PROBLEM 1. The objective of this problem is to prove that, with respect to the Theorem of Graham & Brent, a greedy scheduler achieves the stronger bound:

$$T_p \leq \frac{T_1 - T_\infty}{p} + T_\infty$$

Let $G = (V, E)$ be the DAG representing the instruction stream for a multithreaded program in the fork-join parallelism model. The sets V and E denote the vertices and edges of G respectively. Let T_1 and T_∞ be the work and span of the corresponding multithreaded program. We assume that G is connected. We also assume that G admits a single source (vertex with no predecessors) denoted by s and a single target (vertex with no successors) denoted by t. Recall that T_1 is the total number of elements of V and T_∞ is the maximum number of nodes on a path from s to t (counting s and t).

Let $S_0 = \{s\}$. For $i \geq 0$, we denote by S_{i+1} the set of the vertices ω satisfying the following two properties:

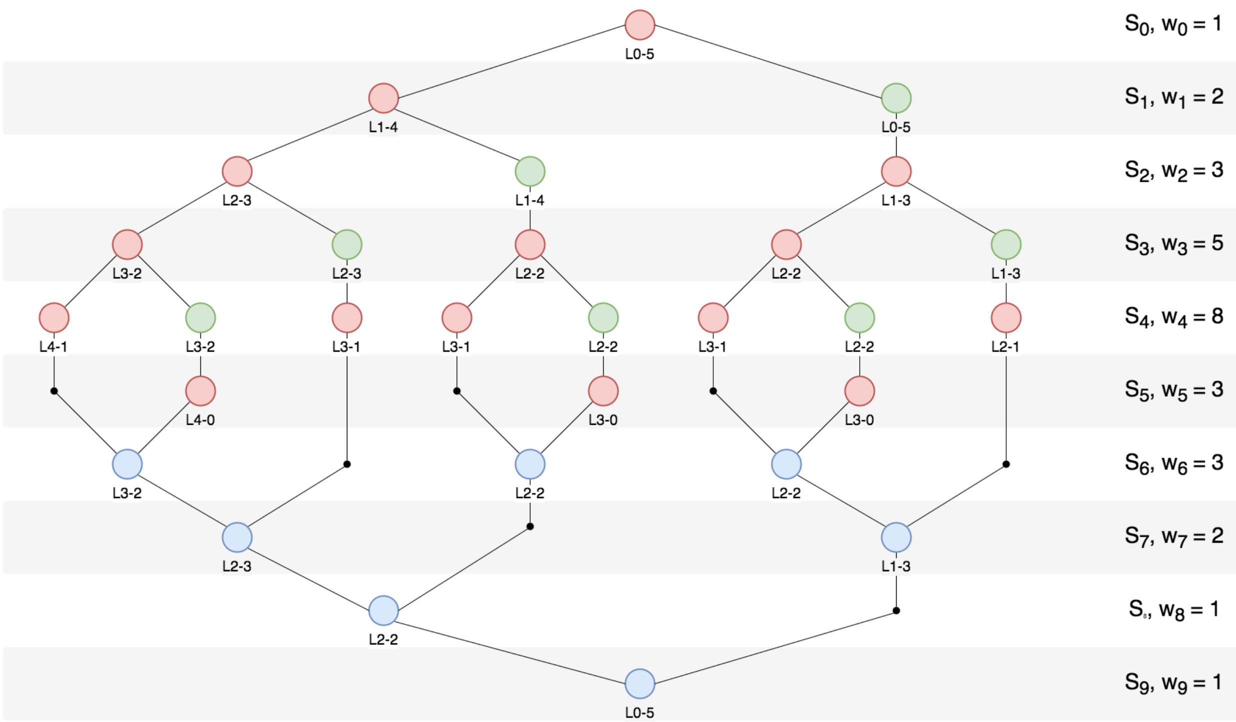
(i) all immediate predecessors of ω belong to $S_i \cup S_{i-1} \cup \ldots \cup S_0$
(ii) at least one immediate predecessor of ω belongs to S_i.

Therefore, the set S_i represents all the unites of work which can be done during the i-th parallel step (and not before that point) on infinitely many processors.

Let $p > 1$ be an integer. For all $i \geq 0$, we denote by ω_i the number of elements in S_i. Let l be the largest integer i such that $\omega_i \neq 0$. Observe that $S_0, S_1 \ldots, S_l$ from a partition of V. Finally, we define the following sequence of integers:

$$c_i = \begin{cases} 0 \text{ if } \omega_i \leq p \\ \left\lceil \frac{\omega_i}{p} \right\rceil - 1 \text{ if } \omega_i > p \end{cases}$$

Question 1. For the computation of the 5-th Fibonacci number (as studied in class) what are $S_0, S_1, S_2, \ldots$?



Question 2. Show that $l + 1 = T_\infty$ and $\omega_0 + \cdots + \omega_i = T_1$ both hold.

T_∞ is the number of nodes on the largest path between $\{s, t\}$. Furthermore, it is observed that $S_0, S_1 \dots, S_i$ form a partition of V by level. It is also observed that for any level S_{i+1} there exists at least one immediate predecessor of ω where $\omega \in S_{i+1}$ that belongs to S_i meaning that the longest path between $\{s, t\}$ must contain at least one vertex of G in every level. Therefore,

$$T_\infty = |S| = |\{S_0, S_1, \dots, S_i\}|$$

Additionally, l is the largest integer i such that $\omega_i \neq 0$. Consequently, l is the i where $S_i = \{t\}$. Hence,

$$|\{S_0, S_1, \dots, S_i\}| = l + 1$$

**The 1 is added as an accomidation for the i counter starting at 0.

$$T_\infty = l + 1$$

T_1 is the number of vertices in the DAG, G. Furthermore, it is observed that $S_0, S_1 \dots, S_i$ form a partition of V, therefore,

$$V = \{S_0, S_1, \dots, S_i\}$$

And,

$$|V| = |S_0| + |S_1| + \cdots + |S_i| = T_1$$

Additionally, since S_i, where $i \geq 0$, represents all the unites of work which can be done during the i-th parallel step on infinitely many processors. Then,

$$|S_0| + |S_1| + \cdots + |S_i| = \omega_0 + \omega_1 + \cdots + \omega_l$$

Hence,

$$\omega_0 + \cdots + \omega_i = T_1$$

Question 3. Show that we have:

$$c_0 + \cdots + c_l \leq \frac{T_1 - T_\infty}{p}$$

Given,

$$c_i = \begin{cases} 0 \text{ if } \omega_i \leq p \\ \left\lceil \frac{\omega_i}{p} \right\rceil - 1 \text{ if } \omega_i > p \end{cases}$$

Upon investigating the equation of c closely, it becomes clear that c calculates the number of cycles required to complete a step upon encountering an incomplete step in a greedy scheduler. Therefore,

$$c_0 + c_1 + \cdots + c_l = T_p - T_\infty$$

Hence,

$$c_0 + c_1 + \cdots + c_l \leq \frac{T_1 - T_\infty}{p}$$

$$T_p - T_\infty \leq \frac{T_1 - T_\infty}{p}$$

Question 4. Prove the desired inequality:

$$T_P \leq \frac{T_1 - T_\infty}{p} + T_\infty$$

With the help of the inequality deduced in the question 3.

$$T_p - T_\infty \leq \frac{T_1 - T_\infty}{p}$$

The inequality could be easily proven via rearrangement

$$T_p - T_\infty \leq \frac{T_1 - T_\infty}{p}$$

$$T_p \leq \frac{T_1 - T_\infty}{p} + T_\infty$$

Question 5. Application; Professor Brown takes some measurements of his (deterministic) multithreaded program, which is scheduled using a greedy scheduler and finds that $T_8 = 80$ seconds and $T_{64} = 20$ seconds. Give lower bound and an upper bound for Professor Brown's computation running time on p processors, for $1 \leq p \leq 100$? Using a plot is recommended.

Using the Theorem of Graham & Brent,

$$T_p \leq \frac{T_1}{p} + T_\infty$$

$$T_p - \frac{T_1}{p} \leq T_\infty$$

The value of p, $1 \leq p \leq 100$, allows us to transform the inequality to an equation because p is small. Therefore, the following system of equations can be used to determine the values of T_1 and T_∞,

$$80 - \frac{T_1}{8} = T_\infty$$

$$20 - \frac{T_1}{64} = T_\infty$$

$$80 - \frac{T_1}{8} = 20 - \frac{T_1}{64}$$

$$80 - \frac{8}{8} \times \frac{T_1}{8} = 20 - \frac{T_1}{64}$$

$$80 - \frac{8 \times T_1}{64} = 20 - \frac{T_1}{64}$$

$$80 - 20 = \frac{8 \times T_1}{64} - \frac{T_1}{64}$$

$$60 = \frac{7 \times T_1}{64}$$

$$\frac{60 \times 64}{7} = T_1$$

$$\frac{3840}{7} = T_1$$

Hence,

$$80 - \frac{\frac{3840}{7}}{8} = T_{\infty}$$

$$80 - \frac{3840}{7 \times 8} = T_{\infty}$$

$$\frac{80 \times 7}{7} - \frac{480}{7} = T_{\infty}$$

$$\frac{80 \times 7}{7} - \frac{480}{7} = T_{\infty}$$

$$\frac{560}{7} - \frac{480}{7} = T_{\infty}$$

$$\frac{80}{7} = T_{\infty}$$

Finally, to determine the upper and lower bounds of the program plot 100 and 1 for the lower bound and upper bound respectively

$$\min\left(\frac{T_1}{p}, T_{\infty}\right) \leq T_p \leq \frac{T_1 - T_{\infty}}{p} + T_{\infty}$$

$$\min\left(\frac{\frac{3840}{7}}{p}, \frac{80}{7}\right) \leq T_p \leq \frac{\frac{3840}{7} - \frac{80}{7}}{p} + \frac{80}{7}$$

$$\min\left(\frac{\frac{3840}{7}}{p}, \frac{80}{7}\right) \leq T_p \leq \frac{\frac{3760}{7}}{p} + \frac{80}{7}$$

$$\min\left(\frac{3840}{7 \times p}, \frac{80}{7}\right) \leq T_p \leq \frac{3760}{7 \times p} + \frac{80}{7}$$

$$\min\left(\frac{3840}{7 \times 100}, \frac{80}{7}\right) \leq T_p \leq \frac{3760}{7 \times 1} + \frac{80}{7}$$

$$\min\left(\frac{3840}{700}, \frac{80}{7}\right) \leq T_p \leq \frac{3760}{7} + \frac{80}{7}$$

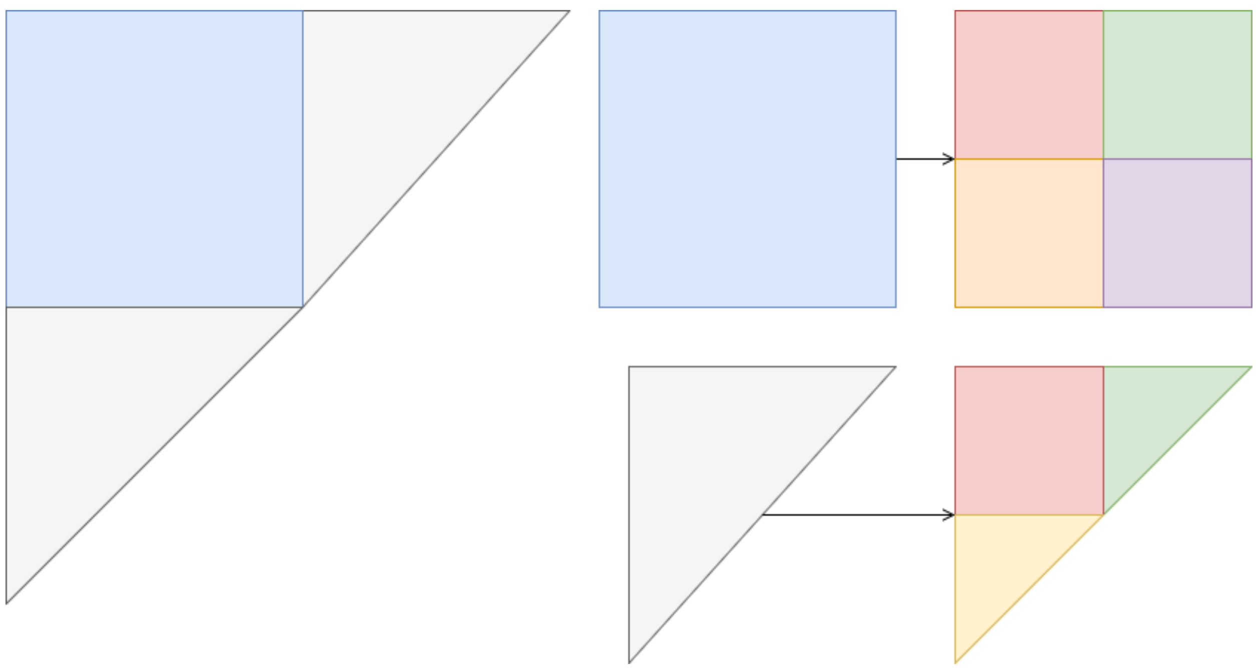
$$\frac{80}{7} \leq T_p \leq \frac{3840}{7}$$

PROBLEM 2. In the chapter *Analysis of Multithreaded Algorithms*, we studied the 2-way and 3-way construction of a tableau.

Question 1. Describe in plain words, how to construct a tableau in a k-way fashion, for an arbitrary integer $k \geq 2$, using the same stencil (the one of the Pascal triangle construction) as in the lectures.

The Pascal stencil tableau can be constructed by dividing and conquering the elements of the Pascal Triangle. The structure of the initial case of the triangle is a half k by k table split diagonally through the middle and including the diagonal cells, from the locations $[0, n]$ to $[n, 0]$, where n is the order of the table. Each of the k^2 elements is solved recursively until a base case is reached. The edge elements use the same structure as the base case while the non-edge elements use a square tableau structure.

The following is a example of a Pascal Triangle with $k = 2$. With one recursive iteration.



Question 2. Determine the work and the span for an input square array of order n.

Work:

$$T_1 = k^2 \times T\left(\frac{n}{k}\right) + \theta(1)$$

Using the master theorem,

$$a = k^2, b = k, c = 1$$

Therefore,

$$T_1 = \theta\left(n^{\log_k{}^{k^2}}\right)$$

$$T_1 = \theta\left(n^{2 \times \log_k{}^{k}}\right)$$

$$T_1 = \theta(n^{2 \times 1})$$

$$T_1 = \theta(n^2)$$

Span:

$$T_\infty = (2 \times n - 1) \times T\left(\frac{n}{k}\right) + \theta(1)$$

Using the master theorem,

$$a = (2 \times n - 1), b = k, c = 1$$

Therefore,

$$T_\infty = \theta\left(n^{\log_k{}^{2 \times n - 1}}\right)$$

$$T_\infty = \theta\left(n^{\log_k{}^{n}}\right)$$

Question 3. Realize a Julia or CilkPlus a multithreaded implementation off that algorithm. Collect running times (both serial and parallel) for increasing values of n (say consecutive powers of 2) and different values of k (at least 2 and 3).

Program could be found under src/pascal
Make command: make
Run command: ./pascal n k, where n = k^x

K	N		TIME
2	32	Serial	0m0.006s
		Parallel	0m0.013s
	256	Serial	0m0.009s
		Parallel	0m0.013s
	2048	Serial	0m0.050s
		Parallel	0m0.084s
	4096	Serial	0m0.174s
		Parallel	0m0.294s
	8192	Serial	0m0.681s
		Parallel	0m1.111s
	16384	Serial	0m2.646s
		Parallel	0m4.588s
	32768	Serial	0m11.206s
		Parallel	0m18.049s
3	9	Serial	0m0.008s
		Parallel	0m0.012s
	81	Serial	0m0.007s
		Parallel	0m0.011s
	729	Serial	0m0.012s
		Parallel	0m0.020s
	6561	Serial	0m0.278s
		Parallel	0m0.611s
8	64	Serial	0m0.007s
		Parallel	0m0.012s
	4096	Serial	0m0.118s
		Parallel	0m0.266s
	32768	Serial	0m7.468s
		Parallel	0m15.083s
32	1024	Serial	0m0.050s
		Parallel	0m0.036s
	32768	Serial	0m7.606s
		Parallel	0m14.105s

PROBLEM 3. Let G be a directed graph with n vertices. For simplicity we identify the vertex set to the set of positive integers $\{1, 2, \dots, n\}$. To each couple (i, j), with $1 \leq i, j \leq n$, we associate a weight $\omega_{i,j}$ such that:

(i) $\omega_{i,j}$ is a non-negative integer if and only if (i, j) is an arc in G,
(ii) $\omega_{i,j}$ is $+\infty$ if and only if (i, j) is not an arc in G.

We assume $\omega_{i,j} = 0$ for all $1 \leq i \leq n$. If $x_1, x_2, \dots, x_m$are $m \geq 2$ vertices of G such that $(x_1, x_2), (x_2, x_3), \dots, (x_{m-1}, x_m)$ are all arcs of G, we say that $p = (x_1, x_2, \dots, x_m)$ is a path in G from x_1 to x_m; moreover the weight of p is denoted by $\omega(p)$ and defined by

$$\omega(p) = \omega_{x_1,x_2} + \omega_{x_2,x_3} + \cdots + \omega_{x_{m-1},x_m}$$

For each couple (i, j) which is not an arc in G it Is natural to ask whether

(1) there is a path in G from i to j, and
(2) if such path exists, then compute the minimal weight of such a path.

This question is often referred as SAP for All-Pair Shortest Paths. The celebrated Floyd-Warshall algorithm solves ASAP by computing a matrix path as follows:

```
for k = 1 to n
    for i = 1 to n
        for j = 1 to n
             path[i][j] = min (path[i][j], path[i][k] + path[k][j]);
```

after initializing `path[i][j]` to $\omega_{i,j}$. For more details, please refer to the Wikipedia page of the Floyd-Warshall algorithm.

Question 1. Is it possible to turn the *Floyd-Warshall algorithm* into a parallel algorithm for the fork-join parallelism? If yes, analyze the work, the span and the parallelism of this algorithm.

Yes. The most inner loop can be parallelized using the divide and conquer method, using a cilk_for. In the case of this improvement. The work will stay at $\theta(n^3)$, while there will be an improvement in the span because of the introduction of the cilk_for, hence, $T_\infty = \theta(n^2 \times \log n)$

Question 2. Discuss the data locality of the above sequential Floyd-Warshall algorithm (not your parallel version of it). Doing a formal cache complexity analysis is not required.

The cache locality in the above algorithm is great when $2 \times n \leq$ cache size. That is because the processor will be able to cache both the i-th and k-th rows of the matrix on cold misses. However, the higher the n value the bigger the chance of introducing capacity misses.

One way to obtain a better algorithm for ASAP (in terms of parallelism and data locality) is to apply a divide and conquer approach. To this end we view $(\omega_{i,j})$ as an $n \times n$-matrix, denoted by W. We also view the targeted results, namely the values (`path[i][j]`) as an $n \times n$-matrix, denoted $\overline{W}$.

Before stating the divide and conquer formulation, we introduce a few notations. Let X, Y be square matrics (of the same order) whose entries are non-negative or $+\infty$. W

- XY the min-plus product of X by Y (obtain from the usual matrix multiplication by replace $+$ (resp. $\times$) by min (resp. $+$))
- $X \vee Y = \min(X, Y)$ the element-wise minimum of the tow matrices X and Y.

We are ready to state the divide and conquer formulation. If we decompose W into four $n/2 \times n/2$-blocks, namely

$$W = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

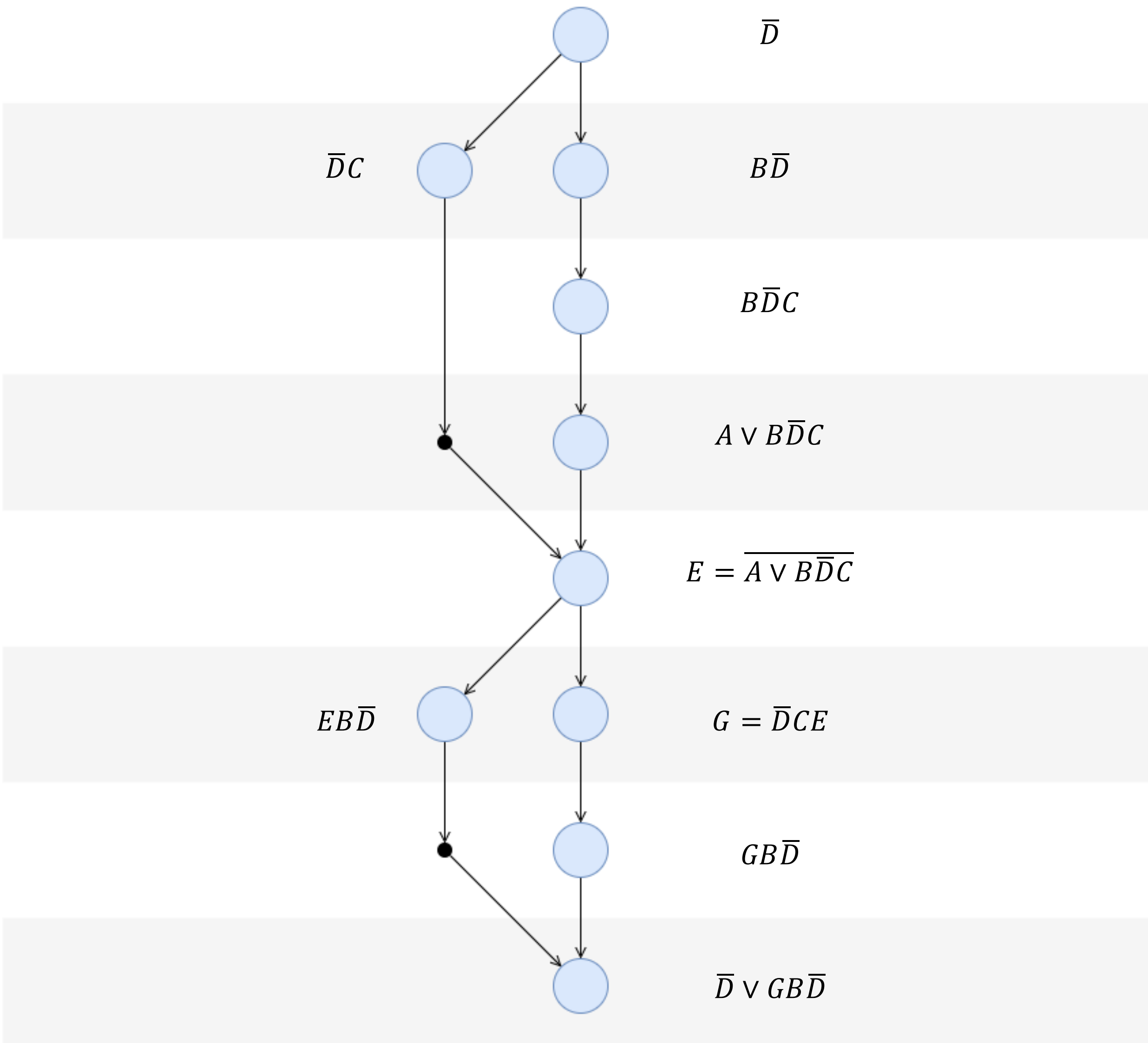
then

$$\overline{W} = \begin{pmatrix} E & EB\overline{D} \\ G & \overline{D} \vee GB\overline{D} \end{pmatrix}$$

where we have $E = \overline{A \vee B\overline{D}C}$ and $G = \overline{D}CE$. We shall admit that these formulas are correct (even though proving them is not that hard).

Question 3. Propose an algorithm for computing $\overline{W}$ in the fork-join parallelism model.

The following DAG proposes a fork-join parallelism model for computing $\overline{W}$. The order at which the variables are computed is chosen to maximize the parallelism as it computes the variables that are essential to the computation of other variables down the pipeline. Every call to the bar function is a blocking call that is computed on the main thread.



Question 4. Analyze the work, the span and parallelism of your algorithm.

Work:

$$T_1 = 2 \times T\left(\frac{n}{2}\right) + \text{Multi}(n) + \text{Min}(n)$$

Trivially,

$$T_1 = 2 \times T\left(\frac{n}{2}\right) + n^3 + n^2$$

Using the master theorem,

$$a = 2, b = 2, c = 3$$

$$\log_b{}^a < c$$

$$\log_2{}^2 < 3$$

$$1 < 3$$

Therefore,

$$T_1 = \theta(n^3)$$

Span:

$$T_\infty = 2 \times T\left(\frac{n}{2}\right) + \text{Multi}(n) + \text{Min}(n)$$

Trivially,

$$T_\infty = 2 \times T\left(\frac{n}{2}\right) + n^2 \times \log n + 1$$

Using the master theorem,

$$a = 2, b = 2, c = 2$$

$$\log_b{}^a < c$$

$$\log_2{}^2 < 3$$

$$1 < 3$$

Therefore,

$$T_\infty = \theta(n^2 \times \log n)$$

There exist alternative algorithms for the ASAP problem which rely on the min-plus multiplication. A simple one is based on the observation that $\overline{W} = W^n$ (and in fact W^{n-1}) where is the *n*-th power W is computed for min-plus multiplication using repeated squaring.

Question 5. Propose such an algorithm. You are welcome to use the literature of simply to use the one suggested above.

The algorithm consists of recursively creating a tree where every leaf value is A. The algorithm then collapses all the leafs and internal nodes using the min-plus function until the root is reached.

Question 6. Analyze the work, the span and parallelism of this third algorithm.

Work:

$$T_1 = 2 \times T\left(\frac{n}{2}\right) + \mathrm{Multi}(n)$$

Trivially,

$$T_1 = 2 \times T\left(\frac{n}{2}\right) + n^3$$

Using the master theorem,

$$a = 2, b = 2, c = 3$$

$$\log_b{}^a < c$$

$$\log_2{}^2 < 3$$

$$1 < 3$$

Therefore,

$$T_1 = \theta(n^3)$$

Span:

$$T_\infty = 2 \times T\left(\frac{n}{2}\right) + \mathrm{Multi}(n)$$

Trivially,

$$T_\infty = 2 \times T\left(\frac{n}{2}\right) + n^2 \times \log n$$

Using the master theorem,

$$a = 2, b = 2, c = 2$$

$${\log_b}^a < c$$

$${\log_2}^2 < 3$$

$$1 < 3$$

Therefore,

$$T_\infty = \theta(n^2 \times \log n)$$

Question 7. Realize a Julia or CilkPlus a multithreaded implementation of that algorithm.

Program could be found under src/ASAP
Make command: make
Run command: ./ASAP < test.txt make