CS5234 Cheat Sheet

Gabriel Yeo

Inequalities

Markov: $P(X \geq \alpha) \leq \frac{E(X)}{\alpha}$

Chebyshev: $P(|X-\mu| \geq k) \leq \frac{Var(X)}{k^2}$

Chernoff: $P(X \leq (1-\delta)\mu) \leq e^{-\frac{\delta^2\mu}{2}} \leq e^{-\frac{\delta^2\mu}{3}}$

$P(X \geq (1+\delta)\mu) \leq e^{-\frac{\delta^2\mu}{2+\delta}} \leq e^{-\frac{\delta^2\mu}{3}}$

$X = X_1 + ... + X_n$,

$X_i \in \{0,1\}$, (ind. Bernoulli)

Chernoff: $P(|X-\mu| \geq \delta\mu) \leq 2e^{-\frac{\delta^2\mu}{3}}$

Hoeffding: $P(|X-\mu| \geq t) \leq 2e^{\frac{-2t^2}{n}}$

$X = X_1 + ... + X_n$,

$X_i \in [0,1]$

Hoeffding: $P(|\overline{X} - E[\overline{X}]| \geq t) \leq 2e^{-2nt^2}$

$\overline{X} = \frac{1}{n}(X_1 + ... + X_n)$

Hoeffding (gen): $P(|\overline{X} - E[\overline{X}]| \geq t) \leq 2e^{-\frac{2t^2}{\sum(a_i-b_i)^2}}$

$X_i \in [a_i, b_i]$

Facts

For $0 < x \leq 1$: $1 - x \leq e^{-x} \leq 1 - \frac{x}{2}$

For $0 < x \leq 1$: $\frac{1}{e^2} \leq (1-x)^{1/x} \leq \frac{1}{e}$

$1/e$: $= 0.36787944117$

$1/e^2$: $= 0.13533528323 \leq 1/6$

$1/e^3$: $= 0.04978706836 \leq 1/20$

Variance: $Var[X] = E[(X - E[X])^2)]$

$= E[X^2] - E[X]$

Lecture 3 Maximal Matching

Return the size of the Maximal Matching.

```
query(e):
    for all neighbours e' of e:
        if hash(e') < hash(e)
            if query(e') = TRUE
                return FALSE
    return TRUE

sum := 0
for j = 1 to s:
    - Choose edge e uniformly at random.
    - if (query(e) = True)
        sum := sum + 1
return m·sum/s
```

$E[cost] = 2\sum_{k=1}^{\infty} \frac{d^k}{k!} = O(e^d)$.

$sum = MM(G) \pm \epsilon m$.

Complexity $= O(\frac{e^d}{\epsilon^2})$.

Can do better: $O(\frac{d^4}{\epsilon^2})$, even $O(\frac{d^2}{\epsilon^2})$, and reduce error to $\pm\epsilon n$.

Lecture 2 Connectivity

G is ϵ-close to connected if you can modify at most ϵnd entries in the adjaceny list to make it connected. If $G(V,E)$ is connected, output True, else if ϵ-far from connected, output False.

```
Connected(G, n, d, ε)
    Repeat 16/εd times:
        - Choose a random node u
        - Do a BFS from u, stopping after 8/εd nodes
          seen.
        - If CC of u has ≤ 8/εd nodes, return FALSE.
    return TRUE
```

BFS cost $= \frac{8}{\epsilon d} \cdot d$

Total complexity $= O(\frac{1}{\epsilon^2 d})$

Lecture 2 Connected Components

Output CC such that: $|CC(G) - C| \leq \epsilon n$, w.p. $> 1 - \delta$

Define $n(u)$ = num nodes in the CC of u.

$cost(u) = 1/n(u)$.

$\sum_{u \in CC_i} cost(u) = 1$.

```
sum = 0
for j = 1 to s:
    - Sample u randomly
    - BFS from u, stop when see up to 2/ε nodes
    - If BFS found > 2/ε node:
        - sum := sum + ε/2
    - Else:
        - sum := sum + cost(u)
return n · (sum/s)
```

Let $\overline{C} = \sum cost(u_j)$

Let $Y_j = cost(u_j)$ of our sample j

$|CC(G) - \overline{C}| \leq \epsilon n/2$

$E[Y_j] = \sum \frac{1}{n} cost(u_i) = \frac{1}{n}\overline{C}$

$E[\sum Y_j] = s \cdot E[Y_j] = \frac{s}{n}\overline{C}$

Since we output $\frac{n}{s}\sum Y_j$, we get $E[\frac{n}{s}\sum Y_j] = \overline{C}$

$$\begin{aligned}
&P(|\overline{C} - \frac{n}{s}\sum Y_j| > \epsilon n/2) \\
&= P(|E[\sum Y_j] - \sum Y_j| > \frac{s}{n}\epsilon n/2) \\
&= P(|E[\sum Y_j] - \sum Y_j| > s\epsilon/2) \\
&\leq 2e^{-2\epsilon^2 s^2/4s} \\
&\leq 2e^{-\epsilon^2 s/2} \leq \delta
\end{aligned}$$

Set $s = \frac{2}{\epsilon^2} ln(2/\delta)$.

Complexity $= 2d/\epsilon \cdot O(\frac{1}{\epsilon^2} ln(1/\delta)) = O(\frac{d}{\epsilon^3} ln(1/\delta))$.

Lecture 3 MST weight

Output M such that: $M = MST(G)(1 \pm \epsilon)$ w.p. $> 1 - \delta$.

Let G_j be the graph with edge weights j and below.

Let C_j be the connected components in G_j.

$MST(G)$ contains $C_j - 1$ edges of weight $> j$.

Therefore:

$$\begin{aligned}
MST(G) &= (n - C_1) + \sum_{j-1}^{W-1} (j+1)(C_j - C_j + 1) \\
&= n - W + \sum_{j-1}^{W-1} C_j
\end{aligned}$$

```
sum := n - W
for j = 1 to W - 1:
    - X_j = ApproxCC(G_j, d, ε'δ')
    - sum := sum + X_j
return sum
```

Sum of errors: $(\epsilon' n)(W-1)$

Set $\epsilon' = \epsilon/W$, then sum of errors $\leq \epsilon n$.

Set $\delta' = \delta/W$. Then $P(anyfail) \leq \sum_{1}^{W-1} \delta/W \leq \delta$.

$MST(G) - \epsilon n \leq sum \leq MST(G) + \epsilon n$

Since $MST(G) \geq n - 1 \geq n/2$, $n \leq 2MST(G)$,

$MST(G)(1 - 2\epsilon) \leq sum \leq MST(G)(1 + 2\epsilon)$

Lecture 4 FM

FM returns $1/X - 1, E[X] = \frac{1}{t+1}, Var[X] \leq \frac{1}{(t+1)^2}$.

FM+: Run FM-subroutine $X_1, X_2, ..., X_a$.

Take average: $Z = \frac{1}{a}\sum_{j=1}^{a} X_j$.

Return $C = 1/Z - 1$. $Var[Z] = \frac{1}{a(t+1)^2}$.

$$\begin{aligned}
Pr[|Z - \frac{1}{t+1}| \geq \epsilon(\frac{1}{t+1})] &\leq Var[Z]\frac{(t+1)^2}{\epsilon^2} \\
&\leq \frac{1}{a\epsilon^2}
\end{aligned}$$

Let $a = 4/\epsilon^2$, $C \in t(1 \pm 4\epsilon)$ w.p. $\geq 1/4$.

FM++: Run FM+-subroutine b times and take median.

Claim: If $> 1/2$ of the C_j's are good, median is good.

Let $b = 36ln(2/\delta)$, FM++ returns $C \in t(1 \pm 4\epsilon)$ w.p. $\geq 1 - \delta$.

Lecture 7 External Memory Model

Problem	EMM
Scan Array	$O(N/B)$
Search Array	$O(log(N/B))$
Sort B-tree	$O(Nlog_B N)$
Search B-tree	$O(log_B N)$
Insert/Delete B-tree	$O(log_B N)$
Search BufferTree	$O(logN)$
Insert BufferTree	$O((1/B)logN)$
Search BufferTree $\sqrt{B}$	$O(log_B N)$
Insert BufferTree $\sqrt{B}$	$O((1/\sqrt{B})logN)$
Ex.MergeSort/BufferTree	$O(\frac{N}{B}log_{M/B}N/B)$
BFS	$O(n+\frac{m}{B}log_{M/B}m/B)$

Lecture 7 Caching

B-Trees: $(a,b)-tree$ with n keys has height $\leq log_a(\frac{n}{a})+1$.
At most n/a leaves. Every node except the root has degree $> a$.
Node at height $log_a(\frac{n}{a})$ has $\geq a^{log_a(\frac{a}{n})} \geq \frac{n}{a}$ leaves.
Corollary: if $a \geq B$, then $(a,b)-tree$ with n keys has height $O(log_B n)$.
BufferTrees: Amortized cost of split/share/merge is $O(1/B)$, thus $O((1/B)logN)$ per operation.
With parent pointers: insert may cost $O(BlogN)$ if every level needs to split.

Lecture 9: van Emde Boas Tree

Cache-oblivious structure for searching.

- Let T_r be the subtree consisting of all the nodes of depth $\leq \lfloor (logn)/2 \rfloor$.
- Let $T_2, T_2, ..., T_k$ be the subtrees consisting of all the nodes of depth $> \lfloor (logn)/2 \rfloor$, each subtree rooted at a unique node of depth $\lfloor (logn)/2 \rfloor + 1$.
- Then arrange the array as follows: $L(T_r), L(T_1), L(T_2), ..., L(T_k)$.

Single value search: $O(log_B n)$.
Range search with k values in range $[k1, k2]$: $O(log_B n + k/B + 1)$.

Lecture 11: Parallel BFS

Span of ProcessFrontier (loose) is $O(log^3 n)$.
Work of ProcessFrontier is $O(mlog^2 n)$.
Total work $W = O(mlog^2 n)$.
Total Span $S = O(Dlog^3 n)$.
Totalruntime with p processors and a good scheduler $= O((m/p)log2n + Dlog^3 n)$

Lecture 10: BFS in the EMM

We can BFS in $O(n+\frac{m}{B}log_{M/B}m/B)$.
Inductive invariant: L_i and L_{i-1} are sorted.
Let $N(L_i)$ be the set of edges adjacent to nodes in L_i.

1. **Enumeration:** Iterate through each $u \in L_i$ to make L_{i+1}^{tmp}, which may contain duplicates and nodes in L_i/L_{i-1}. Cost is $2|L_i| + |N(L_i)|/B$.
2. **Duplicate removal:** Sort L_{i+1}^{tmp} and remove duplicates by scanning. Cost is $sort(|N(L_i)|)$ and $|N(L_i)|/B$ for scanning.
3. **Removing visited nodes:** Finally, we can make L_{i+1} by removing nodes in L_i/L_{i-1} from L_{i+1}^{tmp}. Double concurrent pointers strategy: cost is $O(|N(L_i)|/B + |L_i|/B + |L_{i-1}|/B)$.

Total cost for level L_{i+1}:
$O(|L_i| + |N(L_i)|/B + sort(|N(L_i)|) + |L_i|/B + |L_{i-1}|/B)$.
Total cost:
Note that $\sum_i |L_i| = |V|$ since every node appears in exactly 1 level.
Also, $\sum_i(|N(L_i)|/B) = |E|/B$ and $\sum_i sort(N(L_i)) \leq sort(|E|)$.

Lecture 10: CCs in the EMM

Cost is $O(sort(E)log(|E|/M))$.
Main Idea: Recursively convert into a depth 1 tree. Connected nodes will have the same root.

1. **Divide:** Halve edge list E into E_1 and E_2. Recursively solve E_2, contract E_1 with E_2, recurse on E_1, and merge.
2. **Base case:** $\leq M$ edges, just do in memory.
3. **Contract:** E_2 is a depth 1 tree. Edge list is sorted, so we say for (u,v), u is the root. Modify nodes in E_1 that are connected to nodes in E_2: if $(a,b) \in E_1$ has some $(u,b) \in E_2$, convert (a,b) to (a,u). Do for all different intersecting types. Basically, edges to leaves in E_2 can be converted to edges to the root.
   - Sort E_1 by first components and E_2 by second.
   - Scan through E_1 and E_2 in parallel, marking conversions.
   - Sort E_1 by second component and repeat.
4. Recurse on the contracted E_1.
5. Merge. Do a simple contraction again to make sure we do not have a depth 2 tree.

Cost: $T(|E|) \leq 2T(|E|/2) + O(sort(E))$.
$T(M) = M/B$. Thus, $O(sort(E)log(|E|/M))$.

Lecture 11: PRAM and Fork Join model

Relegate the scheduling problem to a separate scheduling component. Create subtasks that are executed by the system in parallel. Up to p subtasks running together. Always assume scheduler does a good job.

- Work W: how long to complete on 1 processor?
- Span S: how long would it take with $p = \infty$?
- Max Parallelism: W/S.
- Good scheduler will get you $O(W/p + S)$.

Parallel Sort:

```
pMergeSort(A, n):
    if (n==1) then return;
    else
        X = fork pMergeSort(A[1..n/2], n/2)
        Y = fork pMergeSort(A[n/2+1, n], n/2)
        sync;
        M = median of X;
        A1 = pMerge(X[1...M], Y[1...M]);
        A2 = pMerge(X[M...end], Y[M...end]);
```

Work: $W(n) = W(\alpha n) + W((1-\alpha)n) + O(logn)$.
Span: $S(n) = S(3n/4) + O(logn) = O(log^2 n)$.

Lecture 11: Parallel Sets

Set 1 (n items), set 2 (m items), n > m.

Problem	Parallel
insert	$W = S = O(logn)$
delete	$W = S = O(logn)$
divide	$W = S = O(logn)$
union	$W = O(n+m), S = O(logn + logm)$
set diff/sub	$W = O(n+m), S = O(logn + logm)$
intersection	$W = O(n+m), S = O(logn + logm)$

Can implement everything with the following operations in (2,4)-tree with: Work = Span = $O(logn + logm)$.

- **Split(T, x)** → (T1, T2, x)
- **Join(T1, T2)** → (T)
- **Root(T)** → (x)
- **Insert(T, x)** → (T')

```
Union(T1, T2)
    if T1 = null: return T2
    if T2 = null: return T1
    key = root(T1)
    (L, G, x) = split(T2, key)
    fork:
        1. TL = Union(key.left, L)
        2. TR = Union(key.right, R)
    sync
    T = join(TL, TR)
    insert(T, key)
    return T
```

$W(n,m) = 2W(n/2,m) + O(logn + logm) = O(nlogm)$.
$S(n,m) = S(n/2,m) + O(logn + logm) = O(log^2 n)$.