csakanny@cs.bme.hu

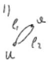
requirements:
1 midterm → ≥ 40% → signature (2 retakes)
cs.bme.hu \ itc2
ctg1

Graph Theory
networks (computer) → graph
roadmaps
acquaintances

set of abstract things — network
computers — connection
cities — roads
people — friendship

Def'n: a graph (a simple graph) $G=(V,E)$ is a pair of sets.
V = set of vertices of G (or points)
E = set of edges, which are unordered pairs (2-elements subset) of V

Drawing of G: 

This definition doesn't allow:
1)  $e_1=\{u,v\}$ → same set, $e_2=\{u,v\}$ parallel or multiple edges
2.)  $e=\{u,u\}$ – loop

a simple graph doesn't contain loops or multiple edges
(ways around: use multisets)

Terminology and notation
$e=\{u,v\}$ – e is incident u,v
– endpoints of e are u,v
– u,v are adjacent

$G=(V,E)$, $V\neq\emptyset$, V is finite

$V=V(G)$, $E=E(G)$ } emphasising G

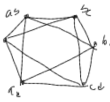
$|V|$, $|E|$ – # vertices, # edges

e.g.

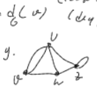
2.) 4 people, pairs of people if ∃ a common person → adjacent

graph: V = pairs
E = common person

$V=\{ab, ac, ad, bc, bd, cd\}$ $|V|=6$



Def'n: degree of vertex v of G ($v\in V(G)$)
= # of edges incident to v (loops are counted twice)
$= d_G(v)$ ($deg_G(v)$)

e.g. 
$d(u)=4$, $d(v)=3$, $d(w)=3$, $d(z)=4$, $|E|=7$
$\sum_{v\in V(G)} d(v)=14$

Thm (Handshake lemma)
$$\sum_{v\in V(G)} d(v)=2\cdot|E(G)|$$
(for arbitrary graphs)

proof: each handshake (= edges) is counted twice at the people (= vertices)

Corollary: The # odd degree vertices in any graph should be even.
proof: otherwise $\sum d(v)$ is odd, ↯ ($2|E|$ is even)

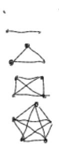
Def'n: $G=(V,E)$ is a directed graph, if the edges of G are ordered pairs of vertices.
notation: if $e=(u,v)$ $u \xrightarrow{e} v$
u → starting / initial point vertex of e
v → end point of e
$e=(u,v)\neq(v,u)=e_1$
out degree (v) = # edges with initial point
in deg (u) = 1
out deg (u) = 3

in deg (v) + out deg (v) = d(v)

$$\sum_{v\in V(G)} \text{in deg}(v)=|E|$$
$$\sum_{v\in V(G)} \text{out deg}(v)=|E|$$

Def'n: the complement of a simple graph $G=(V,E)$ is the graph $\bar{G}=(V',E')$, where $V'=V$ and E' = pairs of vertices which are not in $E(G)$

Def'n: the complete graph K_n on n vertices is $K_n=(V,E)$, when $|V|=n$ and $E(K_n)$ = all the possible pairs of V
i.e. $E(G)\cup E(\bar{G})=E(K_{|V|})$

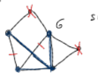
K_n

n=1 K_1
n=2 K_2
n=3 K_3
n=4 K_4
n=5 K_5

Proposition: $d_{K_n}(v)=n-1$ $\forall v\in V(K_n)$
(each vertex is adjacent to all the others)
(terminology): if $d_G(v)=k$ $\forall v\in V(G)$ then G is a k-regular graph
G is regular if it is k-regular for some k.)
2.) $|E(K_n)|=\frac{n(n-1)}{2}$ (Handshake lemma)

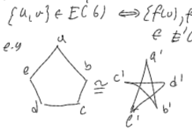
Corollary (for G)
1.) $deg_{\bar{G}}(v)+deg_G(v)=n-1$ $(d_{K_n}(v))$
2.) $|E(\bar{G})|+|E(G)|=\frac{n(n-1)}{2}$ $(=|E(K_n)|)$

Def'n: subgraph of graph $G=(V,E)$ is $G'=(V',E')$ if $V'\subseteq V$, $E'\subseteq E$
if $e=\{u,v\}\in E'$ then $u,v\in V'$
(the edge in the subgraph has to have it's original endpoints in G')

OR:
Def'n: a) deletion of a vertex from G:
delete vertex → delete all edges incident to it.
b) deletion of an edge from G
delete the edge, but keep the 2 endpoints of it.
then a subgraph of G: can be obtained by a sequence of vertex – and edge deletions from G.

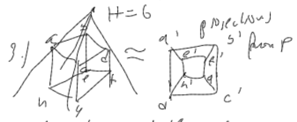


Def'n: if we delete only vertices from G (but not edges), then we get an induced subgraph of G
2.) if we delete only edges, but not vertices, we get a spanning subgraph.

Def'n: Isomorphism of graphs.
$G=(V,E)$ is isomorphic to $G'=(V',E')$, if ∃ a 1-to-1 mapping $f: V\to V'$, which preserves edges;
$\{u,v\}\in E(G) \iff \{f(u),f(v)\}\in E'(G')$
e.g.


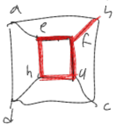
Graph Theory Review
$G=(V,E)$
Quiz: 1)
a) induced subgraphs of K_n
b) spanning subgraphs K_m, $0\le m\le n$
any simple graph on n vertices

2.) H is not induced and spanning subgraphs of G

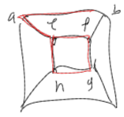
3.) 

bijection of the vertex preserving edges, i.e.
if $G\cong G'$ ⇒ 1. $|V(G)|=|V(G')|$
2. $|E(G)|=|E(G')|$
3. degree sequences are the same (not sufficient)
⇐ 4.) $\bar{G}\cong\bar{G'}$

Connectness of graph

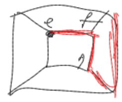
Def'n 1.) a walk in $G=(V,E)$ (or edge sequence) is a sequence
$(v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k)$
where $e_i=\{v_{i-1}, v_i\}$, $i=1\ldots k$
e.g.  e, f, g, h, e, f, b

b) a closed walk $v_0\to v_k$ ($v_k=v_0$)
c) The length of a walk: # of edges in it

2.) Trail in $G=(V,E)$ a walk without repeated edges
(vertices can be repeated but the edges cannot)
e.g.  e, f, g, h, e, a, b

b) a closed trail is called a circuit

3) a path in $G=(V,E)$ without is a walk without repeated vertices (⇒ ∄ repeated edges)

e.g.  e→b: e, f, g, c, b

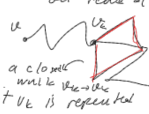
5. a closed path is a cycle
($v_0=v_k$ but otherwise all the vertices are different)
a cycle:  (C_n: cycle on n vertices)

Remarks:
C_0 ⟺ vertex
C_1 ⟺ loop
C_2 ⟺ parallel edges
C_3 ⟺ △
i.e. G is simple ⟺ C_n $n\ge3$

Propositions:
if ∃ a path a→b and b→c ($a,b,c\in V(G)$)
⇒ there is a path a→c

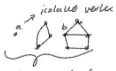
proof: a→b ∪ b→c = walk
There can be repetitions

more general statement: if ∃ a walk v→w then ∃ a path v→w as well
proof:
Get read of the repetitions



We get a shorter walk
repeat this, if necessary → all prov...
between 2 appearances of v_k → delete it

Def'n: The graph $G=(V,E)$ is connected, if ∃ a path between any 2 of its vertices
(remarks: we can use walks between vertices by Proposition)

e.g. 

it has 3 connected subgraphs "largest"
→ 3 compon

Def'n: A (connected) component of $G=(V,E)$ is a maximal induced connected subgraph of G
Def'n: a trivial component: isolated vertex ($deg(v)=0$ in a subgraph)
but not all components are necessarily isolated vertices!

remark: G is connected ⟺ it has 1 component

H is a component of G
⟺ connected subgraph of G, but $\forall v\in V(G)\setminus V(H)$ ∄ edge from v to V(H)

Different approach:
pick $v\in V(G)$
⇒ then subgraph induced by vertices which can be reached from v by a path
In the component of v

Proposition:
Components of v are components (by the Def'n) in G,

More precisely, if C is a component of ⟺ C = component of v

no proof:
but: the component of v are connected by paths

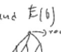

if ∃ path v→a, v→b } ∃ path a→b.

e.g.

component of e
i, h, g, f can be reached from e is a component

Corollary:
The components of G partition V(G) and E(G)

Tree

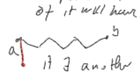


Def'n: a tree is a connected and no cycle-free graph

Remark: trees are simple
no loops or multiple edges (C_1, C_2)

Proposition: In a tree on $n\ge2$ vertices there are $n\ge2$ where ∃ ≥2 vertices of degree 1. (leaves)

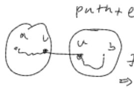
Proof: Consider a maximal (non extendable) path the 2 endpoints of it will have degree 1.



if ∃ another edge incident to a, then either:
1. other endpoint is not in the path → get a longer one ↯
2. other endpoint is on the path → get a cycle ↯
⇒ deg (a) = deg (b) = 1

Thm: if we add a new edge $e=\{u,v\}$ to $G=(V,E)$ ⇒
1. If u, v are in the same component ⇒ new cycle
2. If u, v are in different component → unite components (# of component decreases by 2)

proof: 1.  ∃ a path u→v, path + e → cycle

2.  if $a\in G_1$, $b\in G_2$ ∃ path a→u, v→b ⇒ get a path a→b

Thm: a tree on n vertices has n−1 edges.
proof. use thm 16


start from n isolated vertices, n components

we keep adding edges between different components, n−1 times → get a connected (1) and no cycles (2) } thm

Def'n: A forest: is a cycle-free (components are trees) graph
Def': a spanning tree in $G=(V,E)$ is a spanning subgraph which is a tree.

Prop 1: A forest on n vertices with k components has n−k edges (HW)
Prop 2: A connected graph on n vertices has ≥ n−1 edges.

Preview

$G=(V,E)$ is conn.: $\exists$ a path between each pair of vert.

components: max. con. induc. subgraph

G is connected $\Leftrightarrow$ has only 1 compont

$G=(V,E)$ is a tree - cycle-free graph - con

Proposition: a tree on $n\geq 2$ vertices has ≥ 2 vertices of degree 1 leaves

Theorem: a tree on n vertices has $n-1$ edges

Def'n a forest: a cyclic free graph

remark: if a forest is conn. $\rightarrow$ it is a tree

if a forrest is not connected $\rightarrow$ component of it a tree

Proposition:

a forest on n vertices with k components has $n-k$ edges

(remark: if $=1\rightarrow$ it is a tree $\rightarrow |E(G)|=n-1$)

proof: start with n isolated vertices, add edges 1 by 1 between components ($\sim$ proof of Thm)

$\rightarrow$ by adding an edge we decrease the number of components by one

so: 0 edge $\rightarrow n$ components

1 edge $\rightarrow n-1$ compont,

$n-k$ edg. $\rightarrow k$ compont.

Corollary: A forest on n vertices contains $\leq n-1$ edges

if it contains (exactly) $n-1$ edges $\Rightarrow$ it is a tree

Def'n: a spanning tree of the graph $G=(V,E)$ is a spanning subgraph $\Rightarrow$ (contains all vertices of G) which is a tree

remark: 1. only conn. graph can have a spanning tree (already it's subgraph is conn)

2. if G is not conn $\rightarrow$ then it has a spanning forest (sp. tree in each component)

Thm: a graph $G=(V,E)$ contains a spanning tree $\Leftrightarrow$ it is connected

proof: $\Rightarrow$: ✓ (remark)

$\Leftarrow$: either add edges 1 by 1 from $E(G)$

OR: delete edges 1 by 1 from cycles of G until we get a cycle-free but still connected graph

if we delete an edge from a cycle $\rightarrow$ remove the graph C' is still conn.

delete $e=\{v,w\}$

$\Rightarrow \exists$ a path $v\rightarrow w$ in G'

and from $u'\rightarrow w'$:

$\exists$ path, $u'\rightarrow u, v\rightarrow v, v\rightarrow u'$

if no more cycles $\rightarrow$ stop

remaining: - cycle-free
- connected
- contains all V

} spanning tree

Corollary:

if conn. graph on n vertices $\geq n-1$ edges and if $|E(G)|=n-1$ $\Rightarrow G=T$

Trees are minimally connected graphs

Thm: The following are equivalent for a graph $G=(V,E)$.

1) $G=T$ (cycle-free, connected)

2) G is connected and $|E(G)|=n-1$
↳ a spanning tree "minimally connected graph"

3) G is cycle-free and $|E(G)|=n-1$
↳ forest with $n-2$ edges "maximally cycle-free graph"

4) G is conn. $\forall$ edge = bridge (if we delete it we get 2 components)

proof: 1.) $\Rightarrow$ 2.) corollary 2 (spanning tree)
2.) $\Rightarrow$ 1.) — 11 —

1.) $\Leftrightarrow$ 3) forest with $\geq n-1$ edges

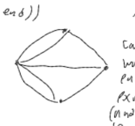
(1. - 4.) $\sim$ proof of $|E(\text{tree})|=n-1$

↳ then delete it $\Rightarrow$ we get an an Euler trail in G

Story: "bridges of Königsberg"

Can we walk through each of the bridges exactly once (and get home in the end))

⇕



Can we walk through each edge exactly once (and get to the starting vertex)

Def: 1) an Euler trail is a trail containing each edge of G (exactly once).

2.) An Euler circuit in a graph $G=(V,E)$ is a circuit containing each edge of G (exactly once)

Remark: an Euler trail can be closed (i.e. a circuit)

$\Leftrightarrow$ 1.) G is connected (apart from isolated vertices)

2.) the degree(v) = even

Proof: $\Rightarrow$: necessity: 1) otherwise if $\exists$ an edge in 2 different components, then we cannot get from one to the other ① — ②

2. if we start a trail from a vertex v when we use 2 edges incident to it, we can pass through more than one), if we use all the edges adjacent to it

$\Rightarrow d_G(v)=2+2+...+2$ = even

since v = starting point = end point in a circuit:

$\Rightarrow \deg(v)=1+2+...+2+1$ = even

$\Leftarrow$: sufficiency: an "algorithm" to connect an Euler circuit in G

Lemma: If all the degrees are even in G, a maximal trail in G can end only in the starting vertex

(if you enter a vertex $\rightarrow$ we odd # of edge $\rightarrow$ you can leave, if the degree is even)

idea: start from an arbitrary vertex v, and go on a non-extendable = maximal trail

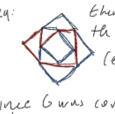
$\Rightarrow$ by lemma $\Rightarrow$ we end up in v (i.e. get a circuit)

2 cases:

- if this circuit contains all the edges $\rightarrow$ it is an euler circuit. ✓

- if we haven't visited all the edges yet:

eq:



Edges in $G\setminus C$ (circuit) all the degrees are even (even - even = even)

Since G was connected

$\Rightarrow$ there must be an edge: $\{u,w\}$ s.t. $u\in V(C)$ but $\{u,w\}\notin E(C)$

$\Rightarrow$ start a trail in u in $G-C\rightarrow$

$\rightarrow$ it can get stuck only in u by lemma

$\rightarrow$ get a circuit C' in $G\setminus C$

$\Rightarrow$ extend our original circuit C: start from v on C $\rightarrow$ if we get to u, we the edges of C' $\rightarrow$ finish C.

$\rightarrow$ if this is not an Euler circuit $\rightarrow$ continue.

$\rightarrow$ eventually, it will contain all the edges of G $\rightarrow$ Euler circuit.

Thm. $G=(V,E)$ contains an Euler trail $\Leftrightarrow$

$\rightarrow$ G is connected (apart from isolated vertex)

$\rightarrow$ G contains ≤ 2 vertices of odd degree.

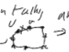
remark: $\Rightarrow$ the # of odd-degree vertices can only be 0 or 2 (1 is impossible, by the handshake lemma)

Proof: $\Rightarrow$: necessity: same

if starting vertex $\neq$ end vertex $\Rightarrow$ their degrees are odd, but the other degrees are even.

$\Leftarrow$: if $\nexists$ odd-degree vertices, then by previous th $\Rightarrow \exists$ Euler circuit, which is also an Euler trail.

- if $\exists$ 2 odd-degree vertices: u,v then add the edge $\{u,v\}$ to $G\rightarrow$ get G' are even $\Rightarrow$ by prev. th. it contains an Euler circuit



$\Rightarrow$ start it with the edge $\{u,v\}$

Condition for directed graphs

Thm: a directed graph G contains a (directed) Euler circuit $\Leftrightarrow$

$\Leftrightarrow$ 1.) G is con. as an undirected graph (+ no isolated vertex)

2.) indeg(v) = outdegree(v) $\forall v\in V(G)$

(for the underlying undirected graph)

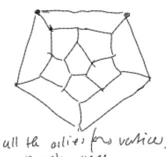
deg(v) = indeg(v) + outdeg(v) $\rightarrow$ even)

Hamilton Cycles and trails

Def'n 1.) A Hamilton cycle in $G=(V,E)$ is a cycle containing all the vertices of G. (exactly once)

2.) Hamilton Path contains all the vertices of G (exactly once)

game



Visit all the cities (vertices) exactly once.

TSP: traveling salesman problem $\rightarrow$ difficult

No necessary and sufficient conditions

Necessary conditions:

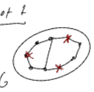
Proposition 1:

if G contains a H cycle, then if we delete k vertices from G, we get $\leq k$ components $(k=1,\dots,\frac{n}{2})$

Proposition 2:

if G contains a H trail, then if we delete k vertices from G, then we can get $\leq k+1$ components $(k=1,\dots,\frac{n}{2}-1)$

proof of 1



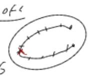
for $k=1$ $\rightarrow$ all the remaining vertices are on one path

if $k=2\rightarrow$ 2 path remain $\rightarrow$ 2 connected vertices

⋮

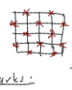
for arbitrary k: path between consecutive vertices remain $\rightarrow$ connected vertices.

proof of 2



From the 2 endings of the path we can get 1 more component (at most)

e.g. square grid



delete 12 vertices

⇓

get 13 isolated vertices

$\Rightarrow$ no H cycle

remarks:

remark: 1.) These condition are not sufficient

Petersen graph condition hold but $\nexists$ H cycle } HW

2.) to use: if G does not contain H cycle/path

H∈G ⇒ ≤ k compont.
≥ k compont. ⇒ H∉G

Hamilton cycles

sufficient conditions:

Thm (Ore) if G is a simple graph on $n\geq 3$ and

⊛ $\deg(u)+\deg(v)\geq n$ $\forall u,v$ $u\nleftrightarrow v$

$\Rightarrow$ G contains a H-cycle

proof: by contradiction:

⊛ holds, but $\nexists$ H cycle in G.

if we keep adding edges to G s.t. the end graph G'

- still has no H-cycle

- but by adding any new edges (it would)

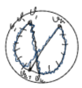
(G' is the saturation of G)

$\rightarrow$ ⊛ hold for G' as well.

if u,v are nonadjacent vertices in G' then $G'\cup\{u,v\}$ has a H-cycle

$\rightarrow G'$ has a H-path with endpoints u,v

G



claim if $\{u,v_k\}$ and $\{v,v_{k+1}\}\in E(G')$

$\Rightarrow \exists$ a H cycle in G'

$v, v_2, v_3, \dots v, v_{k-1}, v_{k-2}, \dots u$.

$\Rightarrow$ by cond on G' ($\nexists$ H cycle)

$\Rightarrow$ cannot have both edges

$\{u,v_k\}$, $\{v,v_{k+1}\}$ in $E(G')$

so if $\deg(u)=d$ $\rightarrow$ then $\deg(v)\leq n-1-d$
no edges to preceding vertices

$\Rightarrow \deg(u)+\deg(v)\leq n-1$, $\{$ w/ ⊛

Th (Dirac) if G is a simple graph on $n\geq 3$ vertices and $\deg(v)\geq\frac{n}{2}$ $\forall v\in V(G)$

$\Rightarrow$ G has a H-cycle

proof: Corollary of Ore's thm.

if $\deg(v)\geq\frac{n}{2}$ $\forall v$

check: ⊛ $\deg(u)+\deg(v)\geq\frac{n}{2}+\frac{n}{2}=n$

$\forall u,v\in V(G)$ $\Rightarrow$ by Ore's Th $\exists$ a H cycle in G.

Remarks: 2) these condition are sufficient but not necessary:

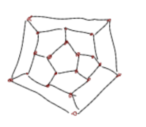
e.g. a cycle C_n $n\geq 5$

- $\exists$ H cycle

$\deg(v)=2$ $\forall v\in V(G)$

$\deg(v)+\deg(u)=4<n$

2) Use the to prove that a G has H cycle.



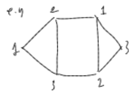
Graph Coloring

vertices, edges, both, ...

Def'n a (proper) coloring of the vertices of G is assignment of colors to the vertices of G s.t. adjacent vertices get different colors.

(edges ~ contacts $\Rightarrow$ want to distinguish the vertices)

e.g



$-G$ 3-colorable (can be colored with 3 colors)

remarks: 1.) can use numbers of colors

2.) no loops $\rightarrow$ (cannot color vertex)

Multiple edges don't matter

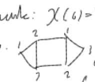
$\Rightarrow$ we will color simple graphs

Def'n: the chromatic number of $G=(V,E)$ is the min number of colors needed to (properly) color G.

notation: $\chi(G)$

remark: $\chi(G)=k$ ↗ 1) G is k-colorable
2) G is not $k-1$ colorable

e.g.



could color with 3-colors $\Rightarrow \chi(G)\leq 3$

$\chi(G)=3$ { 2 colors are not enough. for triangles we need 3 colors $\Rightarrow \chi(G)\geq 3$

2) K_n (complete graph)

$\chi(K_n)=n$ $\rightarrow$ n colors are enough

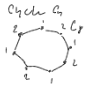
1) $\Rightarrow \chi(G)\leq n$

- $n-1$ is not enough:

each pair of vertices is adjacent $\Rightarrow$ each has to get a different color $\Rightarrow^{2)} \chi(K_n)\geq n$

$\Rightarrow$ 1.2) $\chi(K_n)=n$

Cycle C_n



$\rightarrow \chi(C_{even})=2$
$n\geq 2$

$\rightarrow \chi(C_{odd})=3$

in general: $\chi(C_n)=\begin{cases}2, & \text{if } n \text{ is even}\\ 3, & \text{if } n \text{ is odd}\end{cases}$

$\chi(G)\geq 3\rightarrow$ difficult question

can give bounds on $\chi(G)$

$\rightarrow$ lower bound: $\chi(G)\geq$ size of a maximum complete subgraph $=\omega(G)$

(Def: ω = # the number of vertices in a max complete subgraph) clique

e.g. 1) $\omega(C_n)=2$, $n>3$

$\omega(K_n)=n$

- upper bound: by greedy coloring

(assigning the vertices the smallest possible color in a given order.

$\chi(G)\leq\Delta(G)+1$

⇓

maximum degree in G

proof: later

for $\chi(G)=1,2$:

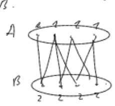
Proposition: $\chi(G)=1 \Leftrightarrow E(G)=\emptyset$

(empty graph - isolated vertices)

$\chi(G)=2$ $\Rightarrow$ bipartite graphs

Def'n The graph $G=(V,E)$ is bipartite, if the vertices of G can be partitioned into 2 classes: A, B.

(i.e. $V(G)=A\cup B$, $A\cap B=\emptyset$), and the edges can only go between A and B.



2-kind of vertices:

boys - girls

people - jobs

⋮

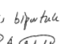
Proposition: $\chi(G)=2 \Leftrightarrow G$ is bipartite (and not the empty graph)

vertices of color = 1 = A

— " — 2 = B

Thm: G is bipartite $\Leftrightarrow$ contains no odd cycles

proof: $\Rightarrow$

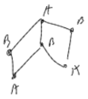


a cycle alternate between A and B.

$A\rightarrow B\rightarrow A\rightarrow B$ must be even

$\Leftarrow$: next week (algorithm)

Graph Coloring



Def'n: A graph $G=(V,E)$ is bipartite, if ∃ a partition on $V(G) = A \cup B$, s.t. all the edges go between A and B.

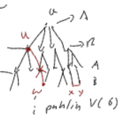
Prop: G is bipartite ⟺ $\chi(G)=2$

Th: G is bipartite ⟺ they contain no odd cycles

proof: ⟹: cycles are even, alternate A B A B A ...

⟸: if G has no odd cycles, then we can find the partition.

Start from v —

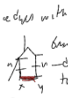


levels: 1st A, 2nd B — ; partition $V(G)$

if G is connected (otherwise we do it in each component)

This partition is good for the def'n:

- ∄ edges jumping over levels: w would be a closer neighbor
- ∄ edges within levels



any +1 edge → cycle; n — distance through levels to the parent; $2n+1$

if z = the first common ancestor of x, y → get an odd cycle: $z \to x \to y \to$ path of the same length

remark: 1.) this gives a simple algorithm for the bipartition → 2 coloring

1. Trees are bipartite (no cycles at all)

Def'n: a (proper) k-coloring of $G=(V,E)$ is an assignment of a set of k colors to vertices s.t. adjacent vertices get different colors.

7.1 Chromatic number of G = min. # of colors needed to color the vertices of G. $\chi(G)$.

i.e. $\chi(G)=k$ ⟺ 1. G is k-colorable, 2. G is not k−1-colorable

1.) Lower bound

$\chi(G) \ge \omega(G)$ = size of the maximum complete subgraph (clique)

2.) How good is this lower bound?

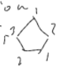
it can be sharp in many cases (i.e. $\chi(G)=\omega(G)$). e.g.: K_n, bipartite ..

— but: sometimes it can be very bad..

$\chi(G)-\omega(G)$ can be arbitrarily large.

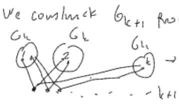
Thm: There is a sequence of graphs: $G_3, G_4, \dots$ s.t. $\chi(G_k)=k$, but clique $\omega(G_k)=2$

Proof: a recursive construction. Zykov's construction. The starting graph: G_3



$\omega(C_5)=2$
$\chi(C_5)=3$

We construct G_{k+1} from G_k



→ k copies of G_k; k+1 all of them.

We pick 1 vertex from each G_k in all possible ways, and we connect them to a new vertex (each time) (i.e. $n_k \cdot n_k \cdots n_k = n_k^k$ new vertices) $N|G_k|$

(no edges between the new vertices)

then $\omega(G_{k+1})=2$ (did not make any △'s)

$\chi(G_{k+1}) \ge k+1$ need an extra color

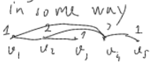
we pick vertices get different colors in the k copies of G_k, then the vertex adjacent to them in G_{k+1} has to get a new color $k+1$.

and $\chi(G_{k+1}) \le k+1$: we can color all the new vertices with color $k+1$ → $k+1$ colors are enough. (same k colors for G_k's).

2.) upper bound $\chi(G) \le \Delta(G)+1$ (max. degree of G)

idea: if we can color G with certain k colors → $\chi(G) \le k$ (min # of colors)

a way to do it is the greedy coloring (do whatever is best at this moment). Color the vertices of G in some way



We assign the smallest possible colors in this order.

Proposition: The greedy coloring uses $\le \Delta(G)+1$ colors

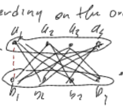
proof:

Color v_j depends only the neighbors of v_j preceding it.

if v_j has d preceding neighbors ⟹ $\le d$ colors are excluded from the possible colors of v_j ⟹ can always select a color $\le d+1$ for v_j. and $d \le \deg(v_j) \le \Delta(G)$

remarks: (.) the greedy coloring can be much worse bound than the optimum $\chi(G)$ depending on the order of vertices.

e.g.



complete bipartite graph − independent edges.

if order: $a_1, b_1, a_2, b_2, a_3, b_3, a_4, b_4$
order 1 1 2 2 3 3 4 4
but $\chi(G)=2$

but the order: $a_1\ a_2\ a_3\ a_4\ b_1 \dots b_4$
1 1 1 1 2 2 → 2

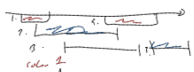
2.) how sharp is $\chi(G) \le \Delta(G)+1$
= for $-K_n$
$-C_{2n+1}$ (odd cycle)

(but for any other conn. graph $\chi(G) \le \Delta(G)$)

Interval Graphs

e.g. jobs to be performed at given times (intervals) distribute them among people/processors.

Overlapping intervals → conflict



$\omega(G)=3$
$\chi(G)=3$

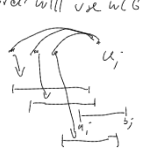
conflict free assignment → coloring of G

Def'n: $G=(V,E)$ is an interval graph, if its vertices can be assigned to (finite, closed) intervals on the real time s.t. 2 vertices are adjacent ⟺ corresponding intervals intersect.

Thm: If G is an interval graph, then $\chi(G)=\omega(G)$, and the optimal coloring can be found using the greedy coloring.

proof: if we know an interval representation of G. i.e. $v_k \leftrightarrow [a_k, b_k]$ → time; then consider ordering of the vertices $v_1, \dots, v_n$ according to the representing intervals (i.e. $a_1 \le a_2 \le \dots \le a_n$)

The greedy coloring in this order will use $\omega(G)$ colors.



α preceding → all must contain a_j
→ they all intersect
→ they intersect each other as well → a $\alpha+1$ clique (including v_j)

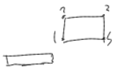
and the color of v_j is $\le \alpha+1$ as before.

⟹ $\chi(G) \le \omega(G)$ (never use more colors than $\omega(G)$)

but $\chi(G) \ge \omega(G)$ ∀G

⟹ $\chi(G)=\omega(G)$ for interval graphs.

Homework: is C_4 an interval graph.

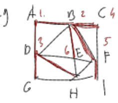


BFS algorithms

coloring of bipartite graphs

BFS: breadth first search

e.g.



algorithm: start from a vertex, say A; visit its neighbors then visit all the neighbors in the order of the already visited vertices (repeated) until all the vertices are visited if the graph is conn.

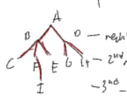
→ FIFO list: first in: first out

e.g. A; B D; C E F; G H I ... (neighbors of A, neighbor of B, neighbors of F)

can be used:

- check if G is connected (if all the vertices get into the list)
- find the components of G (by restarting BFS)
- find the distance of vertices from the starting vertex

$\text{dist}(u,v) = \text{length}(\min(P_{u \to v}))$ (# of edges)



D — neighbors ⟺ dist 1
— 2nd neighbors ⟺ dist 2
— 3rd — ⟺ dist 3

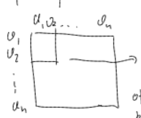
- can find a spanning tree (if G is conn)
- can find a spanning tree (if G is conn) the edges on which we visit a new vertex.
- spanning tree
  - reach all the vertices
  - conn (path from u)
  - no cycle (when we add an edge it's between different components)

\# of steps

a) possible data structure

$v_1 \to$ neighbors (adjacency list):
⋮ — " —
⋮ — " —
v_n — " —

b) Adjacency matrix: $|V| \times |V|$ (n, n)



→ the number of edges between v_i and v_j

\# of steps:

a) check the list of the vertex when we visit it → # of steps $c \cdot$ (total length of the lists)
$= c \cdot 2e = c'e + n$ (if G is not connected)
$= c(n+e)$ steps (size of the adjacency list)
→ linear algorithm

b) # of steps:

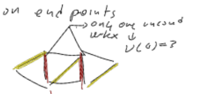
we have to check the row (or column) of the vertex when we visit it ⟹ # of steps $\le c \cdot n \cdot n = cn^2 = cI$

linear: length of input: cn^2 (but the input itself can be much larger)

Matching and other graph parameters.

Def'n 1: a) in a graph $G=(V,E)$ a matching is a set of independent edges, i.e. a set of edges without common end points

e.g.



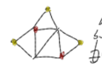
→ only one uncovered vertex; $\nu(G)=3$; matching

b) the # of edges in a maximum matching of G is $\nu(G)$, since $\nu(G) \le \left\lfloor \frac{|V(G)|}{2} \right\rfloor$

c) if for a matching M, $|M| = \frac{|V(G)|}{2}$ ⟹ M → is a perfect matching (size is the max)

Def'n 2: a) An independent set of vertices in $G=(V,E)$ (stable set) is a set of pairwise non-adjacent vertices.

e.g.



$\alpha(G)=3$; to show that ∄4 independent vertices!

b) The maximum number of independent vertices in G is $\alpha(G)$.

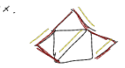
Remark:

independent set of vertices ⟺ induced subgraph is empty
clique ⟺ — " — is complete
$\alpha(G)=\omega(\bar{G})$

Covering edges and vertices

Def'n 3: a) An edge cover in a graph $G=(V,E)$ is a set of edges covering all the vertices, ≥1 edge from it incident to each vertex.

e.g.



$\rho(G)=4$

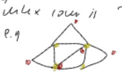
b) the minimum # of edges needed to cover is $\rho(G)$ (rho)

in general: $\rho(G) \ge \left\lceil \frac{|V(G)|}{2} \right\rceil$ (each cover 2 vertices, + might coincide)

Def'n 4: A vertex cover in a graph $G=(V,E)$ is a set of vertices covering each edge, i.e. each edge is incident to ≥1 vertex in the set.

b) the minimum size of a vertex cover is $\tau(G)$ (tau)

e.g.



$\tau(G)=4$

	edges	vertices	
independent	$\nu(G)$	$\alpha(G)$	← max
covering	ρ	τ	← min

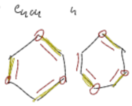
e.g. 1.) K_n: $\nu(K_n) = \left\lfloor \frac{n}{2} \right\rfloor$

$\alpha(K_n) = 1$ (if no loops)
$\rho(K_n) = \left\lceil \frac{n}{2} \right\rceil$ but $\alpha \le 2$
$\tau(K_n) = n-1$ any 2 when we unjoint

(cover the edges incident to the missing vertex at the other endpoint
− n−2 is not enough: the edge between the missing vertices is not covered)

last one is covered by the other edges

2.) cycle n



$\nu(C_n) = \left\lfloor \frac{n}{2} \right\rfloor$
$\alpha(C_n) = \left\lfloor \frac{n}{2} \right\rfloor$
$\rho(C_n) = \left\lceil \frac{n}{2} \right\rceil$
$\tau(C_n) = \left\lceil \frac{n}{2} \right\rceil$

Proposition 1:

$\nu(G) \le \tau(G)$ ∀G

proof: You need a separate vertex to color each edge

Matching

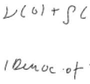
if M is a matching in G ⟹ to cover the edges of M we need ≥|M| vertices because the edges of M have to be covered by distinct vertices.

$\forall |M| \le \nu(G) \le \tau(G)$ (we need ≥ $\nu(G)$ vertices; min # of covering vertices)

Proposition 2:

$\alpha(G) \le \rho(G)$

Proof:



every independent vertex needs a separate edge to cover the vertices in an independent set.

Thm 1:

$\alpha(G)+\tau(G)=n$ (|V(G)|)
if G has no loops (simple graphs)

Thm 2: (Gallai)

$\nu(G)+\rho(G)=n$ → if G has no isolated vertices)

Proof of Thm 1:

X independent; $V(G)-X$ is a vertex cover

Graph parameters

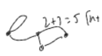
- independent edges (matching) → pairwise nonadjacent → max: $\nu(G)$
- independent vertices —"— max: $\alpha(G)$
- edge cover – set of edges covering all vertices → min: $\rho(G)$
- vertex cover – set of vertices covering all the edges → min: $\tau(G)$

Proposition:

$\nu(G) \le \tau(G)$

$\alpha(G) \le \rho(G)$

Th:



1) $\alpha(G) + \tau(G) = n = |V(G)|$ if ∄ loops

2) $\nu(G) + \rho(G) = n$ $|V(G)|$ if ∄ isolated vertex
how would you edge cover

proof 1:

X is independent ⇔ ∄ edges inside X ⇔ edges: with in V\X have 1 endpoint in V\X
⇔ V\X is a vertex cover

– it can be reversed:

→ So: X is independent ⇔ V\X is a vertex cover

if X is max → $|X| = \alpha(G)$ → $|V \setminus X| \ge \tau(G)$ ⇒ $n - \alpha \ge \tau$

if V – X is min → $|V - X| = \tau(G)$ → $|X| \le \alpha$ ⇒ $n - \tau \le \alpha$

$n - \alpha \ge \tau$

$n - \alpha \le \alpha$

⇒ $n - \alpha = \tau$

$\tau + \alpha = n$

2) a) start from a matching M

→ construct an edge cover C:
- edges of M
- 1 edge for each vertex not covered by M

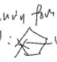
$|C| = |M| +$ # of vertices not covered by M
$= |M| + n - 2|M| = n - |M|$

if M is max ⇒ $|M| = \nu(G)$ →
⇒ $|C| = n - |M| \ge \rho(G)$
⇒ $\nu + \rho \le n$

b) start from an edge cover C as a spanning subgraph

let k be the # of components of C

⇒ $|C| \ge n - k$ = # edges of a spanning forest with k components

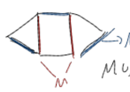
(actually, if C is minimum it's a spanning forest, because it has no cycles:  we can remove)

→ Construct a matching: pick an arbitrary edge from each component of C → independent edges
→ $|M| = k$

if $|C| = \rho(G) = n - k$ → $|M| \ge k$
→ $\nu \ge k$ → $\rho + \nu \ge n$

Matching in bipartite graph

How to find a larger matching? (in any graph)



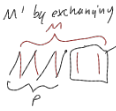
Def'n:

1.) for a given matching $M \in G = (V, E)$:

1.) an alternating path: a path with every 2nd edge from M

2.) an augmenting path: an alternating path + 2 endvertices not covered by M

Proposition: if ∃ an augmented path for M, ⇒ we can collect a 1 larger matching M' by exchanging the edges in M.



(and keep the edges of M not on the path)

Th (Berge)

A matching is M is maximum (in any graph G) ⇔ ∄ augmenting paths to it

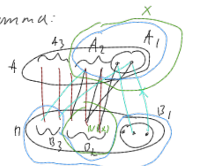
no proof.

→ gives algorithm which (is difficult in general graphs but in bipartite graphs it's simple)

—only bipartite graphs

construct M for which ∄ augmenting path.

lemma:



$G = (A, B, E)$

M is a matching s.t. ∄ an augmented path to it

A_1, B_1: vertices not covered by M

A_2, B_2: vertices that can be reached on alternating paths from A_1

A_3, B_3: sets (covered by M)

Th ∄ edges G from $A_1 \cup A_2$ to $B_1 \cup B_3$

proof: checking cases:

– from A_1 to B_1: would be 1-edge augmented path to M, ↯

– from A_1 to B_3: start of an alternating path ↯

– from A_2 to B_1: ∃ an alternating path to A_1 + 1 edge = augmented path ↯

– from A_2 to B_3: you get a bigger alternating path to B_3 ↯ (by def'n).

Th: König: in bipartite graphs $\nu(G) = \tau(G)$

proof: consider M which cannot be extended by augmenting path, then $A_3 \cup B_2$ is a vertex cover in G.

lemma ⇒ ∄ "outgoing" $A_3 \cup B_2$

$|$vertex cover$| = M$ → for each edge of M we picked 1 endpoint
$|A_3 \cup B_2|$

→ $\tau(G) \le |M| \le \nu(G)$ (min ... max)

but $\nu(G) \le \tau(G)$ in each graph } → $\nu(G) = \tau(G)$

Corollary:

In bipartite graphs $\alpha(G) = \rho(G)$, if ∄ isolated vertices.

proof: $\alpha + \tau = n$ (∄ loops in bipartite graphs)
$\nu + \rho = n$ – ∄ isolated vertices
⇒ $\alpha = \rho$

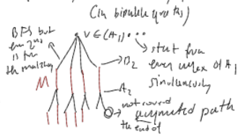
Algorithm of finding a max matching in a bipartite graph G

1) start from the empty matching (or a set of independent edges)

2) keep using augmenting path

3.) if ∄ more augmenting path, then we can find a min vertex cover (= |M|) ⇒ matching is max.

Finding the augmenting path (in bipartite graphs)



→ polynomial algorithm $\le \frac{n}{2} \cdot (\approx BFS)$

Max matching in bipartite graphs

$|M| \le \min\{|A|, |B|\}$
↓
a matching covering A is maximum

Def'n: neighborhood of a set of vertices X in $G = (V, E)$

$N_G(X) = \{v \in V(G) : \exists u \in X, \{u, v\} \in E(G)\}$

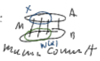
for bipartite graphs, if $X \subseteq A$ → $N(X) \subseteq B$

Th (Hall): in a bipartite graph $G = (A, B; E)$ ∃ a matching covering A

⇔ $|N(X)| \ge |X|$ $\forall X \subseteq A$ (Hall's condition)

proof of Hall's th:

⇒:



if ∃ matching covering A

if $X \subseteq A$ (arbitrary)

→ $N(X) \supseteq$ pairs of X by M

⇒ $|N(X)| \ge |X|$

⇐: prove the contrapositive

if ∄ no matching covering A ⇒ ∃ $X \subseteq A$ s.t. $|N(X)| < |X|$

with the notation of the lemma

let $X = A_1 \cup A_2$

by lemma ⇒ $N_X(X) = B_2$

but $|A_2| = |B_2|$ ⇒ $|A_1 \cup A_2| > |B_2|$ (X, N(X))

Corollary:

1.) Th: (FROBENIUS)

A bipartite graph has a perfect matching

⇔ 1) $|A| = |B|$

2) Hall's condition: $|N(X)| \ge |X|$ $\forall X \subseteq A$

2.) Th:

every regular bipartite graph has a perfect matching

Th (König) in bipartite graphs $\nu(G) = \tau(G)$

Th (Hall) in a —"— $G = (A, B, E)$

∃ a matching covering A

⇔ $\forall X \subseteq A$: $|N(X)| \ge |X|$

Th (Frobenius): in $G = (A, B, E)$ ∃ a perfect matching ⇔

1. $|A| = |B|$

proof: ⇒ if ∃ a perfect matching

→ matches vertices A with B ⇒ $|A| = |B|$ 1.)

 2.): "pairs" of X are in N(X)

N(X) ⇐ 1.) + 2. both ⇒ ∃ a matching covering (A by hall's theorem), and this must cover all the vertices of B as well.

Th: every regular bipartite graph contains a perfect matching.

proof: in the practice.

Edge colouring of graphs

e.g. school: teachers, classes. } bipartite graph

who teaches which class

A. teachers  B. class

class 1 per classes matching

weekly schedule ⇔ covering of the edges by matching
⇕
edge colouring

Monday → red matching

Tuesday → black matching

⋮

Friday → blue edges

Def'n: an edge colouring of $G = (V, E)$: an assignment of colors s.t. any 2 adjacent edges get different colors.

2.) the edge-chromatic number of G = min # of necessary colors

notation: $\chi_e(G)$.

remarks:

1.) instead of colors we can use numbers.

2.) loops: can be, but they can be colored after the other edges are colored. → can

flows in networks

network: - system of roads
 - —"— pipelines } with capacities

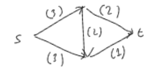
flow: - goods
 - water / oil

problem: want to transfer as much goods / water / oil on the network as possible.

Def'n: a network is an n-tuple $(\vec{G}, s, t, c)$ s.t
G: directed graph $s,t \in V(G)$, s: source, t: sink

C: capacity function on the edges: $c: E(\vec{G}) \to \mathbb{R}_0^+$

e.g.



Def'n: Flow in the network (G,s,t,c) is a function $f: E(G) \to \mathbb{R}_0^+$, s.t

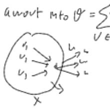
1.) $0 \le f(e) \le c(e)$

2.) conservation (Kirchhoff's law): amount of flow into v = amount of flow out of v $\forall v \in V(G)$, $v \ne s,t$

notation: amount out of $v = \sum_{w \in V(G)} f(v,w) = f^+(v)$

amount into $v = \sum_{u \in V(G)} f(u,v) = f^-(v)$

In general:

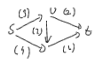


if $X \subseteq V(G)$ $f^+(X) = \sum_{u \in X, v \in V \setminus X} f(u,v)$

$f^-(X) = \sum_{u \in V\setminus X, v \in X} f(u,v)$ $f^+(v) - f^-(v) = 0$

remark: if $X = \{v\} \Rightarrow f^+(X) = f^+(v)$ $f^-(X) = f^-(v)$

Def'n: the amount of the flow f:
$m(f)$ = net flow out of $s = f^+(s) - f^-(s)$



$m(f) = 2+1 = 3$

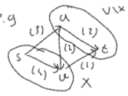
Proposition 1: $m(f)$ = net amount into $t = f^-(t) - f^+(t)$

proof: $\sum_{v \in V(G)} (f^+(v) - f^-(v)) = \underbrace{f^+(s) - f^-(s)}_{m(f)} + f^+(t) - f^-(t)$ ($v \ne s,t$: conservation law)

$= 0$ ($\forall e = (u,v) \to +f(e)$ - at u, $-f(e)$ - at v } cancel)

Def'n: an s,t-cut C in the network (G,s,t,c) is the set of edges between X and $V \setminus X$, where $X \subseteq V(G)$ s.t, $s \in X$, $t \notin X$ (i.e separates the source and the sink)

e.g.

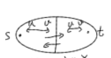


if $X = \{s,u\}$
then $C = \{(s,v), (u,v), (u,t)\}$
$c(C) = 3+1+1 = 5$

Def'n: the capacity of the s,t-cut C,
$c(C) = \sum_{u \in X, v \in V\setminus X} c(u,v)$ $(= c^+(X))$

Proposition 2: $m(f)$ = net flow through the s,t-cut C, $\forall C$
$m(f) = f^+(X) - f^-(X)$

proof:
$\sum_{v \in X} (f^+(v) - f^-(v)) = \underbrace{f^+(s) - f^-(s)}_{s \in X} + \underbrace{0}_{v \ne s}$



- if $u,v \in X \to f(u,v)$ cancels (at u^+, at v^-)
- if $u \in X, v \in V\setminus X \to +f(u,v)$
- if $u \in V\setminus X, v \in X \to -f(u,v)$
- if $u,v \in V \setminus X \to$ doesn't appear

$\underbrace{f^+(X) - f^-(X)}_{\text{net flow through } C} = m(f)$

Proposition 3: for every flow in (G,s,t,c) and $\forall$ s,t-cut C:
$m(f) \le c(C)$

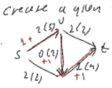
proof: by Prop. 2 $m(f) = f^+(X) - f^-(X)$ (if C is defined by X)

$m(f) = f^+(X) - f^-(X) \le c^+(X) = c(C)$

Algorithm for finding a max flow

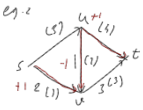
To increase a given flow

e.g. 1.



if ∃ directed path from s→t st ∀e on it $f(e) < c(e)$ (i.e e is not full)
→ can increase over the path

e.g. 2



a path s→t, but has backward edges

- path s→t on forward edges: $f(e) < c(e)$ (not full)
- backward edges: $f(e) > 0$ (not empty)

Def'n: a path like this is an augmented path for a given flow.

remark: on an augmenting path we can increase $m(f)$ by $\min\left\{\min_{e \text{ is forward}}\{c(e) - f(e)\}, \min_{e \text{ is backward}}\{f(e) - 0\}\right\}$

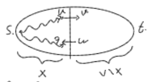
Thm: the following are equivalent for a given flow in (G,s,t,c).
- 1. f is a maximum flow
- 2. ∄ augmenting path for f
- 3. ∃ s,t-cut C: $m(f) = c(C)$

(contrapositive) 1) ⇒ 2) ∃ augm. path ⇒ we could increase the flow

2) ⇒ 3.) ∄ augm. path → f ⇒ we can construct an s,t-cut C (by defining X)

$X = \{u \in V$, s.t. ∃ "beginning of an augmented path from s"$\}$

⇒ $s \in X$, $t \notin X$ (condition) ⇒ X defines an s,t-cut



edges in the cut.
- out of X: (u,v), $u \in X$, $v \in V\setminus X$; ∃ a beginning of an augmented path to u, but we cannot continue it $u \to (u,v)$ is full (not forward edge)
- edge into X: (v,w), $w \in X$, $v \in V\setminus X$; ∃ a beginning of an augmented path to w, but not to $w \to (v,w)$ cannot be a backward edge → $f(e) = 0$

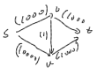
Prop 2. $m(f) = f^+(X) - f^-(X) = c^+(X) + 0 = c(C)$.

3.) ⇒ 1.) Prop 3. $m(f) = c(C)$ for some C ⇒ 3. ($m(f) \le c(C)$ ∀) → f is max

Thm (Ford - Fulkerson or max flow - min cut)

$$\max_{f \text{ flow}} m(f) = \min_{s,t\text{-cut}} c(C)$$

e.g. augmenting paths can take very long:



max $m(f) = 2000$
$c(C) = 2000$ ($X = \{s\}$)
s→u→v→t: 1, s→v→u→t: 1 } 2000 such path
but: s→u→t: 1000, s→v→t: 1000 } 2 paths

Thm (EDMONDS-KARP) if we always use (one of) the shortest path(s), then we can get maximum flow after polynomially many augmentations.
no proof.

Flows in networks,

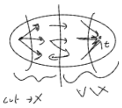
network: (G,s,t,c)

flow: $f: E(G) \to \mathbb{R}_0^+$ s.t

1.) $0 \le f(e) \le c(e)$ $\forall e \in E(G)$

2.) $f^-(v) = f^+(v)$ $\forall v \ne s,t$

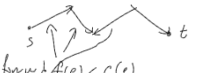
value of flow: $m(f) = f^+(s) - f^-(s)$



$= f^+(X) - f^-(X)$
$\le c(C) = c^+(X)$

s,t cut: X
$s \in X$, $t \in V\setminus X$

Augmenting path.



forward $f(e) < c(e)$
backward $f(e) > 0$

Thm: f is a max. flow in (G,s,t,c)
1. max flow
2. ∄ a augmenting path
1. ∃ st-cut C: $m(f) = c(C)$

Thm (Ford-Fulkerson, max flow - min cut):
$\max_{f \text{ flow}} m(f) = \min_{s,t\text{-cut}} c(C)$

proof: if $m(f) = c(C)$ → f is max by previous
↳ C is min: $m(f) \le c(C)$ $\forall f, C$

if = → f is max, C is min

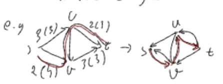
remark: "min max thm"
2.) before: in bipartite graph
$\nu(G) = \tau(G)$
$|M| \le |V_C|$

algorithm for finding a max. flow
1.) start from the all 0 flow.
2.) start using augmenting paths
3.) if ∄ more augmenting paths
→ find a cut with $c(C) = m(f)$
→ f is max (and C is min)

Thm: (EDMONDS - KARP): the above algorithm finds a max flow in any network in polynomial time (with $\le n \cdot e$ augmenting paths) if we always use (one of) the augmenting path(s).
no proof.

Finding an augmenting path for a given flow:
Construct an auxiliary graph G_f:
$V(G_f) = V(G)$
$E(G) = \{(u,v): f(u,v) \le c(u,v)\}$ (forward edges)
$\cup \{(u,v), f(v,u) > 0\}$
backward edges

e.g.



augmenting path ⇔ directed s,t path to f

find the shortest one: use the BFS

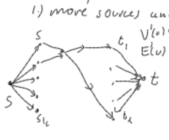
Lemma (Integrality lemma)
if all the capacities in the network (G,s,t,c) are integers ⇒ ∃ a maximum flow s.t $f(e)$ is an integer $\forall e \in E(G)$

proof: augmenting path algorithm: all 0 flow - integer then each augmenting path has integer values on the edges.

$$\min \begin{cases} \min\{\min\{c(e) - f(e)\}\} \\ \min\{\min\{f(e)\}\} \end{cases}$$

this is an integer by induction and $f(e) = \sum \pm$ values of augmenting paths ∋ e (integer)

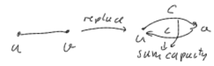
remark: generalization
1.) more sources and more sinks



$V'(G) = V(G) \cup \{s,t\}$
$E(G) = E(G) \cup \{(s,s_i), i = 1,\dots k\} \cup \{(t_j, t), j = 1,\dots l\}$

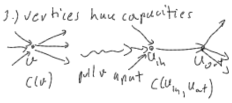
new capacities: large enough

2.) undirected edges



$\{u,v\} \to (u,v) \cup (v,u)$
pair of undirected edges

3.) vertices have capacities



$c(v_{in}, v_{out})$

4.) many types of products: more difficult

Menger's theorems

1) flow → higher connectivity of graphs
2) mini-max theorems
versions: {directed, undirected} graphs - {edges, vertices}

Def'n: G is a {directed, undirected} graph, a set of s-t paths is
- pairwise edge disjoint: no two of the paths share an edge
- pairwise vertex-disjoint: no two of them share an intermediate vertex ($\ne s,t$)

e.g.

3 pairwise edge-disjoint s-t paths
2 - pairwise vertex disjoint s-t paths

Def'n 1). a set of edges covering all s-t paths: ∃ ≥ 1 edge on each s-t path
2.) a set of vertices —"—: ∃ ≥ 1 vertex —"—, not including s and t.

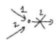
3 edges covering all s-t paths
2 vertices covering all s-t paths

Thm: In a directed graph $G = (V,E)$, $s,t \in V(G)$: the max # of pairwise edge disjoint directed s-t path = min # of edges covering all directed s-t paths.

proof: max ≤ min
we need a separate edge to cover all the p.w edge disjoint path

2: use flows: network (G,s,t,c)
$c(e) = 1$ $\forall e \in E(G)$

→ Fulkerson - Ford thm:
$\max m(f) = \min c(C)$
Integrality lemma
⇒ ∃ max flow f with integer values on each edges.
⇒ $f(e) = \{0, 1\}$ $\forall e \in E(G)$



the edges with $f(e) = 1$ define edge - disjoint s-t paths,
(+ directed cycles → don't contribute $m(f)$)

→ if $m(f) = k$ → k edge-disjoint s-t paths
→ ∃ s,t cut C: $c(C) = k$
∃ X (∋ s, ∌ t)
k edges go out of X
→ these k edges cover all the directed s-t paths.

min # of edges covering all s-t path
≤ k ≤
max # of p.w edge-disjoint s-t path.

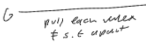
Thm (Menger 2):
G is a directed graph, $s,t \in V(G)$, s,t $(s,t) \notin E(G)$.
the max # of p.w vertex-disjoint s-t paths = min # of vertices (without s and t) covering all s-t paths.

remark: without s and t we cannot cover (s,t) → hence $E(G)$

proof: ≤ ✓ (need a separate vertex to cover each p.w vertex-disjoint paths)

≥: reduce it to the 1
we make edges out of vertices

G → G': pull each vertex ≠ s,t apart



p.w vertex-disjoint path → p.w edge-disjoint path

vertex in G ↔ edge in G'
pick one of the endpoints ≠ s,t

- edges covering s-t path

max # of p.w edge-disjoint s→t paths in G' = the min # of edges covering all s-t paths in G'

in G: max # of p.w vertex-disjoint s-t path ≥ min # of vertex in $G \setminus \{s,t\}$ covering all s-t paths.

+ 2 more versions with undirected graph

Menger's theorem

Th (Menger 1,2) G is a directed graph, $s,t \in V(G)$, $\{(s,t) \in E(G)\}$ then the max # of {edge / vertex} disjoint directed $s \to t$ paths =

= min # of {edges / vertices} that cover all s-t paths {— / without s,t}

Th: (Mengers 3) G is an undirected graph then the max # of edge-disjoint s-t path = min # of edges covering all s-t path.

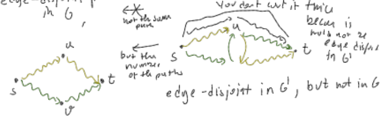
proof:
$\leq$: we need a seperate edge for each of the edges-disjoint paths

$\geq$: reduce it to Menger 1

G undirected ⟶ G' directed graph
V(G) = V(G')

u —— v ⟶ u ⇄ v $E(G) = \{\{(u,v),(v,u)\}, \{u,v\} \in E(G)\}$

edge-disjoint path in G ⟶ edge disjoint paths in G'



this doesn't change the number of edge disjoint s-t paths.

we can solve all such problems 1 by 1 (no new such problem can arive)
it will terminate:
the total length of path decreases (by 2) each time (it has to remain > 0)

→ max # of edge disjoint paths in G (A)
$\geq$
max # of —ıı— $s \to t$ —ıı— G' (A')

in G | in G'
edges covering all s-t path ⟵ (forget the direction) edges covering all directed s-t paths

⇒ min # of edges covering all s-t paths in G B
$\leq$ min # of edges (directed) —ıı— $s \to t$ path in G' B'

by Thm 1

$A \geq A' = B' \geq B$

Thm (Menger 4): G is an undirected graph, $\{s,t\} \notin E(G)$
⇒
max # of vertex-disjoint s-t paths
=
min # of vertices (without s and t) covering all s-t paths

no proof.

Higher connectivity of graphs

network: should be connected
now: if we destroy a not big part of it, then it is still connected. (either edges or vertices)

Def'n: a graph $G=(V,E)$ is k-edge conneted if no matter how we delete $\leq k-1$ edges the remaining graph is still connected.

Def'n: the edge-connectivity number of G: heighest k, s.t G is k-edge-conn

remark: $\lambda(G) = k$:
1. if we delete (any) k-1 edges, still conn.
2. ∃ a set of k edges whose deletion disconnects G

Def'n: a graph $G=(V,E)$ is k-vertex-conneted, if $|V(G)| \geq k+1$, and no matter how we delete $\leq k-1$ vertices, the remaining graph is still conn.

Def'n: the (vertex) connetivity number of G is the heighest k, for which G is k-vertex-conn.
notation: $K(G)$

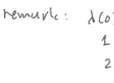
remark $= k \iff |V(G)| \geq k+1$ and
1.) if we delete k-1 vertices, still conn.
2.) ∃ a set of k vertices whose deletion disconnects G.

e.g 1.) trees $\lambda(T)=1$ (→ k=1 edge/vertex-conn) $K(G)=1$ except

⇒ $\lambda(T)=1$ $K(G)=0$
⇒ $\lambda(T)=2$ $K(G)=0$

2.) cycles $\lambda(T)=2$ $K(G)=2$

3.) $\lambda(T)=2$ $K(G)=2$



trees -conn → k = 1 edge/vertex - conn (k-1=0) $1 \leq \lambda(G)$
- delete 1 edge / 1 vertex disconnects it $\lambda(G) \leq 1$

(remark: for k-vertex-conn: $|V| \geq k+1$;
if we delete k-1 vertices ⇒ ≥ 2 vertices are left)

C_n cycle:
1 - {edge / vertex} conn: delete 0 connected
2 - {edge / vertex} conn: if we delete 1 {edge / vertex} → still conn ✓
not 3 - {edge / vertex} conn: delete 2 {edges / vertices} → discon.
→ $\lambda(G) = K(G) = 2$

1 / 0 edge / vertex removed connected
2 / 1 edge / vertex removed: it has HC
3 - edge-conn: { 2 → remaining edges form a spanning tree (still has) ; 1 → edges from HC still has HC ; 0 → other has an HC }
4 - edge-conn: no: min deg(v) = 3

$\lambda(G) \leq$ min deg(v)

$\lambda(v) = 3$ $K(G) = 2$ ⇒ you can find 2 vertices,

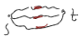
4.)

→ $K(G) = 1$ (from v)
but $\lambda(G) = n-1$

remark: $K(G) \leq \lambda(G) \leq$ min deg(v)

Thm (Menger 5):
G is k-edge-conn ⟺ ∃ ≥k edge-disjoint paths between any 2 vertices

proof: ⇒ contradiction
∃ $s,t \in V(G)$, s.t $\leq k-1$ edge disjoint paths between them, ⇒ by M3. all the s-t paths can be covered by k-1 edges ⇒ if we delete



delete these k-1 edges then disconnect the graph (s,t are in different components) ↯

⟸ k-edge disjoint paths → need ≥ k edges to seperate the vertices

Thm (Menger 6) G is k-vertex-conn ⟺ $|V(G)| \geq k+1$
∃ ≥k vertex-disjoint-paths between any 2 vertices

Corrollary:
if G is k-vertex-connected ⇒ G is k-edge-conn as well

proof:
G is k-vertex-conn $\overset{M6}{\Rightarrow}$ ∃≥k vertex-disjoint paths between any 2 vertices
⇒ ∃≥k edge-disjoint —ıı— ..
$\overset{M5}{\Rightarrow}$ G is k-edge-conn

Corollary:
$K(G) \leq \lambda(G)$

Shortest paths (algorithmic)

length of a path: # of edges
↓
BFS finds the shortest paths from a fixed $s \in V(G)$ to the other vertices

In general: edges have weights
$w: E(G) \to \mathbb{R}$ weight function
length of a path: $\sum_{e \in P} w(e)$

(generalization: if $w(e) = 1$ $\forall e \in E(G)$ → previous)

if $w(e) \geq 0$ $\forall e \in E(G)$ → Dijstra's alg.

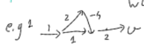
now: $w(e) < 0$ is possible.

(eg: changes of energy in a chemical reaction)

we need an extra condition:
before: consider only directed graphs.
(for undirected graph: u —— v ⇒ u ⇄ v ; undirected edge → pair of directed edges)

Def'n: a weighting on E(G) is conservative if $\sum_{e \in C} w(e) \geq 0$ for every directed cycle C.

e.g 1



$w(\text{cycle}) = -1 < 0$ The length of a shortest walk.

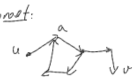
$u \to c \to a \to b - u$: 1
$u \to c \to a \to b \to c \to a \to v$: 0 (−1)
$u \to (c \to a \to b) \to c \to a$: 1−k
smaller and smaller, but not both

eg 2 if $w(e) = -1$ $\forall e$
length of a path $= t(P) = -$(# of edges on P)
the last of this ⟷ longest path in G
max: length of a Hamilton path, if it exist. (difficult)

only OK, if G is a DAG (directed acyclic graph).

Proposition: if the weighting is conservative ⇒ the shortest $u \to v$ walk is a path. More precisely
if ∃ a walk $u \to v$ with ≤k edges of length ≤ C
⇒ ∃ a path $u \to v$ with ≤k edges of length ≤ C

proof:

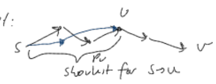


if $a \in V(G)$ apprears again in the walk → we can cut the cycle $a \to a$ from it, → length of the new walk ≤ length of the original (length(cycle) ≥ 0)

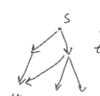
keep doing this, until we get a path.

Corollary: if P is a shortest path $s \to v$ and $u \in P$ ⇒ the initial segment $s \to u$ is a shortest path $s \to u$.

proof:



by contradiction; if ∃ a shorter path P' $s \to u$
⇒ $P' \cup P \setminus P_u$ is a s — v walk shorter than P
can choose a shorter path $s \to v$ by prop ↯



graph of shortest paths from s
tree directed away from s
+ forward edges (if ∃>1 shortest path to a vertex)

algorithm: Bellman-Ford algorithm
G is a directed graph, $w: E(G) \to \mathbb{R}$ a conservative weighting, $s \in V(G)$ fixed
(distance of v from s not symmetric)
$t(v)$ = length of the shortest path $s \to v$ $v \in V(G)$
(if there is ∄ directed s-v path, ⇒ $t(v) = \infty$)

idea: $t_k(v)$ = length of shortest $s \to v$ path ≤k edges
then a recursive formula for t_k's:
- starting: $t_0(v) = \begin{cases} 0; & \text{if } v = s \\ \infty, & \text{if } v \neq s \end{cases}$

$t_1(v) = \begin{cases} 0, & \text{if } v = s \\ w(e), & \text{if } e = (s,v) \in E(G) \\ \infty, & \text{if } (s,v) \notin E(G) \end{cases}$

- recursion:
$t_k(v) = \min_{\substack{u \in V(G) \\ (u,v) \in E(G)}} \{ t_{k-1}(u) + w(u,v) \}$ $\forall v \in V(G)$

end: $t(v) = t_{n-1}(v)$ a path contains ≤ n-1 edges

or we can stop earlier, if $t_{k+1}(v) = t_k(v)$ $\forall v \in V(G)$

this is correct: by corollary the only problem possible:
if v is on the shortest path $s \to u$, $(v,u) \in E(G)$ but that gives a shorter cycle → ∃ a shorter path to u from u', $(u',u) \in E(G)$

# of steps: n rounds (k = 0, ..., n−1) in 1 round:
in 1 round: we use each edge exactly once for the improvement (based on its endpoint)
for each edge: 1. addition, 1 minimum (of 2 numbers) } Ce

→ total number of steps: Cne → polynomial

Some additions:
1.) we can find not only the lengths of the shortest paths, but the shortest paths themselves
general idea (for all shortest path algorithm)
we keep track of the vertex preceding v on the shortest path.

→ $p(v)$: the vertex u on a shortest s-v path of ≤k edges preceding v.
if $t_k = \min_{(u,v) \in E} \{ t_{k-1}(u) + w(u,v) \} = t_{k-1}(u^*) + w(u^*,v)$

then $p_k(v) = \begin{cases} p_{k-1}(v), & \text{if } t_k(v) = t_{k-1}(v) \text{ (no improvements)} \\ u^*, & \text{if } t_k(v) \leq t_{k-1}(v) \end{cases}$

starting value: $p_1(v) = \begin{cases} s, & (s,v) \in E \\ *, & \text{otherwise} \end{cases}$

then the shortest to v: $s \to \dots p_{k_1}(v) \to p_{k-1}(v) \to v$ (backwards from v)

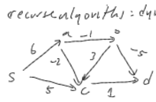
2.) We can check whether the weighting is conservative or not, by using the algorithm:

if we consider $t_n(v)$: n edges → not a path
→ ∃ a directed cycle
if ∃ a directed cycle C, $w(C) < 0$ (and we can reach it from s), then we shall get an improvement in t_n for some v.

if $t_n(v) < t_{n-1}(v)$ for some v ⇒ ∃ must be a directed cycle C $w(C) < 0$, (and it can be reached from s).
in this case is no guarantee that the previous values were correct but if $t_n(v) = t_{n-1}(v)$

remark: recursive algorithms: dynamic programming

e.g



t	s	a	b	c	d
$t_1(v)$	0	6	∞	5	∞
$t_2(v)$	0	$\min\{t_1(s)+6, t_1(b)-1\}$ $\min\{6,\infty\} = 6$	8	$\min\{t_1(s)+5, t_1(a)-2\}$ $\min\{5,4\} = 4$	$\min\{t_1(b)-5, t_1(c)+1\}$ $\min\{\infty, 6\} = 6$
$t_3(v)$	0	$\min\{t_2(s)+6, t_2(b)-1\}$ $\min\{6,7\}$	7	$\min\{t_2(s)+5, t_2(a)-2\} = 4$	$\min\{t_2(b)-5, t_2(c)+1\}$ $= \min\{3,5\}$ 3

P	s	a	b	c	d
$t_1(v)$	s	s	*	s	*
$t_2(v)$	s	s	c	a	c
$t_3(v)$	s	s	c	a	bc
$t_4(v)$					