CS170 Midterm 1 Sheet Wang Jianzhi

Asymptotics Analysis

$f(n) \in O(g(n)) \Leftrightarrow \exists C>0\ \exists N$ s.t. $n>N \Rightarrow |f(n)| \le C\cdot g(n)$

$g(n) \in \Omega(f(n)) \Leftrightarrow f(n) \in O(g(n))$

Limits

$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \Rightarrow f(n) \in O(g(n))$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \in (0,\infty) \Rightarrow f(n) \in \theta(g(n))$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0 \Rightarrow f(n) \in \Omega(g(n))$

Master's Theorem: $T(n) \le aT(\frac{n}{b}) + n^c$, $b>1, a, c>0$

$c > \log_b a \Rightarrow T(n) = O(n^c)$

$c = \log_b a \Rightarrow T(n) = O(n^c \log n)$

$c < \log_b a \Rightarrow T(n) = O(n^{\log_b a})$

Extended Master's: $T(n) \le aT(\frac{n}{b}) + f(n)$; $c_{crit} = \log_b a$

$f(n) = O(n^c)$, $c < c_{crit} \Rightarrow T(n) = O(n^{c_{crit}})$

$f(n) = \theta(n^{c_{crit}} \log^k n)$ $k \ge 0 \Rightarrow T(n) = \theta(n^{c_{crit}} \log^{k+1} n)$

$f(n) = \Omega(n^c)$, $c > c_{crit} \Rightarrow T(n) = \theta(f(n))$

Akra-Bazzis Method

$T(n) = g(n) + \sum_{i=1}^{k} a_i T(b_i n + h_i(n))$ where $a_i > 0$, $0 < b < 1$ constants

Define $p = \arg\left(\sum_{i=1}^{k} a_i b_i^p = 1\right)$

$|g(x)| \in O(x^c)$

$|h_i(x)| \in O\left(\frac{x}{(\log x)^2}\right)$

Then $T(n) \in \theta\left(n^p\left(1 + \int_1^n \frac{g(u)}{u^{p+1}}\,du\right)\right)$

Analysis Toolbox

- $\log(N!) \in O(N \log N)$
- Guessing the form $T(N) = N^b \log N$
- Guessing the bound $T \le C\cdot n^a$
- Runtime Tree / level analysis
- Polynomial estimation i.e. $n^{\frac{1}{3}} > \frac{n}{4}$
- Invariance: $T(n) = T(\frac{3n}{5}) + T(\frac{4n}{5})$
- Binomial Expansion
- Time for $N \to \frac{N}{2}$
- Substituting $T' = T(\log N)$
- Try special large cases

Minimum Spanning Tree

Cut Property: Suppose $X \subset E$ is part of MST of G. Let $S \subset V$ s.t. X has no edge between S and $V \setminus S$. Then $X \cup \{e\}$ is part of MST of G, where e is lightest edge between S and $V \setminus S$

Graph Tricks

- Change of states, even if $|V|^2$ states
- phantom nodes
- edit edge weights (set all negative cycles to have a edge weight, then apply Bellman Ford $2|V|$ times)
- Augmented Graph: consider G^R.

Last Resorts

- Identify problem type.
- Probabilistic algorithm (DnC)
- Check all paradigms (DnC, DP, Greedy)
- Fast Exponentiation
- On the fly
- Huffman coding (Huffman Tree)

Fast Fourier Transform $O(N \log N)$

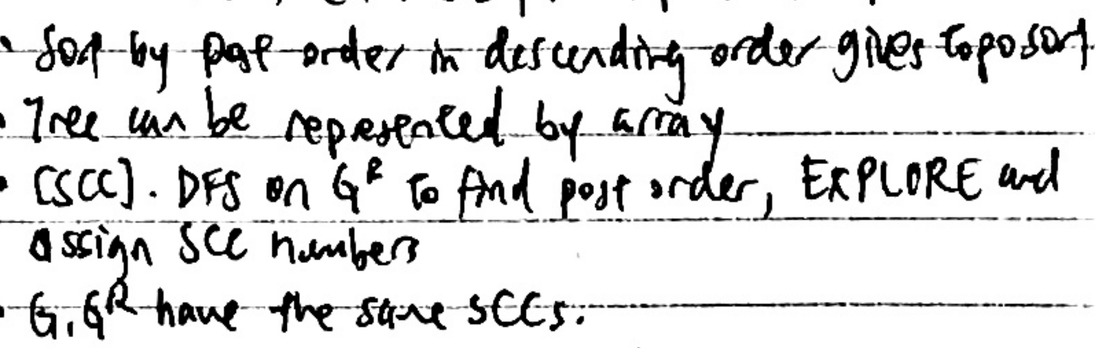
$$\begin{bmatrix} p(1) \\ p(\omega) \\ p(\omega^2) \\ \vdots \\ p(\omega^{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & & & & \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix} = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(n-1)} \\ \vdots & & & & \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} p(1) \\ p(\omega) \\ p(\omega^2) \\ \vdots \\ p(\omega^{n-1}) \end{bmatrix}$$

i.e. IFFT: $\omega \to \omega^{-1}$, multiply by $\frac{1}{n}$.

Modular IFFT: $\omega \to -\omega$, multiply by $n^{-1} \pmod{\cdot}$

[10,3+4i,0,3-i]



Applications: Convolution ($C_j = \sum_i a_i b_{j-i}$), string matching, dot product (sort reverse one of string), generating functions

DFS and variants

Pre-post combinations: $[_u\,[_v\,]_v\,]_u$: Tree / Forward

No other combinations possible: $[_v\,]_v\,[_u\,]_u$: ~~Both~~ Cross

$[_v\,[_u\,]_u\,]_v$: Back

Claims:

- G is a DAG iff no back edge
- In DAG, if $(u,v) \in E$, then post(u) > post(v)
- Sort by post order in descending order gives toposort
- Tree can be represented by array
- [SCC]. DFS on G^R to find post order, EXPLORE and assign SCC numbers
- G, G^R have the same SCCs.
- Let C, C' be SCCs s.t. $C \to C'$. Then after DFS highest post[v] in C > highest post[v] in C'.
- [source finding] DFS on G, get v with highest post order (part of source SCC in G)
- [sink finding] DFS on G^R, get v with highest post order (part of sink SCC in G)
- [Negative cycle Detection] Run Bellman Ford. If dist array changes after one more iteration of relaxing the edges, then exists negative cycle.
- [Negative path from $s \to t$] Run SCC on graph, then for every component of SCC, run Bellman ford on subgraph. If negative cycle, make one edge $-\infty$. Run Bellman Ford on main graph. If dist changes to $-\infty$, then exist negative cycle on path.

Final Check

- Check all edges, see if everything relaxed.
- 3 part Proof.
- Number of times an edge needs to be relaxed (Bellman Ford)

SELECT ($\theta(S)$)

```
SELECT (S, k):  //returns kth smallest number
  pick random pivot v ∈ S
  S< ← {a_i | a_i ∈ S, a_i < v}
  S> ← {a_i | a_i ∈ S, a_i > v}
  S= ← {a_i | a_i ∈ S, a_i = v}
  if k ≤ |S<|: return SELECT(S<, k)
  else if k ≤ |S<| + |S=|: return v
  else return SELECT(S>, k - |S<| - |S=|)
```

DFS ($O(|V|+|E|)$)

```
DFS(G): // DFS on G
  bool visited[n];
  int ccnum[n], pre[n], post[n]
  count ← 1; clock ← 0
  for v ∈ V:
    if visited[v] = false:
      EXPLORE(G, v)
      count ← count + 1
```

BFS ($O(|V|+|E|)$)

```
BFS(G, s):
  dist[s] ← 0
  ∀u ≠ s, dist[u] ← ∞
  Q = [s]
  while Q not empty:
    u ← dequeue(Q)
    for all v s.t. (u,v) ∈ E:
      if dist[v] = ∞:
        enqueue(Q, v)
        dist[v] ← dist[u] + 1
```

Dijkstra $O((|V|+|E|)\log|V|)$ / $O(|E|+|V|\log|V|)$

```
DIJKSTRA(G, l, s):
  dist[s] ← 0
  ∀u ≠ s, dist[u] ← ∞
  V ← {v, dist[v]} ∀v
  while V not empty:
    choose u ∈ V with minimum dist[u]
    remove u from V   [u = V.deleteMin()]
    for v s.t. (u,v) ∈ E:
      dist[v] = min(dist[v], dist[u] + l(u,v))
      decreaseKey(v, dist[v])
```

Bellman Ford $(O(|V||E|))$

```
BELLMAN-FORD(V, E, s):
  for v ∈ V: dist[v] ← ∞, pre[v] ← null
  dist[s] ← 0
  for |V| - 1:
    for (u,v,w) ∈ E:
      if dist[u] + w < dist[v]: dist[v] ← dist[u] + w
                                pre[v] ← u
  for (u,v,w) ∈ E: if dist[u] + w < dist[v]: error "negative cycle"
  return dist, pre
```

EXPLORE

```
EXPLORE(G, u): // DFS on u
  visited[u] ← true
  cc[u] ← count
  pre[u] ← clock; clock ← clock + 1
  for v s.t. (u,v) ∈ E:
    if visited[v] = false:
      EXPLORE(G, v)
  post[u] ← clock; clock ← clock + 1
```

TOPO SORT

```
topo ← []
EXPLORE(G, u): // DFS on u
  visited[u] ← true
  for v s.t. (u,v) ∈ E:
    if visited[v] = false:
      EXPLORE(G, v)
  topo.add(u)
DFS(G); return reverse(topo)
```

SCC

```
SCC(G):
  ∀v ∈ V, visited[v] ← false
  DFS(G^R) // computes post order
  count ← 1
  for u ∈ V in reverse post order:
    if visited[u] = false:
      EXPLORE(G, u)
      count ← count + 1.
```

Prim's Algorithm $O((|V|+|E|)\log|V|)$ / $O(|E|+|V|\log|V|)$

```
PRIM(G, w):
  X ← {}, Q ← priorityqueue()
  for each u ∈ V: Q.insert(u, ∞), from[u] ← null
  pick start vertex s ∈ V
  Q.decreasekey(s, 0)
  while |X| ≤ |V| - 1:
    u ← deleteMin(Q)
    if u ≠ s: X ← X ∪ {(from[u], u)}
    for all v ∈ V s.t. (u,v) ∈ E:
      if v ∈ Q STILL and w(u,v) < v.key():
        Q.decreasekey(v, w(u,v))
        from[v] ← u.
```

Kruskal

```
KRUSKAL(G, w):
  ∀v ∈ V, makeset(v);
  X ← {}. sort edges E by w
  for (u,v) ∈ E:
    if find(u) ≠ find(v):
      add (u,v) to X; union(u,v)
```

UFDS rank[i] = depth of tree below node i

DS	Insert	DelMin	Decrease	Merge
Array	$O(1)$	$O(N)$	$O(1)$	$O(N^2)$
Binary	$O(\log N)$	$O(\log N)$	$O(\log N)$	$O(N\log N)$
Fib	$O(1)$	$O(\log N)$	$O(1)$	$O(N+\log N)$