Amortized Analysis

- Amortized analysis guarantees average performance of each operation in the worst case.
- $\sum$ Amortized Cost $\geq \sum$ Actual Cost.

Aggregate Method

Amortized cost = (total cost of k operations) / k

Accounting Method

- Allows to store credit into a bank for future use, if its assigned amortized cost > actual cost
- Allows to pay for its extra actual cost using existing credit, if its assigned amortized cost < actual cost.
- IMPORTANT property: the bank balance must not go negative, i.e. TOTAL amortized cost must always be $\geq$ TOTAL actual cost.

Charging Method

amortized cost of an operation = actual cost of this operation - total cost charged to past operations + total cost charged by future operations.s

Potential Method

- Amortized cost of an operation = actual cost of this operation + $\Delta\phi$
- Amortized cost of n operation = actual cost of n operations + $\phi(n) - \phi(0)$
- IMPORTANT conditions to be fulfilled by ϕ: $\phi(0) = 0$ and $\phi(i) \geq 0$ for all i.
- To find suitable potential function, try to view carefully the costly operation and see if there is some quantity that is "decreasing" during the operation.

Dynamic Programming

1. Formulate the problem recursively:
    - Don't forget base case :)
    - Argue correctness of recursive formula by using "optimal substructure" → **exact** problem with different (smaller / larger) input sizes.
    - Use the "cut and paste" argument, which goes sth like "If S is a optimal solution, then S - {x} must also be an optimal solution" (this is usually easy to prove by contradiction)

2. Build solutions from the bottom up
    1. Identify the subproblems, choose a memoization data structure
    2. Identify dependencies and find a good evaluation order (from base cases → original problem)
    3. Analyze: The number of distinct subproblems determines the space complexity. Total running time: $\sum$ over all subproblems of running times not counting recursive calls.

Greedy

Start from DP always! Proving greedy is harder.

1. Assume that there is an optimal solution that is different from the greedy solution
2. Find the "first" difference between the 2 solutions
3. Argue that we can exchange the optimal choice for the greedy choice without making the solution worse (although not necessarily better).

e.g. Suppose L is an ordered list of elements. We will argue that if we swap adjacent items of L, then the cost will increase.

Also, don't forget to write the optimal substructure (to justify that we can obtain an optimal solution from a subproblem's optimal solution).

Reducibility and Intractability

$A \leq_P B$ (read: A is polynomial-time reducible to B) : if B is "easily solvable" then so is A!

Pseudopolynomial problem: problems that is polynomial in terms of the input value (but exponential in terms of the number of inputs). FYI: this type of problems are called weakly NP-hard.

Decision problem $\leq_P$ Optimization problem

Karp Reduction between Decision Problems

Given 2 decision problems A and B, $A \leq_P B$ is a transformation from instances α of A to instances β of B s.t.

1. α is a YES-instance for $A \iff \beta$ is a YES-instance for B.
2. The transformation takes polynomial time in the size of α.

> To prove that problem A is NP-hard, reduce a known NP-hard problem to A.

CircuitSAT $\leq_P$ 3-SAT

Encode each gate in input circuit as a clause ("gate gadgets") in the output formula.

3-SAT $\leq_P$ MaxIndependentSet

Reduction: Given an arbitrary 3CNF formula Φ.

- Let k denote the number fo clauses in Φ.
- The graph G contains 3 vertices for each clause, one for each literal.
- Connect 3 literals in the same clause to form a triangle. ("clause gadgets")
- Connect literal to each of its negations. ("variable gadgets")

MaxIndependentSet, MinVertexCover, MaxClique

MaxClique: number of nodes in its largest complete subgraph in a given graph

MinVertexCover: A set S is a vertex cover of $G := (V, E)$ if for every edge $(u, v) \in E$, at least one of u or v is in S.

MaxIndependentSet: A set S is an independent set of $G := (V, E)$ if for every $u, v \in V$, then $(u, v) \notin E$

Reductions:

1. MaxIndSet → MaxClique: Consider the edge-complement graph $\bar{G}$, i.e. (u, v) is an edge in $\bar{G}$ iff (u, v) is NOT an edge in G. The largest independent set in G has the same vertices as the largest clique in $\bar{G}$.
2. MaxIndSet → MinVertexCover: Suppose, the largest independent set in G is I. Then $V \backslash I$ is the smallest vertex cover.

3-SAT $\leq_P$ 3Color

3Color: Given a graph, does it have a proper 3-coloring, i.e. color the vertices of a graph in 1 of 3 possible colors such that every edge has 2 different colors at its endpoints.

Figure 12.12. The truth gadget and a variable gadget for a.

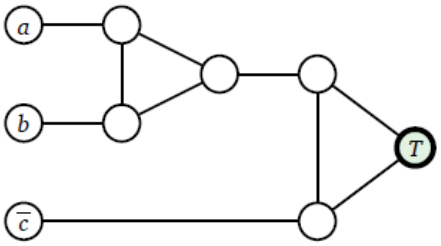

Figure 12.13. A clause gadget for $(a \vee b \vee \bar{c})$.

Reduction:

1. Create a single *truth gadget*.
2. For each variable a, create a *variable gadget*, i.e. a triangle joining two new nodes labeled a and $\bar{a}$ to node X in the *truth gadget*.
3. For each clause in Φ, create a *clause gadget*, joining three literal nodes from the corresponding variable gadgets to node T on the truth gadget as shown in the figure above.

Observation:

- at least one of the 3 literal nodes in every clause gadget is colored TRUE.

MinVertexCover $\leq_P$ DirHamCycle

A Hamiltonian cycle in a graph is a cycle that visits every vertex exactly once.

We can modify the below reduction to prove directed hamiltonian path.
Then use reduction from directed to its undirected counterpart.
Finally, TSP problem is finding a minimum weight hamiltonian cycle in a complete, weighted graph, so it must be NP-hard.

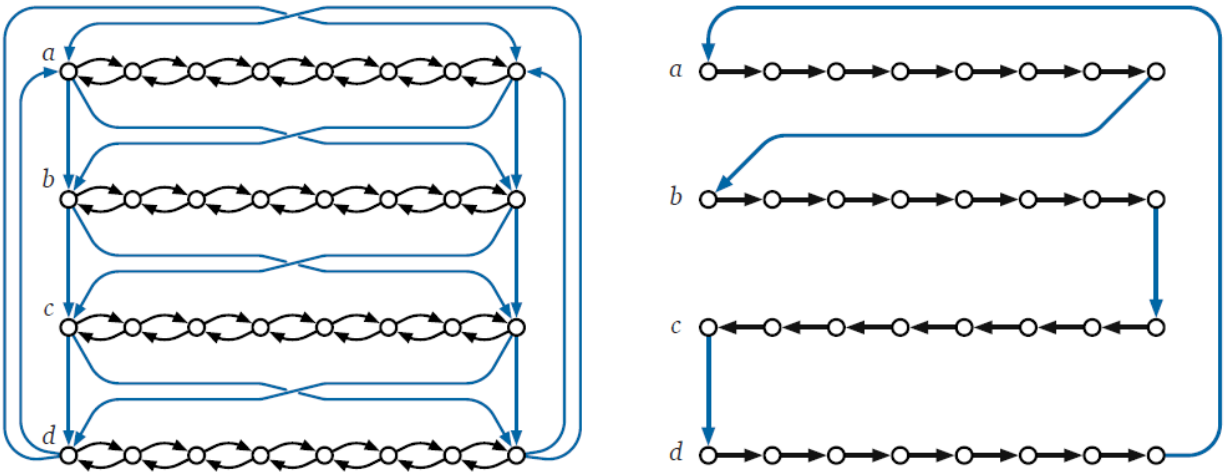

Figure 12.19. Left: Variable gadgets and connectors in G, for any formula with 4 variables and 4 clauses. Right: The Hamiltonian cycle in G corresponding to the assignment $a = b = d = \text{TRUE}$ and $c = \text{FALSE}$

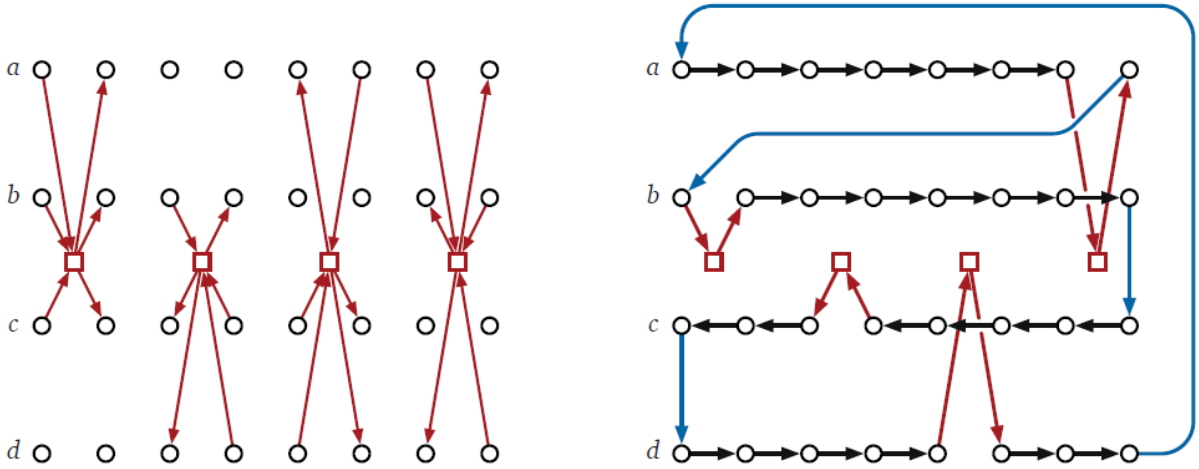

Figure 12.20. Left: Clause gadgets for the formula $(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$. Right: A hamiltonian cycle in H corresponding to the satisfying assignment $a = b = d = \text{TRUE}$ and $c = \text{FALSE}$.

Rules of Thumb

- Assign bits to objects? Choose subset of objects? Partition objects into 2 different subsets → SAT / Partition. E.g. LoadBalancing
- Assign labels to objects from a small fixed set? Partition objects into a small number of subsets → k-Color / 3-Color
- Arrange a set of objects in a particular order? → Dir.HamCycle / DirHamPath / TSP
- Find a SMALL subset satisfying some constraints → MinVertexCover
- Find a LARGE subset satisfying some constraints → MaxIndSet / MaxClique
- If all else fails, try 3SAT / CircuitSAT.

Approximation

Let C' be the optimal cost and C be the cost of the solution given by approximation algorithm A. Then $C \leq \rho(n) \cdot C'$ where $\rho(n) \geq 1$ for minimization problem AND $C \geq \rho(n) \cdot C'$ where $\rho(n) \leq 1$ for maximization problem.

GreedyVertexCover ($\log n$ approximation)

Algorithm: Mark vertex with highest degree and remove all incident edges. Recurse until G is empty.

Proof: For all i , let G_i be the graph after i iterations of the main loop. Let d_i be the max degree of any node in G_{i-1}. Let $|G_i|$ denote the number of edges in G_i. Let C' denote the optimal vertex cover of G, containing OPT vertices. C' is also a vertex cover for G_{i-1}. So $\sum_{v \in C'} \deg_{G_{i-1}}(v) \geq |G_{i-1}|$.
Then $\sum_{i=1}^{OPT} d_i \geq \sum_{i=1}^{OPT} |G_{i-1}|/OPT \geq \sum_{i=1}^{OPT} |G_{OPT}|/OPT = |G_{OPT}| = |G| - \sum_{i=1}^{OPT} d_i$.
In other words, the first OPT iterations of GreedyVC remove at least half the edges of G. After at most $OPT \log |G|$ iterations, all edges of G have been removed. So, GreedyVC computes a vertex cover of size $O(OPT \log n)$.

GreedySetCover and GreedyHittingSet

A **set cover** of a set system (X, F) is a subfamily of sets in F whose unioin is the entire ground set X. Here if we let X be the edges of an undirected graph and F be the vertices, then a vertex cover is a set cover.

A **hitting set** of a set system (X, F) is a subset of the ground set X that intersects every set in F. Here if we let X be the vertices of an undirected graph and F be the edges, then a vertex cover is a hitting set.

DumbVertexCover (2-Approximation)

Algorithm: Pick any random vertex from G. Add both endpoints as vertex cover and remove all incident edges.

Proof: Let $|A|$ be the number of while-loop iterations. Let C' be the optimal vertex cover. We know that each edge must have at least one endpoint in any vertex cover, so $|A| \leq |C'|$. Otoh, any vertex cover C produced by DumbVC will have $|C| \leq 2|A|$.
$\therefore |C| \leq 2|C'|$ as we wanted.