What is a graph?

What isn’t a graph?!?

What is a graph?

Facebook

Vertices = people  Edges = friendships

Facebook

# vertices $n \approx 10^9$  # edges $m \approx 10^{12}$

World Wide Web

1998 paper on PageRank

Vertices = pages  Edges = hyperlinks ("directed graph")
World Wide Web

1998 paper on PageRank

Today: Perhaps $n \approx 10^{12}$, $m \approx 10^{13}$?

Street Maps

Vertices = intersections  Edges = streets

Zachary Karate Club

34 vertices (karatekas)  78 edges (friendships)

Graphs from images

These are “planar” graphs; drawable with no crossing edges.

Zachary Karate Club CLUB
(networkkarate.tumblr.com)

Register allocation problem

A compiler encounters:

```
  temp1 := a + b
  temp2 := -temp1
  c := temp2 + d
```

5 variables; can it be done with 4 registers?

G. Chaitin (IBM, 1980) breakthrough:
Let variables be vertices. Put edge between $u$ and $v$ if they need to be live at same time.
The least number of registers needed is the chromatic number of the graph.
Register allocation problem
A compiler encounters:

\[
\begin{align*}
\text{temp1} & := a + b \\
\text{temp2} & := -\text{temp1} \\
\text{c} & := \text{temp2} + d
\end{align*}
\]

5 variables; can it be done with 4 registers?

(or something like that)

Warning:
The remainder of the lecture is, like, 100 definitions.

If you’ve seen them all before 10 times, play [http://planarity.net](http://planarity.net) on your phone.

Computer Science Life Lesson:
If your problem has a graph, ☺️.
If your problem doesn’t have a graph, try to make it have a graph.

Definitions

Simple Undirected Graphs
Directed Graphs
General Graphs
(aka annoying graphs)

Why should I change?
He’s the one who sucks!

Acoustic Guitar
Electric Guitar

Definitions

Simple Undirected Graphs
Directed Graphs
General Graphs
(aka annoying graphs)
Definitions

A graph $G$ is a pair $(V,E)$ where:
- $V$ is the finite set of vertices/nodes;
- $E$ is the set of edges.

Each edge $e \in E$ is a pair $\{u,v\}$, where $u,v \in V$ are distinct.

Example:
$V = \{1,2,3,4,5,6\}$
$E = \{ \{1,2\}, \{1,4\}, \{2,4\}, \{3,6\}, \{4,5\} \}$

Notation

$n$ almost always denotes $|V|$

$m$ almost always denotes $|E|$

Edge cases

Question:
Can we have a graph with no edges (m=0)?

Answer:
Yes! For example,
$V = \{1,2,3,4,5,6\}$
$E = \emptyset$

Called the “empty graph” with $n$ vertices.

Edge cases

Question:
Can we have a graph with no vertices?

Answer:
Um...... well......
**Edge cases**

**Question:** Can we have a graph with no vertices?

**Answer:** It’s to convenient to say no. We’ll require $V \neq \emptyset$.

One vertex $(n = 1)$ definitely allowed though. Called the “trivial graph”.

**More terminology**

Suppose $e = \{u,v\} \in E$ is an edge.

We say:
- $u$ and $v$ are the **endpoints** of $e$.
- $u$ and $v$ are **adjacent**.
- $u$ and $v$ are **incident** on $e$.
- $u$ is a **neighbor** of $v$.
- $v$ is a **neighbor** of $u$.

**More terminology**

For $u \in V$ we define $N(u) = \{ v : \{ u,v \} \in E \}$, the **neighborhood** of $u$.

E.g., in the below graph, $N(y) = \{ v,w,z \}$,

$N(z) = \{ y \}$,

$N(x) = \emptyset$.

The degree of $u$ is $\deg(u) = |N(u)|$.

E.g., $\deg(y)=3$, $\deg(z) = 1$, $\deg(x) = 0$.

**Theorem:** Let $G = (V,E)$ be a graph. Then $\sum_{u \in V} \deg(u) = 2|E|$.

**Proof of $\sum_{u \in V} \deg(u) = 2|E|$**:

Tell each vertex to put a “token” on each edge it’s incident to. Vertex $u$ places $\deg(u)$ tokens. So one hand,

total number of tokens = $\sum_{u \in V} \deg(u)$.

On the other hand, each edge ends up with exactly 2 tokens, so

total number of tokens = $2|E|$.

Therefore $\sum_{u \in V} \deg(u) = 2|E|$.

Remark: Classic “double counting” proof.
Question:
In an \( n \)-vertex graph, how large can \( m \) be? (That is, what is the max number of edges?)

Answer: \( \binom{n}{2} = \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n = O(n^2) \)

E.g.: \( n = 5, \quad m = \binom{5}{2} = 10 \).

Called the **complete graph** on \( n \) vertices. Notation: \( K_n \)

A bogus “definition”

If \( m = O(n) \) we say \( G \) is “sparse”. If \( m = \Omega(n^2) \) we say \( G \) is “dense”.

This does not actually make sense. E.g., if \( n = 100, \quad m = 1000 \), is it sparse or dense? Or neither?

It **would** make sense if you had a sequence or family of graphs.

Anyway, it’s handy informal terminology.

Let’s go back to talking about \( K_n \).

In \( K_n \), every vertex has the **same degree**.

This is called being a regular graph.

We say \( G \) is \( d \)-regular if all nodes have degree \( d \).

For example: \( K_n \) is \((n-1)\)-regular; the empty graph is 0-regular.

What about \( d \)-regular for other \( d \)?

1-regular graphs

Possible if and only if \(|V|\) is even.

Such a graph is called a **perfect matching**.

2-regular graphs

Called a 5-cycle

2-regular graph is a disjoint collection of cycles.

3-regular graphs

There are lots and lots of possibilities.
A little about “directed graphs”

First, they have a “celebrity couple”-style nickname, a la:

“Kimye”    “Brangelina”

Now an edge is an ordered pair, $e = (u,v)$.

$G = (V,E)$, where:

$V = \{p, q, r, s, t\}$

$E = \{(p, q), (p, r), (q, r), (r, s), (s, t), (t, s)\}$

A little about “directed graphs”

Now there’s out-degree and in-degree

$\text{deg}_\text{out}(u) = |\{v : (u, v) \in E\}|$

$\text{deg}_\text{in}(u) = |\{v : (v, u) \in E\}|$

E.g.:

$\text{deg}_\text{out}(p) = 2$

$\text{deg}_\text{in}(p) = 0$

$\text{deg}_\text{out}(s) = 1$

$\text{deg}_\text{in}(s) = 2$

Storing graphs on a computer

Two traditional methods:

**Adjacency Matrix**

**Adjacency List**

For both, assume $V = \{1, 2, ..., n\}$.

Our example graph:

[Diagram of a directed graph with vertices 1, 2, 3, 4, 5 and edges (1, 2), (1, 3), (2, 4), (3, 5), (3, 4), (4, 5), (5, 1)]

**Adjacency Matrix**

Adjacency matrix $A$ is $n \times n$ array.

$A[i, j] = \begin{cases} 1 & \text{if } i, j \text{ are adjacent} \\ 0 & \text{if } i, j \text{ not adjacent} \end{cases}$

For digraphs, put 1 iff $i \rightarrow j$ is an edge.

For general graphs, put # edges $i \rightarrow j$.

$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$

**Pros:**

Extremely simple.

$O(1)$ time lookup for whether edge is present/absent.

Can apply linear algebra to graph theory... hmm...

**Cons:**

Always uses $n^2$ space (memory).

Very wasteful for “sparse” graphs ($m \ll n^2$).

Takes $O(n)$ time to enumerate neighbors of a vertex.

Adjacency Matrix
Adjacency List

A length-$n$ array $\text{Adj}$, where $\text{Adj}[i]$ stores a pointer to a list of $i$'s neighbors.

$$\text{Adj} = \begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & 1 & 4 & 1 \\
3 & 1 & 2 & 4 \\
4 & 2 & 3 & \end{array}$$

Storing graphs on a computer

Any other possibilities? Sure!

Adjacency matrix and list were good enough for your grandparents.

But you could do something new and fresh. Maybe add in a hash table to your adj. list.

Here's a graph $G = (V,E)$:

$V = \{1,2,3,4,5,6,7\}$

$E = \{ \{1,3\}, \{1,7\}, \{2,4\}, \{2,6\}, \{3,5\}, \{3,7\}, \{4,6\}, \{5,7\} \}$

Notice anything peculiar about it?

This graph is not connected.

Terminology

A graph $G = (V,E)$ is connected if
$$\forall \ u, v \in V, \ v \text{ is reachable from } u.$$ 

Vertex $v$ is reachable from $u$ if there is a path from $u$ to $v$.

That's correct, but let's say instead: “if there is a walk from $u$ to $v$.”
A walk in $G$ is a sequence of vertices $v_0, v_1, v_2, \ldots, v_n$ (with $n \geq 0$) such that $\{v_{t-1}, v_t\} \in E$ for all $1 \leq t \leq n$.

We say it is a walk from $v_0$ to $v_n$, and its length is $n$.

Example: $(p, q, s, r, p, r, s, t)$ is a walk from $p$ to $t$ of length 7.

A path in $G$ is a walk with no repeated vertices.

Fact:
There is a walk from $u$ to $v$ iff there is a path from $u$ to $v$.

Because you can always “shortcut” any repeated vertices in a walk.

Example:
walk $(p, q, s, r, p, r, s, t)$ “shortcuts” to path $(p, q, s, t)$.

A cycle is a walk (of length at least 3) from $u$ to $u$ in which the only repeated vertex is $u$.

Example:
$(p, r, s, q, p)$ is a cycle of length 4.

This 5-vertex graph is connected.
This 11-vertex graph is not connected. It has 3 connected components: \{p,q,r,s,t\}, \{u,v\}, \{w,x,y,z\}

Claim: “is reachable from” is an equivalence relation

Proof:
• u is reachable from u? ✓
• u reachable from v \iff v reachable from u? ✓
• u is reachable from v, v is reachable from w \Rightarrow u is reachable from w? ✓

Connected components are the equivalence classes.

A little more about digraphs

In a digraph, walks have to “follow the arrows”. Given this, the reachable/walk/path/cycle stuff is all the same, except......

\[ u \text{ reachable from } v \not\iff v \text{ reachable from } u \]

G is strongly connected iff \( \forall u,v \in V, u \text{ is reachable from } v \).

Challenge:

Make an n-vertex graph connected using as few edges as possible.

<table>
<thead>
<tr>
<th>n</th>
<th>Connected Graph</th>
<th>m</th>
<th>Necessary and Sufficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Done</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>m = 1</td>
<td></td>
<td>necessary and sufficient</td>
</tr>
<tr>
<td>3</td>
<td>m = 2</td>
<td></td>
<td>necessary and sufficient</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>Connected Graph</th>
<th>m</th>
<th>Necessary and Sufficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Done</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>m = 1</td>
<td></td>
<td>necessary and sufficient</td>
</tr>
<tr>
<td>3</td>
<td>m = 2</td>
<td></td>
<td>necessary and sufficient</td>
</tr>
<tr>
<td>4</td>
<td>m = 3</td>
<td></td>
<td>necessary and sufficient</td>
</tr>
</tbody>
</table>
**Lemma:**
Let $G$ be a graph with $k$ connected components. Let $G'$ be formed by adding an edge between $u,v \in V$. Then $G'$ has either $k$ or $k-1$ connected components.

**Example $G$ with $k=3$ components:**

**Case 1:** $u,v$ in different components

No cycle created, since it would have to involve $u$ & $v$, but they weren’t previously connected.

**Case 2:** $u,v$ in same component

Still have $k$ components.

**Bonus observation:**
Adding $\{u,v\}$ creates a cycle, since $u,v$ were already connected.
Lemma: Let $G$ be a graph with $k$ connected components. Let $G'$ be formed by adding an edge between $u, v \in V$. Then either: a cycle was created, and $G'$ has $k$ components; or no cycle was created, and $G'$ has $k-1$ components.

Theorem: A connected $n$-vertex graph $G$ has $\geq n-1$ edges.

Proof: Imagine adding in $G$'s edges one by one. Initially, $n$ connected components. Each edge can decrease # components by $\leq 1$. Have to get down to 1. Hence $\geq n-1$ edges. 

Bonus: $G$ has exactly $n-1$ edges iff it's acyclic (has no cycles). Such a graph is called a tree.

Definition/Theorem: An $n$-vertex tree is any graph with at least 2 of the following 3 properties: connected; $n-1$ edges; acyclic. It will also automatically have the third.

Trees

Example trees with $n = 9$ vertices.

Leaf: Vertex of degree 1.

Internal node: Vertex of degree $> 1$.

Rooted tree: Tree with any one vertex designated as “root”. Always drawn with root on top, rest of tree “hanging down” from it.

For rooted trees, we use “family tree” terminology: parent, child, sibling, ancestor, descendant, etc.

Rooted tree: Tree with any one vertex designated as “root”. Always drawn with root on top, rest of tree “hanging down” from it.
For rooted trees, we use “family tree” terminology: parent, child, sibling, ancestor, descendant, etc.

**Binary tree:** (cf. Lecture 2)
Rooted tree where each node has at most two children.

---

Time for some actual **computer science**.

Out of **all** computational problems in computer science, my personal favorite is...

**Max-Cut**

**Input:** A graph $G=(V,E)$.

**Output:** A “2-coloring” of $V$: each vertex designated yellow or blue.

**Goal:** Have as many cut edges as possible. An edge is cut if its endpoints have different colors.

---

**Motivation for Max-Cut**
Say you’re producing a TV show with $n$ castaways.
You know the social network of friendships.
You need to split them into two tribes.

Naturally, as producer, you want to break up as many friendships as possible, to maximize drama-lama.

---

Motivating examples might be more natural if the social network recorded **enemysips**, rather than friendships.

There’s an app for that.

**Enemybook**

“Enemybook is an antisocial utility that disconnects you with the people around you.”

Kevin Matulef
Motivation for Max-Cut
For example, given enemyship statuses for the Zachary Karate Club,

computing Max-Cut might give the best prediction for the schism into two clubs.

A “Local Search” Algorithm

Given input graph \( G \) with \( n \) vertices, \( m \) edges...
- Start with an arbitrary 2-coloring (say, all blue).
- Loop:
  - Check each vertex \( u \) to see if switching its color would increase the number of cut edges.
  - If such a vertex \( u \) is found, switch its color.
  - If no such vertex exists, halt.

Question: Why does this algorithm always halt?
Answer: After each loop iteration, # of cut edges increases by \( \geq 1 \). Can’t go above \( m \).
Corollary: Running time is \( O((m+n)^2) \) (quadratic).

A “Local Search” Algorithm

Given input graph \( G \) with \( n \) vertices, \( m \) edges...
- Start with an arbitrary 2-coloring (say, all blue).
- Loop:
  - Check each vertex \( u \) to see if switching its color would increase the number of cut edges.
  - If such a vertex \( u \) is found, switch its color.
  - If no such vertex exists, halt.

Observation: In final 2-coloring, each vertex \( u \) has at least \( \deg(u)/2 \) of its enemyships (edges) cut. (Why?)
Conclusion: Guaranteed to get \( \geq \frac{m}{2} \) cut edges. (Exercise.)

This algorithm is pretty good. Is it optimal?
This algorithm is pretty good. Is it optimal?

Maybe algorithms get as far as this. \( \frac{m}{2} \) edges cut
No single color switch gives any improvement.

But the optimum 2-color cuts all \( m \) edges.

So is there a better polynomial-time algorithm?
Stay tuned…

Definitions:
Seriously, there were about 100 of them.

Theorems:
Sum of degrees = 2|E|.
The Theorem/Definition of trees.
Max-Cut local search analysis.

Study Guide