Homework #4

(Due: May 12)

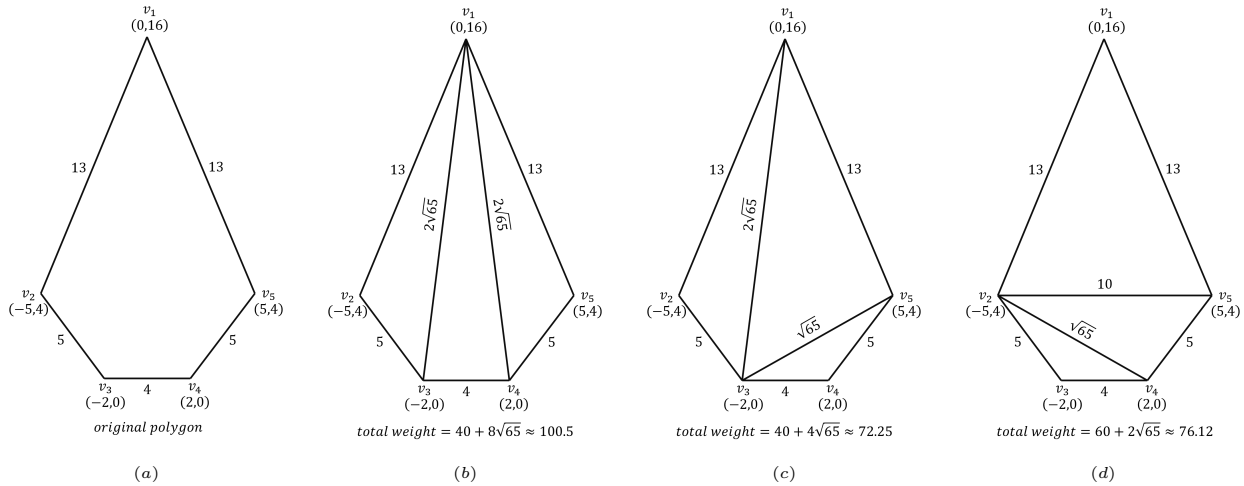

Figure 1: Three possible triangulations of a convex polygon. The triangulation in (c) is the best as it has the smallest weight.

Task 1. [200 Points] Optimal Polygon Triangulation. Let P be a convex polygon defined by a sequence of n vertices $\langle v_1, v_2, \ldots, v_n \rangle$. Since P is convex, for every pair of distinct vertices v_i and v_j $(1 \leq i, j \leq n)$, the straight line joining the two vertices will lie completely inside the polygon. By $len(v_i, v_j)$ we denote the length of the straight line joining v_i and v_j. Figure $1(a)$ shows an example convex polygon containing 5 vertices with $len(v_1, v_2) = 13$, $len(v_2, v_3) = 5$, $len(v_3, v_4) = 4$, $len(v_4, v_5) = 5$ and $len(v_5, v_1) = 13$.

We define a weight function w for triangles formed by the vertices of P. If v_i, v_k and v_j are three distinct vertices of P, we define $w(i, k, j) = len(v_i, v_k) + len(v_k, v_j) + len(v_j, v_i)$. For example, in Figure $1(b)$, $w(1, 2, 3) = len(v_1, v_2) + len(v_2, v_3) + len(v_3, v_1) = 13 + 5 + 2\sqrt{65} = 18 + 2\sqrt{65}$. If the vertices are not distinct (e.g., at least two of them are the same), we define $w(i, k, j) = 0$. Other weight functions are also possible.

The *optimal polygon triangulation* problem asks for decomposing a given convex polygon into non-overlapping triangles such that the total weight of all those triangles is minimized. The triangles must cover the entire polygon. For example, Figures $1(b)$, $1(c)$ and $1(d)$ show three possible triangulations of the convex polygon shown in Figure $1(a)$. Figure $1(b)$ decomposes the polygon into triangles $\triangle v_1 v_2 v_3$, $\triangle v_1 v_3 v_4$ and $\triangle v_1 v_4 v_5$ with a total weight of $w(1, 2, 3) + w(1, 3, 4) + w(1, 4, 5) = (13 + 5 + 2\sqrt{65}) + (2\sqrt{65} + 4 + 2\sqrt{65}) + (13 + 5 + 2\sqrt{65}) = 60 + 8\sqrt{65} \approx 100.5$. Among the three triangulations the one in Figure $1(c)$ has the smallest weight which is, indeed, an optimal triangulation of the given polygon.

Let $t[1:n, 1:n]$ be an $n \times n$ matrix in which entry $t[i,j]$ stores the total weight of an optimal triangulation of the convex polygon formed by the sequence $\langle v_i, v_{i+1}, \ldots, v_j \rangle$, where $1 \leq i < j \leq n$. Then the total weight of an optimal triangulation of P is given by $t[1,n]$ which can be computed using the following recurrence relation:

$$t[i,j] = \begin{cases} \infty & \text{if } 1 \leq i = j \leq n, \\ 0 & \text{if } 1 \leq i = j - 1 < n, \\ \min_{i \leq k \leq j} \{t[i,k] + t[k,j] + w(i,k,j)\} & \text{if } 1 \leq i < j - 1 < n. \end{cases} \quad (1)$$

```
ITERATIVE-TRIANGULATION( t, n )
(Input is an n × n matrix t[1 : n, 1 : n] with t[i, j] = 0 for 1 ≤ i = j − 1 < n and t[i, j] = ∞ otherwise (i.e.,
i ≠ j − 1). This function updates all t[i, j] with 1 ≤ i < j − 1 < n based on Recurrence 1.)
1. for i ← n − 1 downto 1 do
2.     for j ← i + 2 to n do
3.         for k ← i to j do
4.             t[i, j] ← min { t[i, j], t[i, k] + t[k, j] + w(i, k, j) }
```

Figure 2: Iterative implementation of Recurrence 1.

(a) [**20 Points**] An iterative algorithm for computing Recurrence 1 using three nested ***for*** loops is given in Figure 2. Explain how you would correctly parallelize this algorithm using only parallel ***for*** loops. Compute its work, span and parallelism on an input matrix of size $n \times n$.

(b) [**50 Points**] Consider the recursive divide-and-conquer algorithm given in Figure 3 for solving Recurrence 1. Explain how you would correctly parallelize this algorithm using ***spawn*** and ***sync*** . Compute its work, span and parallelism on an input matrix of size $n \times n$ (assuming n to be a power of 2). You are not allowed to use any extra space.

(c) [**50 Points**] If you are allowed to use extra space for storing intermediate values of submatrices of t in the parallel version of the algorithm given in Figure 3 (the same way we did for recursive matrix multiplication), does that lead to a span that is asymptotically shorter than the one you obtained in part (b)? Compute the work, span, parallelism and extra space usage of this new algorithm on an input matrix of size $n \times n$ (assuming n to be a power of 2).

(d) [**15 Points**] Is the parallel algorithm you obtained in part (a) scalable? If so, if you increase p by a factor of 2, by what factor you must increase n so that its efficiency remains constant? Answer the same questions for your parallel algorithms in parts (b) and (c).

(e) [**15 Points**] Analyze the cache complexity of the serial iterative algorithm given in Figure 2 on an input matrix of size $n \times n$.

(f) [**50 Points**] Analyze the cache complexity of the serial recursive algorithm given in Figure 3 on an input matrix of size $n \times n$ (assuming n to be a power of 2).

$A_{opt-tri}(\ X\ )$

(X points to an $m \times m$ square submatrix of the $n \times n$ input matrix t such that the top-left to bottom-right diagonal of X lies on the top-left to bottom-right diagonal of t. Both n and m are assumed to be powers of 2. Initial call to the function is $A_{opt-tri}(t)$.)

1. ***if*** X is not a 1×1 matrix ***then***
2. $\quad A_{opt-tri}(\ X_{11}\ )$
3. $\quad A_{opt-tri}(\ X_{22}\ )$
4. $\quad B_{opt-tri}(\ X_{12},\ X_{11},\ X_{22}\ )$

$B_{opt-tri}(\ X,\ U,\ V\ )$

(X, U and V are $m \times m$ disjoint submatrices of the $n \times n$ input matrix t such that X lies entirely above the top-left to bottom-right diagonal of t, and the top-left to bottom-right diagonals of U and V lie on that of t. Submatrix U lies to the left of X, and both lie on the same rows of t. Submatrix V lies below X, and both lie on the same columns of t. Both n and m are assumed to be powers of 2.)

1. ***if*** X is a 1×1 matrix ***then***
2. $\quad$ Let $X \equiv t[i,j],\ U \equiv t[i,k],\ V \equiv t[k,j]$
3. $\quad t[i,j] \leftarrow \min\{\ t[i,j],\ t[i,k] + t[k,j] + w(i,k,j)\ \}$

else

4. $\quad B_{opt-tri}(\ X_{21},\ U_{22},\ V_{11}\ )$
5. $\quad C_{opt-tri}(\ X_{11},\ U_{12},\ X_{21}\ )$
6. $\quad C_{opt-tri}(\ X_{22},\ X_{21},\ V_{12}\ )$
7. $\quad B_{opt-tri}(\ X_{11},\ U_{11},\ V_{11}\ )$
8. $\quad B_{opt-tri}(\ X_{22},\ U_{22},\ V_{22}\ )$
9. $\quad C_{opt-tri}(\ X_{12},\ U_{12},\ X_{22}\ )$
10. $\quad C_{opt-tri}(\ X_{12},\ X_{11},\ V_{12}\ )$
11. $\quad B_{opt-tri}(\ X_{12},\ U_{11},\ V_{22}\ )$

$C_{opt-tri}(\ X,\ U,\ V\ )$

(X, U and V are $m \times m$ disjoint submatrices of the $n \times n$ input matrix t each of which lies entirely above the top-left to bottom-right diagonal of t. The submatrix U lies to the left of submatrix X, and both lie on the same rows of t. The submatrix V lies below submatrix X, and both lie on the same columns of t. Both n and m are assumed to be powers of 2.)

1. ***if*** X is a 1×1 matrix ***then***
2. $\quad$ Let $X \equiv t[i,j],\ U \equiv t[i,k],\ V \equiv t[k,j]$
3. $\quad t[i,j] \leftarrow \min\{\ t[i,j],\ t[i,k] + t[k,j] + w(i,k,j)\ \}$

else

4. $\quad C_{opt-tri}(\ X_{11},\ U_{11},\ V_{11}\ )$
5. $\quad C_{opt-tri}(\ X_{12},\ U_{11},\ V_{12}\ )$
6. $\quad C_{opt-tri}(\ X_{21},\ U_{21},\ V_{11}\ )$
7. $\quad C_{opt-tri}(\ X_{22},\ U_{21},\ V_{12}\ )$
8. $\quad C_{opt-tri}(\ X_{11},\ U_{12},\ V_{21}\ )$
9. $\quad C_{opt-tri}(\ X_{12},\ U_{12},\ V_{22}\ )$
10. $\quad C_{opt-tri}(\ X_{21},\ U_{22},\ V_{21}\ )$
11. $\quad C_{opt-tri}(\ X_{22},\ U_{22},\ V_{22}\ )$

Figure 3: Recursive divide-and-conquer algorithm for the optimal polygon triangulation problem defined by recurrence 1. Initial call is $A_{opt-tri}(\ t\ )$ for an $n \times n$ input matrix $t[1:n, 1:n]$ with $t[i,j] = 0$ for $1 \leq i = j - 1 < n$ and $t[i,j] = \infty$ otherwise (i.e., $i \neq j - 1$). This function updates all $t[i,j]$ for $1 \leq i < j - 1 < n$ based on Recurrence 1. We assume that n is a power of 2.