4. Describe the conditions under which a parallel algorithm would obtain near perfect speedups.

考虑 Amdahl's Law：

给定并行处理器数 N，串行占比 f（程序中无法并行的部分），理想并行占比 $1-f$，则加速比

$$S(N)=\frac{1}{f+\frac{1-f}{N}}.$$

所以近乎完美加速意味着 $S(N)\approx N$，即 $f \ll 1/N$。

$$W(n) = \sqrt{n}W(\sqrt{n}) + n^2$$

Asymptotic analysis of $T(n) = \sqrt{n}\,T\left(\sqrt{n}\right) + n^2$

Let $n = 2^m$, $T(n)$ derives to

$$T(2^m) = 2^{m/2}\,T\left(2^{m/2}\right) + 2^{2\,m}$$

Let $S(m) = T(2^m)$, $S(m/2) = T\left(2^{m/2}\right)$, substitute all T by S

$$S(m) = 2^{m/2}\,S(m/2) + 2^{2\,m}$$

Let $t = \log_2 m$, thus $t-1$ derives to $\log_2(m/2)$, and m equals to 2^t. Substitute all m by t derives to

$$S(2^t) = 2^{2^{t-1}}\,S\left(2^{t-1}\right) + 2^{2^{t+1}}$$

Let $U(t) = S(2^t)$, thus $U(t-1) = S\left(2^{t-1}\right)$, we have

$$U(t) = 2^{2^{t-1}}\,U(t-1) + 2^{2^{t+1}}$$

and an arbitrary boundary condition $U(0) = 0$

Solve this recurrence equation results in

$$U(t) = 2^{2^t-2}\sum_{i=0}^{t-1} 2^{2+2^{i+1}} + 2^{2^t-1}$$

Substitute U by S yields

$$S(2^t) = 2^{2^t-2}\sum_{i=0}^{\log_2(2^t)-1} 2^{2+2^{i+1}} + 2^{2^t-1} \implies S(m) = 2^{m-2}\sum_{i=0}^{\log_2(m)\,-1} 2^{2+2^{i+1}} + 2^{m-1}$$

Substitute S by T yields

$$T(2^m) = \frac{2^m}{4}\sum_{i=0}^{\log_2(\log_2 2^m)\,-1} 2^{2+2^{i+1}} + \frac{2^m}{2}$$

Substitute 2^m by n yields

$$T(n) = \frac{n}{4}\sum_{i=0}^{\log_2(\log_2 n)\,-1} 2^{2+2^{i+1}} + \frac{n}{2}$$

and there exists a positive constant c such that

$$T(n) = \frac{n}{4}\sum_{i=0}^{\log_2(\log_2 n)\,-1} 2^{2+2^{i+1}} + \frac{n}{2} \le c\times n\times \sum_{i=0}^{\log_2(\log_2 n)\,-1} 2^{2^{i+1}}$$

Now let $a_i = 2^{2^{i+1}}$, we have

$$T(n) \le c\,n \sum_{i=0}^{\log_2(\log_2 n)\,-1} a_i.$$

It is easy to find that a_i is a monotone increasing series, and thus we consider a_i in a descending order (*i.e.*, from larger terms to smaller terms)

$$\text{When } i = \log_2(\log_2 n) - 1,\ a_i = 2^{2^{\log_2 \log_2 n}} = 2^{\log_2 n} = n$$

$$\text{When } i = \log_2(\log_2 n) - 1,\ a_i = 2^{2^{\log_2\left(\frac{\log_2 n}{2}\right)}} = 2^{\frac{\log_2 n}{2}} = n^{1/2}$$

$$\text{When } i = \log_2(\log_2 n) - 2,\ a_i = n^{1/4}$$

...

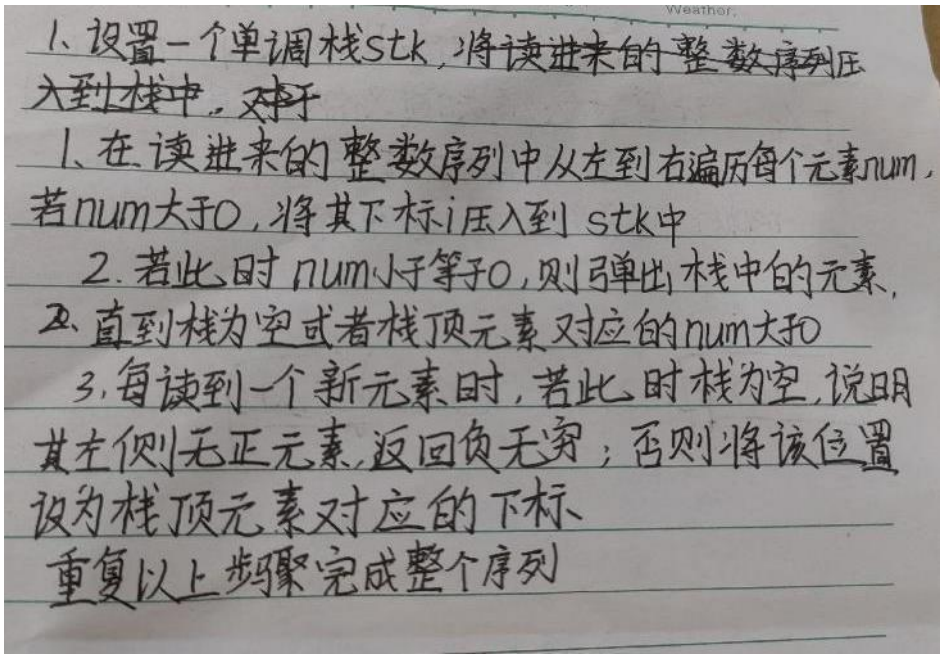
Thus, for $j = 0, 1, 2, \ldots$, we got

$$a_{i-j} = n^{1/2^j}$$

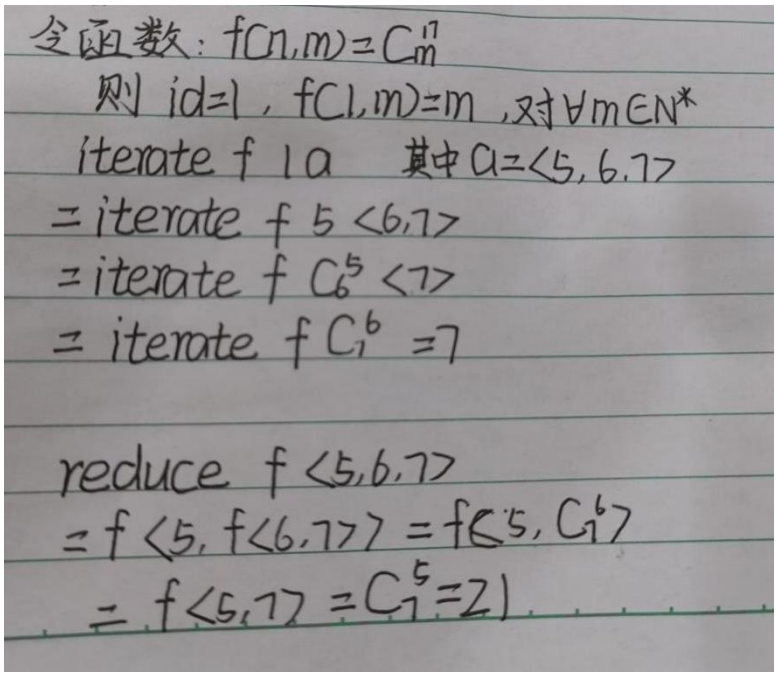
and

$$S = c\,n \sum_{j=0}^{\log_2(\log_2 n)} n^{1/2^j} = c\,n\left(n^1 + n^{1/2} + n^{1/4} + n^{1/8} + \ldots\right) = O(n^2)$$

1. **Design an algorithm that, for each element in a sequence of integers, finds the rightmost positive number to its left. If there is no positive element to the left of an element, the algorithm returns $-\infty$ for that element. For example, given the sequence $\langle 1,0,-1,2,3,0,-5,7\rangle$ the algorithm would return $\langle -\infty,1,1,1,2,3,3,3\rangle$.**



2. **Give an example function f, a left identity x, and an input sequence a such that** iterate $f\ x\ a$ **and** reduce $f\ x\ a$ **return different results.**



4. Analyze the cost of the algorithm for generating 2D points given above.

Points 2D n = flatten(tabulate(lambda x tabulate(lambda y (x,y+1))(n-1))n)
由于 tabulate(lambda y (x,y+1))(n-1) W = O(n) S= O(1) 记为 g(x)
tabulate(lambda x,g(x))n W= $O(n^2)$ S = O(1)
得到的序列元素总个数为 n*(n-1),flatten 前序列个数为 n
故 flatten 的 W = $O(n^2)$ S = O(lg n)
故算法的 Work = $O(n^2)+O(n^2) = O(n^2)$ Span = O(1)+O(lg n) = O(lg n)

5. Present an algorithm that generates all contiguous subsequences of a given sequence.

flatten(tabulate(lambda i,tabulate(lambda j,a[i…i+j])(|a|-i-1)|a|)

6. Analyze the cost of your algorithm for Exercise All contiguous subsequences.

设 g(i) = tabulate(lambda j,a[i…i+j](|a|-i-1)
则其 work = O(|a|-i+1) Span = O(1)
推出 tabulate(lambda I g(i)|a|
work = $O(|a|^2)$ span = O(1)
又由 flatten 前序列个数为|a|,flatten 后序列个数为|a|(|a|-1)
故算法总的 Work = $O(|a|^2)+ O(|a|^2)= O(|a|^2)$ Span = O(1)+O(lg|a|) = O(lg|a|)

8. To generate all composite numbers between 2 and n, prove that it suffices to consider all $i \in \mathbb{N}$ $i \le i \le \sqrt{n}$ and all of i's multiples up to n/i.

参考质数筛法，定理：

如果 n 有一个大于$\sqrt{n}$的 factor，那么它必定有一个小于或等于$\sqrt{n}$的 factor，因为两个大于$\sqrt{n}$的因子相乘会超过 n。因此只需检查到$\sqrt{n}$。

严格证明：

首先证明：只需考虑[1，$\sqrt{n}$]的数 i：

假设在$\sqrt{n}$之后继续考虑数 j，且 j 是 n 的因子，则有 n/j∈[1，$\sqrt{n}$]，已考虑到，矛盾，故条件得证；

继续证明：所有 i 的倍数只需考虑到 n/i：

假设在 n/i 之后继续考虑数 j，且 j 是 i 的倍数、n 的因子，则有 n/j∈[1，i)，已考虑到，矛盾，故条件得证；

综上：题设条件得证。

Example 1.5 (Brute-Force Overlaps). Given two strings a and b, let's define the (maximum) ***overlap*** between a and b as the largest suffix of a that is also a prefix of b. For example, the overlap between "15210" and "2105" is "210", and the overlap between "15210" and "1021" is "10".

We can find the overlap between a and b by using the brute force technique. We first note that the solution space consists of the suffixes of a that are also prefixes of b. To apply brute force, we consider each possible suffix of a and check if it is a prefix of b. We then select the longest such suffix of a that is also a prefix of b.

The work of this algorithm is $|a| \cdot |b|$, i.e., the product of the lengths of the strings. Because we can try all positions in parallel, the span is determined by the span of checking that a particular prefix is also a suffix, which is $O(\lg |b|)$. Selecting the maximum requires $O(\lg |a|)$ span. Thus the total span is $O(\lg (|a| + |b|))$.

Exercise 1.1. The analysis given in the example above is not "tight" in the sense that there is a bound that is asymptotically dominated by $O(|a| \cdot |b|)$. Can you improve on the bound?

可以 KMP 算法改进 overlap 的计算。

使用 KMP 算法来改进 overlap 的计算可以将复杂度进一步减少。KMP 算法通过利用已经匹配的前缀信息来确定跳过的位置，从而避免不必要的比较。

以下是使用 KMP 算法改进 overlap 计算的步骤：

1. 构建 KMP 算法的辅助数组，也称为最长公共前后缀（Longest Proper Prefix which is also Suffix，简称 LPS）数组。该数组用于存储每个位置之前的最长 overlap 长度。

2. 对于字符串 a，计算其对应的 LPS 数组。

3. 初始化两个指针：i 指向字符串 a 的末尾，j 指向字符串 b 的开头。

4. 在循环中，比较 a[i]和 b[j]。如果它们相等，表示找到了一个可能的 overlap。将 i 和 j 都向前移动一位。如果 i 等于 0，则说明 a 的整个字符串都是 overlap 的，可以返回字符串 a 作为 overlap。

5. 如果 a[i]和 b[j]不相等，则根据 LPS 数组确定下一个比较的位置。将 i 移动到 LPS[i-1]的位置。注意，此处的 LPS 数组与字符串 a 相关。

6. 重复步骤 4 和步骤 5，直到遍历完字符串 b 或找到最大 overlap。

使用 KMP 算法改进后，我们不需要每个位置都进行比较，而是利用 LPS 数组来确定下一个比较的位置。这样可以避免不必要的比较，从而将复杂度降低到 O(|a|+|b|)。

因此，使用 KMP 算法改进后的界限是 O(|a|+|b|)，而不是 O(|a|·|b|)。这是一个更紧密的界限，与输入字符串的总长度成线性关系。

- **Describe the changes to the algorithms Algorithm MCS: Brute Force and Algorithm MCSS: Brute Force to implement the strengthening described above. How does strengthening impact the work and span costs?**

MCSS 算法描述如下：

```
MCSSBF a =
    let
        (i, j) = MCSBF a
        sum = reduce' + '0 a[i···j]
    in
        sum
    end.
```

我们可以看出，MCSS 算法的改变是在得到了 MCS 算法结果的基础上进行了 reduce 求和运算，实现 reduce 函数需要的工作量为 O(n)，故 MCSS 算法的 Work 为 O(n^3+n)即 O(n^3)；在 MCSS 中，我们同样以并行的方式生成所有对并用 reduce 求和，故 MCSS 的 span 与 MCS 相同，为 O(lgn)。

• Prove the MCSSE Extension lemma.

我们使用归纳法来证明这个引理：

1. 当 i = 1 时，最大连续子序列和为 ⟨a[1]⟩。这是基础情况，成立。

2. 假设对于某个 i = k 成立，即已知以位置 k 结尾的最大连续子序列和为 Mk。现在我们考虑归纳步骤，即要证明当 i = k + 1 时，该引理也成立。

根据归纳假设，我们知道以位置 k 结尾的最大连续子序列和为 Mk。现在我们需要证明以位置 k + 1 结尾的最大连续子序列和为 Mk+1。

根据题目要求，我们需要根据当前元素 a[k+1] 与 Mk 的和的情况来确定 Mk+1 的值。

1. 如果 Mk 的和 Sum<Mk> 小于等于 0，那么它对序列的最大值贡献为负。因此，我们选择以当前元素 a[k+1] 作为最大连续子序列，即 Mk+1 = ⟨a[k+1]⟩。

2. 如果 Mk 的和 Sum<Mk> 大于 0，那么它对序列的最大值贡献为正。因此，我们选择将当前元素 a[k+1] 添加到 Mk 中形成 Mk+1，即 Mk+1 = Mk ++ ⟨a[k+1]⟩。

通过归纳推理，我们证明了对于任意的 i = k + 1，该引理也成立。

综上所述，根据归纳法，我们可以得出结论：对于任意位置 i，以位置 i 结尾的最大连续子序列和可以根据前一个位置的最大连续子序列和来计算，具体取决于前一个位置的和是否大于零。

• Prove that the pivot tree has O(lgn) height, and is therefore balanced, with high probability.

证明pivot tree有很高的概率是O(lgn)高的，因而是平衡的。

证明思路：证明问题大小的数学期望满足递推关系，进而得出Span(树高)

定义shrink为选取一次pivot对问题进行划分后，问题规模缩小的比例 (之后会用到)

由pivot的选取是任意的，每个位置被选取的概率都为$\frac{1}{n}$

$$
\begin{aligned}
E(shrink) &= \frac{E(\text{子问题规模})}{n} \\
&= \frac{1}{n} * \sum_{k=1}^{n-1} \max(k, n-k-1) * \frac{1}{n} \\
&\leq \frac{3}{4}
\end{aligned}
$$

可以参考这张图

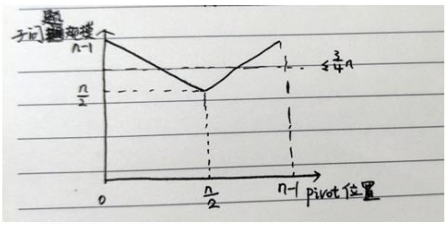


图 1: 参考图

下证：经过d+1次选取pivot进行划分，E(子问题规模)$\leq (\frac{3}{4})^d n$

1.经过1次选取，E(子问题规模)$\leq \frac{3}{4}n$，上面已经证明过了

2.假设经过d次选取E(子问题规模)$\leq (\frac{3}{4})^{d-1}n$成立，证明d+1次成立

定义y为问题的实际规模（y不是一个随机变量）， z是选取的pivot ， f(y，z) 是一个函数其值为问题规模缩小的比例

$$
\begin{aligned}
E\left[Y_{d+1}\right][1] &= \sum_{y,z} y f(y,z) \mathbf{P}_{Y_d,Z_d}(y,z) \\
&= \sum_{y} \sum_{z} y f(y,z) \mathbf{P}_{Y_d}(y) \mathbf{P}_{Z_d|Y_d}(z \mid y) \\
&= \sum_{y} y \mathbf{P}_{Y_d}(y) \sum_{z} f(y,z) \mathbf{P}_{Z_d|Y_d}(z \mid y) \\
&= \sum_{y} y \mathbf{P}_{Y_d}(y) \uparrow \text{(当问题规模为y时,任意选取pivot,问题规模缩小比例的期望)} \\
&\leq \sum_{y} y \mathbf{P}_{Y_d}(y) \mathbf{E}[shrink] \\
&\leq \sum_{y} y \mathbf{P}_{Y_d}(y) \frac{3}{4} \\
&\leq \uparrow \text{(y的期望)} \frac{3}{4} \\
&\leq \frac{3}{4} \mathbf{E}\left[Y_d\right].
\end{aligned}
$$

故d+1次成立

我们这里计算P(h >10lgn+1)的概率，如果概率很小那么我们就可以认为树高大概率是O(lgn)的

当选取若干次次pivot后，仍需选择pivot的条件是问题规模仍大于1

故P(h >10lgn+1) = P(X >1) X为问题规模的随机变量

由markov's law

$P(X > 1) \leq \frac{E(X)}{1} = \frac{n(\frac{3}{4})^{10lgn}}{1} \approx n^{-3.15}$

故当选取10lgn + 1 次pivot进行划分之后，仍需进行划分你的概率小于$\frac{1}{n^3}$,概率很小

故h有很高的概率是O(lgn)，因而是平衡的

最后简单证明一下markov's law

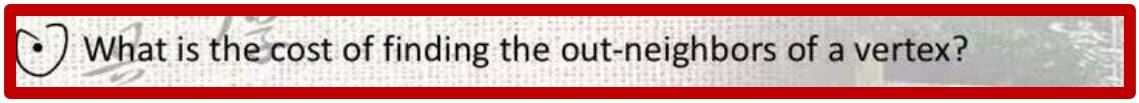
假设X为连续分布且大于零的随机变量

$P(X > k) = \int_k^\infty P(x = j)dj$

$$
\begin{aligned}
E(X) &= \int_0^\infty jP(x = j)dj \\
&= \int_0^k jP(x = j)dj + \int_k^\infty jP(x = j)dj \\
&\geq \int_k^\infty jP(x = j)dj \\
&\geq k \int_k^\infty P(x = j)dj \\
&= k(X > k)
\end{aligned}
$$

故$P(X > k) \leq \frac{E(X)}{k}$

对于离散分布且大于零的随机变量也同样适用



- 题干：使用邻接表存图时，查找一个顶点所有邻居的时间开销有多大？
- 解：

  只需要查询邻接表的第i项，得到的 `seq` 就是这个顶点的全部邻居。

  所以

$$W = \mathcal{O}(1)$$

$$S = \mathcal{O}(1)$$

- Present a multi-source version of the single-source graph search algorithm.

参考 Floyd 多源最短路算法

- The computation of the new F is not quite defined in the same way as in the generic graph search. Prove that the technique used here is consistent with that of the generic algorithm.

一个是 DFS 头递归，一个是尾递归，区别是是否需要在 child 中访问 parent。如果设置一个 visit 标志数组，那么这个数组中的每个值在两个算法中都只会被设置一次，因此它们具有相同的渐进复杂度（O(N+M)）

Pseudo Code 12.38 (Generalized directed DFS).

```
1  function directedDFS(G, Σ0, s) =
2    let
3      function DFS p ((X, Σ), v) =
4        if (v ∈ X) then
5          (X, touch(Σ, v, p))
6        else
7          let
8            Σ' = enter(Σ, v, p)
9            X' = X ∪ {v}
10           (X'', Σ'') = iter (DFS v) (X', Σ') (N_G^+(v))
11           Σ''' = exit(Σ', Σ'', v, p)
12         in (X'', Σ''') end
13   in
14     DFS s ((∅, Σ0), s)
15   end
```

- Prove that the algorithm is correct, i.e., visits all reachable vertices from the source in the order of their distances to the source

证明：假设：对所有距离源点距离小于等于 i，i>1 的顶点都被按距离遍历，则距离源点距离等于 i+1 的点也会按距离遍历。

若不成立，则存在的点，在错误的顺序被遍历，我们设其中一个点 a。

1. 提早被遍历，即有距离小于 i+1 的点在 a 之后才被遍历，这与假设中所有距离源点距离小于等于 i 的顶点都被按距离遍历这一前提矛盾；

2. 延后被遍历，即有距离大于 i+1 的点（不妨设为 b）在 a 之前被遍历. 而根据假设，所有距离等于 i 的点都被遍历，此时根据函数 explore 中的

$F = F \cup \left\{(v, j+1) : v \in N^{+}_{G(u)} \mid v \notin X \wedge (v, \_) \notin F\right\}$,

a 被加入集合 F 中，b 被遍历时必也在集合 F 中，而 b 到源点的距离大于 a，根据

$(u, j) = \mathsf{argmin}_{(v,k)\in F}(k)$,

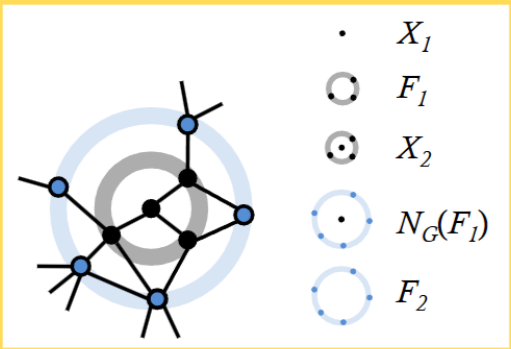
a 会先被加入 X 中, 并早于 b 被遍历，矛盾；

3. 不被遍历，显然所有距离为 i 的点被遍历后，a 会被加入集合 F 中，而加入 F 中的点会被遍历，矛盾。综上，假设成立对于 $i \leq 1$ 时，源点 s 已被加入 F，首先遍历源点，此时所有距离源点为 1 的点也会加入 F，然后被遍历，也是正确的。

综上，对于所有距离 $= i \geq 0$ 的点，都正确。

In general, from which frontiers could the vertices in $N_G(F_i)$ come when the graph is undirected? What if the graph is directed?

Example 11.11. *BFS on an undirected graph with the source vertex at the center.*



Exercise 11.12. *In general, from which frontiers could the vertices in $N_G(F_i)$ come when the graph is undirected? What if the graph is directed?*

有向图会沿着方向走，无向图可能会返回（例如 Example 11.11 的 $N_G(F_1)$）。