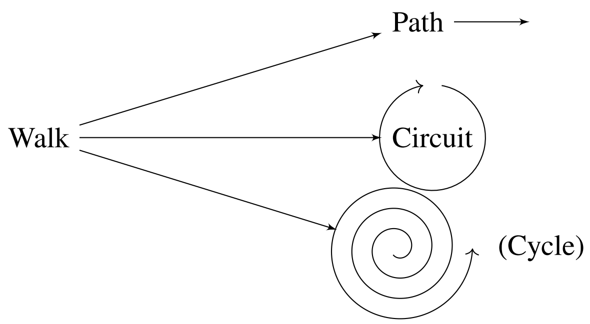


Graph G, undirected // **Digraph** D

$\{G, D\} = (V, E)$ // V, E finite // $e \in E$ as $ij : i, j \in V \Longleftarrow$ ordered pair for D not for G

Empty $E = \emptyset$ (all $v \in V$ are **isolated**); **Complete** K_n

Sub(di)graph $V' \subseteq V, E' \subseteq E$ // **Spanning** subgraph $V' = V$

Edges: have **endvertices** // may have **weights** technically a function

Adjacent vertices are (in-/out-)**neighbours** via their **incident** edges

Connected graph // Connected **component** $\Longrightarrow k_G$ # conn. cpts

Adjacency matrix (symmetric for graph, not for digraph) or **incidence** list (pair of incidence lists for digraphs) or $S - T$ **matrix for bipartite graph**

Tree $\Longleftrightarrow$ connected, has no circuits; **Forest** of trees; **Leaves** have degree 1

Tree in *digraph* may be rooted at ⓡ $\Longrightarrow$ everything except r has in-degree 1

Cuts: $\delta(U, W)$ is edges with one end in U and the other in W. In G, $\delta(U, W) = \delta(W, U)$ but not in D.

IF $V = U \cup W$: we have a **cut**: write $\delta^+(U), \delta^-(U) = \delta(U, W), \delta(W, U)$ [*digraphs*] NB: U, W non-empty, disjoint

in a digraph:

$U \subset V$ **separates** v from w: $v \in U, w \notin U$

(s, t)-cut is a cut $\delta^+(U)$ where U separates s from t

in a graph:

$U \subset V$ **separates** v from w: v XOR $w \in U$

(s, t)-cut is $\delta(S)$ for some subset S of the vertices separating s and t. **Capacity** of an (s, t)-cut is sum of caps on edges in the cut

Digraphs only: **Source**, **sink**

Flows **Network** = $D = (V, E)$ with specified s, t, with capacities $c_{ij} \geq 0$. All vertices other than s, t are **intermediate**. **Flow** f: $f(e) = f_{ij}$ s.t.

$$\sum_{i:ij\in E} f_{ij} = \sum_{k:jk\in E} f_{jk}$$

at intermediate vertices = **conservation equations**.

In *undirected* graph: edge directions: $G \to D$; + flow (in D)

Find that flow out of s = flow into t = **volume** $v(f)$ of the flow.

Feasible flow $0 \leq f_{ij} \leq c_{ij}$ // Feasible & max vol: **maximum** flow

Minimum cut = minimum capacity cut

$\lambda(G; s, t)$ capacity of a minimum (s, t)-cut [remember the cut itself is an edge set, $=\delta(A)$ for some vertices A]

Residual network (NB: not a network) $R(D, f)$ has the same vertices as D and edge ij if net flow can be increased

f**-alterable** path in residual network has **forwards** and **backwards** edges.

f-alterable path to t: f**-augmenting** path

(s, t)**-Edge-connectivity**: (D or G) min # edges that can be deleted to leave no (s, t)-path.

(s, t)**-Vertex-connectivity**: (D or G) min # vertices [NOT s, t] that can be deleted to leave no (s, t)-path.

Global edge conn.: (G): min # edges that can be deleted from G so G becomes disconnected

Current: assignment of a flow f_{ij} to edges s.t. there is a net flow of d_v for each vertex (*demands* on vertices) [c.f.conservation eqns $d_v = 0$ all v]

Circulation: in D a digraph w/o source/sink: flow f with $l(e) \le f(e) \le u(e)$, satisfying conservation equations

Circular if there is a directed circuit C of D with $f = \epsilon (> 0)$ round C and 0 otherwise.

Closed set in digraph $= F \subseteq V$ s.t. $i \in F, ij \in E \implies j \in F$. [it's a sort of *backwards* dependency: "finish what you'e started"]

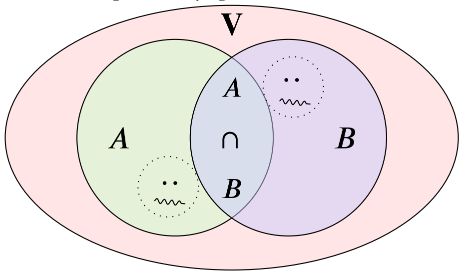
Now add **unit costs** to edges:

min. cost flow problem: feasible flow with given vol v at min cost

Vertex **identification**: glue them together into w, add edges wj for every merged vj (can then merge parallel edges)

$G_S = G$ with $v \in S$ identified

Legal ordering: start v_1 anywhere and then v_i has largest total capacity joining it with $v_1, \ldots, v_{i-1}$.



Gomory–Hu tree: weighted tree $T = (V, F)$ s.t. for any edge $e = st \in F$, take $T \setminus \{e\} = 2$ conn. cpts U, V, $\delta(U) = \delta(V)$ is a min (s,t)-cut.

for $R \subseteq V$, **Gomory–Hu tree for** G, R is T, and a partition of V into **parties** with **leaders**; R is the set of leaders

Subsets $A, B \subseteq V$ **cross** if $A \cap B$, $A - B$, $B - A$, $V - (A \cup B)$ are all non-empty

Matching M in G: edge subset; no two incident with same vertex;

vertices are **covered** by M or **exposed**.

Not to be confused with: a (vertex) **cover**: a subset of *vertices* s.t. you can get all edges. [*not* required: all vertices!].

Min. cover has fewest poss. vertices. think cut-and-cover

M may be **maximum**; a maximum matching may be **perfect**.

$\iff$ flows: M-alternating; M-augmenting;

General G (not bipartite): $k_o(G)$ (o for odd) is the number of conn. cpts with odd number of vertices.

blossom *for matching* M= circuit with odd number $2k+1$ of vertices and k edges of M. Has a **base**. identifying blossom to its base is **shrinking** it.

NB: Maxi*mum* (biggest) vs Maxi*mal* (can't add anything)

NB: Maximal might not be maximum: below left is maximal (can't add) but not maximum (there's an aug. path)

$\circ - \circ \sim \circ - \circ$ vs $\circ \sim \circ - \circ \sim \circ$

Big-O Exists $c > 0, A$ s.t.: $\forall x > A, |f(x)| \le cg(x) \implies O(g)$

Matroids & friends

Hereditary system: $I \in \mathcal{I}, J \subseteq I \implies J \in \mathcal{I}$ always contains empty set

e.g.: edge sets of spanning forests

e.g.: lin. independent columns of a matrix

Matroid: Hereditary system AND: for every $I, J \in \mathcal{I}$ s.t. $|I| < |J|$, can find an element in J to add to I and get a new ind. set

Transversal matroid: $(S, \mathcal{I})$ (for $G = (S, T, E)$): $I \subseteq S$ is independent $\iff$ there exists M in G s.t. $I \subset S(M)$

Has **rank** $r(M)$ = size of biggest ind. set. (Transversal) matroid of the (bipartite) graph G: mat(G)

We write $S(M)$ for the vertices of S covered by the matching M.

1. **Bubblesort**

In: List of n integers

for $i = n-1 : -1 : 1$:
bubble from $1 : i$ ($= i$ comparisons)

Out: Sorted list

Time: $(n-1)\sum_i i = \boxed{O(n^2)}$

Correctness: by induction

---

2. **Kruskal: min. cost spanning tree**

In: G: connected, weighted

$T \leftarrow \emptyset$;
while $E \neq \emptyset$:
delete cheapest edge e from E;
if $T \cup \{e\}$ has no circuits: $T \leftarrow T \cup \{e\}$

Out: $T \subseteq E$

Time: naively, $O(mn)$ (using heapsort; BFS for conn. cpt check); better: cpt labelling $\implies \boxed{O(m \log n)}$

---

3. **Prim:min. cost spanning tree**

In: G: connected, weighted; $v \in V$

$T \leftarrow \emptyset$;
while (V, T) not connected:
find cheapest edge e to link to T: $T \leftarrow T \cup \{e\}$

Out: $T \subseteq E$

Time: **?**

---

4. **GMST (red-blue) algo**

In: G: weighted, connected; all weights distinct (perturb if nec.

Red rule (Circuits): find circuit with no red edges; colour max. cost edge red
Blue rule (Cuts): find cut with no blue edges; colour min. cost edge blue

Out: Blue edges: $T \subseteq E$

Time: -

All edges get coloured!

---

5. **General greedy algo**

In: Hereditary system $M = (E, \mathcal{I}) +$ *non-neg.* weights

$I \leftarrow \emptyset$

while $E \neq \emptyset$:

delete costliest edge from E;

If $I \cup \{e\} \in \mathcal{I}$: $\mathcal{I} \leftarrow I \cup \{e\}$

Out: max weight $I \in \mathcal{I}$

Time: ?

vs Kruskal (See Ex 2.14 for how to $\leftrightarrow$):

max. not min — must be non-neg — Forest not Tree

Works iff: M is matroid

---

6. **BFS: shortest paths**

In: **Unweighted** D, r=start

Init: P(i) = -1 (0 for r)

Loop on queue Q:

Pull off head v of queue; for all out-neighbours of v, ?update P & add to queue

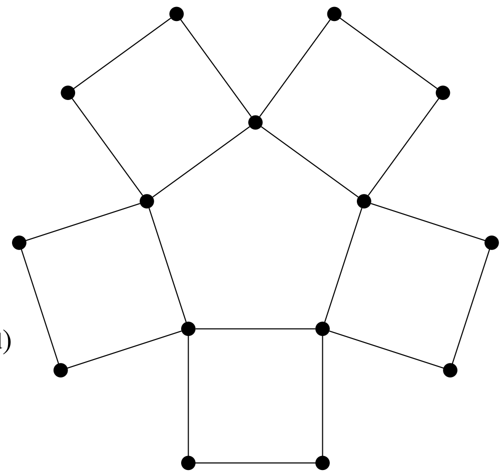
Out: Tree of paths to *all other* vertices as vector P of parents

Time: init n + all out-neighbours $m = \boxed{O(n+m}$

Can easily mod for G; can use to determine connectedness

BFS to find circuits in G:

- G connected: count the edges! No BFS needed ☺
- BUT to *find* the circuits: was e inspected but $e \notin$ output T?
  Then $e \in C$: trace up tree for common ancestor.
- G not connected: run BFS from some v; collect up circuits;
  throw away all $v \in$ tree found; rinse&repeat
- Shortest circuit? Try for all edges ij in turn: delete $e = ij$,
  find shortest ij path; compare. (NB this works *because* G is undirected)
- Shortest odd-length circuit: harder than you think!
  Consider *special walk* = length $2k+1$.



7. **Ex 3.3: TSP**

In: weighted D, alg A to find shortest *paths*

Init: replace all weights with -1;

Run A (we don't know what A is: -ve circuits could exist)

Out: i, j-path passing through all $v \in V$

Time: dep. A (Basically intractable)

---

8. **Ford-Bellman: shortest path tree**

In: weighted D with no -ve circuits, $r \in V$

Init: $u_r, P(r) = 0$;

all other $u_i = \infty$; all other $P(i) =$ -1;

for 1 : $n - 1$:

check **every** edge: ij: $u_i + c_{ij} \overset{?}{<} u_j$:

$\rightarrow$ update $u_j, P(j)$

NB: **Ex 3.8** Can stop if no changes or (sneaky) no changes except to sinks!

Out: Tree as vector P, lengths vector u_i

Time: $O(nm)$

Order of edges makes difference! But don't know "right" order until afterwards ;-)

Can alternatively be used to detect -ve circuits: run loop n times

---

9. **Dijkstra**

In: D with no -ve lengths; r

Init: $u_r, P(r) = 0$;

Temps T: $V - \{r\}$

u_i cost on edge from r, or ∞; $P(i) = r$ (*note difference from FB init!*)

for 1:n-2:

find min T, index is j: finalise j by:

Delete from T;

Check all remaining T: $u_j + c_{jk} \overset{?}{<} u_k$:

$\rightarrow$ update $u_k, P(k)$

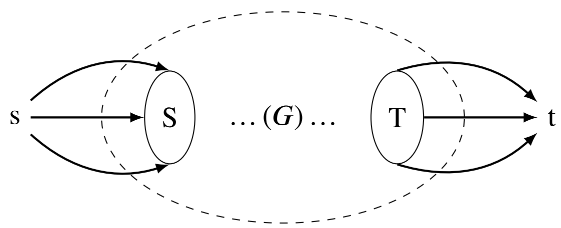
Out: **u** and P as F-B

Time: $O(n^2)$

Ex 3.10 Shortest path S to T

In: G, sets $S, T \subseteq V$

def G':



Find (s, t)-path in G'.

10. **Ex 3.11 Ordered vertex labelling**

In: D (no further constraints)

Init: identify sources by counting in-neighbours: list L
Loop: label a source $v \in L$, ditch it $\rightarrow D'$; add any new sources in D'.
Terminate: either out of vertices ✓ or no new sources ↻

Out: Vertex labelling

Time: $O(n^2)$ naively $\implies O(n+m)$

Smart runtime: use incidence lists, not adj. matrix $\rightarrow$ go through lists of in &out-neighbours of v

$\rightsquigarrow$ then this stage becomes $2m$ *total* (not each step)

---

11. **Ex 3.12 1 to everywhere $O(n^2)$**

In: $D, r \in V$

Init: $D \rightarrow D' = (V', E)$ s.t. $ij \in E \implies i < j$; take $r = 1$
Thm: $u_{ij} = min_{k:i<k<j}\{u_{ij} + c_{kj}\}$
for $j = 1 : n$:
set $u_{1j} = \min_{1:i<k<j}\{u_{1j} + c_{kj}\}$

Out: vector $\mathbf{u}$ of shortest paths

Time: $O(n^2)$ naively $\implies O(n+m)$

Smarter: use incidence lists of out-neighbours of j, instead of trying all values of k

$\rightsquigarrow$ as above this gives total $O(m)$ comparisons so we have $O(n+m)$ init, $O(n+m)$ run $\implies O(n+m)$

∘ **Ex 3.13 longest paths** Replace max by min and set cost to $-\infty$ if no edge

---

12. **Ex 3.14 Critical path analysis**

In: Project dependency D (NB unweighted) + weight vector (times) on V

Init: push weights from V to edges, eg with node-splitting trick
Find longest path: alg in 3.13
Its length is sum of weights of its vertices (found automatically with node-splitting version)

Out: Shortest possible project time (+ paths tree)

Time: $O(n+m)$

---

13. **Ex 3.15 Minimise the max. length of an edge**

In: D (no constraints); $r, s \in V$

Init: collect up edge lengths
Binary search: create subgraph D' by ditching all edges over length l; setting weights of the survivors =1; see if there's a path – BFS;
Terminate when there is a path with edges length l, but not $l-1$

Out: (r, s)-path minimising max. length of edge

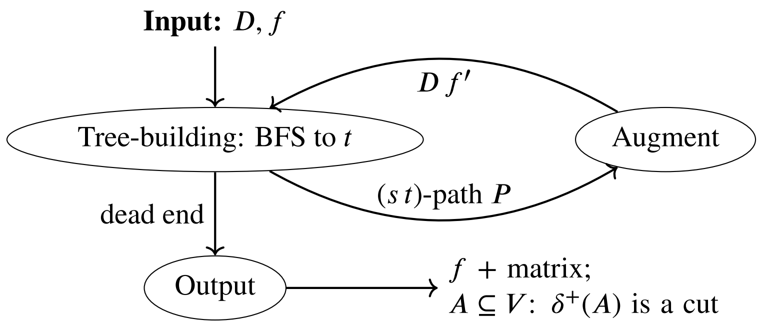
Time: $O((\log n)(n+m))$

Could try and find shortest path in winning D', but only if we know no -ve length circuits…

---

15. **Augmenting paths algorithm (Ford/Fulkerson)**

In: D = a *network*; f a flow(=0)



Out: max flow matrix, $C \subseteq V \to$ cut

Time: $O(nm^2)$ (Using BFS)

Converting to flow problems:

- **Edge connectivity**: give all edges cap 1
- **Vertex conn.**: split vertex trick, cap 1 between split
- **Closure *(project planning)***: add source+sink, give $s \to$ +ve v cap r, $t \to$ −ve v cap $-r$ (i.e. +ve value)

  $\sum +ve - v(flow) = \sum v \in$ closure = in cut $\neq s$
- **Current**: source → -ve, sink ← +ve *(opp. from closure!)*
- **Sports teams**: $s \xrightarrow{\text{wins needed}} \{\text{Teams}\} \times \{\text{Matches}\} \xrightarrow{\#\text{matches}} t$
- **Digraph building**: $s \xrightarrow{\text{out-degree}} \{V\} \overset{\max ij}{\times} \{V\} \xrightarrow{\text{in-degree}} t$

  Bipartite: put S and T rather than V both sides — don't forget backward edges on ∞ middles!
- **Matching**: $s \xrightarrow{1}$ (S) $\overset{\infty}{\times}$ (T) $\xrightarrow{1} t$

16. **Legal ordering**

In: G

Init: pick $v_1 \in V$ (arbitrary);

init table with other vertices

$n-2$ times: *last one is evident*

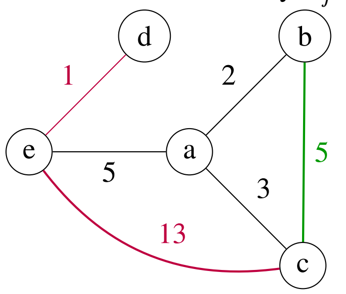
pick vertex v with max cap (current table row):

add to list;

create new table row: blank out v; for remaining vertices in table, entry $v_j + = cap(vv_j)$

	b	c	d	e
$[a]$	2	3	0	(5)
$[ae]$	2	(13)	1	-
$[aec]$	(7)	-	1	-
$[aecb]$				

$\Longrightarrow [a, e, c, b, d]$



Out: Vertex list $v_1, \ldots, v_n$

Time: $O(n^2)$

---

17. **Global minimum cut**

In: G, with caps ≥ 0

Init: $M = \infty$, $A = \emptyset$

Loop:

Find legal ordering; test $c(\delta(\{v_n\})) \overset{?}{<} M$:

$\rightarrow$ update M, new $A = \delta(\{v_n\})$;

Identify $v_n, v_{n-1} \rightarrow G'$; *loop*

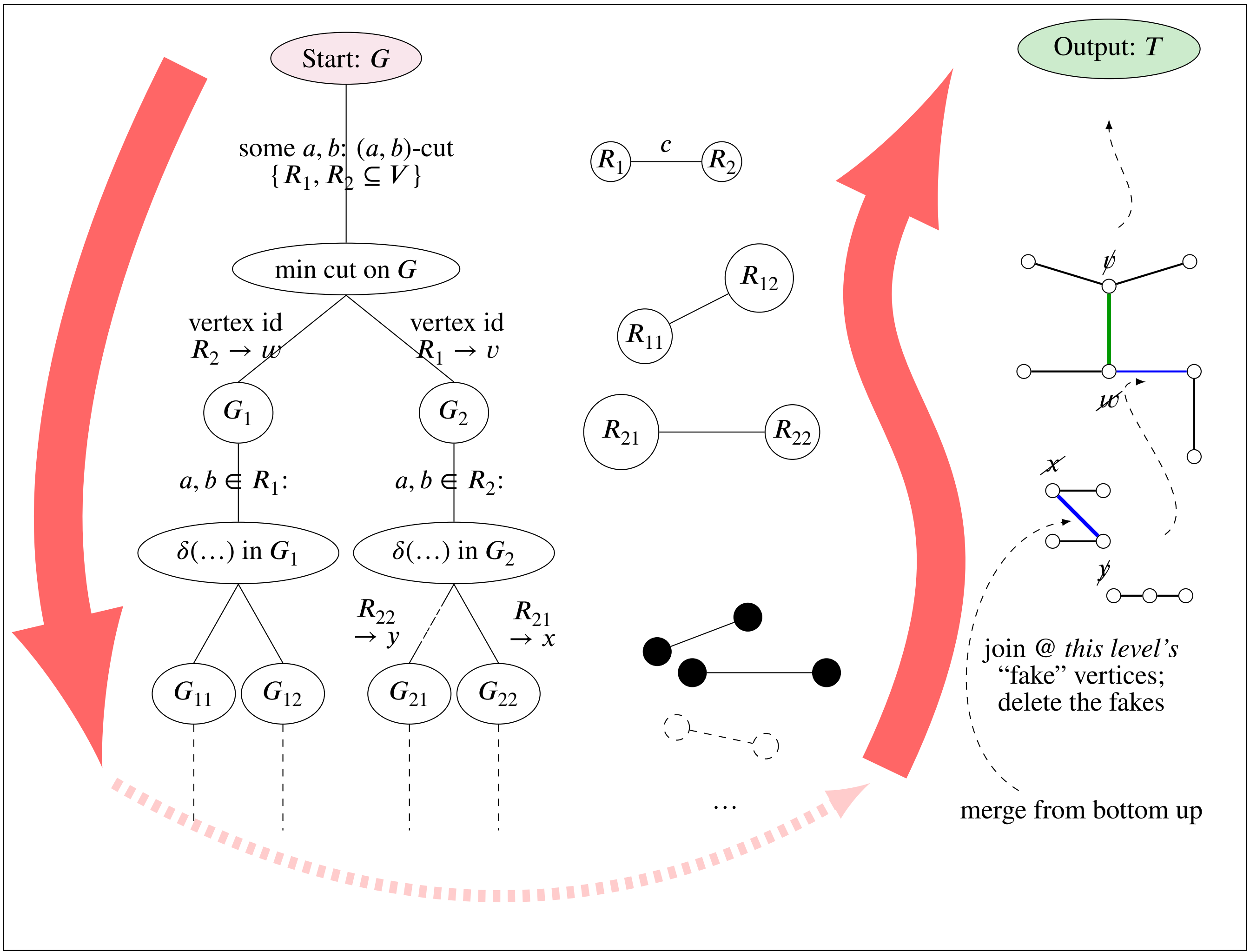
Out: $A \subseteq E$, a min cut

Time: $O(n^3)$

---

18. **Gomory–Hu**

In: G



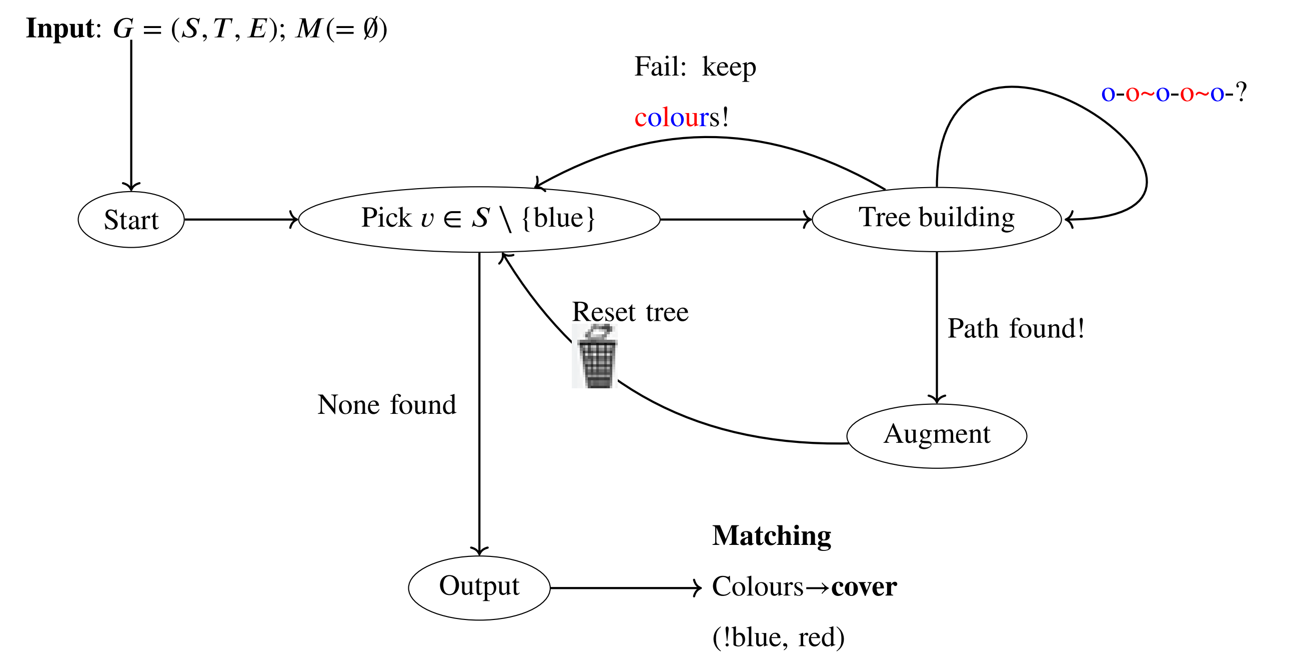
Out: T, a G–H tree

Time: $(n-1)$ min. cuts (e.g. via aug paths) (technically, $(n-1) *$ max. flow $=$ $\boxed{O(n^2m^2)}$

---

19. **Hungarian matching alg**

In: $G = (S, T, E)$ bipartite, $M (= \emptyset)$



Out: M, K

Time: $O(n^3)$

---

20. **Assignment problem: min. cost perfect matching**

In: $G = (S, T, E)$; $|S| = |T|$ $E = \{s_i t_j \,:\, s_i \in S, t_j \in T\}$ ("complete")

Init: Set up $\mathbf{u}, \mathbf{v}$: A good init: u_i as min. entry in ith row, v_i as min of $c_{ij} - u_i$ in each col.

LOOP:

- Calculate reduced cost matrix $\bar{c}_{ij} = c_{ij} - u_i - v_j$ use as basis of bipartite graph G_E;
- Seek perfect matching in G_E. FOUND $\implies$ DONE; else
- **update u, v**: C is vertices coloured during matching step:

  $\epsilon = \min\{\bar{c}_{ij} \,:\, S_i \in S, T_j \in T - C\}$:
  increase u_i at blue vertices by ϵ;

  decrease v_j at red vertices by ϵ:

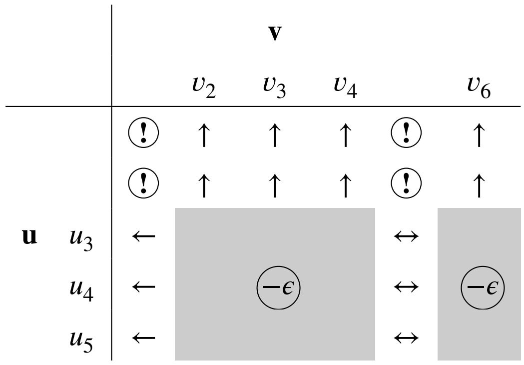
  NB: when updating $\bar{c}_{ij}$ on next loop:

  i, j used to calc. ϵ: 1 $\iff = \bar{c}_{ij} - \epsilon$

  i XOR j used to calc. ϵ: c_{ij}

  neither i nor j used to calc. ϵ: $c_{ij} + \epsilon$

- $\implies$ *loop*



Out: M a matching

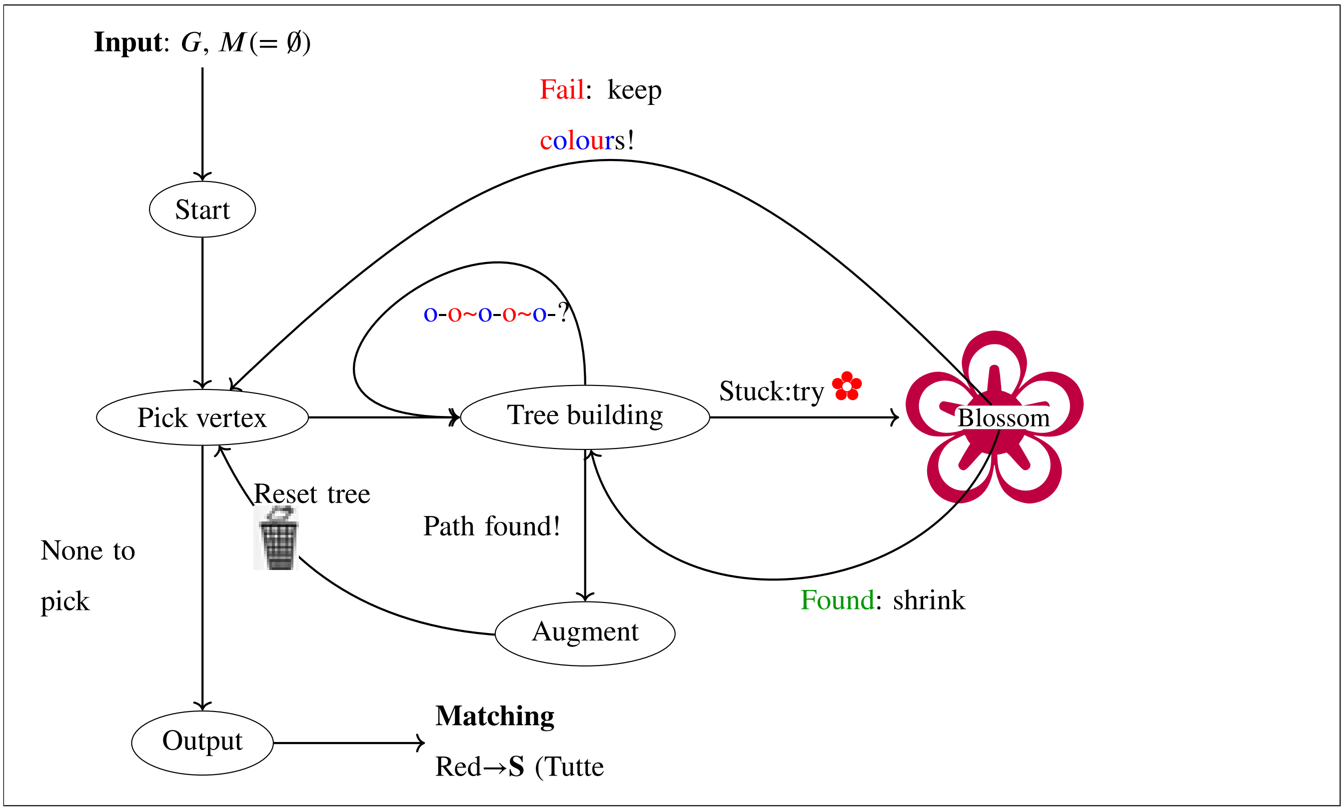
Time: $O(n^4)$

Also known as ***Hungarian alg*** for assignment problem

Check: $\sum_i u_i + \sum_j v_j = \text{cost}$ (**u**, **v** not updated after perfect matching found!)

21. **Edmonds' blossom algorithm**

In: G



Out: M, $S \subseteq V$

Time: $O(n^4)$

Shrink: $G \to G'$ and this $T \to T'$

Unshrink: round even # edges

---

22. **Build-up algorithm**

In: D a network (integral caps), int v target flow

Init $f = 0$ // empty flow matrix;

Build-up:

- construct $R(D, f)$ (with costs);
- find shortest path P (FB as -ve edges);
- augment along P;

⇝ if $f = v$ DONE;

else: *loop*

Out: Flow matrix

Time: $\boxed{O(fnm)}$ (Paths with FB)

Blank slate: build up to min. cost feasible v

Check: cost of flow*edges = sum of aug*path-cost

---

23. **Circuit-cancelling algo**

In: D a network (integral caps), f a flow mat @v

Loop:

- construct $R(D, f)$ (with costs);
- find -ve circuit C (FB with n iters):

if none: DONE;

⇝ augment round C; *loop*

Out: flow mat f

Time: ?

Lemma 1.2.2. Can break every walk down into paths and circuits

Lemma 1.3.1 // 1.3.6. 3-for-2

G has $n-1$ edges	$n-m$ edges
G is connected	has $k(G)=m$ conn. cpts
G has no circuits (is a tree)	is a forest

Lemma 1.3.4. T a spanning tree of G, add one "extra" edge e from $G \implies T \cup \{e\}$ contains unique circuit C; can remove *any* edge of C & get spanning tree.

Prop 1.4.1. i) f polynomial degree k, f is $O(x^i) \iff i \geq k$

ii) x^k is $O(e^x)$ ($k \in \mathbb{R}$)

iii) $\ln x$ is $O(x^\epsilon$ ($\epsilon > 0$)

Prop 1.4.2. f_1:$O(g_1)$, f_2: $O(g_2) \implies$

i) $f_1 + f_2$ is $O(\max\{g_1, g_2\})$;

ii) $f_1 f_2$ is $O(g_1 g_2)$

Ex 1.9. $n \leq \left(\frac{n}{e}\right)^n$

Ex 2.3. If weights are distinct, tree found by Kruskal is unique

Lemma 2.3.2. C edges of a circuit, C^* cut of a graph, $|C \cap C^*|$ is even

Thm 2.3.3. G connected: GMST colours all the edges and the blue edges form a min. cost spanning tree of G.

Ex 2.7. Kruskal & Prim are both special cases of GMST

Lemma 3.2.1. D has no -ve length circuit: if you can get $i \to j$, there's a shortest path. [just remove circuits] *of interest because algs find walks…*

Lemma 3.2.2. D has no -ve length circuits and a walk from r to everywhere in V: there is a collection of shortest paths from r to every other vertex whose union forms a tree rooted at r.

Ex 3.11. No directed circuits $\implies$ must be at least 1 sink / at least 1 source

Ex 3.11. Possible to input D and either relabel vertices s.t. $ij \in E \implies i < j$, or deduce that D has a (directed) circuit: runtime $O(n+m)$ [easier; $O(n^2)$]

Thm 3.6.1. D: no -ve weight circuits: FRW alg finds all shortest paths; runtime $O(n^3)$.

Prop 2.5.2. Graph G: hereditary system $(E, \mathcal{F})$ of spanning forests is a matroid

Prop 2.5.3. M a matrix over field F: (labels of) lin. ind. sets of columns $\implies$ matroid. Remember in binary field 2 cols are lin. ind. *unless* equal

Lemma 2.6.1. Hered. system $(E, \mathcal{I})$ is a matroid $\iff$ for every subset A of E, all maximal independent subsets of A have the same size.

Thm 2.6.2. Hered. system $M = (E, \mathcal{I})$ is a matroid $\iff$ for every non-negative weight function on E, the greedy alg determines the maximum weight ind. set.

Ex Assmt 1. Useful fact: $M = M(A) \implies$ there exists A' over F with $M = M(A')$ and A' has as many rows as the largest independent set in M [proof not given//in Assmt solns]

Thm 9.5.2. $G = (S, T, V)$ (back to *bipartite* graphs now): $\mathcal{I} = \{S(M) : M \text{is a matching}\} \to (S, \mathcal{I})$ is a matroid ("*transversal matroid*").

Prop 9.5.3. If M is a transversal matroid then there is a bipartite graph G such that $M = \text{mat}(G)$ with $|T| = r(M)$

Thm 4.3.3. Max-flow min-cut .

Thm 5.1.1. Integral capacities $\implies$ aug. paths takes $\leq v(f^*)$ iterations; f^* is integral

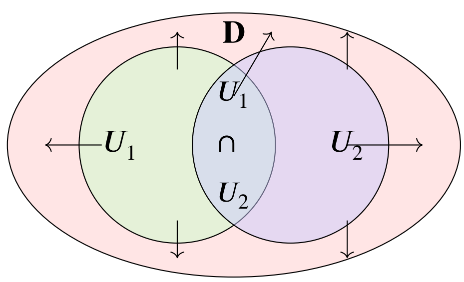
Cor. 5.1.2. Integrality theorem If all caps are integral, there is a max. flow in which all flow values are integral.

Lemma 5.1.4. Let $d(v, w)$ be the shortest path length in the residual digraph; let f, f' be a feasible flow and its augmentation: For every $v \in V$: $d(s, v) \leq d'(s, v)$ and $d(v, t) \leq d'(v, t)$

Lemma 5.1.5. Additionally, let $A(f)$ be the union of edges in $R(D, f)$ in all augmenting paths length d: if $d(s, t) = d'(s, t)$, then $A(f') \subset A(f)$

Thm 5.1.6. The augmenting path alg using BFS has runtime $O(nm^2)$

Ex 5.1. U_1, U_2 both separate s from t: $\text{cap}(\delta^+(U_1 \cap U_2) + \text{cap}(\delta^+(U_1 \cup U_2)) \leq \text{cap}(\delta^+(U_1)) + \text{cap}(\delta^+(U_2))$
Proof: think about drawing a picture



Lemma 7.4.3. $A, B \subseteq V(G)$: $c(\delta(A)) + c(\delta(B)) \geq c(\delta(A \cup B)) + c(\delta(A \cap B))$

Ex 5.3. f_1 and f_2 both max flows: sets of vertices to which they have f-alterable paths are identical.

Lemma 5.2.1. If every (s,t)-cut in D has infinite cap, there is an (s,t)-path containing only infinite cap edges.

Thm 5.2.2. Let D have an (s,t)-cut with finite cap: then (i) there is a max. flow (=min cap of a cut); (ii) if all finite caps are integral, there is an integral max. flow; (iii) need time $O(nm^2)$ to find it

Menger's theorems

Thm 5.3.1. (D) (s,t) edge connectivity = max # (s,t) paths with pairwise disjoint sets of edges *NB: thm, not the def!*

Thm 7.2.1. Same for (G)

Thm 5.3.3. (D) (s,t) vertex conn. = max. # internally disjoint (s,t)-paths.

Thm 7.2.3. Same for (G)

Page U7/5. Global edge conn. = cap of a *global* min. cut in G with all caps set to 1.

Thm 5.4.2. Gale's thm Current $\iff \sum_v d_v = 0$ and for every $S \subseteq V$, $\sum_{v \notin S} d_v \leq \text{cap}(\delta^+(S))$

(prove by $D \cup s, t \to D'$; cap of (s,t)-cut in D')

Thm 5.4.3. Hoffman circulation thm Given D and lower/upper bounds on each edge, there is a circulation $f \iff$ for every $S \subseteq V$, $l(\delta^-(S) \leq u(\delta^+(S))$.

Page U6/5. Team t can win the league $\iff$ the corresponding network has a feasible flow with volume $\sum_{ij} r_{ij}$ (r_{ij} are the remaining matches to play)

Page U7/2. $\delta(S)$ in G is an (s,t)-cut $\iff$ one of $\delta^+(S), \delta^-(S)$ is same in $D \implies$ capacity of cut in G = cap of cut in $D \implies$ max-flow min-cut holds.

Page U7/5. Possible to find a min. cut by solving $n-1$ max flow problems (pick some s, find the min (s,t)-cut for all choices of t, Bob's your uncle …)

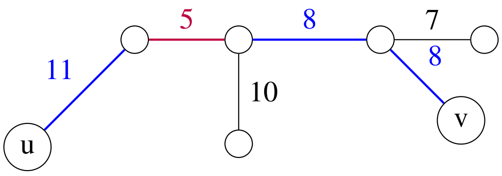
Page U7/5. 1:1 correspondence between cuts of G that are not (s,t)-cuts and cuts of $G_{s,t}$; preserves capacity

---

Lemma 7.3.5. $\lambda(G; u, v) \geq \min\{\lambda(G; u, w), \lambda(G; v, w)\}$

Thm 7.3.6. $v_1, \ldots, v_n$ legal ordering $(n! = 1) \implies \delta(\{v_n\}$is a min. (v_n, v_{n-1})-cut in G.

Lemma 7.4.1. let $v_0, \ldots, v_k$ be vertices of G: $\lambda(v_0, v_k) \geq \min\{\lambda(v_0, v_1), \lambda(v_1, v_2), \ldots, \lambda(v_{k-1}, v_k\}$

Thm 7.4.2. T a G–H tree: 

min edge on $(u - v)$ path is cap of min (u, v)-cut; vertices of the cut are everything on side of the edge

---

Lemma 7.4.4. $\delta(S)$ a minimum (s, t)-cut; $v, w \in S$; There is a minimum (v, w)-cut of the form $\delta(W)$ for $W \subseteq S$

Ex 7.12. $s, t, v, w \in V(G)$; $s \neq t$; $v \neq w$; $\delta(S)$ min (s, t)-cut: there is a min (v, w)-cut $\delta(T)$ for some T s.t. S and T do not cross.

Thm 7.4.5. $G = (V, E)$: for each $\emptyset \neq R \subseteq V$, there is a G–H tree for G, R

Thm 7.4.6. A G–H tree for G can be found by computing $n - 1$ min. cuts.

Page U8/1. G is bipartite $\iff$ no circuits with odd length.

Lemma 8.1.2. M a matching, P M-augmenting: $M' = M \triangle P$ is a matching with $|M'| = |M| + 1$.

Thm 8.1.3. Berge 1957 M is maximum $\iff$ no M-augmenting path

---

Lemma 8.1.4. M matching, K cover: $|M| \leq |K|$.

Ex 8.2. $S \subseteq V$ covered by some M: must exist maximum matching covering S.

Thm 8.2.3. König Max matching = min cover

Thm 8.2.4. Equivalently: binary matrix A rep $S \to T$: maximum sized set of ones from A with no two ones in same row or col is equal to min sized set of rows and cols containing every one of A.

Ex 8.6. Can solve matching problem using flows

Ex 8.7. In fact, Hungarian alg & augmenting flow alg do same thing (up to BFS)

Ex 8.8. König is cor. of max-flow min-cut

Thm 8.2.5. Hall perfect M in bipartite graph exists $\iff$ for every $X \subseteq S$, $|N(X)| \geq |X|$.

Cor. 8.2.6. $r \geq 0$; G has matching $|M| \geq |S| - r$ $\iff$ for every $X \subseteq S$, $|N(X)| \geq |X| - r$. Proof: stick r extra vertices in T

Ex 8.11. G bipartite; every vertex has degree k: G has k disjoint perfect matchings

---

Ex 8.13. If G is bipartite and every vertex has degree k: G has an edge colouring using k colours.

Ex 8.14. G bipartite has k disjoint perfect matchings $\iff$ for every $A \subseteq S, B \subseteq T$, there are at least $k(|A| + |B| - |S|)$ edges straddling A, B;

Lemma 9.1.1. Given $M, G, S \subseteq V$:

$$|M| \leq \frac{1}{2}(|V| + |S| - k_o(G \setminus S))$$

Lemma 9.1.3. S a blossom in G w.r.t. M; M_S-augmenting path P in M_S: P can be extended to an M-augmenting path of G by expanding the blossom.

Thm 9.4.1. Tutte, Berge For any G,

$$\max_M |M| = \min_{S \subseteq V} \frac{1}{2}(|V| + |S| - k_o(G \setminus S))$$

Ex 9.2. Tutte's theorem Necessary and sufficient condition for graph to have a perfect matching: for all $S \subseteq V$, $k_o(G \setminus S) \leq |S|$ AND even # of vertices (otherwise $k_o = 1$ for $S = \emptyset \implies$ we have already lost) (=**Thm 9.6.1**)

Ex 9.5. Petersen If every $v \in V$ has 3 neighbours and for every $e \in E$, $G \setminus e$ is connected: G has a perfect matching.

Ex 10.1. Shortest path problem is a special case of the min. cost flow problem

---

Lemma 10.1.2. f a circulation in $D = f_1 + \ldots + f_k$ with the f_i circular circulations

---

Thm 10.1.3. f feasible, vol. v has min. cost $\iff$ no -ve cost (directed) circuit in $R(D, f)$

Lemma 10.3.1. n iterations of F-B on D: D has -ve length circuit $\iff$ some u_i changes in the nth iteration, which can be found by climbing tree from i.

Thm 2.1.2. G connected; Kruskal finds min weight spanning tree.

Thm 2.2.1. Kruskal (can be) $O(m \log n)$ (G connected)

Thm 3.3.2. D with no -ve length circuits: F-B outputs length of shortest (r, v)-path for every v that has a path, plus a rooted tree of the paths; runtime $O(nm)$.

Thm 3.5.1. D with no -ve lengths: Dijkstra finds shortest paths from r to all other vertices; runtime $O(n^2)$.

Prop 7.3.4. A legal ordering of a graph G can be found in time $O(n^2)$

Thm 8.2.1. The Hungarian alg returns M, K with $|M| = |K|$; runtime $O(n^3)$

Prove: (1) M; (2) K; (3) $|M| = |K|$; (4) runtime

Thm 8.3.2. Hungarian algorithm for assignment problem (weighted edges) (1) determines an optimal solution (2) in time $O(n^4)$

Thm 9.3.1. Edmonds alg finds a max matching and S s.t. (Lemma 9.1.1) ==. Runtime $O(n^4)$

Proof: via 2 claims: (1) for each blue vertex v in a tree with root r there is an alternating path from r to v, first edge not in matching; (2) for each matching edge, either both endvertices are uncoloured or one is blue and one red; Mainly need to show that this preserved by blossom step

---

Thm 10.2.1. The build-up algo returns optimal flow and it's integral

---

Thm 10.3.3. Circuit-cancelling algo returns optimal flow of vol v and it's integral

---

Tips and tricks

- double up vertices and put capacity between them
- add source & sink and caps to them
  [also for finding paths to/from groups of vertices]
- turn into flow network
- finding paths by assigning edge weight 1
- finding most-edges paths by assigning edge weight -1 (FB)
- $O(n^2)$ vs $O(nm)$: is the graph sparse or close to complete?
- run aug path (flow/matching) alg once/to the end to get the list of interesting vertices // confirm result
- sportsteams draws → just pretend it's two matches
- MSc student problem with 2 types requirement = 2 types $s \to St \implies$ double up the student vertices
- Augmenting paths in G graphs: take care! Net poss change takes both directions into account (can change direction of arrow); flow of 0 can *always* add the edge (ditto).
- "Modify" $\implies$ **modify graph** then run vanilla alg, *don't mess with alg*!
- NB for thms: often need to specify:
  - non-empty
  - non-negative capacities
  - $u \neq v$