L02 Basics of Algorithm Analysis

① *Desirable scaling property.* When the input size doubles, the algorithm should only slow down by some constant factor C.

Poly time. There exists constants $c > 0$ and $d > 0$ such that on every input of size N, its running time is bounded by cN^d steps.

Thm. An algorithm is poly time iff. the above scaling property holds.

② *Def.* An algorithm is efficient if its running time is polynomial.

③ *Upper bounds.* T(n) is $O(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $T(n) \leq c \cdot f(n)$.

Lower bounds. T(n) is $\Omega(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $T(n) \geq c \cdot f(n)$.

Tight bounds. T(n) is $\Theta(f(n))$ if T(n) is both $O(f(n))$ and $\Omega(f(n))$.

L03 Greedy Algorithms

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

① Interval Scheduling

Consider jobs in increasing order of finish time. Take each job provided its compatible with the ones already taken. O(n log n).

Proof by contradiction.

② Scheduling to Minimize Lateness

Goal: schedule all jobs to minimize maximum lateness $L = \max l_j$.

Earliest deadline first.

Def. An inversion in schedule S is a pair of jobs i and j such that: i < j but j scheduled before i.

Observation. There exists an optimal schedule with no idle time. Greedy schedule has no inversions. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.

Claim. Swapping two adjacent, inverted jobs reduces the number of inversions by one and does not increase the max lateness.

③ Optimal Caching

Goal. Eviction schedule that minimizes number of evictions.

Farthest-in-future. Evict item in the cache that is not requested until farthest in the future.

Def. A reduced schedule is a schedule that only inserts an item into the cache in a step in which that item is requested.

Lemma. Let S be a reduced schedule that makes the same schedule as S_{FF} through the first j requests. Then there is a reduced schedule S' that makes the same schedule as S_{FF} through the first j+1 requests, and incurs no more evictions than S does.

④ Clustering

Clustering of maximum spacing. Given an integer k, divide objects into k non-empty groups s.t. maximizing the min distance between any pair of points in different clusters. (find a k-clustering of maximum spacing)

Single-link k-clustering algorithm.

-Create n clusters, one for each object.

-Find the closest pair of objects such that each object is in a different cluster; add an edge between them and merge the two clusters.

-Repeat n-k times until there are exactly k clusters.

Key observation. This procedure is precisely Kruskals algorithm (except we stop when there are k connected components). Equivalent to finding an MST and deleting the k-1 most expensive edges.

L04 Divide and Conquer

设 $a \geq 1$ 和 $b > 1$ 为常数，设 f(n)为一函数，T(n)由递归式

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

其中$\frac{n}{b}$指$\left\lfloor\frac{n}{b}\right\rfloor$和$\left\lceil\frac{n}{b}\right\rceil$，可以证明，略去上下去整不会对结果造成影响。那么 T(n)可能有如下的渐进界

(1)若 $f(n) < n^{\log_b^a}$，且是多项式的小于。即

$\exists\, \varepsilon > 0$，有 $f(n) = O\left(n^{\log_b^a - \varepsilon}\right)$，则 $T(n) = \Theta\left(n^{\log_b^a}\right)$

(2)若 $f(n) = n^{\log_b^a}$，则 $T(n) = \Theta\left(n^{\log_b^a} \log n\right)$

(3)若 $f(n) > n^{\log_b^a}$，且是多项式的大于。即

$\exists\, \varepsilon > 0$，有 $f(n) = \Omega\left(n^{\log_b^a + \varepsilon}\right)$，且对$\forall\, c < 1$ 与所有足够大的 n，有 $af\left(\frac{n}{b}\right) \leq cf(n)$，则 $T(n) = \Theta\left(f(n)\right)$

① Merge sort

Def. T(n) = number of comparisons to mergesort an input of size n.

$$T(n) = \begin{cases} 0 & if \ \ n = 1 \\ T\left(\left\lceil\frac{n}{2}\right\rceil\right) + T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + n & otherwise \end{cases}$$

$T(n) = O(n \log_2 n)$.

② Closest Pair of Points

Assumption. No two points have same x coordinate.

Algorithm. Compute separation line L such that half the points are on one side and half on the other side; Conquer: find closest pair in each side recursively; Delete all points further than δ from separation line L; Sort remaining points by y-coordinate. Scan points in y-order and compare distance between each point and next 11 neighbors. If any of these distances is less than δ, update δ. return δ.

Improvement. Sort all the points twice before recursive call, once by x coordinate and once by y coordinate. Reuse the sorted sequences when needed (linear time)

$$T(n) \leq 2T(n/2) + O(n) \rightarrow T(n) = O(n\log n)$$

③ Integer Multiplication

Divide each n-digit integer into two ½n-digit integers (Karatsuba-Ofman, 1962)

$$xy = 2^n \cdot x_1 y_1 + 2^{\frac{n}{2}} \cdot \left((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0\right) + x_0 y_0$$

$$T(n) \leq T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + T\left(\left\lceil\frac{n}{2}\right\rceil\right) + T\left(1 + \left\lceil\frac{n}{2}\right\rceil\right) + \Theta(n)$$

$$T(n) = O\left(n^{\log_2 3}\right) = O(n^{1.585})$$

④ Matrix Multiplication

Divide each n-by-n matrix into four ½n-by-½n blocks

7 multiplications, 18 additions.

$$T(n) \leq 7T\left(\frac{n}{2}\right) + \Theta(n^2) \rightarrow T(n) = \Theta\left(n^{\log_2 7}\right) = O(n^{2.81})$$

Best known. $O(n^{2.3728596})$ [Alman & Williams, 2020]

Conjecture. $O(n^{2+\epsilon})$ for any $\epsilon > 0$.

⑤ Convolution and FFT

$$A(x) = A_{even}(x^2) + xA_{odd}(x^2).$$

$$A(-x) = A_{even}(x^2) - xA_{odd}(x^2).$$

Def. An nth root of unity is a complex number x such that $x_n = 1$.

Fact. The nth roots of unity are: $\omega^0, \omega^1, \ldots, \omega^{n-1}$ where $\omega = e^{2\pi i/n}$.

Fact. The 1/2nth roots of unity are: $v^0, v^1, \ldots, v^{\frac{n}{2}-1}$ where $v = e^{4\pi i/n}$. $\omega^2 = v$.

Conquer. Evaluate degree $A_{even}(x)$ and $A_{odd}(x)$ at the 1/2nth roots of unity: $v^0, v^1, \ldots, v^{\frac{n}{2}-1}$.

Combine.

$$A(\omega^k) = A_{even}(v^k) + \omega^k A_{odd}(v^k), 0 \leq k < n/2$$

$$A(\omega^{k+n/2}) = A_{even}(v^k) - \omega^k A_{odd}(v^k), 0 \leq k < n/2$$

Integer multiplication

Convert to binary polynomial, then multiply. O(n log n) complex arithmetic steps.

L05 Dynamic Programming

Top-down: May skip unnecessary sub-problems

Bottom-up: Save the overhead in recursion

① Weighted Interval Scheduling - $O(n \log n)$ [O(n) if pre-sorted start & finish]

Goal: find maximum weight subset of mutually compatible jobs.

Sort jobs by finish times so that $f_1 \leq f_2 \leq \cdots \leq f_n$.

Def. p(j) = largest index i < j such that job i is compatible with j.

Notation. OPT(j) = value of optimal solution to the problem consisting of job requests 1, 2, ..., j.

$$OPT(j) = \begin{cases} 0 & if \ \ j = 0 \\ \max\{v_j + OPT(p(j)), OPT(j-1)\} & otherwise \end{cases}$$

② Knapsack Problem - $\Theta(nW)$

Def. OPT(i, w) = max profit subset of items 1, …, i with weight limit w.

$$OPT(i, w) = \begin{cases} 0 & if \ \ i = 0 \\ OPT(i-1, w) & if \ \ w_i > w \\ \max\{OPT(i-1, w), v_i + OPT(i-1, w - w_i)\} & otherwise \end{cases}$$

③ RNA Secondary Structure – $O(n^3)$

[Watson-Crick.] A-U, U-A, C-G, or G-C.

[No sharp turns.] If $(b_i, b_j) \in S$, then i<j-4.

[Non-crossing.] If $(b_i, b_j), (b_k, b_l)$ are two pairs in S, then we can't have i < k < j < l.

Goal. Given an RNA molecule $B = b_1 b_2 \ldots b_n$, find a secondary structure S that maximizes the number of base pairs.

Notation. OPT(i, j) = maximum number of base pairs in a secondary structure of the substring $b_i b_{i+1} \ldots b_j$.

$$OPT(i, j) = \begin{cases} 0 & i \geq j - 4 \\ \max(OPT(i, j-1), 1 + \max_t\{OPT(i, t-1) + OPT(t+1, j-1)\}) & i < j - 4 \end{cases}$$

take max over t such that $i \leq t < j - 4$ and b_t and b_j are Watson-Crick complements.

Do shortest intervals first.

④ Sequence Alignment - $\Theta(mn)$ time and space

Edit distance. Gap penalty δ; mismatch penalty α_{pq}.

Def. OPT(i, j) = min cost of aligning strings $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_j$.

$$OPT(i, j) = \begin{cases} j\delta & if \ \ i = 0 \\ min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & otherwise \\ i\delta & if \ \ j = 0 \end{cases}$$

⑤ Sequence Alignment in Linear Space

O(m+n) space. Easy to compute OPT.

Recover alignment (combination of divide-and-conquer and dynamic programming):

-$O(mn)$ time to compute $f(\bullet, n/2)$ and $g(\bullet, n/2)$ and find index q.

- $T(q, n/2) + T(m - q, n/2)$ time for two recursive calls.

$$T(m, n) \leq cmn + T(q, n/2) + T(m - q, n/2)$$

⑥ Shortest Paths

Def. OPT(i, v) = length of shortest v-t path P using at most i edges.

$$OPT(i, v) = \begin{cases} \infty & if \ \ i = 0, v \neq t \\ 0 & if \ \ v = t \\ \min\left\{OPT(i-1, v), \min_{(v,w) \in E}\{OPT(i-1, w) + c_{vw}\}\right\} & otherwise \end{cases}$$

if no negative cycles, then OPT(n-1, v) = length of shortest v-t path.

$\Theta(mn)$ time, $\Theta(n^2)$ space.

O(n) extra space, $O(mn)$ time

Claim. Throughout the algorithm, M[v] is length of some v-t path, and after i rounds of

```
BELLMAN–FORD–MOORE(V, E, c, t)
FOREACH node v ∈ V :
    d[v] ← ∞.
    successor[v] ← null.
d[t] ← 0.
FOR i = 1 TO n – 1
    FOREACH node w ∈ V :
        IF (d[w] was updated in previous pass)
            FOREACH edge (v, w) ∈ E :              pass i
                IF (d[v] > d[w] + ℓvw)             O(m) time
                    d[v] ← d[w] + ℓvw.
                    successor[v] ← w.
    IF (no d[·] value changed in pass i) STOP.
```

updates, the value M[v] is <u>*no larger than*</u> the length of shortest v-t path using $\leq$ i edges.

⑦ Distance Vector Protocol

Bellman-Ford "Routing by rumor." each router performs n separate computations, one for each potential destination node. "counting to infinity" Each router also stores the entire path. Requires significantly more storage.

⑧ Negative Cycles in a Graph

Lemma. If OPT(n,v) = OPT(n-1,v) for all v, then there is no negative cycle with a path to t. If OPT(n,v) <OPT(n-1,v) for some node v, then (any) shortest path from v to t contains a cycle W. Moreover W has negative cost.

Theorem. Can detect negative cost cycle in O(mn) time. Add new node t and connect all nodes to t with 0-cost edge. Check if OPT(n, v) = OPT(n-1, v) for all nodes v.

- if yes, then no negative cycles
- if no, then extract cycle from shortest path from v to t.

L06 Network Flow

① Residual Graph, Augmenting Path, Ford-Fulkerson Algorithm

Def. The capacity of a cut (A, B) is: $\mathrm{cap}(A,b) = \sum_{e \text{ out of } A} c(e)$.

Flow value lemma.

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)$$

Weak duality. Let f be any flow. Then, for any s-t cut (A, B) we have $v(f) \leq \mathrm{cap}(A, B)$.

Max-flow min-cut theorem. The value of the max flow is equal to the value of the min cut.

Integrality theorem. If all capacities are integers, then there exists a max flow f for which every flow value f(e) is an integer.

Capacity Scaling: Let the Δ-residual graph $_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least Δ. $O(m^2 \log C)$ time.

② Bipartite Matching

$M \subseteq E$ is a matching if each node appears in at most one edge in M.

-Create digraph $' = (L \cup R \cup \{s,t\}, E')$.

-Direct all edges from L to R, and assign infinite (or unit) capacity.

-Add source s, and unit capacity edges from s to each node in L.

-Add sink t, and unit capacity edges from each node in R to t.

Perfect Matching: $|L| = |R|$, G has a perfect matching iff $|N(S)| \geq |S|$for all subsets $S \subseteq L$.

③ Extensions to Max Flow

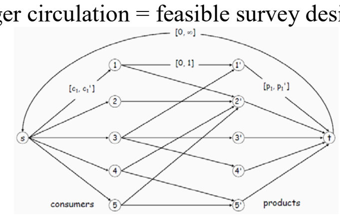
Circulation with Demands. Add new source s and sink t. For each v with d(v) < 0, add edge (s, v) with capacity -d(v). For each v with d(v) > 0, add edge (v, t) with capacity d(v). Claim: G has circulation iff G has max flow of value D. (saturates all edges leaving s and entering t)

Characterization. Given (V, E, c, d), there does not exists a circulation iff there exists a node partition (A, B) such that $\sum_{v \in B} d_v > cap(A, B)$.

Circulation problem with lower bounds.
Send $l(e)$ units of flow along edge e.
Update demands of both endpoints.左加右减

④ Survey Design

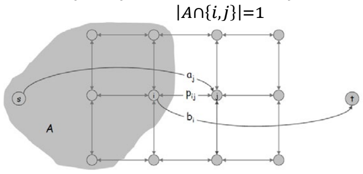
Integer circulation = feasible survey design.



⑤ Image Segmentation

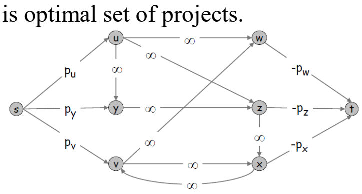
Find partition (A, B) that maximizes:

$\sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}|=1}} p_{ij}$



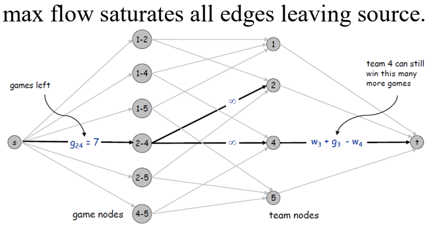
⑥ Project Selection

Min cut formulation. (A, B) is min cut iff $A - \{s\}$ is optimal set of projects.



⑦ Baseball Elimination

Assume team 3 wins all remaining games. $w_3 + g_3$ wins. Team 3 is not eliminated iff max flow saturates all edges leaving source.



Explanation for Sports Writers

Team z is eliminated iff there exists a subset T* such that $\frac{w(T^*)+g(T^*)}{|T^*|} > w_z + g_z$.

Define T* = team nodes on source side of min cut. Observe $x - y \in A$ iff both $x \in T^*$ and $y \in T^*$. $g(S - \{z\}) > cap(A, B)$.

L07 NP and Computational Intractability

① Polynomial-Time Reductions

Problem X polynomial-time <u>*reduces to*</u> problem Y if arbitrary instances of problem X can be solved using polynomial number of standard computational steps, plus polynomial number of calls to oracle that solves problem Y. (not reduce from)

② NP stands for nondeterministic polynomial-time.

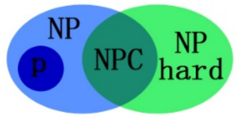
P. Decision problems for which there is a poly-time algorithm.

EXP. Decision problems for which there is an exponential-time algorithm.

NP. Decision problems for which there is a poly-time certifier.

③ Terminology

NP-complete. A problem in NP such that every problem in NP polynomial reduces to it. **NP-hard.** A problem such that every problem in NP reduces to it. **co-NP.** Complements of decision problems in NP. for *no* instance, there is a succinct disqualifier. (e.g. TAUTOLOGY, NO-HAM-CYCLE, PRIMES.)



- If NP≠co-NP, then P≠NP.
- $P \subseteq NP \cap$ co-NP.
- Factoring is in NP∩co-NP, but not known to be in P. (Factor: Given two integers x and y, does x have a nontrivial factor less than y?)

③ NP-complete.

Step 1. Show that Y is in NP. (polytime cert)
Step 2. Choose an NP-complete problem X.
Step 3. Prove that $X \leq_p Y$.

CIRCUIT-SAT. Given a combinational circuit built out of AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1?

3-SAT. Given CNF formula Φ, each clause contains exactly 3 literals, does it have a satisfying truth assignment?

INDEPENDENT SET. Given a graph G = (V, E) and an integer k, is there a subset of vertices$S \subseteq V$such that $|S| \geq k$, and for each edge at most one of its endpoints is in S?

VERTEX COVER. Given a graph G = (V, E) and an integer k, is there a subset of vertices$S \subseteq V$such that $|S| \geq k$, and for each edge, at least one of its endpoints is in S?

SET COVER. Given a set U of elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of U, and an integer k, does there exist a collection of $\leq$ k of these sets whose union is equal to U?

HAM-CYCLE. given an undirected graph G = (V, E), does there exist a simple cycle Γ that contains every node in V.

DIR-HAM-CYCLE. given a digraph G = (V, E), does there exists a simple directed cycle Γ that contains every node in V?

TSP. Given a set of n cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq$ D?

LONGEST-PATH. Given a digraph G = (V, E), does there exists a simple path of length at least k edges?

3D-MATCHING. Given disjoint sets X, Y, and Z, each of size n and a set $T \subseteq X \times Y \times Z$ of triples, does there exist a set of n triples in T such that each element of $X \cup Y \cup Z$ is in exactly one of these triples?

3-COLOR. Given an undirected graph G does there exist a way to color the nodes red, green, and blue so that no adjacent nodes have the same color?

k-REGISTER-ALLOCATION. Assign program variables to machine register so that no more than k registers are used and no two program variables that are needed at the same time are assigned to the same register.

SUBSET-SUM. Given natural numbers $w_1, \ldots, w_n$ and an integer W, is there a subset that adds up to exactly W?

PARTITION. Given natural numbers $v_1, \ldots, v_m$, can they be partitioned into two subsets that add up to the same value?

SCHEDULE-RELEASE-TIMES. Given a set of n jobs with processing time t_i, release time r_i, and deadline d_i, is it possible to schedule all jobs on a single machine such that job i is processed with a contiguous slot of t_i time units in the interval $[r_i, d_i]$?