Graphs & Algo cheat-sheet

Check bipartite: Look for odd cycles. G is bipartite iff G has no odd cycles. (2-colou r, P)
Independent Set: $I \subseteq V$ is independent iff $\neg\exists$ edge between vertices in I. NP
Max degree d, then $\mathbf{max}$ independent set $|I| \geq \frac{n}{d+1}$.

Vertex Cover: Set $S \subseteq V$ such that every edge $(v, w) \in E : v \in S \vee w \in S$. NP
S is a vertex cover iff $V \setminus S$ is independent.

Matching: Set $M \subseteq E$ such that no vertex V has more than one edge in E. P
(Also called *independent edge set* .)
Perfect Matching: Every vertex is exactly incident to one edge in M.
X**-saturating matching** : Match all vertices of X in a bipartite graph (X, Y).

- **max** Matching $\leq$ **min** Vertex Cover. (Equality holds for bipartite graphs.)
- Maximal matching of M? Then **min** vertex cover $|C|$ is: $|M| \leq |C| \leq |2M|$
- **Hall's Theorem** : G has an X-saturating matching iff $\forall S \subseteq X$: $|N(S)| \geq |S|$
- **Tutte-Berge theorem:** $\mathbf{max}\, M = \mathbf{min}_U \frac{1}{2}(n + U - o(G \setminus U))$
- **Tutte's matching theorem:** G has a perfect matching iff $\forall U \subseteq V$: $o(G \setminus U) \leq |U|$

Dominating set: Set $S \subseteq V$ such that each vertex is either in S or has a neighbou r in S.

Probability theory

$\binom{n}{k} \leq \frac{n^k}{k!}$ and $\binom{n}{k} \leq (\frac{en}{k})^k$ and $\binom{n}{k} \leq 2^n$ and finally for $p \in [0, 1] : (1 - p)^n \leq e^{-pn}$.

Markov's inequality: $\mathbf{Pr}[X \geq a] \leq \frac{\mathbb{E}[X]}{a}$ (X non-negative and $a > 0$.)

Probabilistic idea is to randomly assign (or assign with probability p) stuff:

1. $\exists i : n_i \geq \mathbb{E}[X]$ and $\exists_j : n_j \leq \mathbb{E}[X]$. Useful for expected sizes of certain sets.
2. Consider event $A(X)$. If $\mathbf{Pr}[A(X)] > 0$, then there must exist a value of X where A occurs. Similar vice-versa.

Lovasz Local Lemma (LLL): Consider n bad events $A_1, \ldots, A_n$. Futhermore, it holds that $\forall_{1 \leq i \leq n} : \mathbf{Pr}[A_i] \leq p$ and each A_i depends on at most d other events A_j.

- Then $ep(d + 1) \leq 1 \implies \mathbf{Pr}[\bigcap_i \bar{A}_i] > 0$.

Exponential time algorithms

3-colouring: Whether you can colou r the vertices of a graph with 3 colou rs.
Checking if $\forall X \subseteq V$ whether X is independent $\rightarrow$ if $G(V \setminus X)$ is `2-colourable`? $O^*(2^n)$

Vertex cover: Considering the k decision variant, i.e. vertex cover of at most size k.
Brute force: $O^*(2^n)$ by checking all 2^n subsets or all $\sum_{i=0}^{k} \binom{n}{k}$ subsets.
For a random edge, add either u or v and continue recursively on $k-1$ edges: $O^*(2^k)$.
For $u \in V$ where $deg(u) \geq 2$, either add u or add all neighbours: $O^*(1.62^k)$.

Cluster editing: Graph is cluster graph if all connected components are cliques. Look for X such that $(V, E \triangle X)$ is a cluster graph. Look for induced $P_3 : uv, vw \in E \wedge uw \notin E$. Recurse by either add uw, remove uv or remove vw on $k-1$: $O^*(3^k)$.

Feedback Vertex Set: Set $X \subseteq V$ such that $G[V \setminus X]$ is a forest, i.e. any cycle has $\geq$ one vertex in X. Solvable with *iterative compression*: given solution of $k+1$, compress to k.

Dynamic Programming + Inclusion / Exclusion

Knapsack: Given $w_1, \ldots, w_n$ and $v_1, \ldots, v_n$, pick $X \subseteq \{1, \ldots, n\}$ to maximize $\sum_{e \in X} v_e$ while $\sum_{e \in X} w_e \leq W$. $A[i, j]$ is $\textbf{max}\left\{\sum_{e \in X} v_e \textbf{ with } X \subseteq \{1, \ldots, i\} \textbf{ and } \sum_{e \in X} w_e \leq j\right\}$.

k-colouring: $A_k[X]$ is true of it has a k-colouring. Recursion: try all subsets $Y \subseteq X$ and look for $A_{k-1}[X \setminus Y] \wedge A_1[Y]$, where the latter checks if it is an independent set. $O^*(3^n)$.

TSP: Find cycle C such that $V[C] = V$ and $\omega(V[C])$ is minimized.
Walk: Sequence vertices such that their edges are consecutively connected on G.
Path: Walk while only visiting each vertex once.
Cyclic walk: Walk such that we start and end in the first vertex.
Cycle: Path such that we start and end in the first vertex.
Pick arbitrary s, define $A_t[X] = \textbf{min}\{\omega(E[P]) : P \textbf{ is a path from } s \textbf{ to } t \textbf{ using vertices } \mathrm{V[X]}\}$.
Recursion works by looking for the minimum path of: $\{\text{in-neighbours } t' + \omega(t', t)\}$ where t' also has to be in X. $O^*(2^n)$, as we need to consider all subsets $X \subseteq V$.

Inclusion/Exclusion: calculate union via sum of intersection and vice versa:

$$\left|\bigcap_{i=1}^{n} P_i\right| = \sum_{F \subseteq \{1, \ldots, n\}} (-1)^{|F|} \left|\bigcap_{i \in F} \bar{P}_i\right|$$

One can think of P_i as *good properties*. Note: $\bar{P}_i = U \setminus P_i$ and $|\bigcap_i P_i| = |U \setminus \bigcup_i \bar{P}_i|$.

Hamiltonian cycle: Consider P_i as a cyclic walk of length n visiting vertex i. Then $|\bigcup_{i \in F} \bar{P}_i|$ are all cyclic walks not visiting $F \equiv$ all cyclic walks of length n of $V \setminus F$.
DP solution: $w_F(s, t', k)$ is #walks $s \rightarrow t$ of length k, and recurse / sum over all $N^-(t) \setminus F$.

Treewidth

Tree decomposition: Pair X, T where $X = \{X_1, \ldots, X_l\}$ are bags, with $X_i \subseteq V$. T is the tree on X_i. 1. $\bigcup_{i=1}^{l} X_i = V$. 2. Edge in G? Then u and v should be in at least one X_i together. 3. All X_i containing v are connected.
Width: $\max_{i=1}^{n} |X_i| - 1$. **Treewidth:** Minimum width of all tree decompositions.

Cops and robbers: $w+1$ cops can win iff graph has treewidth $\leq w$.

We can create **Nice tree decompositions** in polynomial time:
Introduce: Bag with one child and $X_i = X_j \cup v$. **Leaf:** $|X_i| = 1$.
Forget: Bag with one child and $X_i = X_j \setminus v$. **Join:** Bag with two children, $X_i = X_j = X_{j'}$.

Given planar graph and rooted spanning tree S of G of at most height h ($\max$ # edges root $\to$ leaf), then a tree decomposition of G of width $\leq 3h$ can be found in poly time.
Removing vertices and edges from a graph does not increase treewidth.
Planar graph: Can be drawn without overlapping edges. ($O(n^2)$ or even $O(n)$)
Euler's formula: If G is planar and connected, then $n - m + f = 2$ (m edges, f faces).

Contracting an edge: merge u and v into w with neighbou rs of u and v.
Graph H is a minor of G if it can be obtained by contracting only.
Grid Minor Theorem: $\forall l \in \mathbb{N}:$ all planar graphs ha ve either $(l \times l)$-grid as a minor or Treewidth at most $9l$. This can be found in polynomial time
Kuratowski's theorem: G is planar iff it has no K_5 or $K_{3,3}$ as minor.

Randomized Algorithms (expected running times here)

Incremental construction: First randomly permute the input objects. Then add objects one by one in a list, maintaining a partial solution $1 \ldots i$. Then add $i+1$ to the list.

Backward analysis: Calculate $\mathbb{E}[\mathcal{I}_i]$ in backward direction. Cu rrently at $Sol(i)$, want to go back to $Sol(i-1)$. Sorting example: $s_1, \ldots, s_i$ are already sorted. Remove s_k, affects points $L_{k-1} = (s_{k-1}, s_{k+1})$. Prob of picking interval is $\frac{1}{i}$. $\mathbb{E}[\mathcal{I}_i] = \frac{|L_1|}{i} + \cdots + \frac{|L_i|}{i} \leq \frac{2n}{i}$.

Convex hull: For each point within C_i, make a bidirectional pointer to edge e of C_i, by intersection with origin. New point p_{i+1} outside? Delete intersected edge + points and walk along C to remove vertices until convex hull good again $O(n)$. Then update all pointers that pointed to the removed edges to the 2 new edges.
Backwards analysis: Ha ve $|C_i| \leq i$. Remove random point p_k. Inside? Done. On C_i? Remove two edges. Let $|e|$ be #points pointing to e. Then $\mathbb{E}[\mathcal{I}_i] = \sum_{e \in C_i} \frac{2|e|}{i} \leq O(\frac{n}{i})$.

Karger's min cut algorithm: Want to find S such that edges between S and $\bar{S}$ is minimized. **Contraction:** Same as in Treewidth, but we can ha ve multiple edges now. Choose random edge e and contract until 2 vertices are left. Suppose k min-cut. Then $\forall_{v \in V} d(v) \geq k$ and thus $m \geq \frac{nk}{2}$. $O(n^4)$. Speed up version after recu rsion: $O(n^2 \log n)$.

Probabilistic exponential time algorithms

*Algorithms that output **true** with a certain probability ($1 - \frac{1}{e}$), but never a false positive.*

Stirling's approximation: $n! \approx \binom{n}{e}$ for $O^*(n)$ algorithms.

k-path: Given directed graph, determine whether k-path exists. N
Note: for DAGs, solvable in $O^*(1)$ time via simple DP.
General k-path 1: randomly assign numbers $\{1, \ldots, k\}$ to vertices. Transform into DAG by only keeping consecutive edges. Look for k-path. Probability $\frac{1}{k^k}$, so need to repeat

this k^k time.
General k-path 2: randomly assign colou rs $\{1, \dots, k\}$ to vertices. Look for *colourful* k-path via DP: Check whether all different-colou red in-neighbou rs have a solution for $k-1$ colours, namely $X \setminus c(v)$. $O^*(2^k)$. Doing this check e^k times for c.e.p., so $O^*((2e)^k)$.

FVS 2: Lemma: $|X| \geq \frac{|E|}{2}$. Algorithm: first remove edge cases (v, v), $deg(v) \leq 2$. Then pick an edge + its endpoint at random, probability of $\frac{1}{4}$ of being in X. Recurse on $k-1$. Need to repeat this at most 4^k time as probability is $\frac{1}{4^k}$ of getting correct FVS. $O^*(4^k)$W

Isolation: given family $F \subseteq 2^U$ and weight function ω, ω isolates F if there is a unique $S \in F$ such that $\omega(S) = \min_{S' \in F} \omega(S')$. I.e. there is a unique set that has the minimum value of all sets in F.
Isolation Lemma: For every $e \in U$, choose $\omega(e) \in \{1, \dots, N\}$ at random. So randomly assignment numbers a set results in a unique minimizing set $\Pr[\omega \textbf{ isolates } F] \geq 1 - \frac{|U|}{N}$

Matrix multiplications

We can multiply two matrices in $O(n^\omega)$ time, where $\omega \leq 2.37$. ($\omega \approx 2.81$ in lecture notes)
For graph G with adjacency matrix A. Define $B = A^2$, then $b_{ik} = \sum_{j=1}^n a_{ij} a_{jk}$ is the # of walks on two edges from v_i to v_k. A^3 contains # of 3-walks.
Triangle count: Compute $trace(A^3)/6$, where the /6 comes from the fact that one can start at three vertices in two directions and $trace(A) = \sum_{i=1}^m a_{ii}$, sum of diagonal.
Square: We can compute the square of a graph G in $O(n^\omega)$ time. The square of G is G' where we put an edge between each vertex of distance ≤ 2.

Spectral Graph Theory

Laplacian Matrix: Diagonal entires are the degree of vertex i, while if there exists an edge between v_i and v_j, then $(L_G)_{ij} = -1$. Eigenvalues are $\lambda_1, \leq \lambda_2 \leq \dots \leq \lambda_n$, whose corresponding eigenvector v_i are orthogonal to each other and ha ve length $||v_i||^2 = 1$.

Laplacian quadratic form: For any vector $x \in \mathbb{R}^n : x^T L_G x = \sum_{(i,j) \in E} (x_i - x_j)^2$.
Can think of this as placing a value of x_i on vertex i, then $(x_i - x_j)^2$ defines the length of the edge. Also, we can decompose every graph as $L_G = \sum_{(i,j) \in E} L_{i,j}$.
Every eigenvalue is non-negative and it holds that: $\lambda_i = v_i^T L_G v_i$

$\lambda_1 = 0$ and v_1 is a scaling of the all-ones vector. For the rest: $\lambda_i = \min\limits_{x : x \perp v_1, \dots, v_{i-1}} \dfrac{x^T L x}{x^T x}$

Suppose the graph has k disconnected components, then $\lambda_1 = \dots = \lambda_k = 0$.

Let G be a connected, d-regular graph. $\lambda_n = 2d$ iff G is bipartite.

Expansion: $\phi_G(S) = \frac{E[S, V \setminus S]}{d|S|}$. For d-regular graph G: $\dfrac{\lambda_2}{2d} \leq \phi(G) \leq \sqrt{\dfrac{2\lambda_2}{d}}$