Final Exam
CSE 202 Algorithms and Data Structures, Winter, 2014

Hao Zhuang
PID: A53029734
Email: zhuangh@ucsd.edu

March 20, 2014
1 K-Select (Problem 1)

Background
The algorithm for this problem uses an sub-routine $DS(A, i)$, which is the deterministic select algorithm, to select the $i$-st largest element in an unsorted array $A$, which was discussed in the class. It is shown as follow:

Deterministic Select Algorithm $DS(A, i)$

- Step 1: Group and divide the array into $n/5$ groups with size 5, and find the median of each groups. Then put them into $medians$.
- Step 2: Recursive use $DS$ to find the $p$, which is the median of the medians: $p \leftarrow DS(medians, |medians|/2)$;
- Step 3:
  - Step 3.1: Partition via the pivot $p$ to screen the elements no less than $p$ in $A$ into $B$, $B \leftarrow \{A[k] \mid A[k] \geq p\}$;
  - Step 3.2: Partition via the pivot $p$ to screen the elements less than $p$ in $A$ into $S$, $S \leftarrow \{A[k] \mid A[k] < p\}$;
- Step 4:
  - If $i < |B|$ Return $DS(B, i)$;
  - If $i = |B|$ Return $p$;
  - If $i > |B|$ Return $DS(S, i - |B|)$;

The complexity is $O(n)$ to find the $i$-st largest element in an unsorted array $A$. We use it as a sub-routine in this problem.

Note: After $DS(A, i)$, besides the $i$-st largest element $p$ in $A$, we also have $B$ which $B[j] \geq p \ (j = 1, \cdots, |B|)$, and $S[j] < p \ (j = 1, \cdots, |S|)$. We leverage $B$ and $S$ to reduce the problem size for later.

1.1 Algorithm and Correctness (4 points)

Algorithm Description for k-Select Problem

- Step 1: Run Algorithm $k-Select(A, I)$ as follows,
  - Step 1.1: Select the $I_{mid}$ with the middle index in $I$, terminates itself when $I$ is empty or $I_{mid}$ is already marked as found in $I$.
  - Step 1.2: Run $DS(A, I_{mid})$ to select the $I_{mid}$-st largest element $p$ in $A$.
    * Step 1.2.1: The $p$ is obtained, and $I_{mid}$ is marked as found in $I$. 

2 of 9
Correctness Proof

Given the correct DS function, it can be used to find the $i$-st largest element $p$ in an unsorted array. After running DS, we also have $B[j] \geq p$, where $j = 1, \cdots, |B|$, and $S[j] < p$ ($j = 1, \cdots, |S|$).

$k - \text{Select}(A, I)$ is the function to find the middle element of $I$, which is $I_{mid}$. First, it uses DS to find the $I_{mid}$-st largest elements $p$ in $A$, and also obtains two sub-arrays $B$ and $S$, where $B[j] \geq p$, $(j = 1, \cdots, |B|)$, and $S[j] < p$ ($j = 1, \cdots, |S|$). Then, the problem is divided into solving two smaller problems $k - \text{Select}(B, I_B)$ and $k - \text{Select}(S, I_S')$, where $I = I_B + I_{mid} + I_S$. We need to prove that by only running the $k - \text{Select}(B, I_B)$ and $k - \text{Select}(S, I_S')$ with smaller problem sizes are correct.

$B$ has all the elements smaller than $I_{mid}$-st largest element $p$ in $A$. Besides, $I$ is an increasing array, so that the elements in $I$ with the index smaller than $mid$ all have smaller values than $I_{mid}$. Therefore, $k - \text{Select}(B, I_B)$ can be used to find the $I_B[j]$-st largest elements inside $B$.

On the other side, using the same analysis, $k - \text{Select}(S, I_S')$ can be used to find the $I_S'[j]$-st largest elements inside $S$, which is the sub-array by eliminating the first $|B|$-st largest elements from $A$. Therefore, finding the $I_S'[j] = (I_S[j] - |B|)$-st largest elements inside $S$ is equivalent to finding the $I_S[j]$-st largest element inside $A$. So the left problem is solved by running $k - \text{Select}(S, I_B)$ and $k - \text{Select}(S, I_S')$ are correct and enough. By induction, the following subproblems of the subproblems are still correct by the same reasoning.

1.2 Efficiency and Time Analysis (16 points)

The time complexity of this algorithm is $O(n\log(k))$.

$k - \text{Select}(A, I)$ is a divide-and-conquer method. At the first level, the subroutine $DS$ takes $O(n)$ to find the $I_{mid}$-st largest element $p$, and screen the elements at $A$ into $B$ and $C$, where $B[j] \geq p$, $(j = 1, \cdots, |B|)$, and $S[j] < p$ ($j = 1, \cdots, |S|$). The left problem can be solved by running $k - \text{Select}(B, I_B)$ and $k - \text{Select}(S, I_S')$ at the next level.

At 2nd level, $k - \text{Select}(B, I_B)$ it costs $O(|B|)$ to find $I_{B,mid}$-st largest element in $B$, and leads to another two smaller problems. For $k - \text{Select}(S, I_S')$, it costs
$O(|S|)$ to find $I_{g,\text{mid}}^{s,\text{mid}}$-st largest element in $S$, and also leads to another two smaller problems. The total complexity in this level is $O(n)$. It leads to four smaller problems in the next level with the same total problem size $n$. Therefore, at each level, the total complexity is $O(n)$ and we have $\log(n)$-level to run DS until all $I_i$ are found. The time complexity is

$$T(k) \leq Cn + 2T(k/2)$$

The overall complexity is $O(n\log k)$. 
2 Sum of Independent, Identically Distributed Random Variables (Problem 3)

Background

$X$ is an independent random variable with distribution probabilities $X_0, X_1, \cdots, X_{k-1}$, where $X_i = \text{Prob}[X = i]$. Therefore, the joint probability distribution $S_n$, which is the sum of $n$ copies of $X$, can be calculated via convolution\(^1\). For example, given two independent random variables $Y, Z$. $Y$ has a value range from 0 to $y - 1$ and $Z$ has a value range from 0 to $z - 1$. The joint distributions of them is

$c = Y \ast Z = \{c_0, c_1, \cdots, c_{(y-1)+(z-1)}\}$

where,

$$c_j = \sum_{(a,b):a+b=j,0\leq a<y,0\leq b<z} Y_a Z_b$$

We can extend our method to calculate the $S_n$ by convolution of a pair $S_a$ and $S_b$, where $a + b = n$, because $X$ is an independent random variable.

$$S_n = S_a \ast S_b = \{S_{n,0}, S_{n,1}, \cdots, S_{n,n(k-1)}\}$$

where $S_{n,j} = \text{Prob}[S_n = j]$ and

$$S_{n,j} = \sum_{(a,b):a+b=n,0\leq j\leq n(k-1),0\leq i\leq j} S_{a,j} S_{b,j-i}$$

We know the convolution $c = Y \ast Z$ can be computed via Fast Fourier Transform (FFT)\(^2\) method in time complexity $O(\log y)$ when $y > z$. Therefore, it costs $O(a(k-1)\log(a(k-1)))$ to compute $S_n = S_a \ast S_b$ when $a > b$.

---


2.1 Polynomial Time Algorithm and Correctness (8 points)

Algorithm $S(n)$ to compute $S_n$.

- Step 1: If $n = 1$, then return $S_1 = \{X_0, X_1, \cdots, X_{k-1}\}$;
- Step 2: If $n = 2$, then return $S_2 = S_1 \ast S_1$ via FFT;
- Step 3: If $n > 2$ and $S_n$ has been computed, return $S_n$;
  otherwise, $S_n$ has not been computed, then
  - Step 3.1: If $n$ is an even number ($n = 2u$, $u$ is a integer), then return $S_n = S(u) \ast S(u)$ via FFT;
  - Step 3.2: If $n$ is an odd number ($n = 2u + 1$, $u$ is a integer), then return $S_n = S(u) \ast S(u + 1)$ via FFT;

Correctness

As mentioned in this problem’s background, when $X$ is an independent random variable, $S_n$ is computed by convolution of a pair $S_a$ and $S_b$, where $a + b = n$.

$$S_n = S_a \ast S_b = \{S_{n,0}, S_{n,1}, \cdots, S_{n,n(k-1)}\}$$

where $S_{n,j} = \text{Prob}[S_n = j]$ and

$$S_{n,j} = \sum_{(a,b):a+b=n;0 \leq j \leq n(k-1); 0 \leq i \leq j} S_{a,j} S_{b,j-i}$$

When $n = 1$, $S_1 = X_0, X_1, \cdots, X_{k-1}$; When $n = 2$, $S_2 = S_1 \ast S_1$. The base cases are all correct.

Then, we use $a = u$ and $b = u$ (or $b = u + 1$ when $n = a + b$ is odd). Recursively, in the algorithm, it guarantees that the equality of $n = a + b$ is maintained. Thus, algorithm can produce the correct values for $S_n$, which is reused to compute other components.

2.2 Efficiency and Time Analysis (12 points)

The time complexity of this algorithm is $O(nk\log(nk))$.

Time Analysis: The algorithm can be described as a recursive tree, which has total $O(\log(n))$ levels. Each level $i$ of the tree has only at most two kind of copies of $S_u$ and $S_{u+1}$. The total number of copies is $2^i$. Once $S_u$ or $S_{u+1}$ is computed by FFT, it should be reused to avoid re-computations in the future.
It is a recursive process, we argue from bottom up. At the $\log(n)$-th lowest level, $S_2 = S_1 \ast S_1$ via FFT using $O(k\log k)$, if there is only $S_2$ in this level. The complexity is only $O(k\log k)$. In the level, which contains $S_2$, the other possible element is $S_1$, which is directly from $X_0, X_1, \cdots, X_{k-1}$, so this level has complexity $O(k\log k) < O(2k\log k) = O(2 \times 2^0k\log(2^0k))$.

For the level contains $S_3$, $S_3 = S_2 \ast S_1$ costs $O(2k\log(2k))$ by reusing $S_2$ and $S_1$. In the level, which contains $S_3$, the other possible element is $S_4$, so this level has complexity $O(2k\log(2k)+2k\log(2k)) \leq O(2 \times 2k\log(2k)) = O(2 \times 2^1k\log(2^1k))$.

By reusing $S_u, S_{u+1}$ from $(i+1)$-th level, the $i$-th level has total time complexity of $O(2 \times 2^p k\log(pk))$, where $p = \log(n) + 1 - i$.

The overall complexity
\[
T(n) < O(2 \times 2^0\log(2^0k) + 2 \times 2^1k\log(2^1k) + \cdots + 2 \times 2^{\log(n)k}\log(2^{\log(n)-1}k)) \\
< O(2 \times (1 + 2^1 + \cdots + 2^{\log(n)-1}))\log(nk) \\
< O(2nk\log(nk))
\]

Therefore, the time complexity is $O(nk\log(nk))$. 

3 Load Balancing (Problem 4)

Background

Reduce this problem to network maximum flow problem.

Directed Weighted Graph $G$ Construction:

- Vertices Construction
  $V = \{s, t\} \cup J \cup M$, where $J$ is the set of jobs, and $M$ is the set of machines.

- Edges Construction
  Add an edge from $J_i$ to $M_j$, when machines $j$ can perform the job $i$.
  For $i = \{1, \cdots, n\}$, we add an edge from $s$ to $J_i$ with the capacity of 1.
  For $j = \{1, \cdots, m\}$, we add an edge from $M_j$ to $t$ with the capacity of $T$.

- Discussion about capacity $T$ from $M$ to $t$
  When the algorithm has maximum flow $\text{Flow}(f) = |J| = n$. The flow $f$ is an job assignment when the maximum number of jobs assigned to any machine is not greater than $T$.

3.1 Algorithm Description

Based on the graph $G$ in Directed Weighted Graph $G$ Construction:

- Step 1: $T$ starts from 1;
- Step 2: Run the Ford-Fulkerson (FF) algorithm to obtain network flow $f$.
- Step 3:
  - If the $\text{Flow}(f) = n$, the job assignment with minimum $T$ is obtained and terminate the algorithm.
  - Otherwise $\text{Flow}(f) < n$, we increase $T$ by 1. Update the residual graph $G'$ for $G$, and go to Step 2, run FF using augmentation.

3.2 Correctness and Proof

Flow $f$ to Job Assignment

If there is a flow $f$, $\text{Flow}(f) = n$ in $G$. There are $n$ edges leaving $s$ with unit capacity. Therefore, each edges from $s$ to $J$ are used.

Besides, $J_i$ has and only has one edge with the capacity of 1 flows into it, thus, there must be 1 unit flow leaving $J_i$ and goes to the machine $M_j$, which can perform the task $J_i$, thus each job can only be assigned to one machine.

8 of 9
The $G$ limits each edge from $M$ to $t$ with the capacity of $T$. This means $M_j$ has at most $T$ flow in and out of it, so the maximum number of jobs assigned to $M_j$ is $T$.

We also run $T$ starting from 1, and increase $T$ by 1 till we find the first $f$ Flow($f$), so it is the minimum $T$, we can have for such job assignment.

**Job Assignment to Flow $f$:**

Given a job assignment $J_A$, while each machine $M_i$ in the assignment $J_A$ with at most $T$ jobs. Flow $f$ is the corresponding flow in $G$.

Because $J_A$ is a job assignment, $f$ assigns one unit flow from $s$ to $J_i$ for $i = 1, \cdots, n$ (each $J_i$ has one unit flow into itself). $f$ uses all the edges out of $s$, thus Flow($f$) = $n$.

$f$ also assigns one unit flow from $J_i$ to the $M_j$ in the assignment $J_A$ (In $J_A$, each job is only assigned to one machine). The flows into $M_j$ within $J_A$ has the same flow value out of it to $t$. The flow satisfies the conservation property.

Besides, $J_A$ assigns at most $T$ jobs to any one machine, so there are at most $T$ flows going $M_j$. Then, the flow out of $M_j$ to $t$ is not greater than $T$. This meets the capacity constraints.

### 3.3 Efficiency and Time Analysis

The time complexity of this algorithm is $O(n^2m)$.

**Time analysis:** Starting from $T = 1$, we run the Ford-Fulkerson algorithm, which costs $O(|Edges|f)$ at most, where $O(|Edges|) = O(nm)$ (and $O(|V|) = O(n + m)$). If the Flow($f$) < $n$, we increase $T$ by 1, then there are at most new $m$ edges ($M_j$ to $t$) in residual graph $G'$.

Based on the solution at last iteration, we use DFS to search new paths from $s$ to $t$, and do the augmentations, which costs $O(|mn|)$ and finish the FF method for this iteration.

If the Flow($f$) < $n$, we increase $T$ by 1, and do it again (each iteration costs $O(|mn|)$) until Flow($f$) = $n$. It has at most $n$ iterations. Therefore, the overall time complexity $O(n^2m)$.