1. (20 points)

(a) (5 points) Prove that if G is an undirected graph with n vertices and n edges with no vertices of degree 0 or 1, then the degree of every vertex is 2.

(b) (5 points) Let G be an undirected graph with at least two vertices. Prove that it is impossible for every vertex of G to have a different degree.

(c) (10 points) In a group of 10 people, each one has 7 friends among the other nine people. Prove that there exist 4 people who are friends of each other.

LI. Yuntong
20944800

(a): Proof: ① Let $G=(V,E)$ be an undirected graph with n vertices and n edges.

② The sum of the degrees of all vertices in G is $2|E| = 2n$ (By the Handshaking Lemma).

③ Since no vertex has degree 0 or 1, the minimum degree of any vertex is at least 2.

④ If any vertex has a degree greater than 2, the sum of degree will exceed $2n$. which is contradicts the Handing Lemma.

⑤ Therefore, every vertex must have degree exactly 2

(b): Proof: G is an undirected graph with at least 2 vertices, every vertex has a unique degree.

n = number of vertices.

Degree of vertices in G will range from 0 to $n-1$.

If one vertex has degree $n-1$, no vertex can have degree 0, as it would contradict the connectivity of the graph.

This leaves $n-1$ possible degrees for n vertices, which is a contradiction.

Thus, it is impossible for all vertices to have different degree

(c): Proof: We can have: G has 10 vertices, each vertex has degree 7, meaning G is 7-regular graph. The total number of edge is: $\frac{10\times 7}{2} = 35$

Assume G has no K_4.

By Turan's Theorem, the maximum number of edges in a graph with no K_4 is: $\frac{n^2}{2}\left(1-\frac{1}{r}\right)$. $r=3$. $n=10$ $\Rightarrow$ $\frac{10^2}{2}\left(1-\frac{1}{3}\right)\approx 33.3$

G has 35 edges, which exceeds this bound. This contradiction implies that G must contain at least one K_4.

Thus, there exist 4 people who are friends of each other.

2. (20 points) Let $G = (V, E)$ be an undirected connected graph. Let n be the number of vertices in G. Let m be the number of edges in G. Design an algorithm to output a set of cycles $C_1, C_2, C_3, \ldots$ in G such that for every edge e of G, if e is contained in some cycle in G, then e is contained in some output cycle C_i. Explain the correctness of your algorithms. Analyze its running time which should be polynomial in n and m. Note that you are not required to output all cycles in G, and an edge of G may appear in multiple output cycles.

① Perfomed a DFS on G to find a spanning tree T of G.

② Identify the non-tree edges $e \in E \setminus T$

③ For each non-degree edge: add e to T. forming a cycle C. Tranverse the graph to find the unique cycle C formed by e in T. add C to the output set of cycles.

④. Output the set of cycles

Correctness: The spanning tree T ensures connectivity.
Each non-tree edge is processed to ensure all cycles edges.

Running Time: ① Building the spanning tree T: $O(n+m)$.
② processing each edge in $E \setminus T$: $O(m)$
Total: $O(n+m)$

3. (20 points) Let $G = (V, E)$ be an undirected connected graph with n vertices and m edges. Each edge in G is also given an non-negative integer weight. Given a path P in G from a vertex u to a vertex v, the *bottleneck weight* of P, denoted by $wt(P)$, is the minimum edge weight in P. A *maximum bottleneck path* between u and v is the path Q between u and v such that $wt(Q) \geq wt(P)$ for all paths between u and v. Our problem is to report the maximum bottleneck paths between all pairs of vertices in G. Show that this problem can be solved by finding the minimum spanning tree of some graph. Explain the running time of your algorithm.

① Compute the MST of G: Use Kruskal's or Prim's algorithm to compute the MST of G. Let the resulting MST be T.

② Preprocess the MST for path queries: Construct a Max-Edge Tree from the T. For each pair of vertices u and v, the bottleneck weight of the path between u and v in T is the maximum edge weight encountered along the path in T.

Use binary lifting or LCA to preprocess the MST for efficient path queries.

③ For each pair of vertices u and v. find the path between u and v in the T. The bottleneck weight of this path is the minimum edge weight along the path in T. return this value as the result for the maximum bottleneck between u and v.

Running Time: ① compute the MST: $O(m \log n)$

② Preprocess the MST: $O(n \log n)$

③ For query the maximum bottleneck path: $O(\log n)$

Total: processing the MST: $O(m \log n) + O(n \log n) = O(m \log n)$ [since $m \geq n$]

There are total $O(n^2)$ pairs of vertices, each query takes $O(\log n)$. Total query time: $O(n^2 \log n)$.

$\therefore O(m \log n + n^2 \log n)$.

4. (20 points) Let $G = (V, E)$ be a directed graph with positive edge weights.

   (a) (10 points) The cost of a cycle is the sum of the weights of edges on that cycle. A cycle is called shortest if its cost is the minimum possible, Design an algorithm to return the cost of the shortest cycle in G. If G is acyclic, your algorithm should say so. Your algorithm should run in $O(n^3)$ time, where n is the number of vertices in G. Explain the correctness of your algorithm. Derive its running time.

   (b) (10 points) Suppose that the edge weights in G are integers from the given range $[0, W]$. Describe an implementation of Dijkstra's algorithm that runs in $O((n + m) \log W)$ time.

(a) ① Use Floyd-Warshall to compute the shortest paths between all pairs of vertices

dist[u][v] = length of shortest path from u to v [This takes $O(n^3)$ time]

② For each edge $(u,v) \in E$, check if dist[u][v] $\neq \infty$.

if true: a cycle exists involving u and v, and its cost is: cost cycle = dist[u][v] + weight(u,v)

keep track of the minimum cycle cost.

③ if no cycle is found, the graph is acyclic

Correctness: Floyd-Warshall ensures that dist[u][v] is the shortest path from u to v. adding the weight of (u,v) gives the total cost of the cycle formed by $u \to v$ and $v \to u$

Time Complexity: Floyd-Warshall runs in $O(n^3)$

For checking the all edges for cycles runs in $O(m)$. [m is the number of the edges]

So in total, the complexcity is $O(n^3)$

(b) ① Initialize a distance array dist[v] = ∞ for all vertice v, except the sourse s, where dist[s] = 0

Initialize a bucket array B[0... W], where each bucket corresponds to a distance value modulo W.

② Insert the source vertex s into bucket B[0].

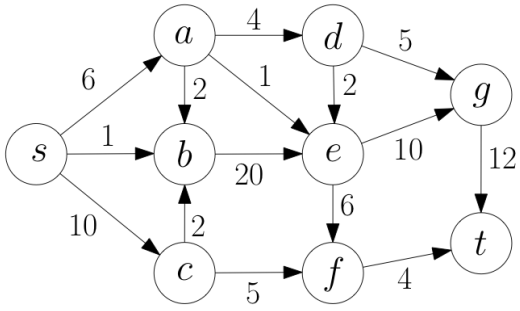
③ For each bucket B[i] (starting from i=0), process all vertices u in B[i]:

For each neighbor v of u, relax the edge (u,v): dist[v] = min (dist[v], dist[u] + weight(u,v))

Insert v into the appropriate bucket B[j], where $j = \left(\text{dist}[v]\right) \bmod W$.

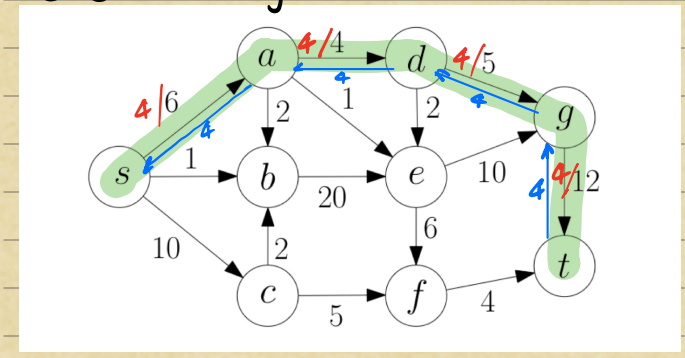
④ continue until all vertices are processed.

5. (20 points) Find the maximum flow from s to t in the following flow network. Determine the corresponding minimum cut as well. Follow the notation in the lecture slides to show your intermediate steps.

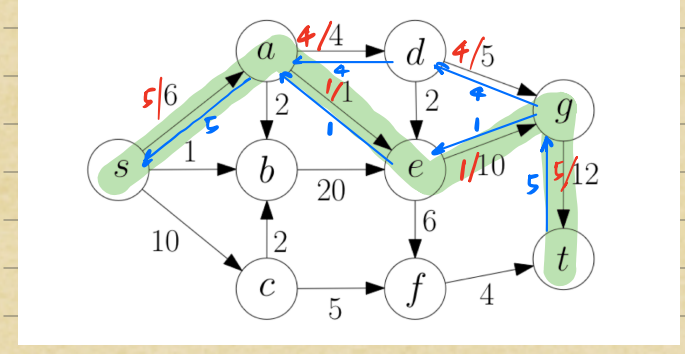


[For the maximum flow]

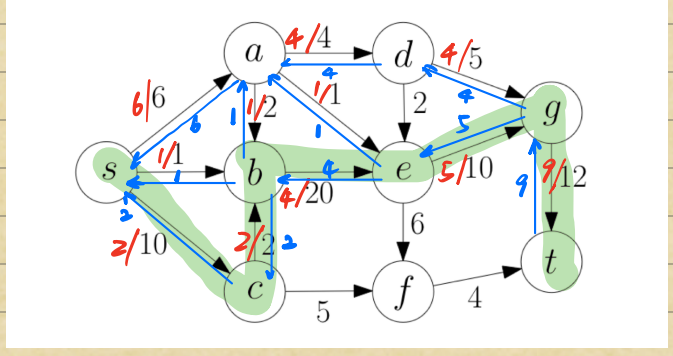
① $s \to a \to d \to g \to t$



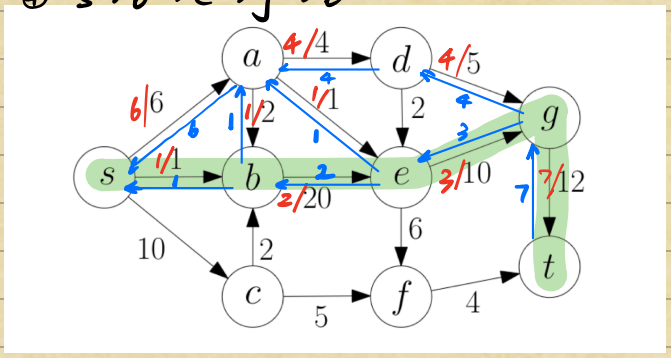
② $s \to a \to e \to g \to t$



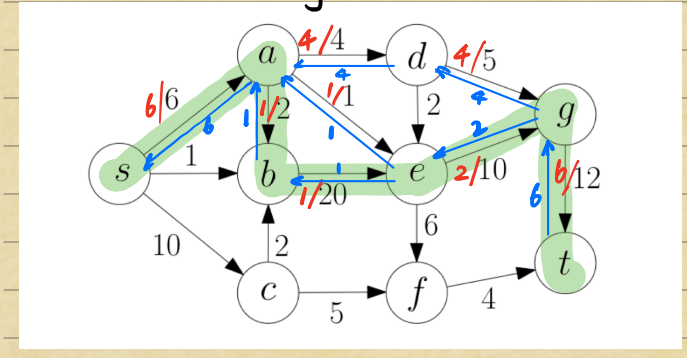
③ $s \to c \to b \to e \to g \to t$



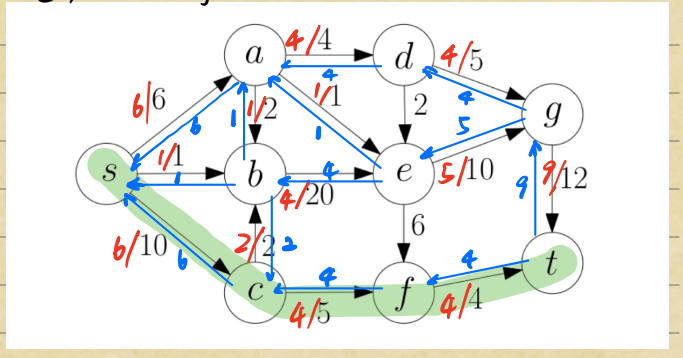
④ $s \to b \to e \to g \to t$



⑤ $s \to a \to b \to e \to g \to t$

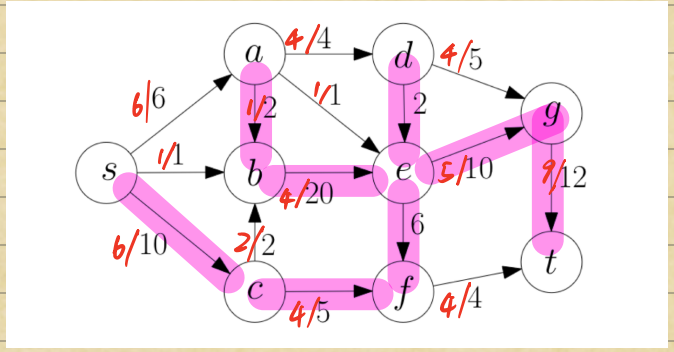


⑥ $s \to c \to f \to t$

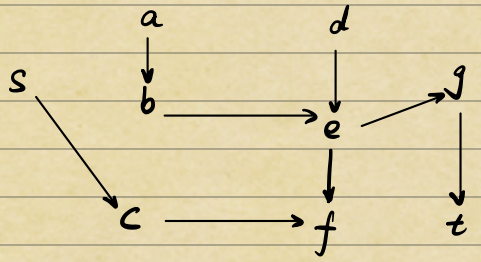


So: in total, the maximum flow is: $9+4=13$

[For the minimum cut].

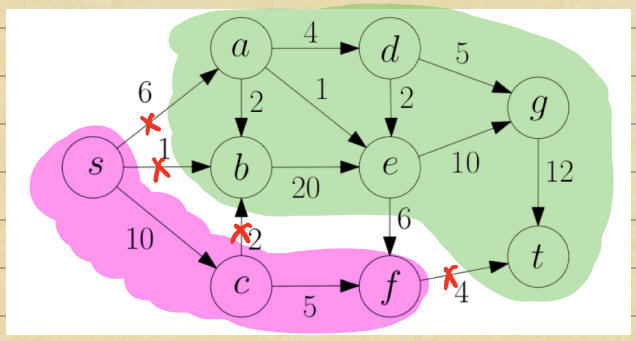


⇒



So: $S = \{ s, c, f \}$

$T = \{ a, b, d, e, g, t \}$



so: the minimum cut is: $6+1+2+4 = 13$ = maximum flow