Reference Sheet

$BFS(s)$

```
Initialize: All vertices marked "undiscovered"
Mark s discovered
queue ← {s}
while queue not empty do
    u ← removeFront(queue)
    for all edge (u, x) do
        if x is "undiscovered" then
            Mark x "discovered"
            Append x on queue
        end if
    end for
    Mark u "fully explored"
end while
```

$TopologicalOrder(G)$

```
count[w] ← (remaining) number of incoming edges to w
S ← set of (remaining) nodes with no incoming edges
while S not empty do
    Remove some v from S
    make v next in topological order
    for all edges from v to some w do
        decrement count[w]
        if count[w] = 0 then
            add w to S
        end if
    end for
end while
```

DFS(v):

```
v.dfs# =dfscounter++
for all edge (v, x) do
    if x.dfs# = −1 then
        DFS(x)
    else
        (code for back edges, etc.)
    end if
    Mark v "completed"
end for
```

Shortest Weighted Path(G, s, l)

```
Let S be the set of explored nodes
S = {s} and d(s) = 0
while S ≠ V do
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d(u) + l_e
    is as small as possible.
    Add v to S and define d(v) = d'(v)
end while
```

Min Spanning Tree(G, l) (Prim's Algorithm)

```
Arbitrarily choose some starting node x.
Let V_new = {x}, E_tree = {}.
while  doV_new ≠ V
    Choose edge e = (u, v) with minimal weight such that u ∈ V_new and v ∉ V_new
    Add v to V_new and e to E_tree.
end while
```

Huffman(C, f)

```
Insert node for each letter into priority queue by freq
while queue length > 1 do
    Remove smallest 2 nodes, call them x, y
    Make new node z with children x, y.
    f(z) = f(x) + f(y)
    Insert z into queue
end while
```

O, Ω, Θ

- $f(n)$ is $O(g(n))$ iff there is a constant $c > 0$ so that $f(n)$ is eventually always $\leq c\ g(n)$
- $f(n)$ is $\Omega(g(n))$ iff there is a constant $c > 0$ so that $f(n)$ is eventually always $\geq c\ g(n)$
- $f(n)$ is $\Theta(g(n))$ iff there is are constants $c_1, c_2 > 0$ so that eventually always $c_1\ g(n) \leq f(n) \leq c_2\ g(n)$

Greedy Analysis Strategies

- Greedy algorithm *stays ahead*. Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.
- *Structural*. Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.

- *Exchange argument.* Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

Master Recurrence

- If $T(n) = aT(n/b) + cn^k$ for $n > b$ then
  - if $a > b^k$ then $T(n)$ is $\Theta(n^{\log_b a})$.
    (many subproblems $\Rightarrow$ leaves dominate)
  - if $a < b^k$ then $T(n)$ is $\Theta(n^k)$
    (few subproblems $\Rightarrow$ top level dominates)
  - if $a = b^k$ then $T(n)$ is $\Theta(n^k logn)$
    (balanced $\Rightarrow$ all $\log n$ levels contribute)

Minimum Stamp Recurrence

$$Opt(i) = min \left(\begin{cases} 0 & i = 0 \\ 1 + Opt(i-5) & i \geq 5 \\ 1 + Opt(i-4) & i \geq 4 \\ 1 + Opt(i-1) & i \geq 1 \end{cases} \right)$$

Minimum Stamp: Memoized Code

Initialize M to array of "empty" values
procedure MEMOIZESTAMP(n)
  if $M[n] =$ "empty" **then** $M[n] = min \left(\begin{cases} 0 & i = 0 \\ 1 + MemoizeStamp(i-5) & i \geq 5 \\ 1 + MemoizeStamp(i-4) & i \geq 4 \\ 1 + MemoizeStamp(i-1) & i \geq 1 \end{cases} \right)$
  end if
  return $M[n]$
end procedure

Minimum Stamp: Iterative Code

---

for $i = 1$ to n **do**
  $M(i) = min \left(\begin{cases} 0 & i = 0 \\ 1 + M(i-5) & i \geq 5 \\ 1 + M(i-4) & i \geq 4 \\ 1 + M(i-1) & i \geq 1 \end{cases} \right)$
end for

RNA folding recurrence

$$Opt[i,j] = \begin{cases} 0 & \text{if } i \geq j-4 \\ max \begin{cases} Opt[i,j-1] \\ max_t(1 + Opt[i,t-1] + Opt[t+1,j-1]) \end{cases} \end{cases}$$

NP

A decision problem is in NP if and only if there is a polynomial time procedure $verify()$ and an integer k such that

1. For every YES problem instance x there is a hint h with $|h| \leq |x|^k$ such that $verify(x,h) = YES$
2. For every NO problem instance x there is no hint h with $|h| \leq |x|^k$ such that $verify(x,h) = YES$

NP-hard

A problem B is NP-hard if and only if every problem in NP is polynomial time reducible to B.

NP-complete

A problem B is NP-complete if and only if both:

1. B is in NP
2. B is NP-hard

Polynomial-Time Reductions

Given two decision problems, A, B, we say that A is polynomial-time reducible to B, $A \leq_P B$ if there exists some polynomial-time function f that converts each instance x of problem A into an instance $f(x)$ of problem B such that x is a YES instance of A if and only if $f(x)$ is a YES instance of B.