\cdot m_{i_j} + (n-j)\cdot m_{i_{j+1}} + \Big(\sum_{k=j+2}^{n}(n-k+1)m_{i_k}\Big) \end{aligned}$$

The cost of permutation Π' is

$$C(\Pi') = \Big(\sum_{k=1}^{j-1}(n-k+1)m_{i_k}\Big) + (n-j+1)\cdot m_{i_{j+1}} + (n-j)\cdot m_{i_j} + \Big(\sum_{k=j+2}^{n}(n-k+1)m_{i_k}\Big)$$

Hence,

$$C(\Pi) - C(\Pi') = m_{i_j} - m_{i_{j+1}} > 0 \qquad \text{(by the choice of } j\text{)}$$

Hence, $C(\Pi) > C(\Pi')$ and Π is not a permutation of minimum cost. ■

Problem 3. (Exercize 17.2-2 in [CLR], page 336)

Key idea: Let i be the highest-numbered item in an optimal solution S for W pounds and items $1,\dots,n$. Then $S' = S - \{i\}$ is an optimal solution for $W - w_i$ pounds and items $1,\dots,i-1$, and the value of the solution S is v_i plus the value of the subproblem solution S'.

We can express this in the following formula: Define $c[i,w]$ to be the value of the solution for items $1,\dots,i$ and maximum weight w. Then:

$$c[i,w] = \begin{cases} 0, & \text{if } i = 0 \text{ or } w = 0, \\ c[i-1,w], & \text{if } w_i > w, \\ max\{v_i + c[i-1,w-w_i], c[i-1,w]\}, & \text{if } i > 0 \text{ and } w \geq w_i \end{cases}$$

This says that the value of a solution for i items either includes item i, in which case it is v_i plus a subproblem solution for $(i-1)$ items and the weight excluding w_i, or does not include item i, in which case it is a subproblem solution for $(i-1)$ items and the same weight.

The algorithm takes as input the maximum weight W, the number of items n, and the two sequences $v = \langle v_1, v_2, \dots, v_n\rangle$ and $w = \langle w_1, w_2, \dots, w_n\rangle$. It stores the $c[i,j]$ values in a table, i.e. a 2–dimensional array, $c[0..n, 0..W]$ whose entries are computed in a row-major order. (That is, the first row of c is filled in from left to right, then the second row, and so on.) At the end of the computation, $c[n,W]$ contains the maximum value that can be packed into the knapsack.

```
DYNAMIC-0-1-KNAPSACK (v, w, n, W)
for w ← 0 to W
    do c[0, w] ← 0
for i ← 1 to n
    do c[i, 0] ← 0
       for w ← 1 to W
           do if w_i ≤ w
                then if v_i + c[i − 1, w − w_i] > c[i − 1, w]
                        then c[i, w] ← v_i + c[i − 1, w − w_i]
                        else c[i, w] ← c[i − 1, w]
                else c[i, w] ← c[i − 1, w]
```

The set of items to take can be deduced from the c table, starting at $c[n, W]$ and tracing backwards where the optimal values came from. If $c[i, w] = c[i-1, w]$, item i is not part of the solution, and we continue tracing with $c[i-1, w]$. Otherwise item i is part of the solution, and we continue tracing with $c[i-1, w-w_i]$.

The above algorithm takes $\Theta(nW)$ time, broken up as follow:

- $\Theta(nW)$ time to fill the c table, which has $(n+1) \cdot (W+1)$ entries, each requiring $\Theta(1)$ time to compute.
- $\mathcal{O}(n)$ time to trace the solution, because the tracing process starts in row n of the table and moves up 1 row at each step.

Problem 4. (Exercize 25.2-4 in [CLR], page 531)

First Solution

To find the most reliable path between s and t, run Dijkstra's algorithm with edge weights $w(u, v) = -\log(r(u, v))$ in order to find shortest paths from s in $\mathcal{O}(E + V \log(V))$ time. The most reliable path is the shortest path from s to t, and that path's reliability is the product of the reliabilities of its edges.

Correctness of the algorithm just described: Because the probabilities are independed, the probability that a path will not fail is the product of the probabilities that its edges will not fail. We want therefore to find a path p from s to t such that:

$$r(v_1, v_2) \cdot r(v_2, v_3) \cdot \ldots \cdot r(v_{k-1}, v_k)$$

is maximized, where $v_1 = s, v_k = t$, and the path p is the sequence of edges:

$$(v_1, v_2),\ (v_2, v_3),\ \ldots\ (v_{k-1}, v_k)$$

This is equivalent to maximizing (why? review logarithms at the bottom of page 34):

$$\log(r(v_1, v_2)) + \log(r(v_2, v_3)) + \ldots + \log(r(v_(k-1), v_k))$$

which is equivalent to minimizing (why?):

$$(-\log(r(v_1, v_2))) + (-\log(r(v_2, v_3))) + \ldots + (-\log(r(v_{k-1}, v_k))$$

Thus if we assign weights $w(u,v) = -\log(r(u,v))$, we have a shortest-path problem, to which we can apply the algorithms of Chapter 25. Note that $r(u,v)$ can be 0, and $\log(0)$ is undefined. So in this algorithm, define $\log(0) = -\infty$.

Since $\log(1) = 0$, and $\log(x) < 0$ for $0 < x < 1$, and we have defined $\log(0) = -\infty$, all the weights w are non-negative, and we can use Dijkstra's algorithm to find the shortest paths from s in $\mathcal{O}(E + V\log(V))$ time.

Second Solution

You can also work with the original probabilities by running a modified version of Dijkstra's algorithm that maximizes the product of reliabilities along a path instead of minimizing the sum of weights along a path. In Dijkstra's algorithm, use the reliabilities as edge weights and substitute:

- max (and EXTRACT-MAX) for min (and EXTRACT-MIN) in relaxation and the queue,
- $*$ for $+$ in relaxation,
- 1 (identity for $*$) for 0 (identity for $+$),
- $-\infty$ (identity for min) for $+\infty$ (identity for max).

For example, the following is used instead of the usual RELAX procedure:

```
RELAX -RELIABILITY (u, v, r)
1.   if d[v] < d[u] * r(u, v)
2.      then d[v] ← d[u] * r(u, v)
3.           pi[v] ← u
```

This algorithm is essentially the same as the one described in the first solution above: It performs the same operations except that it is working with the original probabilities instead of the transformed ones.

Assaf Kfoury
Created: 99.02.10