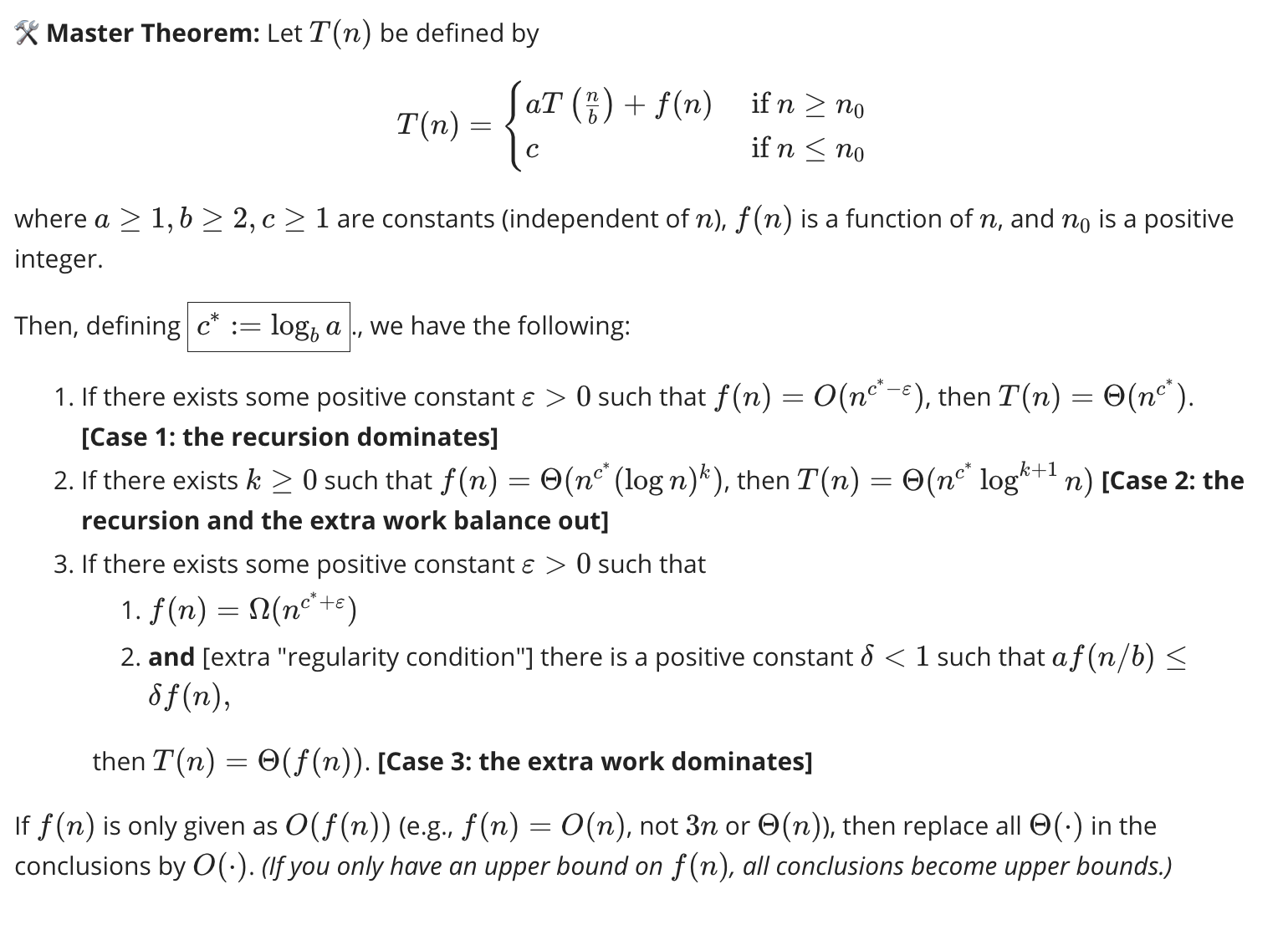
**Dynamic Programming**

1. Define subproblems.
2. Define recurrence.
3. Define base case.
4. Prove base cases are reached.
5. Prove correctness.
6. Find call that solves original problem.
7. Analyse time complexity.

****

**Flows**

Max flow = min cut

F is max flow iff there is no augmenting path in the residual network

Flow value lemma: for any s-t cut the net flow sent across the cut is equal to the amount leaving s

Integrality: If all capacities are integers then there exists a max flow f for which every edge has an integer flow

Ford-Fulkerson: Find and s-t path in the residual network. Find the bottleneck of that path. Increase the flow through each edge by the bottleneck (decrease for backwards). O(F(m+n)) runtime. Pseudo polynomial algo.

Edmund Karp O((m^2logF)) || Edmunds Karp2 O(m+n) || Better Fulk O((m+n)logF)

**Bipartite:**Undirected graph G = (V, E) |M ⊆ E is a matching if each node appears in at most one edge in M.

Bipartite graph G = (L ∪ R, E) | Find max cardinality matching.

Max Flow Formulation: 1. Create digraph G' = (L ∪ R ∪ {s, t}, E' ). 2. Direct all edges from L to R, and assign **unit** capacity. 3. Add source s, and unit capacity edges from s to each node in L. 4. Add sink t, and unit capacity edges from each node in R to t. To show >= V

1. **Flow integrality** (No Fraction Flows) 2. Constraints of flownet **(capacity, conservation)** satisfy matching constraints (each vertex incident to at most one matching edge) 3. **Flow Value Lemma** matching has same size as value of flow

Marriage Theorem: G = (L ∪ R, E), bipartite graph with |L| = |R|. G has a **perfect matching** iff |N(S)| ≥ |S| for **all** subsets S ⊆ L.

**Edge Disjoint:** Two paths = edge-disjoint if no edge in common (can share node). Q eg. find max edge disjoint paths

^ Max flow formulation: assign unit capacity to every edge | edge-disjoint s-t paths equals max flow value

**Network Connectivity:** Find min number of edges whose removal = no s-t paths.

F ⊆ E d/c t from s if all s-t paths uses at least one edge in F | Max num of edge-dsjnt s-t paths = min number of edges to remove. | Constraints for -V- = (edge capacity, **and demand conservation** sum of +dv = -dv)

**Circulations:** Directed graph G = (V, E). Edge capacities c(e). Supply and demands d(v). Given (V, E, c, d) is there circ?

1. Add new source s and sink t. 2. Each v with d(v) < 0, add source edge 2. each v with d(v) > 0, add sink edge.

G has circulation iff G' has max flow of value D.

**NP**

Decision: does there exist? Output yes/no

Optimization: what is the size? Output number

Search: find the object. Output object

Strategies:

1. Reduction by simple equivalence (S independent set iff V\S is a vertex cover)
2. Reduction from special to general case (Vertex cover < independent set - make each node a set and each edge an number in the set. Make U the set of all edges)
3. Reduction by encoding with gadgets (3-Sat < Independent set – Make a group of 3 vertices for each clause connect literal to its negations in other clauses then 3-sat is satisfiable iff there is an independent set with k = num clauses )

Restricted Cases

1. Independent set on trees – if v is a leaf there exists a max size independent set containing v. Can be solved in O(n) time
2. Weighted IS on trees – If (u, v) is an edge with leaf v then OPT either includes u or all leaf nodes incident to u. OPTin(u) = max weight independent set rooted at u, containing u. OPTout(u) = max weight independent set rooted at u, not containing u. OPT(u) = max{OPTin, OPTout}. Runs in O(n) time
3. Finding small vertex covers - Let (u,v) be an edge of G. G has a vertex cover of size ≤ k iff at least one of G\{ u } and G\{ v } has a vertex cover of size ≤ k-1. Check graph without each node and return if one of them has a vertex cover. Runs in O(2kkn) fine if k is small

Approximating solution:

Sacrifice optimality but still run in polynomial time

Approximation ratio = cost(soln) / cost(opt soln)

**Constraint Satisfaction**

SAT: Given a Boolean formula **f**, the problem is to identify if the formula **f** has a satisfying assignment or not.

3-SAT: Given a conjunction of clauses, where each clause is a disjunction of exactly 3 literals determine if there is a satisfying assignment of literals

**Packing**

Set-packing: given a set U consisting of n elements and m subsets of U S1, …, Sm­ and a positive integer k determine if there exists a collections of at least k subsets such that no two subsets intersect

Independent set: Given a graph G = (V, E) determine if there exists an independent set S⊆V such that no two vertices in S share and edge

**Covering**

Vertex-cover: Given a graph G and integer k is there a subset of vertices S such that | S | ≤ k and for each edge at least one of its endpoints is in S

Set cover: Given a set U of elements and a collection S1, …, Sm of subsets of U and integer k does there exist a collection of k of these sets whose union is equal to U

**Sequencing**

Hamiltonian cycle: Given a graph G = (V, E), the problem is to determine if graph G contains a Hamiltonian cycle (a cycle visiting each vertex exactly once) consisting of all the vertices belonging to V.

Travelling salesman: Given a graph G = (V, E) where each edge in E has a weight, what is Hamiltonian cycle in G with the minimum total path weight

**Partitioning**

3D-matching: Let X, Y, and Z be finite sets, and let T be a subset of X × Y × Z. Now *M* ⊆ *T* is a 3-dimensional matching if the following holds: for any two distinct triples (*x*1, *y*1, *z*1) ∈ *M* and (*x*2, *y*2, *z*2) ∈ *M*, we have *x*1 ≠ *x*2, *y*1 ≠ *y*2, and *z*1 ≠ *z*2. Given an integer k decide if there exists a such a set M with |M | ≥ k.

3-Colour: The graph 3-colorability problem is a decision problem in graph theory which asks if it is possible to assign a colour to each vertex of a given graph using at most three colours, satisfying the condition that every two adjacent vertices have different colours.

Set partition: Given a set S find if there is a partition into {A, B} where S = A U B and the sum of A = the sum of B

**Numerical Problems**

Subset Sum: Given a set of non-negative integers and a value sum, the task is to check if there is a subset of the given set whose sum is equal to the given sum.

Knapsack: Given a set of n items each with a weight and benefit what is the best selection of items such that the benefit is maximised but the weight does not exceed some W