McGill University COMP360 Winter 2011 **Instructor: Phuong Nguyen**

Assignment 8 Solution

Question 1 (10pt) You are given a color picture consisting of an $m \times n$ array A of pixels, where each pixel specifies a triple of red, green, and blue (RGB) intensities. Suppose that we wish to compress this picture slightly. Specifically, we wish to remove one pixel from each of the m rows, so that the whole picture become one pixel narrower. To avoid disturbing visual effects, however, we require that the pixels removed in two adjacent rows be in the same or adjacent columns; the pixels removed from a "seam" from the top row to the bottom row where successive pixels in the seam are adjacent vertically or diagonally.

(a) [2pt] Show that the number of such possible seams grows at least exponentially in m (the number of rows), assuming that $n > 1$.

(b) [8pt] Suppose now that along with each pixel $A[i, j]$, we have calculated a real valued disruption measure $d[i, j]$, indicating how disruptive it would be to remove pixel $A[i, j]$. Intuitively, the lower a pixel's disruption measure, the more similar the pixel is to its neighbors. Suppose further that we define the disruption measure of a seam to be the sum of the disruption measures of its pixels.

Give a dynamic programming algorithm to find a seam with the lowest disruption measure. The space used by your algorithm must be $\mathcal{O}(mn)$. (For space requirement, assuming that the total disruption measure of any seam requires a constant amount of space.) The running time of your algorithm must be a polynomial in mn; calculate it.

Solution (a) For each pixel on one row there are at least 2 choices for the pixel on the next row. Thus the total number of seams is at least 2^m.

(b) Step 1: The related problems that we are solving is to consider for each row i and each $j \leq n$ the minimum disruption measure of a seam that starts from the first row and ends at row i at the pixel $A[i, j]$. Let $B[i, j]$ denote this quantity.

Step 2: For the base case, $i = 1$, and we have $B[1, j] = d[1, j]$.

For the induction step, the seam that ends in pixel $A[i, j]$ on row i must arrive on row $i - 1$ at one of the pixels $A[i - 1, j - 1], A[i - 1, j]$ or $A[i - 1, j + 1]$, provided that $2 \leq j \leq n - 1$. If $j = 1$ then the pixels on row $i - 1$ must be $A[i - 1, 1]$ or $A[i - 1, 2]$. Similarly, if $j = n$ then the pixel on row $i - 1$ must be $A[i - 1, n]$ or $A[i - 1, n - 1]$.

So we have the following recurrence formula: If $2 \leq j \leq n - 1$ then

$$B[i, j] = \min\big(B[i - 1, j - 1], B[i - 1, j], B[i - 1, j + 1]\big) + d[i, j]$$

Also,

$$B[i, 1] = \min\big(B[i - 1, 1] + B[i - 1, 2]\big) + d[i, 1]$$

and

$$B[i, n] = \min\big(B[i - 1, n] + B[i - 1, n - 1]\big) + d[i, n]$$

Step 3: The program for computing B is as follows.

1. Let B be an array of size $m \times n$
2. for j from 1 to n let $B[1, j] \leftarrow d[i, j]$
3. for i from 2 to n do
4. $\quad B[i, 1] \leftarrow \min\big(B[i-1, 1] + B[i-1, 2]\big) + d[i, 1]$
5. $\quad B[i, n] \leftarrow \min\big(B[i-1, n] + B[i-1, n-1]\big) + d[i, n]$
6. $\quad$ for j from 2 to $n-1$ do
7. $\quad\quad B[i, j] \leftarrow \min\big(B[i-1, j-1], B[i-1, j], B[i-1, j+1]\big) + d[i, j]$
8. $\quad$ end for
9. end for

Step 4: To use the array B to compute a seam of minimum disruption, we look for the smallest value of $B[n, j]$ to find the pixel of the seam on row n. Then we trace the computation of such $B[n, j]$ to find the seam's pixel on row $n-1$, etc. In general, suppose that pixel $A[i, j_i]$ belongs to the seam of minimum disruption, then the pixel on row $i-1$ is such neighbor $A[i-1, j]$ that makes

$$A[i, j_i] = A[i-1, j] + d[i, j]$$

The program is as follows. The seam will be stored in an array S of size m, where $S[i]$ is the column index of the pixel on row i.

1. Let S be an array of size m.
2. Let $S[m]$ be a j such that $A[m, j]$ is minimal among $A[m, 1], \ldots, A[m, n]$
3. for i from m down to 2 do
4. $\quad$ if $S[i] = 1$
5. $\quad\quad$ if $B[i-1, 1] < B[i-1, 2]$ then $S[i-1] \leftarrow 1$
6. $\quad\quad$ else $S[i-1] \leftarrow 2$
7. $\quad$ else if $S[i] = n$
8. $\quad\quad$ if $B[i-1, n-1] < B[i-1, n]$ then $S[i-1] \leftarrow n-1$
9. $\quad\quad$ else $S[i-1] \leftarrow n$
10. $\quad$ else
11. $\quad\quad$ find $j \in \{S[i]-1, S[i], S[i]+1$ with smallest $B[i-1, j]$
12. $\quad\quad S[i-1] \leftarrow j$
13. $\quad$ end if

14. end for

Analysis The space required by the algorithm is $\mathcal{O}(mn)$ because it needs an array B of size $m \times n$ where each cell occupies constant space. The construction of B takes time $\mathcal{O}(mn)$ because each operation on lines 4,5,7 takes constant time. Finally, the building of the seam in Step 4 takes time $\mathcal{O}(m)$.

Question 2 (10pt) Consider a directed graph G where each edge e is labeled with a label called $\ell(e)$ that comes from a set L of labels. We define the label of a directed path to be the concatenation of the labels of the edges on that path.

(a) [2pt] Describe a polytime algorithm that, given an edge-labeled directed graph G with a distinguished vertex s and a sequence of labels $(\ell_1, \ell_2, \ldots, \ell_k)$, returns a path in G that begins with s and has the sequence $(\ell_1, \ell_2, \ldots, \ell_k)$ as its label, if any such path exists. Otherwise the algorithm must return NO-SUCH-PATH. Analyze the running time of your algorithm.

(b) [8pt] Now suppose that every edge e in the graph is associated with a positive weight $w(e)$. The weight of a path is defined to be the sum of the weights of its edges. Extend your algorithm in (a) so that it returns a path of maximum weight. Your algorithm must still run in time polynomial in the size of the input.

Here we assume that each label ℓ in L, each weight $w(e)$, as well as the total weight of any path of length n ($n = |V|$ is the total number of vertices in G) occupy a constant amount of space, and that adding/comparing two weights can be done in constant time.

Solution (a) For each distance $i \leq k$ we will list all vertices v of G such that there is a path of length i from s to v whose label is $(\ell_1, \ell_2, \ldots, \ell_i)$. If the list for $i = k$ is empty then we will output NO-SUCH-PATH. Otherwise we trace through the lists to compute a path.

More formally, we have a boolean array $A[1 \ldots k, 1 \ldots n]$ of dimension $k \times n$, where n is the number of vertices of G. On column i of A we list the vertices of the i-th list above: the bit $A[i, v] = 1$ iff v is in the list. This can be computed as follows:

1. initial A so that $A[i, v] = 0$ for all $1 \leq i \leq k, 1 \leq v \leq n$
2. for all neighbors v of s such that $\ell(s, v) = \ell_1$ do $A[1, v] \leftarrow 1$
3. for i from 1 to $k - 1$ do
4. for v from 1 to n do
5. if $A[i, v] = 1$
6. for all u such that (v, u) is an edge of G and $\ell(v, u) = \ell_{i+1}$ do $A[i + 1, u] \leftarrow 1$
7. end if
8. end for
9. end for

Now if for all v, $A[k, v] = 0$ then there is no path satisfying the condition of the problem and we output NO-SUCH-PATH. Suppose now that for some v, $A[k, v] = 1$ then there is a path from s to v with labels $(\ell_1, \ell_2, \ldots, \ell_k)$. We show how to compute such a path. We will use an array P of length $k + 1$ so that $P[i]$ is the i-th vertex on the path ($P[0]$ is s).

1. let $P[k]$ be the first v such that $A[k, v] = 1$
2. $u \leftarrow P[k]$
3. for i from $k - 1$ to 1 do
4. find a v such that $(v, u) \in G$ and $\ell(v, u) = \ell_{i+1}$ and $A[i, v] = 1$
5. $P[i] \leftarrow v$
6. $u \leftarrow v$
7. end for

(b) We modify the algorithm in (a) slightly.

Step 1: The array A is no longer boolean here: instead, each $A[i, v]$ contains the maximum weight of a s-v path with label $(\ell_1, \ell_2, \ldots, \ell_i)$.

Step 2: The recurrence for A is as follows.

$$A[1, v] = \begin{cases} w(s, v) & \text{if } \ell(s, v) = \ell_1 \\ 0 & \text{otherwise} \end{cases}$$

For $1 \leq i \leq k - 1$ and $1 \leq u \leq n$:

$$A[i + 1, u] = max_{v:\ell(v,u)=\ell_{i+1}}(A[i, v] + w(v, u))$$

Step 3: The program for computing A:

1. initial A so that $A[i, v] = 0$ for all $1 \leq i \leq k, 1 \leq v \leq n$
2. for all neighbors v of s such that $\ell(s, v) = \ell_1$ do $A[1, v] \leftarrow w(s, v)$
3. for i from 1 to $k - 1$ do
4. for v from 1 to n do
5. if $A[i, v] > 0$
6. for all u such that (v, u) is an edge of G and $\ell(v, u) = \ell_{i+1}$ do
7. if $A[i + 1, u] < A[i, v] + w(v, u)$ then $A[i + 1, u] \leftarrow A[i, v] + w(v, u)$
8. end for
9. end if

10. end for

11. end for

Step 4: To compute a path of maximum weight, we proceed pretty much in the same way as in (a), but here we need to find out the maximum total weight first. The program is as follows.

1. find v such that $A[k, v]$ is maximum
2. if $A[k, v] = 0$ then output NO-SUCH-PATH
3. $P[k] \leftarrow v$
4. $u \leftarrow P[k]$
5. for i from $k - 1$ to 1 do
6. find a v such that $(v, u) \in G \& \ell(v, u) = \ell_{i+1} \& A[i, v] > 0 \& A[i + 1, u] = A[i, v] + w(v, u)$
7. $P[i] \leftarrow v$
8. $u \leftarrow v$
9. end for