Classics

Asymptotic Analysis

$$f(n) \in O(g(n)) \Leftrightarrow$$
$$\exists c > 0 \quad \exists N \text{ s.t. } n > N \Rightarrow |f(n)| \le c \cdot g(n)$$

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

$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \Rightarrow f(n) \in O(g(n))$
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \in (0,\infty) \Rightarrow f(n) \in \Theta(g(n))$
$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0 \Rightarrow f(n) \in \Omega(g(n))$

Master's Theorem

$$T(n) \le a\,T\left(\frac{n}{b}\right) + n^c$$
$$b > 1, a, c > 0$$

#1	$c > \log_b a$	$T(n) = O(n^c)$
#2	$c = \log_b a$	$T(n) = O(n^c \log n)$
#3	$c < \log_b a$	$T(n) = O(n^{\log_b a})$

$$T(n) \le a\,T\left(\frac{n}{b}\right) + f(n)$$
$$c_{\text{crit}} = \log_b a$$

$f(n) = O(n^c), c < c_{\text{crit}}$	$T(n) = \Theta(n^{c_{\text{crit}}})$
$f(n) = \Theta(n^{c_{\text{crit}}} \log^k n)$ $k \ge 0$	$T(n) = \Theta(n^{c_{\text{crit}}} \log^{k+1} n)$
$f(n) = \Omega(n^c), c > c_{\text{crit}}$	$T(n) = \Theta(f(n))$

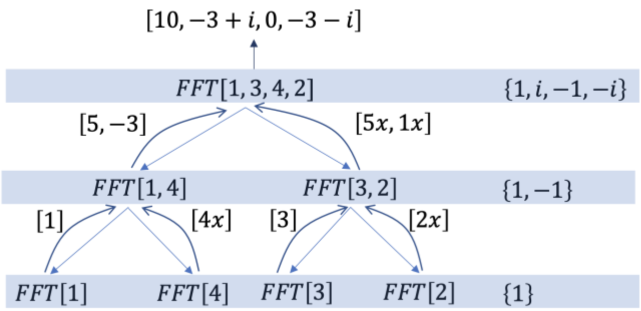
Fast Fourier Transform (FFT) $O(N \log N)$

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

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

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

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



Applications: Convolution ($c_j = \sum_i a_i b_{j-i}$), string matching, dot product (just reverse)

Akra-Bazzi's Method

$$T(n) = g(n) + \sum_{i=1}^{k} a_i T(b_i n + h_i(n))$$

where $a_i > 0, 0 < b_i < 1$ constants, $|g(x)| \in O(x^c), |h_i(x)| \in O\left(\frac{x}{(\log x)^2}\right)$

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

$$T(n) \in \Theta\left(n^p \left(1 + \int_1^n \frac{g(u)}{u^{p+1}} \mathrm{d}u\right)\right)$$

Analysis Toolbox

- $\log(N!) \in \Theta(N \log N)$
- Invariance trick: $T(N) = T\left(\frac{3N}{5}\right) + T\left(\frac{4N}{5}\right)$
- Guessing the form: $T(N) = N^b \log N$
- Guessing the bound: $T \le c \times n^a$
- Binomial expansion
- Time for $N \to \frac{N}{2}$
- Runtime tree; level analysis
- Substitution of $T' = T(\log n)$ and apply techniques on T' instead

DFS and Variants

Pre-, post- ordering combinations: $(u, v) \in E$

$[_u\ [_v\ ]_v\ ]_u$	Tree, Forward
$[_v\ ]_v\ [_u\ ]_u$	Cross
$[_v\ [_u\ ]_u\ ]_v$	Back

No other combinations possible

Claims

- G is a DAG iff no back edges (cycle-finding)
- In DAG, $(u, v) \in E$, then $\text{post}(u) > \text{post}(v)$
- Sort by post order in descending order gives a topological sort (linearization)
- Trees can be represented by an 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 such that $C \to C'$. Then, after DFS, highest post[v] in C > highest post[v] in C'

- Polynomial estimation $n^{\frac{1}{3}} > \frac{n}{4}$; then apply Master's Theorem to get a bound
- Try special large values and find generalizations: 3^{3^3} etc. 3^{3^k}

Minimum Spanning Tree

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

Graph Tricks

- Change of states, even if $|V|^2$ states
- Edit edge weights (e.g. set all negative cycles to have a $-\infty$ edge weights, then apply Bellman Ford $2|V|$ times)
- Phantom nodes; higher dimension nodes
- Augmented graph; consider G^R

- [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^R)

Negative cycle detection

Run Bellman-Ford. If dist array changes after one more iteration of relaxing the edges, exists negative cycle.

Negative cycle on path $s \to t$

SCC in negative cycle on path. Run SCC on graph, then for every component of the SCC, consider the subgraph. Run Bellman ford, if negative cycle, then make one edge $-\infty$. Run Bellman ford on main graph again. If $-\infty$, we are done. (there is negative cycle on path).

Appendix of Pseudo-codes

SELECT $\Theta(|S|)$

```
SELECT(S, k): // return 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_=|)
```

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
```

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

```
DFS(G): // DFS on G
   boolean 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
```

TOPOSORT

```
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)
```

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
```

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
```

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

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

Implementations for Dijkstra

```
PriorityQueue():
   insert(elem, key)
   deleteMin()
   decreaseKey(elem, key)
```

DS	Insert	DelMin	Decr	Total
Array	1	N	1	N^2
Binary	$\log N$	$\log N$	$\log N$	$(N+M)\log N$
Fibo	1	$\log N$	1	$N\log N + M$

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

```
BELLMAN_FORD(V, E, s):
```

Horn SAT

```
HORN_SAT(X):
```

```
  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 "contains negative weight"
  return dist, pre
```

```
  ∀i, x_i ← False
  while ∃ unsatisfied (x_i ∧ … ∧ x_j) ⇒ x_k:
    x_k ← True
  if every (x̄_i ∨ … x̄_j) satisfied:
    return (x_1, … , x_n)
  else return "not satisfiable"
```

Kruskal's Algorithm $O(\|E\| \log\|V\|)$	Meta Algorithm

```
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)
```

```
META(G, w):
  X = { }
  repeat until |X| = |V| − 1:
    pick S ⊂ V s.t. X has no edges from S to V\S
    let e ∈ E be lightest edge from S to V\S
    X = X ∪ {e}
```

Prim's Algorithm $O((\|V\| + \|E\|)\log\|V\|)/O(\|E\| + \|V\|\log\|V\|)$	UFDS $O(n + m \cdot \alpha(m, n))$

```
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
```

```
UFDS(n):
  ∀i p[i] ← i, rank[i] ← 0


FIND(x):
  if p[x] = x: return x
  return p[x] ← FIND(p[x])

UNION(x, y):
  x ← FIND(x)
  y ← FIND(y)
  if x = y: return
  if rank[x] = rank[y]:
    p[y] ← x
    rank[x] ← rank[x] + 1
  else if rank[x] < rank[y]:
    p[x] ← y
  else:
    p[y] ← x
```